E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

Desarrollo de un software de control y visualización para sistemas de monitorización de tráfico basados en sensores de fibra óptica.

Grado en Ingeniería en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Autor: Isabel Jaso Gallego

Directores: Rosa Ana Pérez Herrera

Mikel Bravo Acha

Pamplona, 23-Octubre-2020

upna Universidad Pública de Navarra Nafarroako Unibertsitate Publikoa

Resumen

En este documento se describen dos sub-proyectos dedicados a la adquisición y procesado de datos generados por un sistema de monitorización del tráfico basado en redes de difracción de Bragg en fibra (FBGs) situadas en estructuras de asfalto.

El primer sub-proyecto consiste en el desarrollo en MATLAB de un software para un interrogador comercial de FBGs de alta velocidad con el fin de obtener los datos generados en la red de sensores y almacenarlos para su uso y análisis en el laboratorio.

El segundo sub-proyecto describe el desarrollo de un sistema para acceder a los datos generados por el sistema de forma sencilla y comprensible desde un ordenador fuera del laboratorio. El lenguaje de programación empleado para ello es Python.

Para garantizar que este documento sea comprensible, el primer capítulo contiene una introducción teórica a conceptos como la fibra óptica, los sensores de fibra óptica, las redes de difracción de Bragg... etc. El objetivo de este capítulo es ejercer como una introducción rápida y accesible a aquellas personas que no sean familiares con dichos conceptos. El segundo y tercer capítulo describen el funcionamiento y desarrollo del software diseñado en el primer y segundo sub-proyecto respectivamente. Los códigos diseñados para cada uno han sido incluidos como dos anexos.

Abstract

This document describes two different sub-projects devoted to the acquisition and processing of data generated by a traffic monitoring based on FBG sensor arrays in asphalt structures.

The first sub-project consists of the development of a MATLAB software for a commercial FBG interrogator with the objective of obtaining data from the sensor array and storing it, for its latter use and analysis in a lab environment.

The second sub-project describes the development of a system aimed to offer an easy and comprehensible access to the system's data from a computer situated outside the lab. The scripting language used is *Python*.

In order to guarantee the comprehensibly of this document, the first chapter contains an introduction of such topics as optic fiber, optic sensors, FBG ... etc. This chapter is meant to be a fast and accessible introduction for readers unfamiliar with such concepts. The second and third chapters describe the inner workings and development of the first and second sub-projects respectively. The codes designed for each project are contained in this document as two addendums.

Lista de palabras clave:

Python, *data analysis*, Sensores basados en redes de difracción de Bragg, Fibra óptica, IoT.

Lista de tablas

Lista de figuras

Lista de ecuaciones

1 Tabla de contenido

2 Introducción

Durante décadas el estándar de transducción ha sido el empleo de sensores electrónicos los cuales estaban seriamente limitados por factores como su tamaño, sensibilidad ante interferencias y pérdidas de transmisión. Sin embargo, la aparición de nuevas tecnologías como los sensores de fibra óptica nos permiten realizar medidas en entornos que antes nos eran inaccesibles.

Estos nuevos sensores combinados con la accesibilidad de internet nos permiten no sólo obtener nueva información acerca del entorno que nos rodea sino acceder a dicha información desde cualquier lugar. No obstante, una vez obtenidos dichos datos surge la necesidad de poder extraer la información relevante.

En este documento se describe el desarrollo de dos proyectos dedicados a la recopilación y análisis de datos obtenidos de una red de sensores ópticos. La razón por la que este TFG está dividido en dos proyectos se debe al cierre del campus universitario causado por la emergencia sanitaria por COVID-19 en marzo del 2020. Más detalles en las secciones 4 y 6.2.1

Ambos sub-proyectos parten del mismo proyecto de monitorización del tráfico basado en sensores ópticos. Sin embargo, están destinados a distintos usuarios:

- El primer proyecto está dirigido al personal del laboratorio de la UPNA.
- El segundo proyecto está dirigido al personal de gestión de tráfico.

Esta diferencia de usuarios se refleja en las diferencias entre los dos sistemas desarrollados:

Proyecto 1: Software basado en MATLAB para la toma de datos generados por un interrogador Si155.	Proyecto 2: Interfaz de usuario basada en *Python* para la visualización de datos del tráfico.
Requiere funcionar en la misma red que el interrogador.	No requiere ninguna configuración de la red. Basta con ser un usuario aceptado por el sistema y tener acceso a internet.
Asume un nivel de conocimiento por parte del usuario.	No requiere conocimiento en la materia para su uso.
Trabaja con los datos generados directamente por el sensor. Es decir, los datos de la señal óptica.	Los datos que recibe el usuario ya han sido procesados para extraer la información relevante sobre el tráfico detectado.

Tabla 1: Diferencias entre los dos proyectos.

En ambos proyectos el objetivo primario es mostrar al destinatario la información obtenida por el sensor de manera que se adapte a sus necesidades.

En el caso del segundo sub-proyecto también existen una serie de objetivos secundarios:

- Soportar la monitorización de datos correspondientes a eventos pasados.
- Soportar la monitorización de datos en tiempo real.
- Permitir al usuario modificar la configuración de la conexión.
- Permitir al usuario introducir nuevos sensores en el sistema.
- Aplicar redes de sensores de fibra óptica a tecnologías IoT(*Internet of Things*)[1] .

3 Fundamentos teóricos

3.1 Fibra óptica

La fibra óptica es una guía de ondas cilíndrica dieléctrica típicamente fabricadas mediante sílice o polímeros. Se emplea para la transmisión de ondas electromagnéticas.

Consta de dos partes principales: el núcleo (*core*) y la cubierta (*cladding*). La fibra también posee una tercera sección llamada 'recubrimiento' o *buffer* cuya función es proteger la fibra de la humedad y daños físicos.[2]

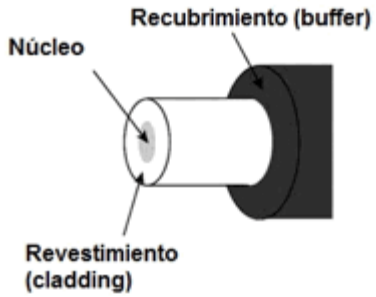


Figura 1: estructura básica de la fibra óptica[2]

La cubierta y el núcleo poseen índices de refracción ligeramente distintos. De acuerdo con la ley de Snell, si el ángulo de incidencia de la luz al entrar en la fibra es mayor que el ángulo crítico se produce una refracción total de la luz en la cubierta (toda la luz revota sobre la superficie de la cubierta). De esta forma la luz queda confinada dentro del núcleo y se transmite a lo largo del núcleo[3].

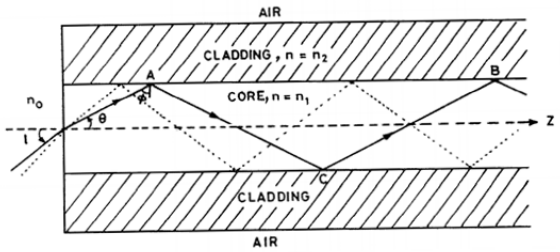


Figura 2: Transmisión de la luz por el interior de la fibra óptica[3].

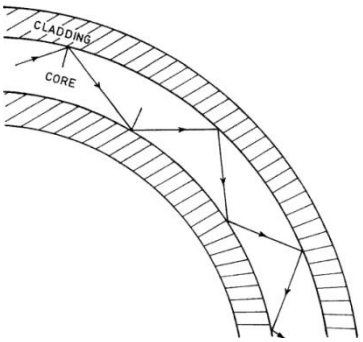


Figura 3: Transmisión de la luz por el interior de una fibra óptica doblada[3].

Al aplicarse la teoría de rayos sobre este sistema se describe esta transmisión de pulsos ópticos como una serie de rayos o modos de luz que atraviesan la fibra óptica realizando reflexiones al incidir sobre la frontera entre el núcleo y la cubierta. Si bien la teoría de rayos describe la propagación de la luz a través de la fibra óptica de una forma sencilla permite simplificar los cálculos no es la más adecuada. La aplicación de las ecuaciones de Maxwell con las condiciones de contorno particulares de cada fibra da lugar a una interpretación mucho más correcta. Sin embargo, debido a que este documento sólo busca realizar una introducción breve al concepto de la fibra óptica, no se va a entrar en más detalle.

Según la cantidad de modos que circulan por una fibra esta puede clasificarse como:

- Monomodo: si circula un único modo por su interior.
- Multimodo: si circulan varios modos por su interior.

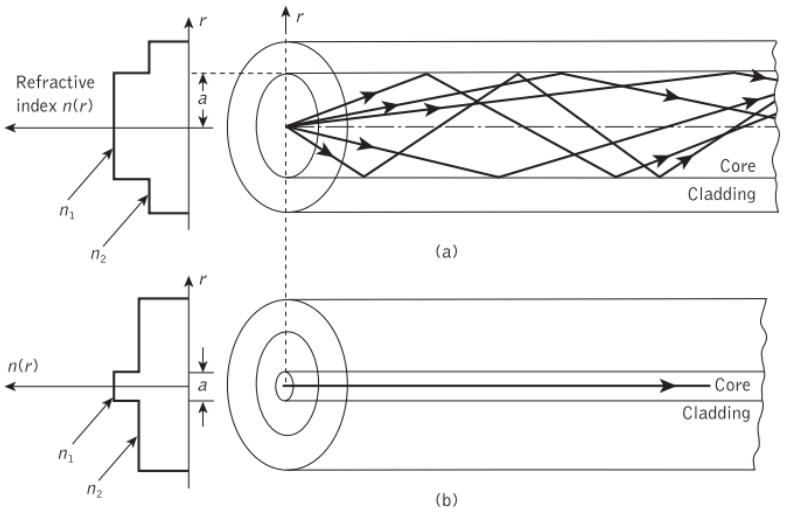


Figura 4: propagación de rayos por el interior de la fibra óptica: (a) fibra multimodo y (b) fibra monomodo [4].

La fibra óptica posee varias características beneficiosas que otros medios tradicionales como el cobre no poseen:

Gran ancho de banda	Posee un ancho de banda potencial muy superior al de sistemas de cables metálicos.
Bajas perdidas de transmisión	Se han llegado a fabricar fibras ópticas con pérdidas de transmisión tan bajas como $0{,}15\ dB\ km^{-1}$
Poco tamaño y peso	Posee un diámetro muy reducido y son mucho más ligeras que los cables de cobre
Aislamiento eléctrico	Al ser fabricadas con plástico o cristal son aislantes eléctricos.
Inmunidad frente a interferencias y *crosstalk*	Forman una guía de onda dieléctrica por lo que están libres de interferencias electromagnéticas e interferencias de radiofrecuencia.
Seguridad de señal	Al contrario que el cable de cobre, no es posible obtener la señal transmitida por la fibra de forma no invasiva.
Robustez y flexibilidad	Si bien, es necesario el uso de capas de recubrimiento para proteger la fibra de la humedad y otros daños físicos, la fibra que soporta mucha tensión.
Fiabilidad del sistema.	Debido a la reducida necesidad de repetidores o amplificadores el sistema es mucho más fiable y los componentes ópticos suelen tener una esperanza de vida de entre 20 y 30 años.
Bajo coste	Al ser fabricada con materiales comunes, el precio de la fibra óptica es mucho más reducido que el del cobre.

Tabla 2: Ventajas de la fibra óptica[5].

3.2 Sensores de fibra óptica

Un sensor de fibra óptica es un dispositivo empleado para medir un factor químico, físico o biológico. La interacción entre el parámetro a medir con el campo de luz provoca una modulación en la atenuación, longitud de onda o fase. Esta modificación puede ser detectada, por ejemplo, al interferir la señal óptica recibida con una señal de referencia[6].

FIBER OPTIC SENSORS				
Magnitude to be measured		Spatial distribution	Transduction mechanism	Measurand induced modulation
Physical	Bio Chemical			
Curvature / bend	Gas	Point	Intrinsic	Intensity (amplitude)
Displacement/strain	Molecular	Distributed	Extrinsic	Interferometric (phase)
Electromagnetic	DNA	Quasi distributed		Polarimetric (polarization)
Pressure	Protein			Spectroscopy (wavelength)
Temperature	Humidity and pH			
Torsion/twist				
Transversal load				
Refractive index				
Vibration				

Tabla 3: Tipos de sensores de fibra óptica[7].

El reducido tamaño y peso de la fibra óptica combinado con su flexibilidad la convierte en una opción ideal para desarrollar sensores que operen en ambientes donde la colocación de sensores eléctricos sería complicada [8] . En particular sus características dieléctricas lo convierten en una gran alternativa para realizar mediciones en ambientes donde no pueden colocarse sensores tradicionales como en estructuras de asfalto [9].

Existen múltiples diseños de sensores basados en fibra óptica, pero en este trabajo nos vamos a centrar en redes de difracción de Bragg (FBG según sus siglas en inglés).

3.2.1 Sensores basados en redes de difracción de Bragg

Este tipo de sensores pueden ser empleados para realizar multitud de funciones actuado como reflector, filtro o sensor [10]. Consisten en una fibra óptica cuyo núcleo posee una variación periódica del índice de refracción que actúa como un filtro de banda.

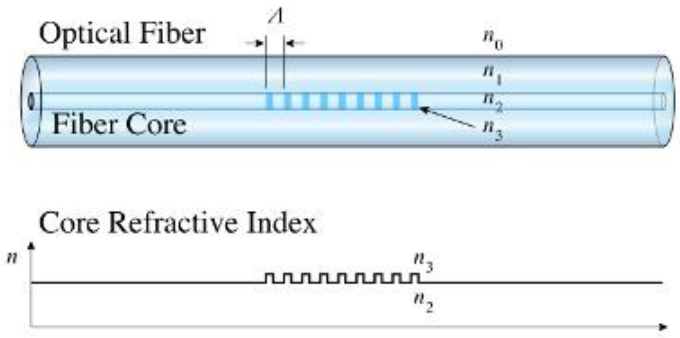


Figura 5: Perfil de una red de difracción de Bragg[11].

El principio básico de operación empleado en los sistemas de sensores basados en redes de difracción de Bragg es la monitorización del cambio en la longitud de onda de la señal recibida. De esta manera se pueden monitorizar el parámetro que se está controlando (típicamente temperatura o presión). La longitud de onda reflejada (también conocida

como longitud de onda de Bragg: λ_{Bragg}) está definida por Ecuación 1 donde Δ representa el periodo del índice de modulación refractivo y n es el índice de refracción efectivo en el núcleo de la fibra [12].

$$\lambda_{Bragg} = 2n\Delta$$ Ecuación 1: Longitud de onda de Bragg

El funcionamiento del sensor aparece descrito en la Figura 6. Se conecta una fuente de luz espectralmente ancha a la fibra. Al llegar a la red de difracción, parte la componente espectral correspondiente con la longitud de onda de Bragg es reflejada por la estructura mientras que el resto de las componentes la atraviesan. La presión y la temperatura provocan modificaciones en la longitud de onda de Bragg que pueden ser detectadas tanto en el espectro reflejado como el transmitido[12].

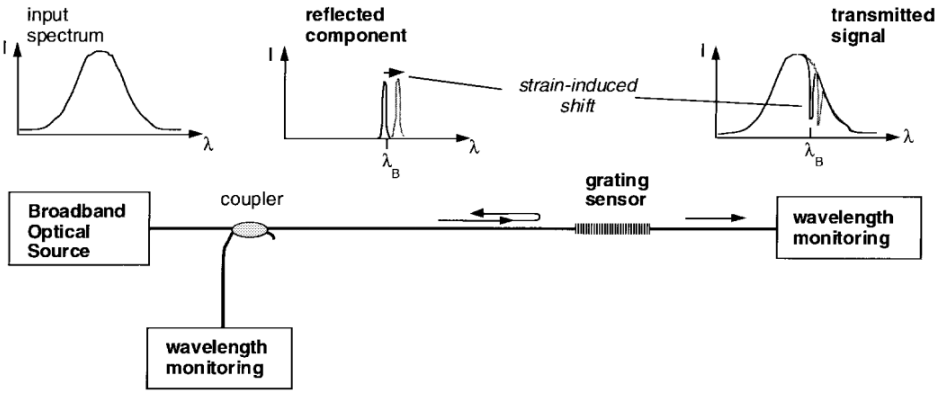


Figura 6: Funcionamiento de una sensor basado en red de difracción de Bragg [12].

3.2.2 Aplicación de FBG para la monitorización del tráfico.

En el año 2019 un estudio realizado en la UPNA consiguió demostrar el correcto funcionamiento de redes de sensores de fibra óptica instaladas en estructuras de asfalto.

Esta demostración se realizó mediante la colocación de dos *arrays* de 10 sensores FBG instalados en el asfalto y espaciados entre ellos 50 cm tal como se puede ver en Figura 7 y Figura 8. Para garantizar la protección de los sensores y su máxima vida útil estos *arrays* estaban protegidos con una cubierta de fibra de vidrio.

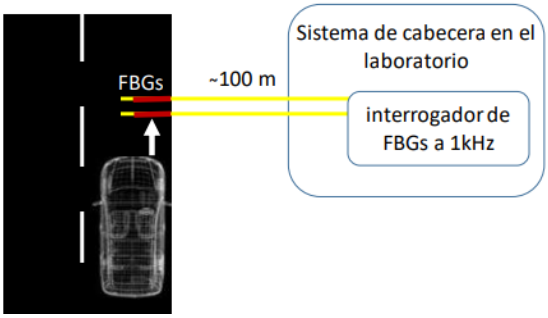


Figura 7: Esquema del sistema de monitorización de tráfico [9].

Figura 8: Fotografía de los sensores instalados sobre el asfalto[9].

Los sensores son capaces de detectar cuando pasa un vehículo sobre ellos, tal como se ve en Figura 9. En esta gráfica cada pico representa uno de los ejes del motor, de forma que en este intervalo de tiempo se han detectado cinco vehículos

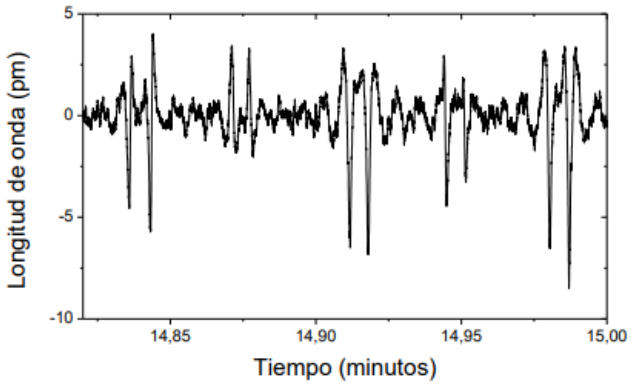


Figura 9: Resultados del funcionamiento de un array de sensores[13].

Al emplear dos *arrays* de sensores es posible calcular la velocidad del vehículo analizando el retardo entre las dos líneas de sensores separadas 50 cm, tal como se puede apreciar en Figura 10. Un detalle importante de dicha figura es la diferencia de amplitud y velocidad entre los dos ejes del vehículo. La diferencia de velocidad puede explicarse como una desaceleración del vehículo, mientras que la diferencia de amplitud representa una diferencia de peso (el primer eje es típicamente donde va el motor y por tanto soporta más peso).

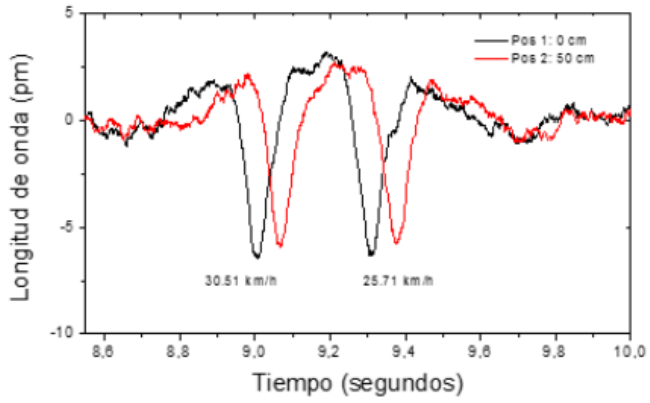


Figura 10: Análisis de la detección de un vehículo[13].

El empleo de esta estructura de sensores no sólo permite obtener los mismos datos que radares de tráfico tradicionales, sino que también permitiría realizar un pesado de los vehículos en movimiento.

3.3 Comunicación a través de internet

3.3.1 OSI

OSI (***O**pen **S**ystem **I**nterconnection*) es un modelo de referencia para los protocolos de transmisión de red reconocido en 1978 por ISO (*International Organization for Standardization*).

Este modelo divide el protocolo en una serie de capas, las cuales podemos definir como una colección de subsistemas del mismo rango. Estas capas están organizadas de tal manera que cada capa da valores extra a las capas inferiores, mientras que las capas inferiores son capaces de funcionar de forma independiente a las superiores [14].

Figura 11: Modelo OSI[15].

3.3.2 TCP/IP

Antes de la aparición de internet, los ordenadores funcionaban de forma independiente unos de otros y sólo podían comunicarse si tenían el mismo hardware y sistema operativo. No fue hasta la década de 1960 que se empezó a desarrollar un sistema de comunicación entre ordenadores de diferentes características. El resultado fue el protocolo TCP/IP (*Transmission Control Protocol/Internet Protocol*) el cual se sigue usando a día de hoy[16].

TCP/IP posee varios componentes:

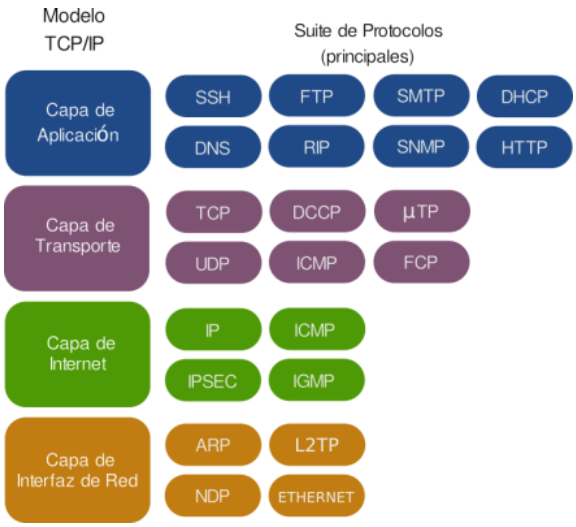


Figura 12: Familias de protocolos de Internet[17].

Existen multitud de protocolos que permiten realizar una conexión TCP/IP, aquí se explicarán sólo los empleados en este proyecto:

HTTP	El protocolo de aplicación se encarga de garantizar la comunicación entre cliente y servidor
TCP	El protocolo de transporte se encarga del control de flujo de paquetes.
IP	El protocolo de red se encarga del enrutamiento del mensaje.
Ethernet	Interfaz de la red estándar de redes de área local.

Tabla 4: Componentes de TCP/IP.

Dentro de una comunicación a través de internet existen dos roles: cliente y servidor. El cliente es aquel que inicia la conexión y solicita un recurso, mientras que el servidor es el que proporciona dicho recurso.

3.3.2.1 HTML

HTML (*Hypertext Transfer Protocol*) es el protocolo de comunicación por internet más empleado hoy en día. Su funcionamiento se describe en Figura 13.

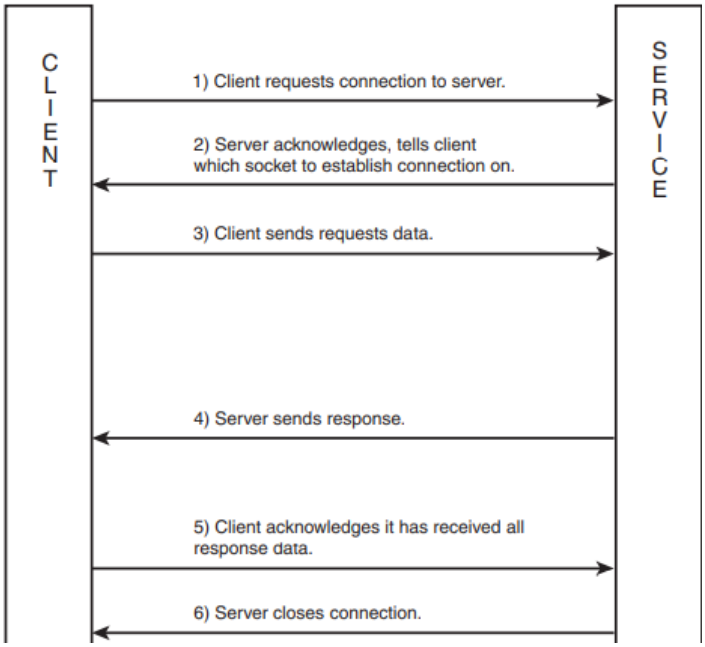


Figura 13: Funcionamiento de HTML[16].

3.3.3 *Web Service*

Un *Web Service*, es un sistema de software diseñado para soportar interacciones entre máquinas a través de la red. Al contrario que las páginas web tradicionales, no está diseñado para ser comprendido por humanos sino por máquinas [18].

Existen varias arquitecturas de comunicación entre maquinas. Sin embargo, hoy en día las más reconocidas son REST y SOAP

3.3.3.1 REST

REST (*Representational State Transfer*) es un estilo de arquitectura software empleado en la creación de *Web Services.* Esta arquitectura emplea las funciones ya definidas dentro del protocolo HTML para realizar peticiones (más información en Tabla 14). En práctica, esto significa que las variables son enviadas como parte del URL[19].

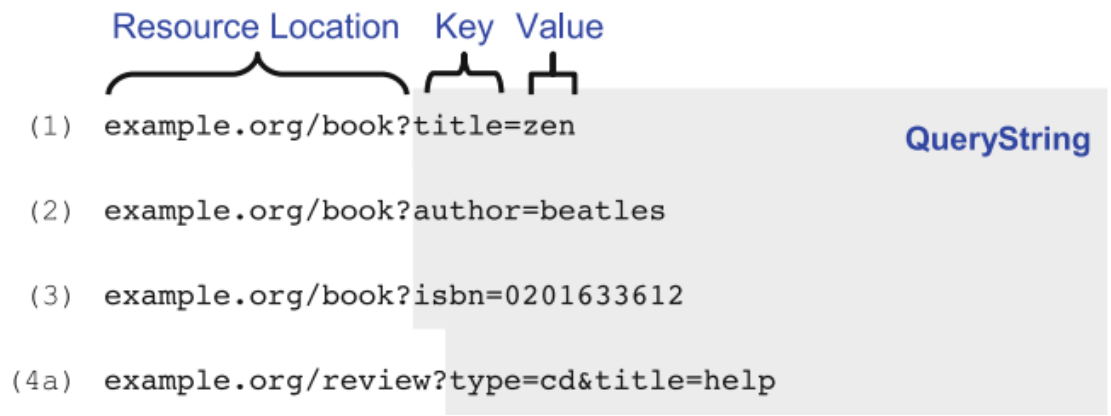


Figura 14: URL enviadas a una API con arquitectura REST [19].

3.3.3.2 SOAP

SOAP (*Simple Object Access Protocol*) es un protocolo de comunicación empleado en *Web Service*. La comunicación se realiza a través de documentos XML (ejemplo en Figura 15). Esta estructura permite enviar variables de tipo complejo.

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Figura 15: Ejemplo de un mensaje XML creado con SOAP.

Para ver una comparativa de las propiedades de SOAP y REST ver Tabla 10.

3.3.4 Bases de datos

Una base de datos es una colección de datos. Se consideran 'datos' a aquellos hechos relacionados que se pueden grabar y tienen significado implícito. Por definición una base de datos debe tener tres características:

- Representa algún aspecto del mundo real.
- Debe ser una colección de datos lógicamente coherente con algún tipo de significado inherente.
- Se diseña, construye y rellena para un propósito específico[20].

Un software DBMS (*Database Manager System*) es una colección de programas que permiten al usuario crear y mantener una base de datos[20].

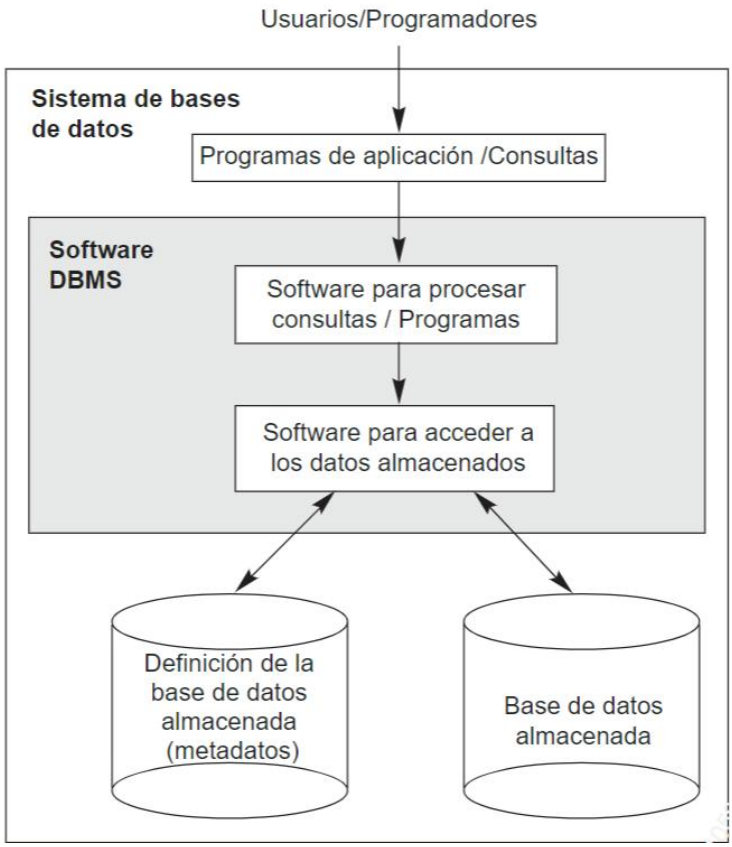


Figura 16: Entorno de una base de datos simplificado [20].

Existen múltiples tipos de bases de datos, pero todas las empleadas en este trabajo pueden clasificarse como bases de datos relacionales. Una base de datos es aquella en la cada fila representa una colección de datos relacionados[21].

3.3.4.1 SQL

SQL (*Structured Query Language*) es un lenguaje de base de datos global que permite la creación, consulta y modificación de bases de datos mediante el uso de peticiones o *queries*. SQL puede considerarse uno de los principales motivos detrás del éxito de las bases de datos relacionales. Es un lenguaje claro e intuitivo que permite trabajar con bases de datos de forma sencilla e incluso establecer restricciones dentro de la propia base de datos [22].

3.3.5 API

API (*Application Programming Interface*). En *Web Services* se define un API como una interfaz de programación qué define como se comunican dos aplicaciones[23]. En otras palabras, se encarga de procesar las peticiones de los clientes y de hacer de intermediario entre el cliente y la base de datos.

4 Sub-Proyecto 1: Sistema de control y adquisición de datos a alta velocidad para interrogadores de FBGs modelo si155 de Micron Optics

Este desarrollo de este proyecto estaba planeado en dos fases:

1. Fase de programación: Desarrollo en MATLAB de un software para el interrogador si155 de redes de difracción Bragg en fibra.
2. Fase experimental: Realización de una serie de mediciones sobre unos sensores basados en FBG.

Debido a la orden de confinamiento causada por la emergencia sanitaria por COVID-19 en marzo del 2020 el acceso a los laboratorios de la UPNA fue bloqueado. Este proyecto tuvo que ser interrumpido antes de ser completado debido a la incompatibilidad de escritorio remoto y el acceso a la red del sensor.

4.1 Fase de programación

4.1.1 Consideraciones previas

El interrogador es el modelo si155 de *Micron Optics.*

Figura 17: Interrogador si155

Este modelo tiene una serie de características:

si	Sensing Instrument
Serie 1xx	4 canales paralelos. Interrogador de tamaño pequeño. Capaz de operar sin un ventilador.
Núcleo x55	Un láser sintonizable estrecho capaz de escanear un ancho de banda entre 1460-1620 nm a una velocidad máxima de 1 kHz con un nivel de ruido y *jitter* bajos.

Tabla 5: Características del interrogador si155[24].

Laser Type	Fiber Laser
Laser Class (IEC 60825-1, 21 CFR 1040.10)	1
Max Output Power	0.25 mW
Min Output Power	0.06 mW
Mode Field Diameter	~ 10.4 µm
Numerical Aperture	0.14
Wavelength	1460 – 1620 nm

Figura 18: Características del láser del interrogador[24].

Si155 tiene dos sistemas de adquisición de datos:

- Detección de picos: El interrogador posee un algoritmo interno que permite detectar picos en el espectro de la señal. La Velocidad máxima alcanzable de este modo es 1Khz
- Espectro total: Esta función transmite el espectro completo de forma continua. La máxima alcanzable en este modo 10 Hz.

Si155 soporta comunicación a través del protocolo TCP/IP en los puertos 51971, 51972, 51973 y 51974. El puerto 51971 está dedicado a una comunicación basado en comandos y respuestas mientras que el resto están dedicados a la obtención de datos por flujo. En concreto el puerto 51974 está dedicado a la transmisión del espectro completo mientras que el 51972 transmite los picos detectados por el sistema[24].

El lenguaje escogido para el desarrollo del software ha sido MATLAB ya que se trata de una herramienta de software matemático que cuenta con su propio entorno de desarrollo integrado, así como funciones integradas para la representación de datos, funciones, matrices, etc.

4.1.2 Implementación

El interrogador Si155 soporta dos sistemas de comunicación:

- Comunicación petición-respuesta
- Comunicación por flujo de datos

Con el fin de permitir una comunicación con si155 mediante ambos sistemas han desarrollado una serie de funciones en MATLAB:

Conn	Abrir una conexión con el interrogador.
Disc	Cerrar una conexión con el interrogador.
LilEndian	Pasar una cadena de bytes a formato Little Endian.
Peticion[1]	Genera peticiones a partir del comando y los argumentos y la envía al interrogador.
Response	Almacena la respuesta enviada por el interrogador.

Tabla 6: Funciones desarrolladlas para la comunicación con SI155.

4.1.2.1 Establecer conexión

Si155 implementa el protocolo TCP/IP, por lo que para establecer una conexión con el interrogador es necesario crear un objeto de la clase TCPIP (parte de la librería básica de MATLAB) [25]. Para crear este objeto necesitamos conocer una serie de parámetros:

- La IP del interrogador: el manual del interrogador especifica que para conectar el ordenador con el interrogador es necesario conectarse a la IP 10.0.0.121 con la máscara 255.255.255.0[24].
- El puerto al que conectarse: dependiendo del tipo de datos se deseen obtener, será el 51974, el 51973, el 51972, o el 51971.

[1] No lleva tilde para evitar problemas con MATLAB

- El rol del objeto en la conexión: tanto para comunicación por flujo de datos como petición-respuesta debemos fijar el rol de la conexión como cliente (*Network client*).
- El tamaño del buffer: El tamaño máximo posible son 2 GB, el tamaño recomendado es 32768.

Para ello hemos definido la función 'conn', que recibe por inyección la id del interrogador y el puerto al que se desea conectarse. Conn crea un objeto con estas características e inmediatamente lo usa para abrir una conexión con el interrogador.

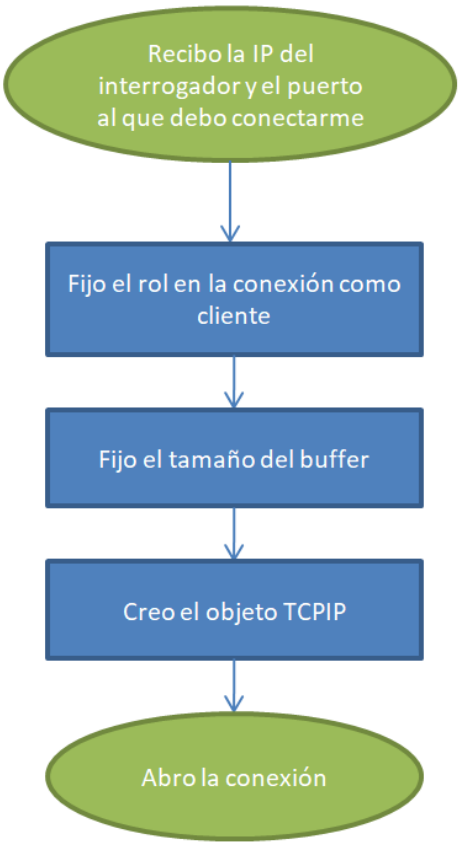


Figura 19: Diagrama de flujo de la conexión con el interrogador.

4.1.2.2 Comunicación por petición-respuesta.

Uno de los métodos de comunicación aceptados por el interrogador es un sistema de peticiones y respuestas intercambiadas entre el interrogador y el cliente:

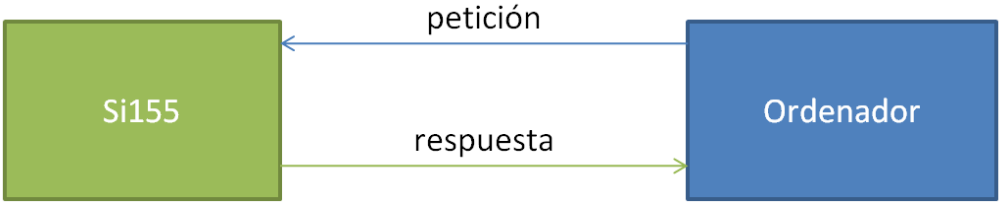


Figura 20: Diagrama de comunicación petición-respuesta del interrogador.

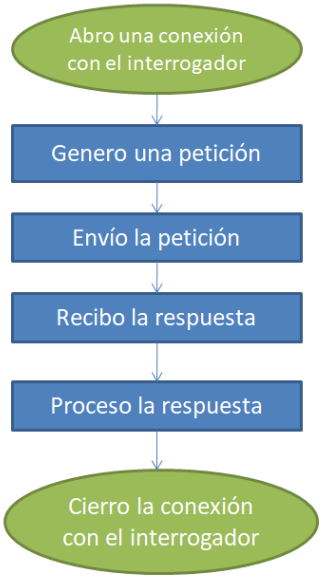


Figura 21: Diagrama de flujo de una comunicación petición-respuesta con el interrogador.

4.1.2.2.1 Formato

El interrogador acepta sólo peticiones codificadas mediante *little endian (*formato que ordena los bytes de menos significativo a más significativo[26]). Sin embargo, MATLAB trabaja con *big endian* por lo que antes de realizar una petición es necesario modificar el formato:

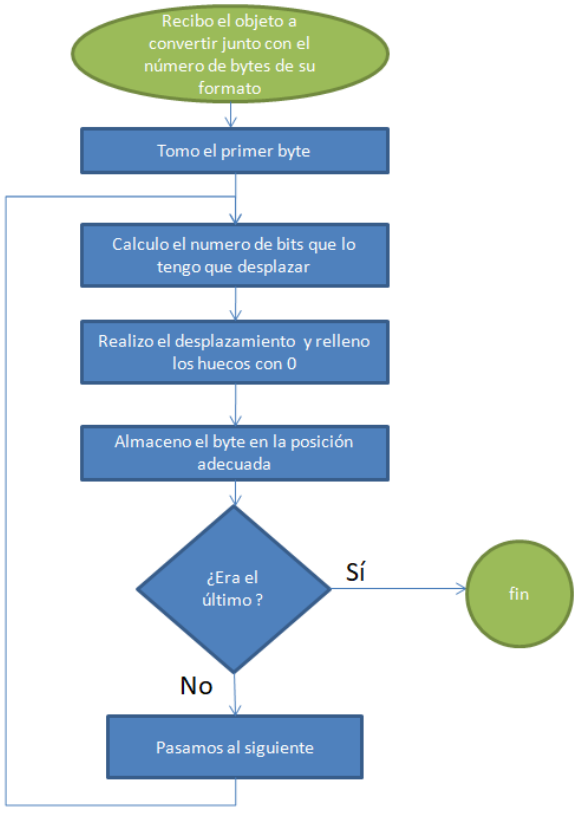


Figura 22: Diagrama de flujo de la conversión a formato *little endian*.

4.1.2.2.2 Peticiones

El interrogador acepta una serie de comandos (más detalle en el manual) a los que se pueden añadir argumentos. Algunos ejemplos de comandos son:

Comando	Argumentos	Respuesta
#Help	Grupo específico de comandos (opcional)	Lista de todos los comandos disponibles. Se puede limitar la lista a un subgrupo de comandos específicos.
#GetSerialNumber		Número de serie del instrumento.
#GetInstrumentName		Nombre asignado al instrumento (máximo 20 caracteres).
#SetInstrumentName	Nombre asignado.	Asigna un nombre al instrumento.
#IsReady		Si el instrumento es capaz de transmitir datos.
#Reboot		Reiniciar el interrogador.
#GetUserData	Número de la localización	Obtener los datos de usuario en una localización especifica
#SetUserData	Número de la localización y los datos de usuario	Establecer los datos de usuario en una localización especifica
#GetPeakOffsets	Número del canal	*Offset* de los picos de un canal especifico
#ClearPeakOffsets	Número del canal	Eliminar el *offset* de los picos de un canal especifico
#GetPeaks		Picos y valles detectados por el instrumento
#GetSpectrum	Número de canal (opcional)	Espectro completo de un canal especifico o si no hay argumentos de todos los canales
#GetAvaliableLaserScanSpeeds		Velocidades de escaneo soportadas por el instrumento
#GetActiveNetworkSettings		Configuración de la red actual

Tabla 7: Ejemplos de comandos aceptados por el interrogador

Para solicitar un comando al interrogador es necesario crear una petición con una estructura determinada:

Byte Offset	Size (bytes)	Type	Description
0	1	u8	Request Option. 0-None, 1-Supress Message, 2-Supress Content, 4-Compress
1	1	N/A	Reserved
2	2	u16	Size of command, *Nc*
4	4	u32	Size of argument, *Na*
8	*Nc*	ASCII	Command
8+*Nc*	*Na*	ASCII	Arguments

Figura 23: estructura de las peticiones aceptadas por el interrogador[24].

Es decir, que las peticiones tienen un tamaño de 8 + Nc + Na bytes (siendo Nc y Na el número de bytes que ocupan el comando y los argumentos respectivamente).

Se ha desarrollado una función llamada 'peticion'[2] capaz de generar una petición con la estructura correcta y enviarla a partir del comando, los argumentos, y el objeto TCPIP:

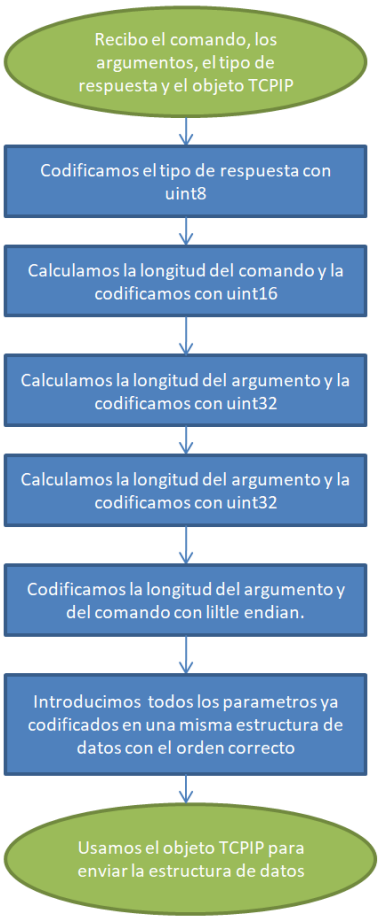


Figura 24: Diagrama de flujo de la generación y envío de una petición.

4.1.2.2.3 Respuesta

Debido a la multitud de comandos existentes la respuesta puede tener distinta longitud. Sin embargo, siempre tendrán la misma estructura de cabecera:

Byte Offset	Size (bytes)	Type	Description
0	1 bytes	u8	Status
1	1 bytes	u8	Option (Request Option echo)
2	2 bytes	u16	Message length, *Nm*
4	4 bytes	u32	Content length, *Nc*
8	*Nm* bytes	ASCII	Message string
8 + *Nm*	*Nc* bytes	variable	Content

Figura 25: Estructura de la cabecera de las respuestas del interrogador[24].

Gracias a esta estructura de cabecera es posible conocer la longitud de los datos enviados por el interrogador lo que facilita la lectura de la respuesta.

[2] El nombre de la función no lleva tilde para evitar problemas con MATLAB

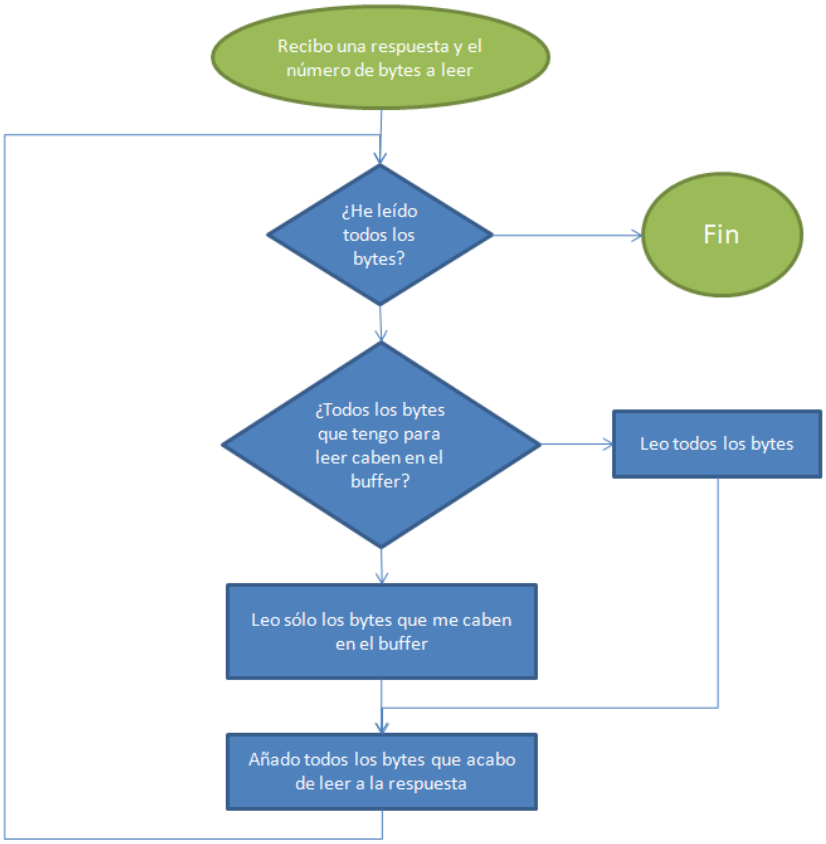


Figura 26: Diagrama de flujo de la lectura de una respuesta

4.1.2.3 Comunicación por flujo de datos

El interrogador ofrece una segunda opción de comunicación: por flujo de datos. El sistema está programado de tal manera que ciertos puertos están constantemente transmitiendo datos sin necesidad de lanzar un comando.

Puerto	Datos	Comando equivalente
51972	Picos detectados por el sistema	#GetPeaks
51973	Espectro completo	#GetSpectrum
51974	Datos del sensor	#GetSensorData

Tabla 8: Puertos y opciones de flujo de datos en el interrogador.

Al omitir la necesidad de realizar una petición para obtener una traza de datos logramos reducir el tráfico en la red y aumentamos la velocidad de obtención de datos.

Al ser equivalentes con su respectivo comando la función creada para la lectura de datos es la misma que la empleada en la lectura de respuestas de la sección anterior. Sin embargo, para leer un flujo continuo de datos la función es ejecutada en bucle:

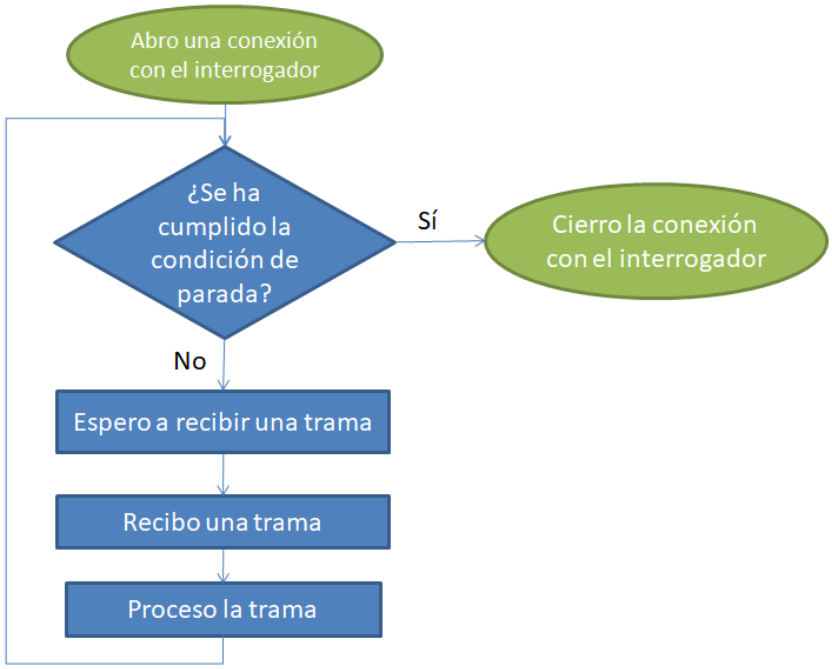


Figura 27: Diagrama de flujo de una comunicación por flujo de datos con el interrogador.

4.1.2.4 Desactivar conexión

Para desactivar la conexión sólo es necesario cerrar el *socket* de la comunicación y eliminar el objeto TCPIP que lo representa.

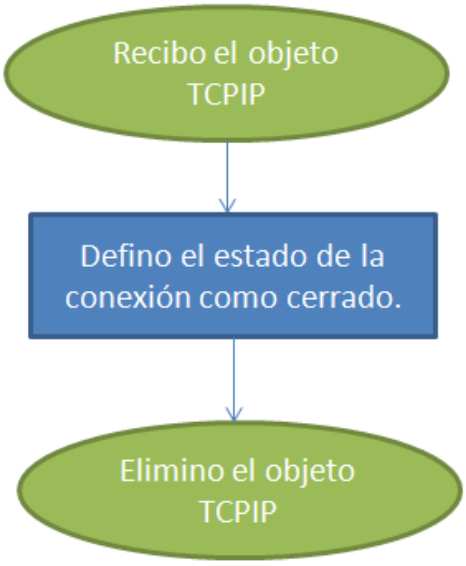


Figura 28: Diagrama de flujo de la desactivación de la conexión con el interrogador.

4.2 Fase experimental

Debido al confinamiento, esta fase no pudo ser realizada.

5 Sub-Proyecto 2: Desarrollo de un sistema para el acceso a datos generados por una serie de sensores de monitorización del tráfico

El objetivo de este proyecto es permitir al usuario acceder a los datos generados por una serie de sensores basados en redes de difracción de Bragg (FBG, según sus siglas en inglés) colocados bajo el asfalto y conectados con una *Raspberry Pi* capaz de procesar los datos del sensor con el fin de detectar cuando pasa un vehículo, a qué velocidad, qué aceleración y el tipo de vehículo.

Una *Raspberry Pi* es un ordenador de placa reducida (una computadora completa en un solo circuito) de bajo coste. Una de las principales ventajas de este sistema es capacidad de expansión y adaptabilidad. Aparte de conectores estándar como USB, HDMI y ranuras SD, Raspberry Pi también incluye un circuito GPiO (*general-purpose input/ output*). Esta característica permite que la *Raspberry* pueda conectarse con multitud de circuitos, desde circuitos sencillos hasta sensores y equipamiento de laboratorio[27].

5.1 Estructura básica del sistema

El usuario debe ser capaz de acceder a los datos según se vayan generando, así como a los datos de eventos pasados. Para conseguir esto, se ha desarrollado un sistema formado por tres elementos básicos:

- Los sensores que generan los datos. En la versión actual del proyecto el usuario tiene acceso a los datos generados por 4 sensores: Avenida Cataluña 1(AvCat1), Avenida Cataluña 2 (AvCat2), Avenida de Sancho el Fuerte (AvSancho) y Bajada de Labrit (Labrit); de los cuales sólo AvCat1 corresponde a un sensor real, el resto son datos generados por un *software.* En este caso, los sensores disponen de un módulo de comunicaciones basado en una *Raspberry Pi.*
- Una *Raspberry Pi* central que realiza los roles de servidor, *broker*, simulador y base de datos.
- Una interfaz de usuario que permite acceder a los datos de forma sencilla e intuitiva.

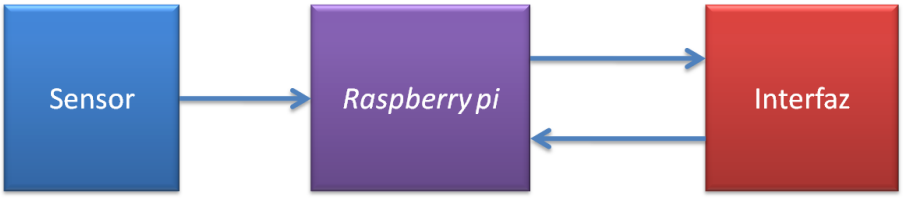


Figura 29: Estructura básica del sistema.

5.1.1 Consideraciones previas

El objetivo es crear una plataforma que permita al usuario pueda acceder a los datos de forma sencilla e intuitiva. Existen dos tipos de datos a los que acceder:

- Datos históricos: datos de eventos de tráfico pasados almacenados en una base de datos.

- Datos en tiempo real: el usuario debe ser capaz de ver los eventos de tráfico según se van generando en tiempo real por el sensor.

Para lograr estos objetivos debemos diseñar un sistema en el que el usuario sea capaz de comunicarse con un servidor para solicitar datos históricos y comunicarse con los sensores para obtener los datos en tiempo real. Debido a que el usuario no puede comunicarse directamente con el servidor es necesaria la implementación de una interfaz que sea intuitiva y sencilla de manejar.

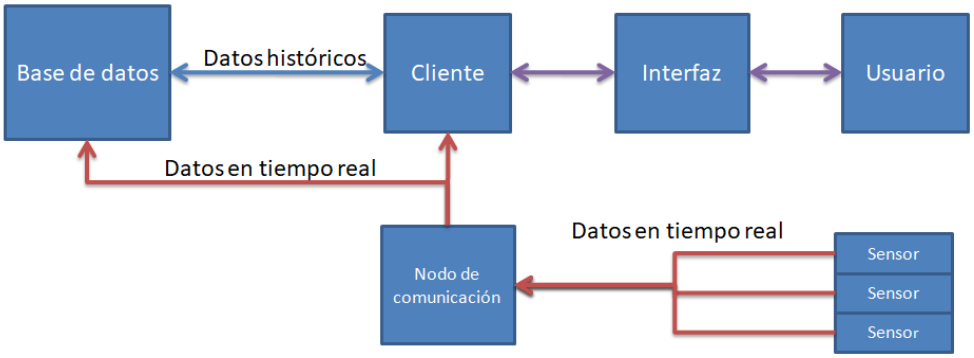


Figura 30: Diagrama básico del flujo de datos del sistema

5.1.1.1 *Interfaz de Usuario*

A la hora de implementar la interfaz de usuario existen varias opciones a considerar:

- Una página web
  - Ventajas:
    - Accesible desde cualquier lugar.
    - Sin necesidad de instalación.
    - Experiencia en el diseño de páginas web.
  - Desventajas:
    - Si realizamos el procesado de datos en el lado del servidor esto supone una mayor carga de programación en los servidores lo que puede causar problemas si un gran número de usuarios intenta acceder simultáneamente.
    - Si realizamos el procesado de datos en el lado del cliente esto haría que el código de procesado de datos fuera accesible desde el cliente.
    - Para asegurar la mejor experiencia sería necesario diseñar la página de tal manera que se adapte a distintos dispositivos: móvil, tablet, ordenador, etc.
- Una aplicación móvil:
  - Ventajas:
    - Accesible desde cualquier lugar.
  - Desventajas:
    - Poca experiencia en el diseño de aplicaciones móvil.

        - Los sistemas operativos tienen su propio lenguaje, por lo que una aplicación programada en *Android* no sería compatible con un dispositivo *Apple*.
        - Dependiendo de la versión del sistema operativo es posible que algunas funciones del programa no funcionen correctamente.
- Un programa de ordenador:
    - Ventajas:
        - Al realizar el procesado de los datos en el lado del cliente se requiere poco ancho de banda para funcionar.
        - Experiencia en programación orientada a objetos.
    - Desventajas:
        - Necesidad de instalación.

Tras considerar las distintas ventajas y desventajas de cada posible solución el sistema escogido ha sido el realizar un programa

Una vez escogido el tipo de interfaz, debemos considerar el lenguaje de programación con el que la vamos a implementar. En este caso el lenguaje escogido es *Python*.

Python es un lenguaje de programación orientado a objetos de código abierto y lenguaje de alto nivel fuertemente tipado con tipado dinámico. Es multiplataforma y versátil[28].

- Es versátil-> puede emplearse para desarrollar aplicaciones web, aplicaciones de servidor, aplicación de escritorio.
- Es multiplataforma-> es posible ejecutar un programa creado en *Python* en diferentes sistemas operativos
- Lenguaje de alto nivel->La gramática del código se asemeja más a un lenguaje natural que al lenguaje máquina. Su gramática es muy sencilla y legible ya que prescinde de poner puntos y comas al final de cada línea, etc.
- Lenguaje orientado a objetos-> no se definen funciones ni lógica sino clases. Una clase es una entidad definida por sus características y las funciones que es capaz de realizar, objeto es una instancia de una clase. Citando *python crash course:*

  *"In object-oriented programming you write classes that represent real-world things and situations, and you create objects based on these classes. When you write a class, you define the general behavior that a whole category of objects can hav*e."[29]

  "En programación orientada a objetos se escriben clases que representan situaciones y cosa reales y se crean objetos basados en dichas clases. Al escribir una clase, se define el comportamiento general que puede poseer toda una categoría de objetos"

  Por ejemplo: a la hora de definir la clase semáforo sus atributos serían sus tres luces y sus funciones serían apagar y encender cada una de ellas. La clase

semáforo sería el diseño de un semáforo, mientras que un objeto de la clase semáforo será un semáforo individual.

- Soporta herencia múltiple-> que una clase puede definirse como subclase de otras, lo que implica que tendrá los mismos atributos y funciones automáticamente sin que haga falta definirlas.

Python es el lenguaje escogido porque es un lenguaje orientado a objetos que es con lo que tengo más experiencia trabajando y posee multitud de librerías de código abierto que contienen clases especializadas en distintas funciones incluyendo una para MQTT[30].

Lista de módulos a emplear:

tkinter	Módulo gráfico de la interfaz. Pertenece a la librería estándar de *Python*. Es compatible con la mayoría de sistemas Unix y Windows[31]. *Python* posee multitud de librerías para generar interfaces gráficas. *Tkinter* ha sido seleccionada debido a que es compatible con pandastable [32] y matplotlib [33].
paho	Módulo dedicado a la comunicación con un sistema MQTT[30].
pandas	Módulo que incluye estructuras de datos similares a las bases de datos de SQL [34][35].
pandastable	Módulo que incluye elementos gráficos capaces de representar estructuras de datos de pandas[32].
queue	Módulo que implementa colas de mensajes, es parte de la librería estándar de *Python*[36].
Flask	Módulo que contiene un *framework* para trabajar con páginas web. Ha sido escogido por su sencillez y minimalismo [37].
requests	Módulo diseñado para trabajar de forma sencilla con HTML en *Python*[38].
mysql .connector	Módulo que permite a programas de *Python* acceder a bases de datos MySQL de acuerdo con PEP 249 - *Python Database API Specification*[39][40].
matplotlib	Módulo dedicado a la visualización de datos a través de gráficos. Posee multitud de gráficos y una documentación extensa y detallada.[41]
tkcalendar	Modulo que contiene las clases DateEntry y Calendar. Es compatible con tkinter [42].
json	Módulo que contiene un codificador y decodificador de json. Forma parte de la librería estándar de *Python* [43].

Tabla 9: Lista de módulos a emplear

5.1.1.2 Datos en tiempo real

Los datos que se almacenan en el servidor son los generados por el sensor, por lo que es necesario que tanto el usuario como el servidor puedan comunicarse con el sensor simultáneamente. Para esto el estándar de comunicación escogido para la comunicación en tiempo real es MQTT.

MQTT: es un protocolo OSIS de transmisión de mensajes para IoT –*Internet of Things*-. Ha sido diseñado para ser un sistema de publicación/subscrición de mensajes que permita

interconectar sistemas remotos empleando el mínimo coste computacional y el menor ancho de banda [44].

En este protocolo existen 2 tipos de entidades: *brokers* y clientes. Los mensajes que circulan por el sistema tienen una etiqueta llamada '*topic*'. Los clientes son aquellos elementos de la red que publican y reciben mensajes mientras que el *broker* es responsable de retransmitir cada mensaje a todos los clientes que estén subscritos a su *'topic'*. Los *brokers* están permanentemente conectados al sistema mientras que los clientes pueden conectarse y desconectarse siempre y cuando sean usuarios autorizados [45] .

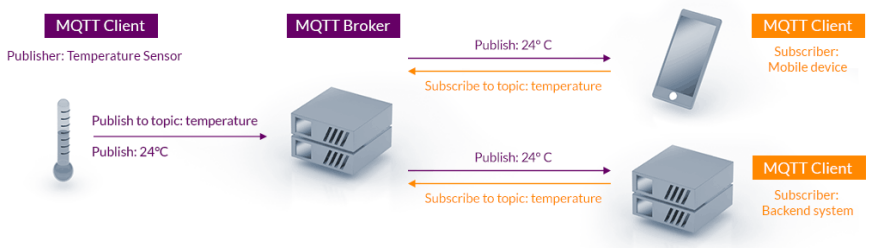


Figura 31: Funcionamiento del protocolo MQTT[44]

Este protocolo ha sido seleccionado porque requiere pocos recursos computacionales (compatible con microcontroladores) y emplea poco ancho de banda. Además permite cifrar mensajes y autentificar usuarios [44] .Existe una librería de *Python* especializada llamada 'paho' [30].

5.1.1.3 Datos históricos

Para la obtención de datos históricos la solución más simple es la creación de un API (*Application Programming Interface*) con la que el cliente pueda comunicarse y solicitar información almacenada en la base de datos del servidor. Un API es un conjunto de acciones que nos dan acceso a determinadas tareas de un software, es una forma de dar acceso a una aplicación a un usuario externo, donde dicho usuario solo puede usar y ejecutar ciertas funciones [46].

A la hora de definir la estructura de comunicación de la api hay dos arquitecturas principales a considerar: SOAP y REST

SOAP	REST
Las cabeceras requieren más ancho de banda (entre el doble y el triple que REST).	Las cabeceras ocupan poco ancho de banda.
Permite mandar variables del tipo complejo.	No admite variables tipo complejo.
Es propio formato comprueba que las variables sean del tipo correcto.	La comprobación del tipo puede realizarse en el servidor.
Hoy en día se usa poco.	Es un sistema muy usado [47].
Compatible con *Python*.	Compatible con *Python*.
Las respuestas tienen una estructura compleja y a veces pueden enviar más información de la necesaria [48].	Las respuestas tienen una estructura simple.

Tabla 10: Comparación entre REST y SOAP

Vamos a emplear REST porque ocupa menos ancho de banda, su estructura es más simple, es el sistema más comúnmente empleado, es compatible con *Python* y la diferencia en seguridad con SOAP es fácilmente compensada en el servidor.

El API se programará utilizando *Python*. El principal motivo detrás de esta decisión es coherencia, emplear el mismo lenguaje de programación para todo el sistema.

5.2 Implementación

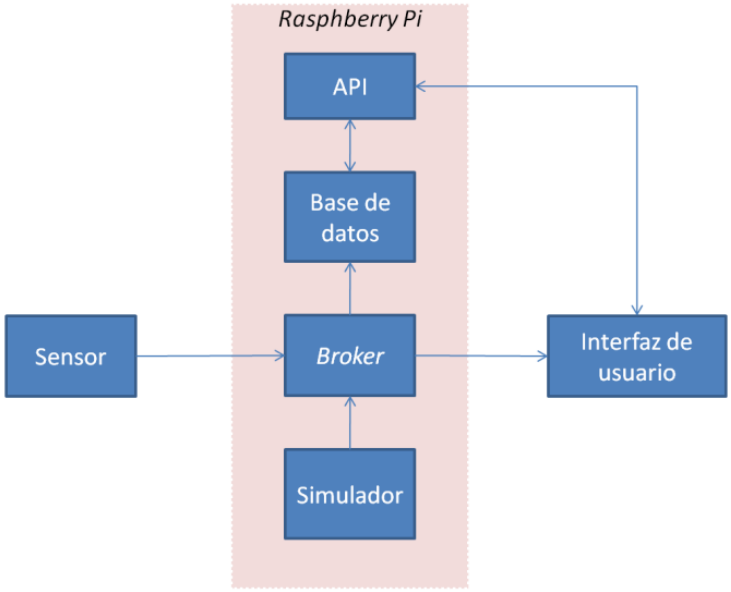


Figura 32: Estructura del sistema.

5.2.1 Sensor

El sensor actúa como un cliente dentro del protocolo MQTT, publicando mensajes cada vez que detecta una incidencia.

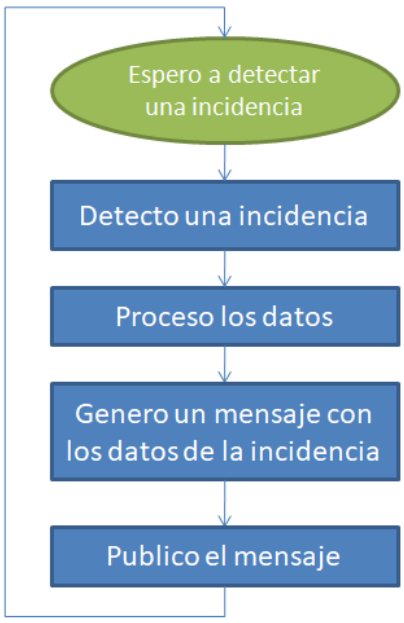


Figura 33: Diagrama de flujo del sensor

5.2.2 *Raspberry Pi* Central

5.2.2.1 MQTT

Como hemos indicado en la estructura básica del sistema, de los cuatro sensores sólo uno corresponde con un sensor real y el resto son simulados por la *Raspberry.* Es decir, que la *Raspberry* actúa simultáneamente de *broker* y de cliente.

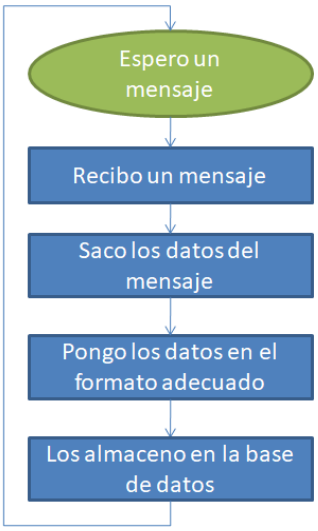


Figura 34: Diagrama de flujo del *broker*.

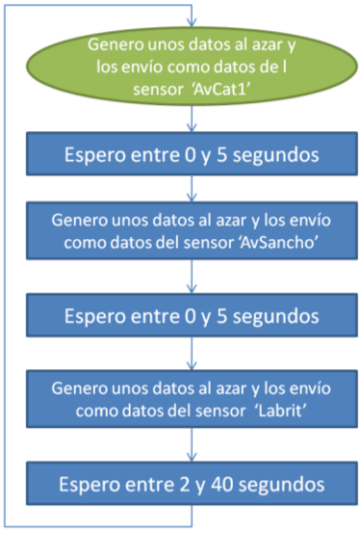


Figura 35: Diagrama de flujo del simulador.

Dado que solamente el *broker* necesita comunicarse con el exterior de la *Raspberry* sólo le asignamos un puerto físico al *broker*.

Un detalle que destacar es que la simulación de datos utiliza una distribución uniforme.

5.2.2.2 Base de datos

El servidor posee una base de datos en SQL donde se almacenan los datos del sistema mediante una serie de tablas:

- Tablas de sensor: Cada sensor debe tener asignada una tabla (identificada por el nombre de dicho sensor) donde almacenar sus datos.

Campo	Tipo	Descripción
id	int(20)	Contiene el código numérico que identifica la incidencia. No puede tomar varias veces el mismo valor. No puede estar vacío. Si se recibe una incidencia sin este campo automáticamente se le asigna el número continuo a la última incidencia registrada.
tme	Datetime (6)	Objeto de tipo *datetime* que indica el instante de tiempo en el que se detectó la incidencia.
vel	Decimal (7,4)	Numero decimal que indica la velocidad del vehículo.
acc	Decimal (7,4)	Numero decimal que indica la aceleración del vehículo.

tipo	Int (11)	Código numérico que indica el tipo de vehículo.

Tabla 11: Descripción de las tablas de sensores.

- Tabla de descripción: Tabla única llamada 'Limite' donde se recoge la información acerca de la ubicación del sensor y los límites de tráfico admitidos en su localización.

Campo	Tipo	Descripción
Id	Int(20)	Código numérico que identifica al sensor. No puede tomar varias veces el mismo valor. No puede estar vacío. Si se recibe una incidencia sin este campo automáticamente se le asigna el número continuo a la última incidencia registrada.
calle	varchar(50)	Nombre del sensor .Tiene la característica *not null* (no puede estar vacío).
vel	int(7)	Máxima velocidad admitida en la zona donde se localiza el sensor.
vehículos	varchar(50)	Cada uno de los códigos numéricos correspondientes al tipo de vehículo no permitido en la localización de los sensores separados por una coma.
x	int(6)	Coordenada x donde se localiza el sensor dentro del mapa de la interfaz.
y	int(6)	Coordenada y dentro se localiza el sensor dentro del mapa de la interfaz.

Tabla 12: Descripción de la tabla de descripción.

5.2.2.3 API

Todas las funciones del API son realizadas por una misma clase de *Python*. De esta manera, todas las peticiones se escuchan por el mismo puerto de la *Raspberry* independientemente de la función que se solicite. Si en un futuro se desearía modificar esta cualidad, en la interfaz sólo sería necesario modificar la configuración de la conexión (más detalles en la sección 5.2.3.1).

Clases de librerías externas:

Flask	Obtenida del módulo *flask*. Implementa una aplicación WSGI (*Web Server Gateway Interface*)[37].
request	Obtenida del módulo *flask*. Permite obtener los argumentos de una petición[37].
MyEncoder	Clase *open source*. Convierte objetos de clase Decimal y Datetime en objetos compatibles con json[49].
json	Obtenida de la librería estándar de *Python*. Permite leer y escribir objetos de formato json[43].

Tabla 13: Clases externas empleadas en el API.

Este archivo contiene:

- una clase *open source* llamada *MyEncoder* [49] dedicada a hacer que los objetos almacenados en la base de datos tengan un formato compatible con json.

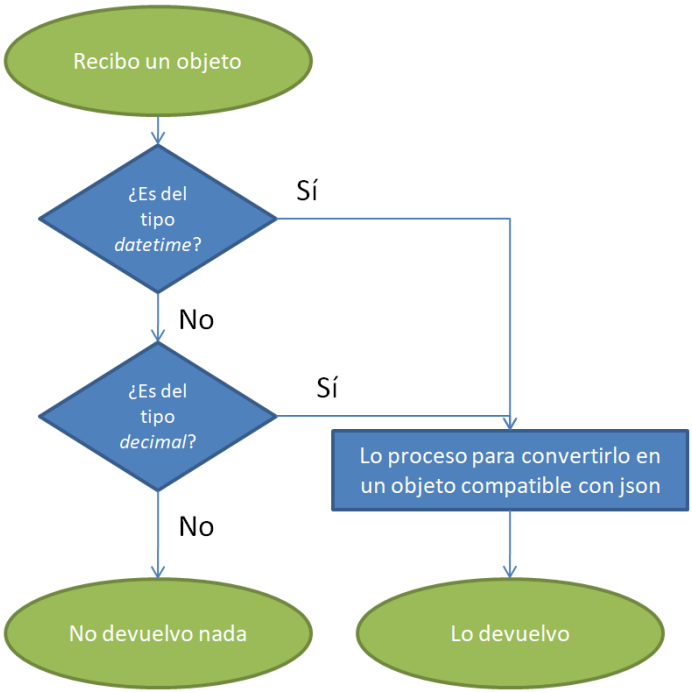


Figura 36: Diagrama de flujo de *MyEncoder*.

- una función llamada *get_data* que recibe la *query* (petición en SQL) y devuelve un objeto json que contiene la respuesta del programa.

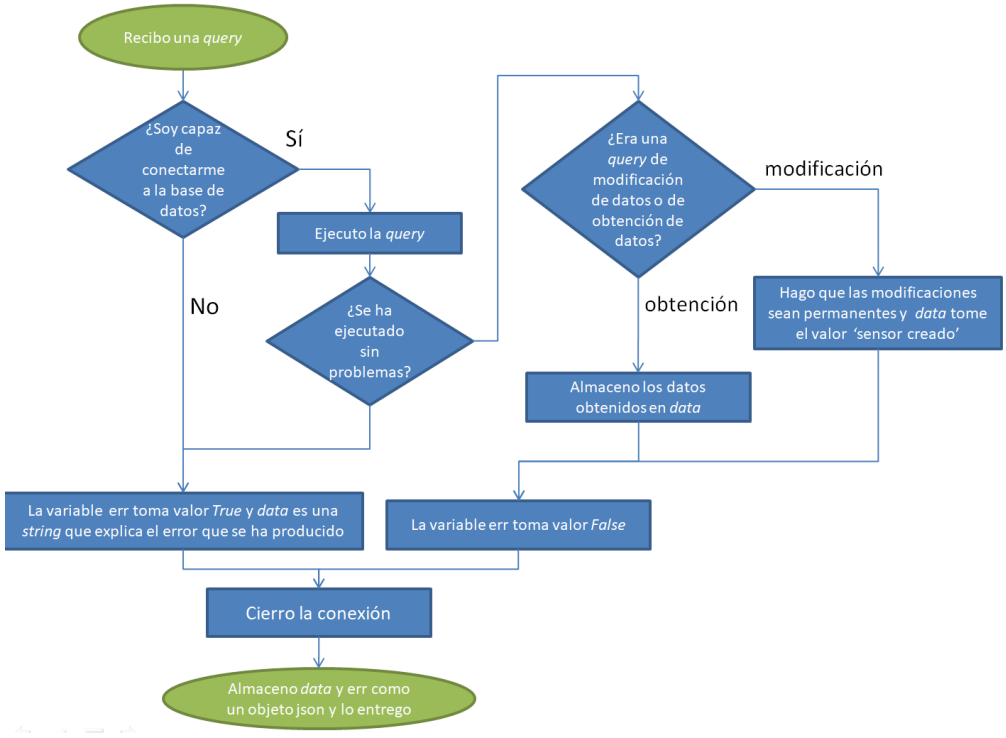


Figura 37: Diagrama de flujo de *get_data*.

- Un objeto de la clase Flask que implementa una aplicación WSGI (*Web Server Gateway Interface*) y actúa como el objeto central dedicado a recibir y procesar peticiones.

- 3 funciones que definen cómo se debe procesar la petición dependiendo del directorio al que vayan dirigida.

Para definir las 4 funciones primero es necesario decidir el método que van a emplear. En un API REST existen varios métodos que el usuario puede solicitar.

GET	Obtener un recurso.
POST	Crear nuevos recursos.
PUT	Actualizar un recurso. Requiere enviar el recurso modificado y el recurso sin modificar.
DELETE	Eliminar un recurso.
PATCH	Modificar un recurso. Requiere sólo enviar los cambios que se quieren aplicar, no el recurso entero.

Tabla 14: Métodos de un API REST

5.2.2.3.1 Límites y localización

Directorio: http://keltxo.no-ip.org:23040/limites

El objetivo de esta función es que el usuario pueda solicitar una lista de todos los sensores existentes, así como su localización en el mapa y sus límites en cuanto a velocidad y tipos de vehículo. Recibe como parámetro obligatorio el nombre del sensor y como opcionales el inicio y final de un intervalo de tiempo. Devuelve los datos correspondientes como un objeto json. Para ello empleamos el método *GET*.

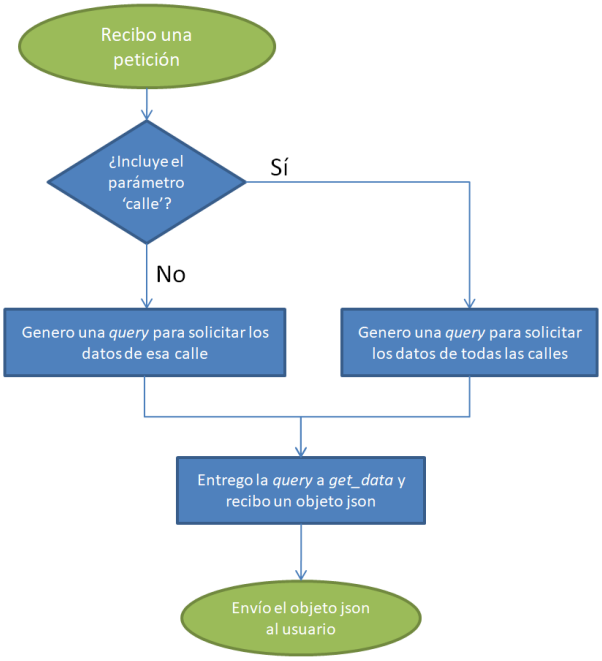


Figura 38: Diagrama de flujo de la función de límites y localización.

5.2.2.3.2 Datos históricos

Directorio: http://keltxo.no-ip.org:23040/

El objetivo de esta función es que el usuario pueda solicitar un listado de las incidencias detectadas en un sensor durante un intervalo de tiempo específico. Para ello empleamos el método *GET*.

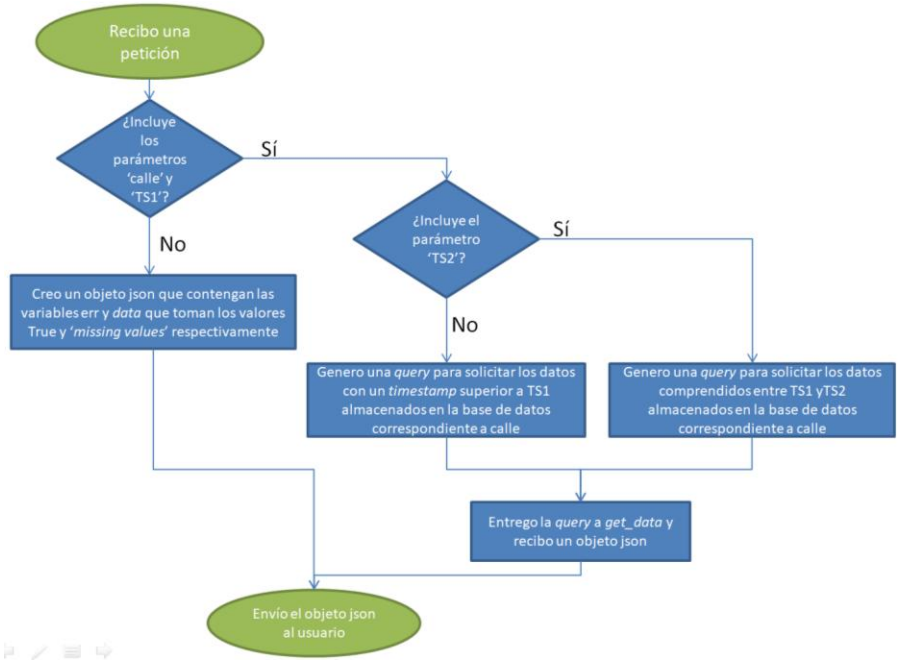


Figura 39: Diagrama de flujo de obtención de datos históricos.

5.2.2.3.3 Nuevos sensores

Directorio: http://keltxo.no-ip.org:23040/new

El objetivo de esta función es permitir al usuario crear un nuevo sensor en el sistema lo que se traduce en crear una nueva tabla dentro de la base de datos correspondiente a dicho sensor y añadir una nueva entrada a la tabla de límites. Para ello empleamos el método POST.

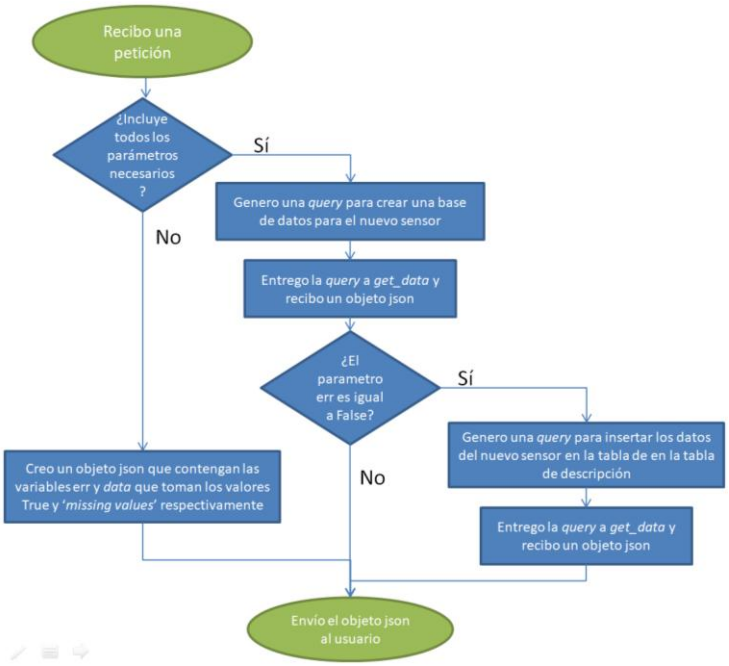


Figura 40: Diagrama de flujo para crear nuevos sensores.

5.2.3 Interfaz de usuario

La interfaz debe ser capaz de realizar 4 funciones:

- Configurar: modificar la configuración de la conexión con el resto del sistema.
- Datos históricos: solicitar y visualizar datos históricos.
- MQTT: Activar y desactivar la obtención de datos en tiempo real y visualizarlos.
- Generar nuevos sensores: solicitar la creación de un nuevo sensor junto con su base de datos.

Se ha diseñado el programa de tal manera que cada una de las cuatro funciones se realice en una pestaña distinta dentro de una misma ventana.

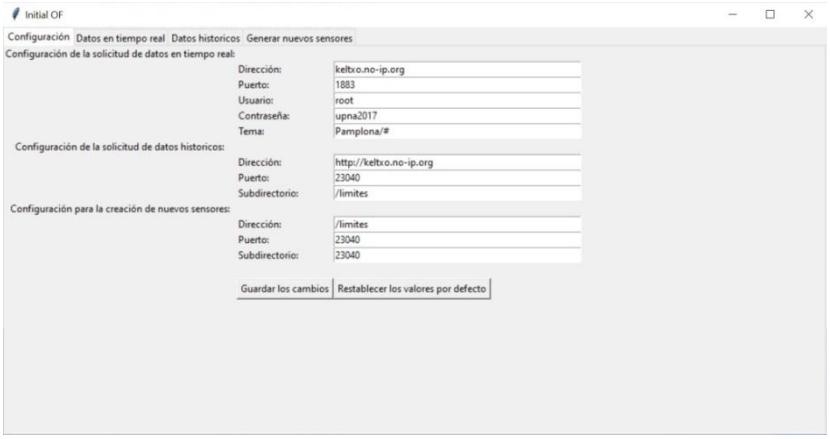


Figura 41: Captura del programa.

A nivel de programación es posible realizar las 4 funciones mediante una misma clase. Sin embargo, se ha decidido emplear 4 clases distintas con el fin de hacer que estas funcionen de la forma más independiente posible, así como facilitar tanto la lectura del código y el añadido de funcionalidades de cara al futuro.

Hay una clase central que tendrá como atributos un objeto de cada una de estas clases y que recibe el nombre "*Central_Hub*". Para su creación necesitamos emplear varias clases procedentes de librerías externas:

Frame	Obtenida de tkinter. Representa un espacio rectangular dentro de la ventana de la interfaz.
messagebox	Obtenida de tkinter. Esta clase proporciona una serie de modelos de ventanas para mostrar mensajes a los usuarios. Lo usamos para avisar al usuario de que se ha producido un error.
Notebook	Obtenida de tkinter. Permite generar un espacio dentro de un *Frame* donde se pueden añadir objetos *Frame* como pestañas.

Tabla 15: Clases de librerías externas usadas en *Central_Hub*.

Central_Hub es una clase que representa un *Frame* (un espacio rectangular dentro de la ventana de la interfaz) sobre el cual colocaremos las pestañas correspondientes a cada uno de los objetos.

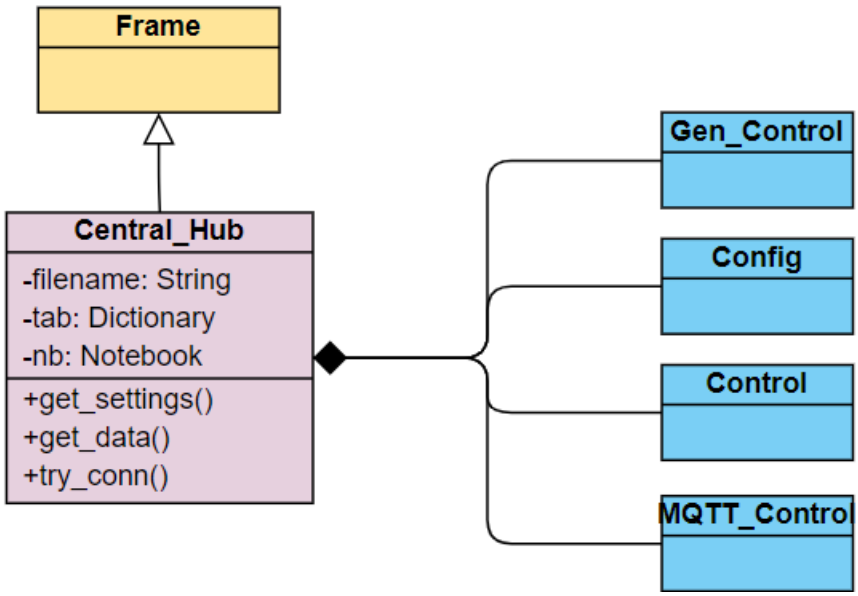


Figura 42: Diagrama de clases de *Central_Hub.*

El objetivo de *Central_Hub* es simplemente hacer de contenedor de los cuatro objetos y probar la conexión con el servidor. De los 4 objetos, el único que no necesita una conexión para funcionar es la clase 'Config' por lo que no tendría sentido iniciar los tres objetos restantes sin haber comprobado la conexión antes.

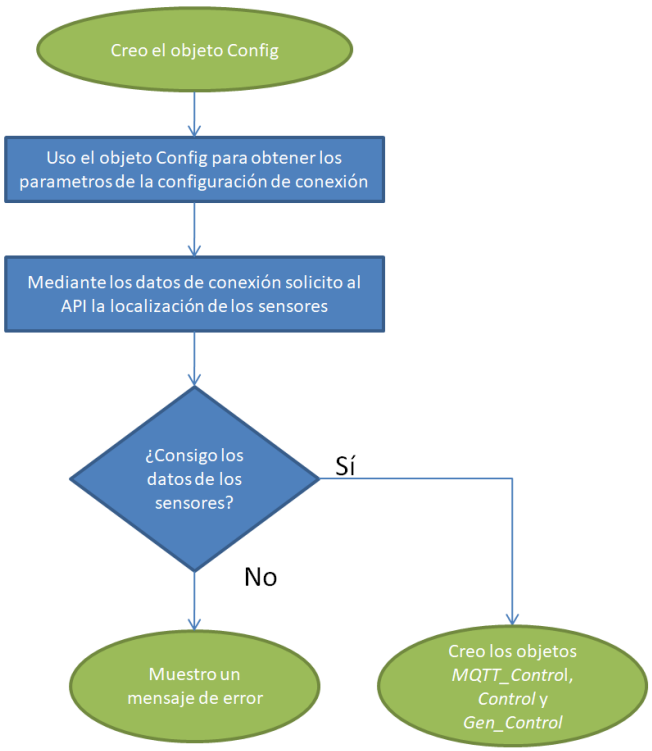


Figura 43: Diagrama de flujo del constructor de *Central_Hub.*

La prueba de la conexión se hace intentando obtener los datos de los límites y localización de cada sensor del servidor. Al obtener los límites y la localización se los pasamos a las

secciones del programa que las necesitan (datos en tiempo real y datos históricos) de esta manera evitamos tener que solicitarlos múltiples veces.

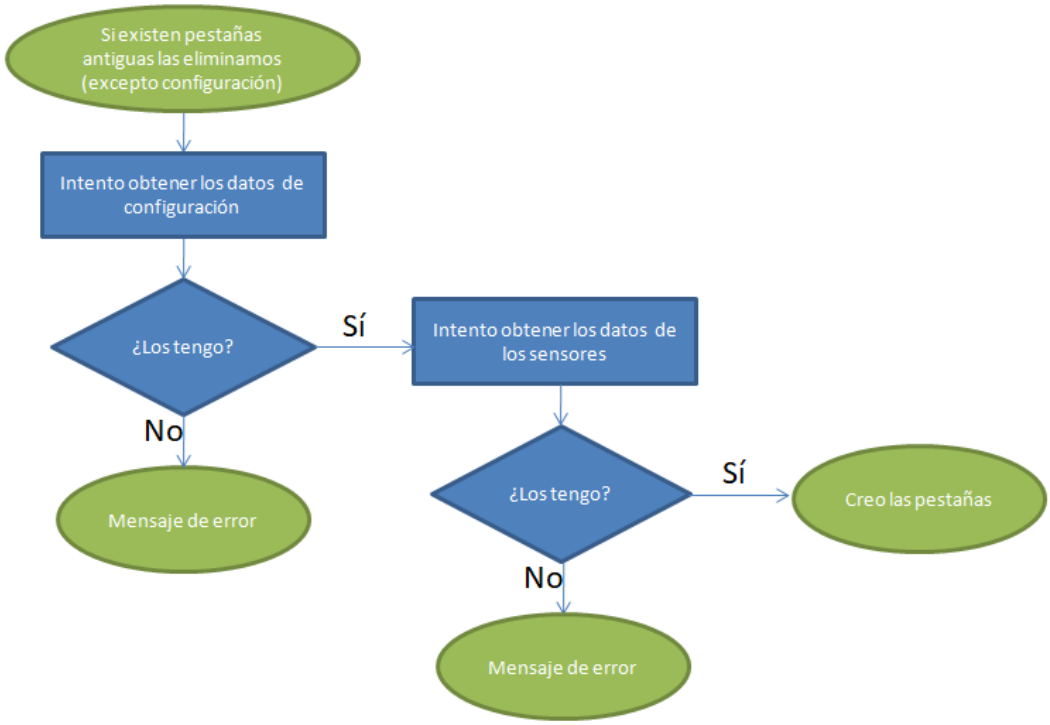


Figura 44: Diagrama de flujo de la prueba de conexión de *Central_Hub*.

5.2.3.1 *Configuración*

Clases de librerías externas empleadas:

Button	Obtenida de tkinter. Genera un botón interactivo [31].
entry	Obtenida de tkinter. Genera un espacio donde el usuario puede escribir[31].
Frame	Obtenida de tkinter. Representa un espacio rectangular dentro de la ventana de la interfaz[31].
json	Obtenida del módulo json. Permite leer y escribir objetos de formato json [43].
messagebox	Obtenida de tkinter. Esta clase proporciona una serie de modelos de ventanas para mostrar mensajes a los usuarios. Lo usamos para avisar al usuario de que se ha producido un error[31].

Tabla 16: Clases de librerías externas usadas para la configuración.

Clases creadas para esta sección:

Config	Permite leer y editar un archivo de extensión '.json' donde se almacenan los detalles de la conexión con el servidor y el *broker*.

Tabla 17: Clases creadas para la configuración.

Para que el usuario sea capaz de modificar los parámetros de conexión y que dichas modificaciones se almacenen para la siguiente vez que se abra el programa lo mejor es almacenar dichos parámetros en un archivo y modificar dicho archivo. Para realizar esta función sólo es necesario la creación de una única clase: 'Config'.

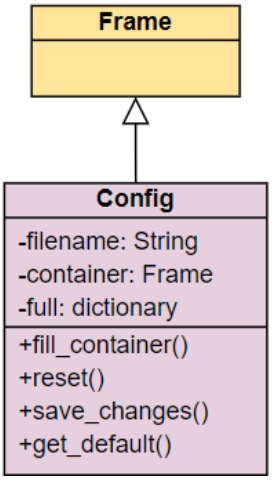


Figura 45: Diagrama de clases para la configuración del cliente.

El archivo donde se almacena los parámetros de conexión recibe el nombre '*settings.json*'. Si bien hubiera sido posible emplear otras extensiones como '.txt', '.json' ha sido seleccionada porque es muy similar a la estructura de datos 'diccionario' que es típica de *Python* y porque la librería estándar de *Python* contiene una clase capaz de leer y escribir objetos de tipo json.

Nada más crear un objeto de la clase Config se comprueba que *settings.json* existe y contiene todos los parámetros necesarios.

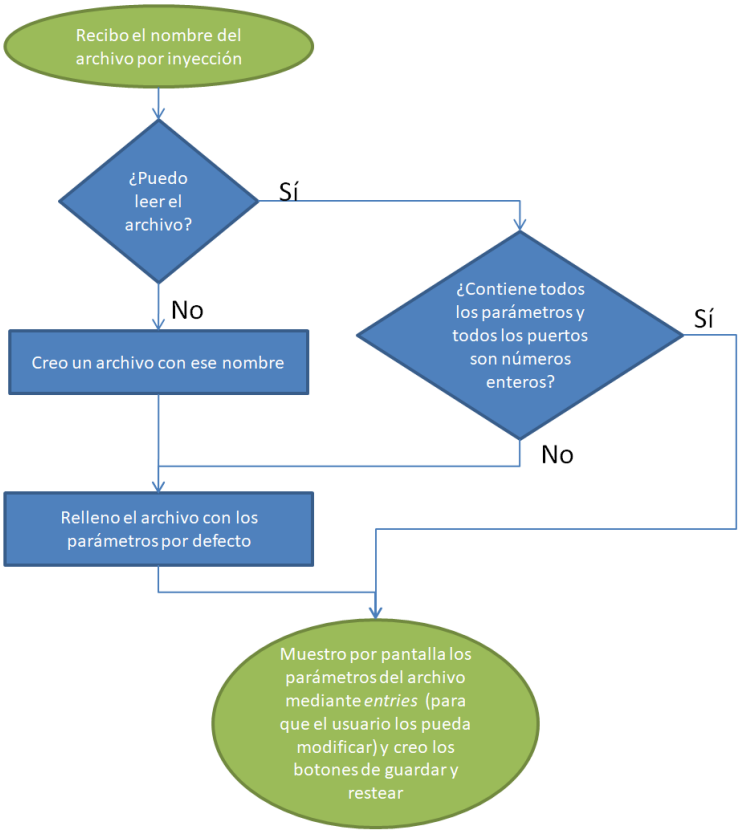


Figura 46: Diagrama de flujo del constructor de la clase Config.

Una vez que se hayan obtenido los parámetros se procede a rellenar la pestaña con estos colocados dentro de *entries* con el fin de que el usuario pueda editarlos. Por último, se añaden dos botones a la parte inferior de la ventana: 'Guardar los cambios' y 'Restablecer los valores por defecto'.

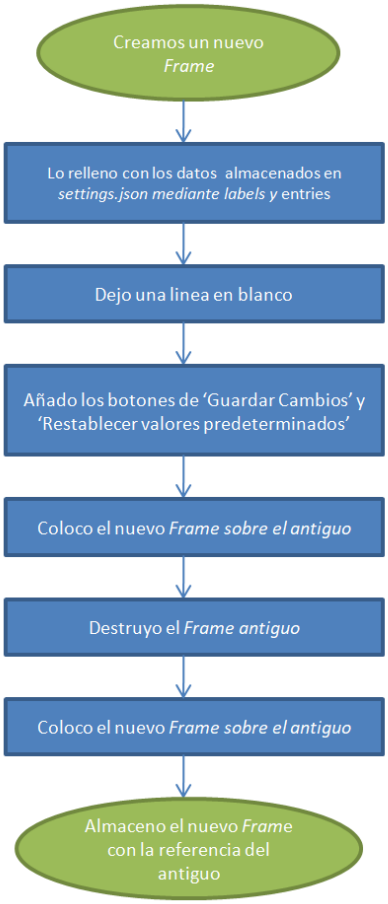


Figura 47: Diagrama de flujo de la generación de un nuevo *Frame* en la pestaña de configuración.

El proceso de colocar el nuevo *Frame* se hace de esta manera para evitar un efecto de parpadeo en la ventana al llamar a la función 'Restablecer los valores por defecto'.

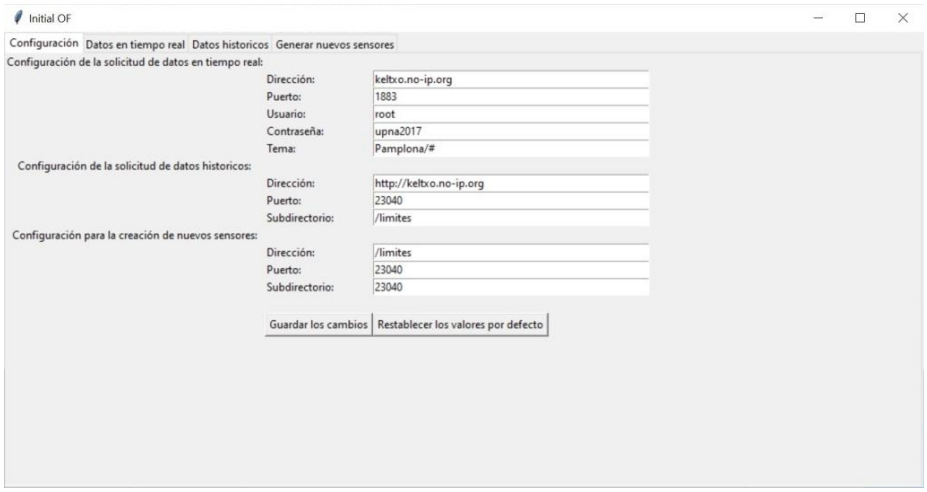


Figura 48: Pestaña de configuración.

Como se puede ver en Figura 48, todas las secciones relacionadas con el API tienen un campo de puerto, de dirección y subdirectorio. De esta manera si se realiza alguna modificación al API no es necesario modificar el código del programa, sino que el propio usuario puede modificar la configuración de la conexión.

Guardar los cambios edita los contenidos de *settings.json* de acuerdo con los datos introducidos en el programa, pero sólo si todos los parámetros tienen valores aceptables. Una vez editado el programa se pide a *Central_Hub* que pruebe la conexión. Si la conexión falla *Central_Hub* muestra al usuario un mensaje de error.

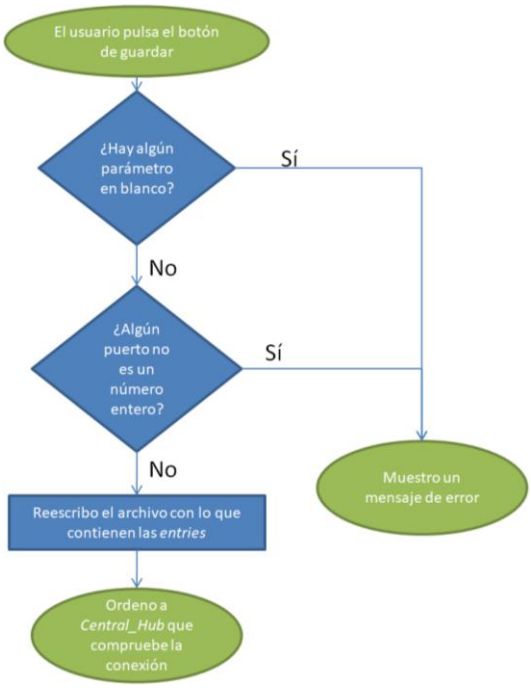


Figura 49: Diagrama de flujo de la edición de parámetros de configuración.

Restablecer los valores por defecto, modifica *settings.json* y vuelve a cargar lo contenido en *entries* para reflejar la modificación.

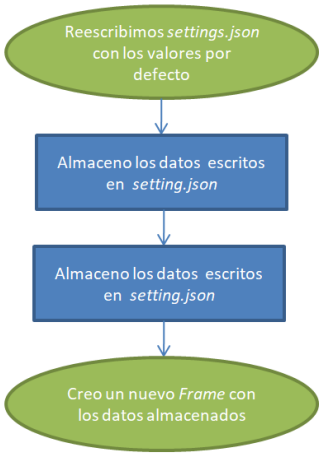


Figura 50: Diagrama de flujo de restablecer los datos por defecto en configuración.

5.2.3.2 Datos en tiempo real

Clases de librerías externas empleadas:

annotate	Obtenida de matplotlib. Permite realizar anotaciones dentro de un gráfico[41].
ax	Obtenida de matplotlib. Contiene la mayoría de los elementos de figure y fija el sistema de coordenadas. Actúa de intermediario entre *Figure* y el gráfico[41].
Dataframe	Obtenida del módulo pandas. Es una estructura de datos bidimensional con ejes que se pueden etiquetar. Permite realizar operaciones lógicas y aritméticas [35].
Figure, FigureCanvasTkAgg	Obtenidas de matplotlib. *Figure* permite definir un espacio donde se va a colocar un gráfico, mientras que *FigureCanvasTkAgg* permite colocar Figure dentro del entorno definido por tkinter [33].
Frame	Obtenida de tkinter. Representa un espacio rectangular dentro de la ventana de la interfaz[31].
imread	Obtenida de matplotlib. Permite leer una imagen y colocarla en un gráfico[41].
messagebox	Obtenida de tkinter. Esta clase proporciona una serie de modelos de ventanas para mostrar mensajes a los usuarios. Lo usamos para avisar al usuario de que se ha producido un error[31].
Queue	Obtenida de la librería estándar de *Python*. Es una cola de mensajes que permite comunicarse a dos hilos distintos [36].
scatter	Obtenida de matplotlib. Permite dibujar un gráfico de puntos de acuerdo con los datos que recibe por inyección[41].
Table, TableModel	Obtenidas del módulo pandastable. *Table* permite generar un elemento grafico capaz de representar un *Dataframe* como una tabla dentro de la interfaz y *TableModel* permite realizar modificaciones a dicho elemento [32].
thread_with_exception	Clase *open source* heredera de *Thread*. Al igual que *Thread* que genera un hilo (un proceso aparte del bucle central del programa) sin embargo esta clase incluye la función de parar el hilo [50].

Tabla 18: Clases de librerías externas empleadas para la obtención de datos en tiempo real.

Clases creadas para esta sección:

MQTT_Control	Actúa de contenedor de *MQTT_Display* y de los controles del usuario dentro de la ventana.
MQTT_Display	Actúa de contenedor de *MQTT_Map* y de *MyTk*.
MQTT_Map	Clase dedicada a representar en un mapa los sensores del sistema y los últimos datos recibidos de los mismos.
MyTk	Clase dedicada a mostrar los contenidos de un *Dataframe* por pantalla. Hace que los datos sean descargables en formato CVS. Incluye la capacidad de hacer *scroll.*
MyMQTT	Clase dedicada a la comunicación con el *broker*.
MQTT_Parser	Clase dedicada a poner los mensajes recibidos en el formato correcto.

Tabla 19: Clases creadas para este proyecto para la obtención de datos en tiempo real.

El objetivo de esta sección es darle al usuario la capacidad de activar y detener la obtención de datos en tiempo real, así como mostrar los datos obtenidos.

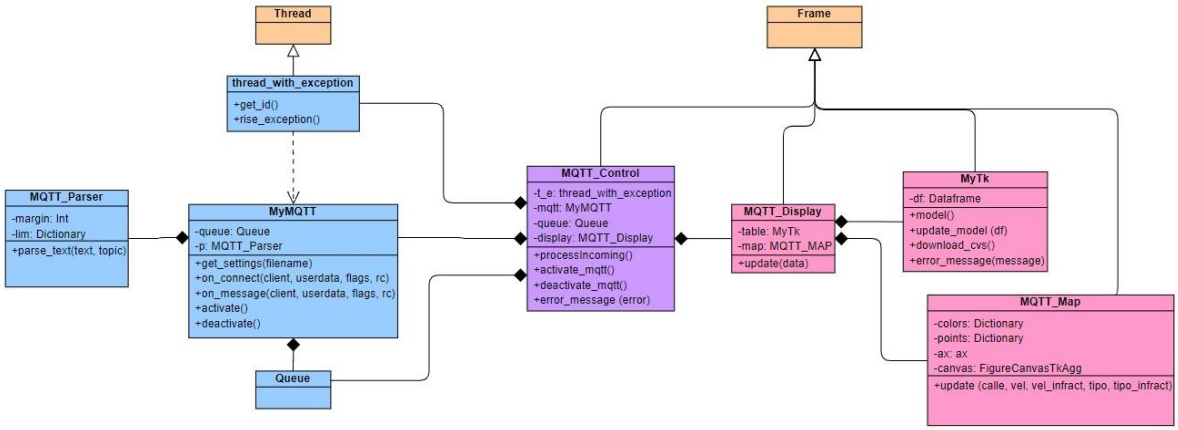


Figura 51: Diagrama de clases para la obtención de datos en tiempo real.

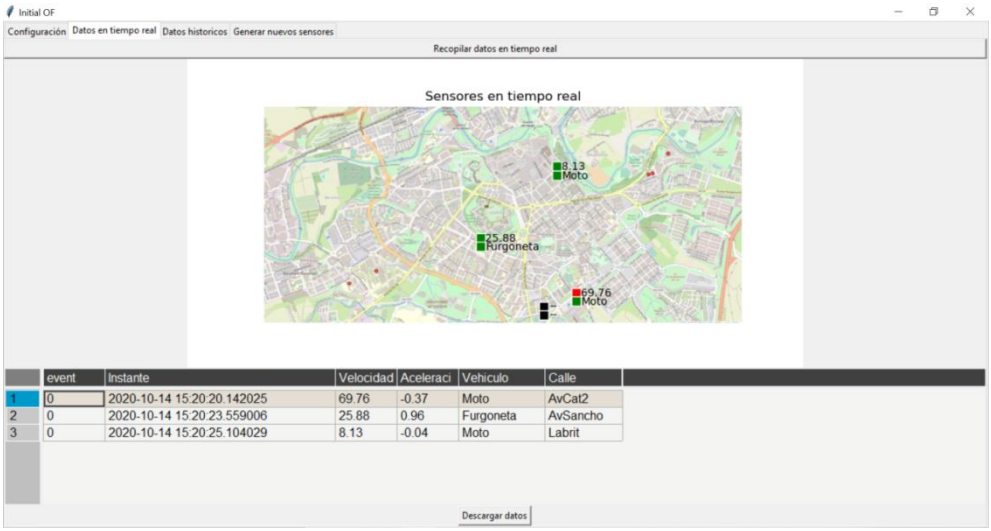


Figura 52: Pestaña de datos obtenidos en tiempo real.

Podemos dividir el conjunto de clases empleadas para esta función en tres secciones: lógica (azul en Figura 51), control (morado en Figura 51) y *display (rosa en* Figura 51*)*. Lógica es el conjunto de clases que se encargan de la comunicación con el *broker* y del procesado de los mensajes. *Display* se encarga de mostrar en pantalla los datos recibidos. Control se encarga de procesar las interacciones del usuario y de hacer de puente entre las secciones de *display* y lógica.

Control se encarga de activar y desactivar la conexión con el *broker* de acuerdo a lo que indique el usuario. Esto lo hace activando y desactivando el objeto de la clase *thread_with_exception*, que se encarga de ejecutar el bucle de *MyMQTT* de conectarse al sistema y recibir mensajes. El motivo por el cual debemos emplear un hilo externo en lugar de usar el hilo principal del programa para conectarnos y recibir mensajes de MQTT es porque ambas acciones (administrar la ventana del programa y escuchar la conexión MQTT) son dos procesos que deben estar en constante ejecución y cada hilo sólo puede realizar un único proceso.

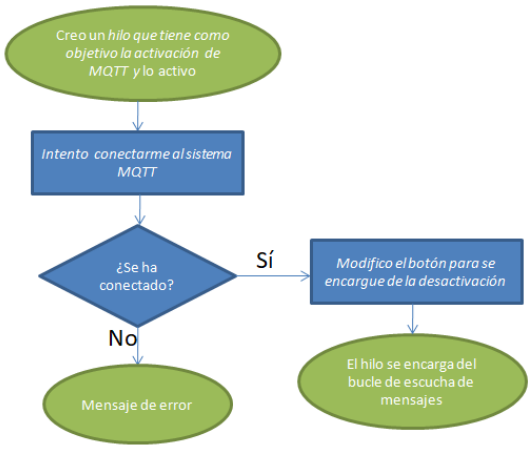


Figura 53: Diagrama de flujo de la activación de MQTT.

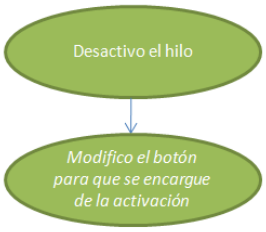


Figura 54: Diagrama de flujo de la desactivación de MQTT.

La sección de lógica engloba todo el proceso de escucha y procesado de mensajes. Esta sección es controlada por el objeto thread_with_exception. El procesado de mensajes se lleva a cabo por el objeto *MQTT_Parser* que se encarga de convertir el mensaje de texto recibido por *MyMQTT* en un objeto de tipo diccionario que contiene los datos necesarios para actualizar la sección de *display*.

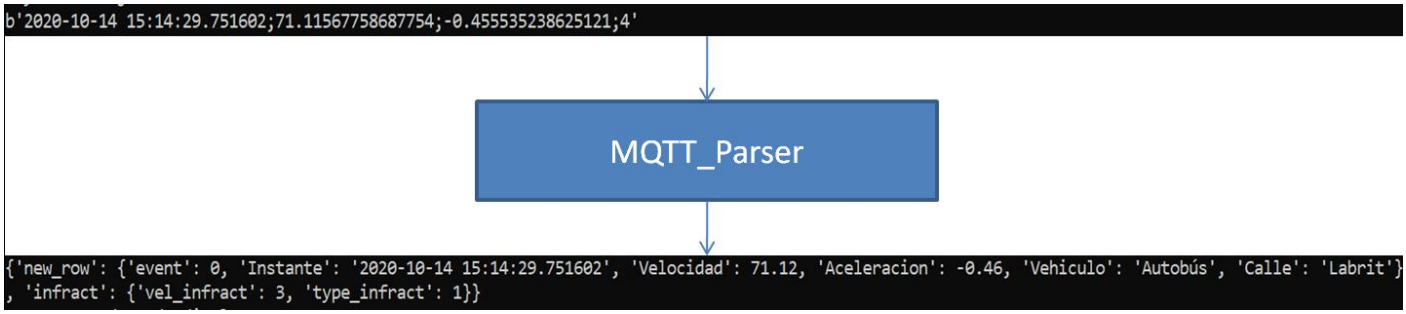


Figura 55: Ejemplo del funcionamiento de MQTT_Parser

Sin embargo, no podemos hacer que el hilo de lógica llame directamente a las funciones de los objetos de *display* porque esto haría que el programa se quedara bloqueado. Para poder hacer que los dos hilos sean capaces comunicarse datos entre ellos necesitamos implementar una cola de mensajes.

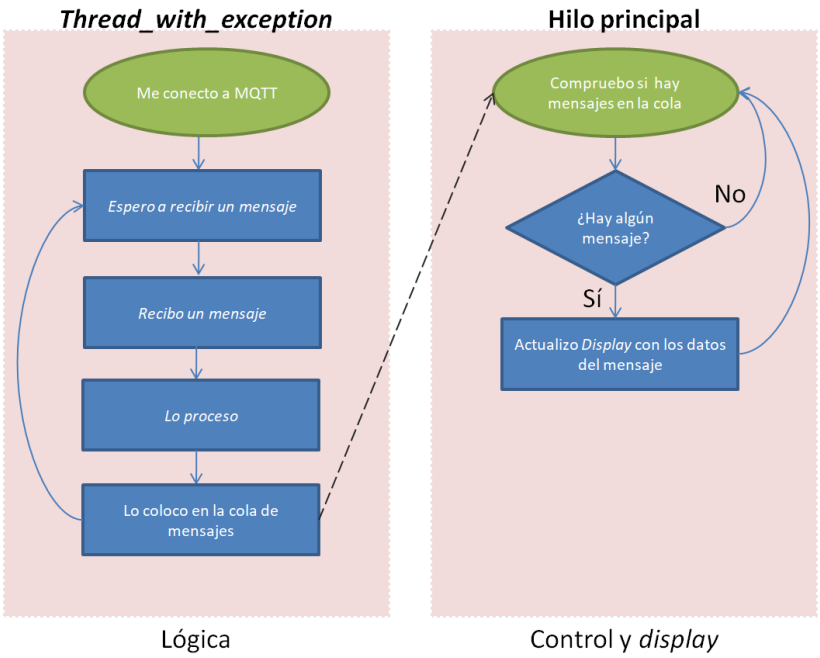


Figura 56: Diagrama de flujo de la obtención de datos en tiempo real

Cuando control le pasa a *display* los datos generados por lógica, *MQTT_Display* se encarga de pasar los datos necesarios a *MQTT_Map* y *MyTk* para que estas se actualicen.

Un dato por destacar sobre esta sección es *MQTT_Map*. Si bien es cierto que Python tiene librerías que permiten generar mapas interactivos (por ejemplo, folium), ninguna de estas es compatible con tkinter. Debido a esta limitación, MQTT_Map en realidad no es un mapa, sino un gráfico con una imagen del mapa de pamplona de fondo del cual modificamos ciertos parámetros con cada orden de actualizar *display*.

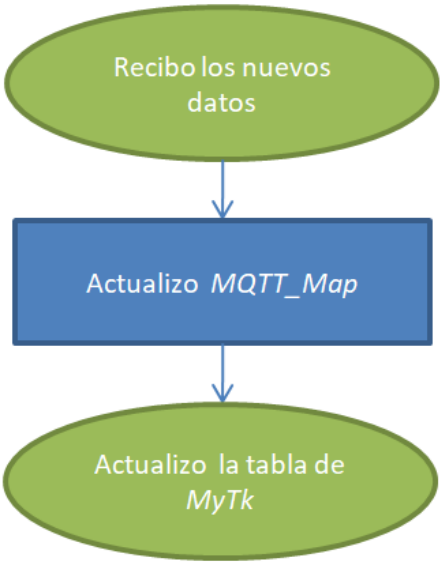


Figura 57: Diagrama de flujo de la actualización de *display*.

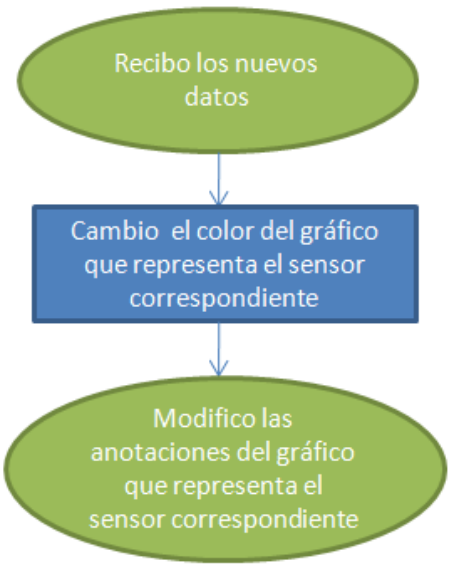


Figura 58: Diagrama de flujo de la actualización de MQTT_Map.

5.2.3.3 Datos históricos

Clases de librerías externas empleadas:

annotate	Obtenida de matplotlib. Permite realizar anotaciones dentro de un gráfico[41].
ax	Obtenida de matplotlib. Contiene la mayoría de los elementos de *Figure* y fija el sistema de coordenadas. Actúa de intermediario entre *Figure* y el gráfico[41].
Dataframe	Obtenida del módulo pandas. Es una estructura de datos bidimensional con ejes que se pueden etiquetar. Permite realizar operaciones lógicas y aritméticas [35].
DateEntry	Obtenida de tkcalendar. Permite escoger una fecha dentro de un calendario[42].
entry	Obtenida de tkinter. Genera un espacio donde el usuario puede escribir[31].
Figure, FigureCanvasTkAgg	Obtenidas de matplotlib. *Figure* permite definir un espacio donde se va a colocar un gráfico, mientras que *FigureCanvasTkAgg* permite colocar Figure dentro del entorno definido por tkinter [33].
Frame	Obtenida de tkinter. Representa un espacio rectangular dentro de la ventana de la interfaz[31].
imread	Obtenida de matplotlib. Permite leer una imagen y colocarla en un gráfico[41].
json	Obtenida del módulo json. Permite leer y escribir objetos de formato json[43].
messagebox	Obtenida de tkinter. Esta clase proporciona una serie de modelos de ventanas para mostrar mensajes a los usuarios. Lo usamos para avisar al usuario de que se ha producido un error[31].
Notebook	Obtenida de tkinter. Permite generar un espacio dentro de un *Frame* donde se pueden añadir objetos *Frame* como pestañas[31].
requests	Obtenida de requests. Permite comunicarse con http[38].
scatter	Obtenida de matplotlib. Permite dibujar un gráfico de puntos de acuerdo con los datos que recibe por inyección[41].
Table, TableModel	Obtenidas del módulo pandastable. *Table* permite generar un elemento grafico capaz de representar un *Dataframe* como una tabla dentro de la interfaz y *TableModel* permite realizar modificaciones a dicho elemento [32].

Tabla 20: Clases de librerías externas empleadas para obtener datos históricos.

Clases creadas para esta sección:

Control	Permite al usuario generar peticiones. Genera una pestaña por cada sensor seleccionado en *Map*.
Display	Actúa de contenedor de las clases *Graphs*, *Flux*, *Infract* y *MyTk*. Mostrando cada una como una pestaña.
Map	Mapa interactivo que se usa para seleccionar los sensores a visualizar de forma clara e intuitiva.
Graph	*Frame* con *scroll*.
Graphs	Heredera de *Graph*. Permite la visualización de los datos a través de una serie de gráficos.

Flux	Heredera de *Graph*. Permite la visualización del flujo de tráfico a través de una serie de gráficos interactivos.
Infract	Permite visualizar las infracciones
MyTk	Clase dedicada a mostrar los contenidos de un Dataframe por pantalla como *frame* una tabla con los datos según van llegando. Hace que los datos sean descargables en formato CVS. Incluye la capacidad de hacer *scroll.*
Interplot	Gráfico interactivo que permite escoger las líneas que son visibles y las que no a través de la leyenda

Tabla 21: Clases creadas para la obtención de datos históricos.

El objetivo de esta sección es darle al usuario la capacidad de solicitar datos históricos a la base de datos, así como de visualizarlos de manera clara e intuitiva.

Para comprender el funcionamiento del programa es mejor dividirlo en dos partes: Obtención de datos (morado en Figura 59) y visualización de datos (rosa en Figura 59).

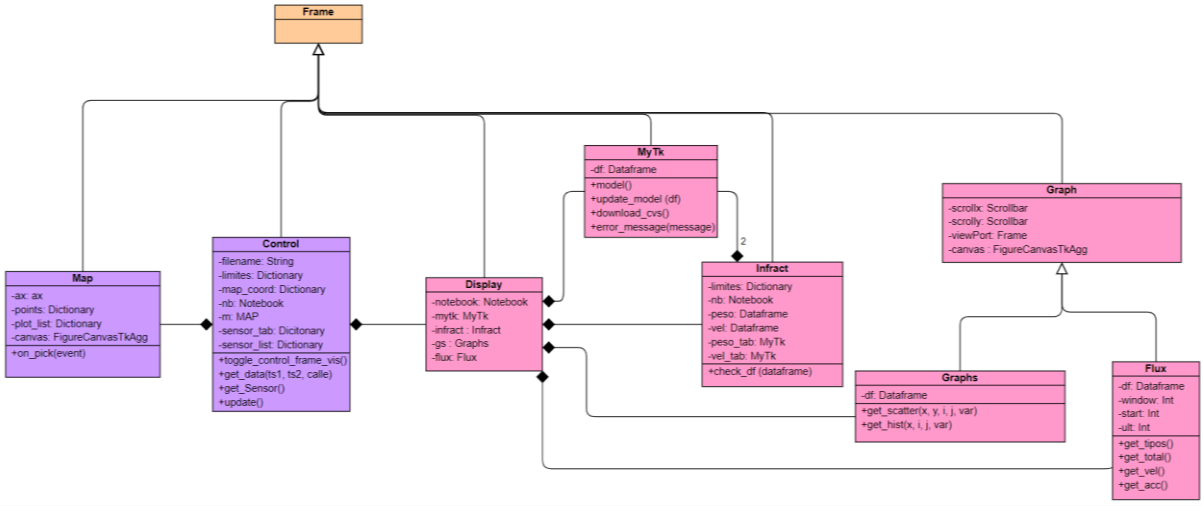


Figura 59: Diagrama de clases de la obtención de datos históricos.

Obtención de datos es la parte del programa dedicada a la generación de peticiones de datos y la obtención de datos. La clase *Control* es la clase central de esta sección y se encarga de actuar como contenedor gráfico, enviar peticiones al API y procesar la respuesta. Para permitir al usuario especificar los datos que desea obtener utilizamos dos secciones:

1. las clases *entry* y *DateEntry* para permitir al usuario escoger la hora y fecha correspondiente al inicio y final del intervalo a solicitar.
2. la clase *Map* para permitir al usuario seleccionar aquellos sensores de los que deseamos obtener los datos.

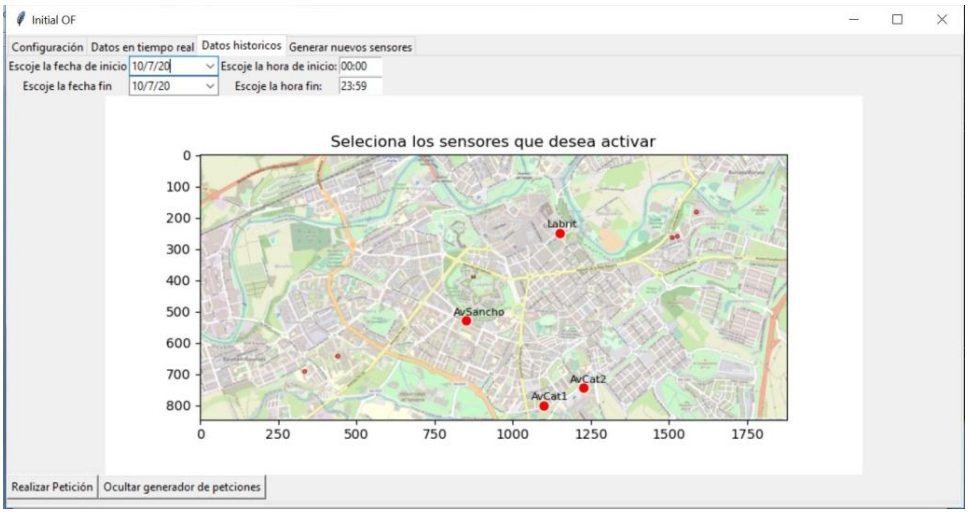


Figura 60: Captura de la pestaña de datos históricos.

Al igual que *MQTT_Map, Map* no es un mapa interactivo sino un gráfico un mapa de pamplona de fondo. En este caso, cuando el usuario hace *click* sobre uno de los puntos que representa un sensor, se produce y se procesa un evento de *click*:

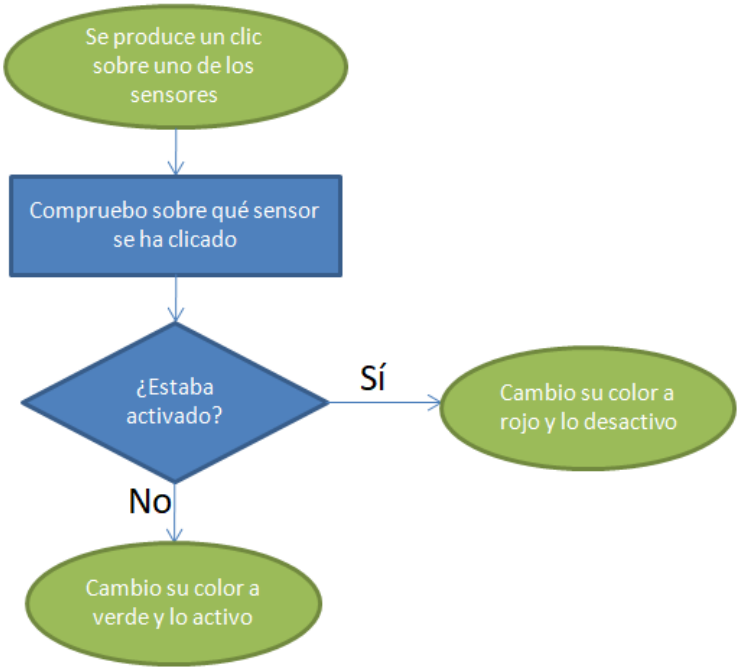


Figura 61: Diagrama de flujo del mapa empleado para la obtención de datos históricos.

Una vez el usuario ha generado la petición, procedemos a transmitirla al api. Si se produce un error en la conexión o no existen datos para la petición solicitada se lo indicamos al usuario mediante la clase *messagebox.*

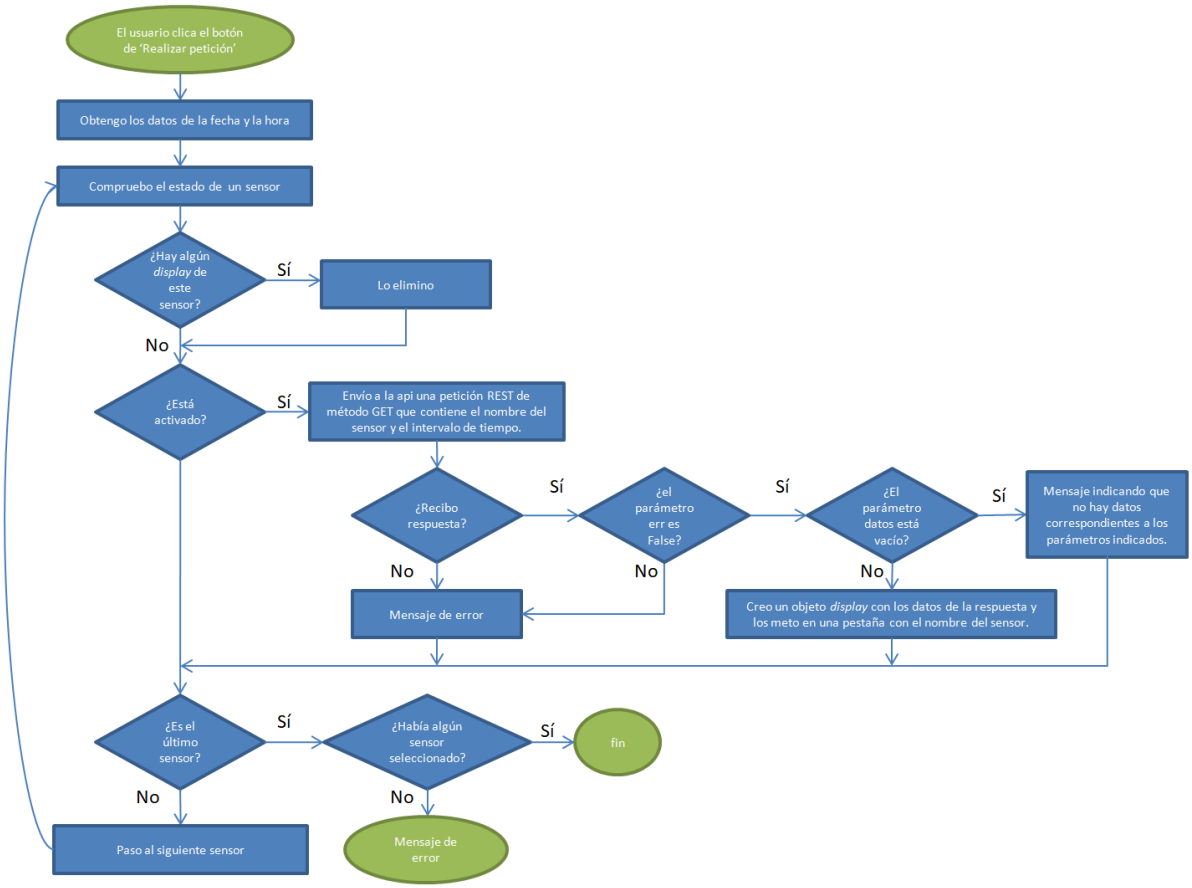


Figura 62: Diagrama de flujo de la obtención y visualización de datos históricos.

Una vez obtenidos estos datos pasamos a la sección de visualización. Dado que esta sección ocupa mucho espacio dentro de la ventana, se ha creado el botón 'Ocultar generador de peticiones'. Este botón permite ocultar las partes de la ventana dedicada a generar peticiones y volver a hacerlas visibles.

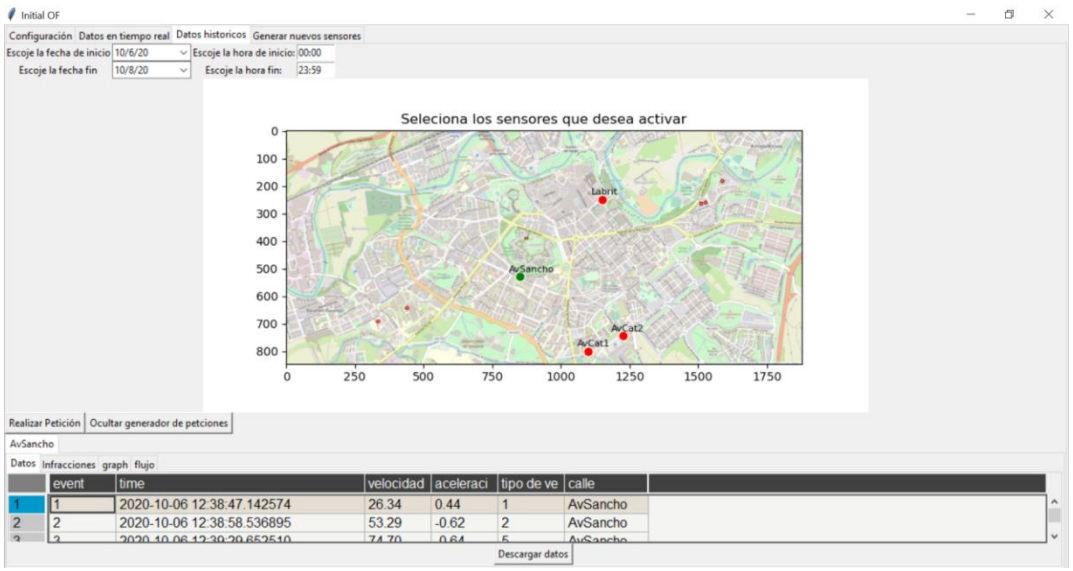


Figura 63: Pestaña de Datos históricos completa.

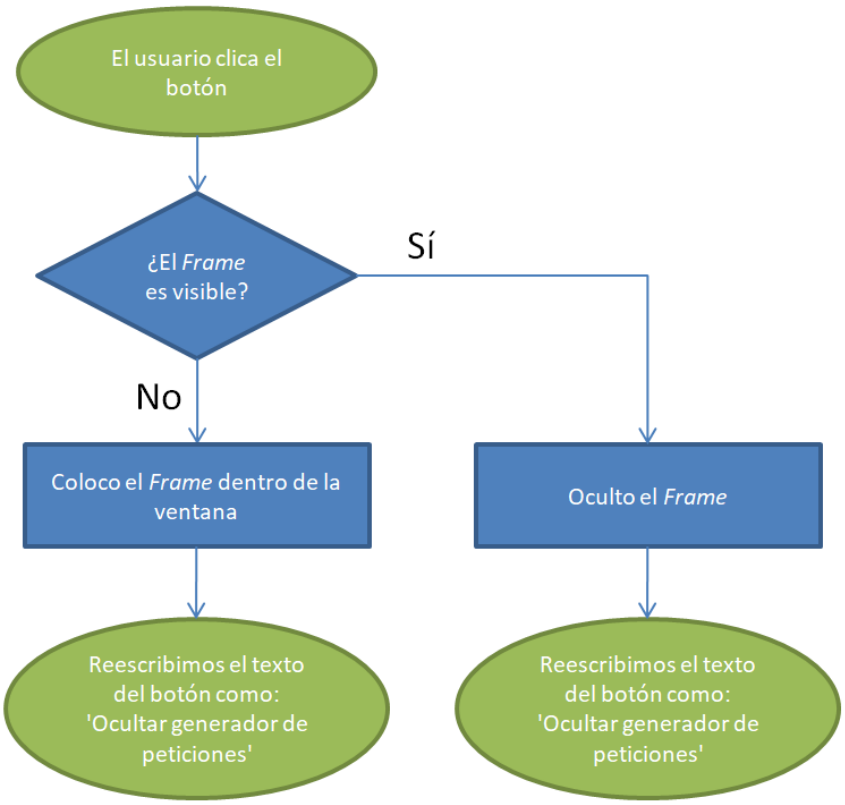
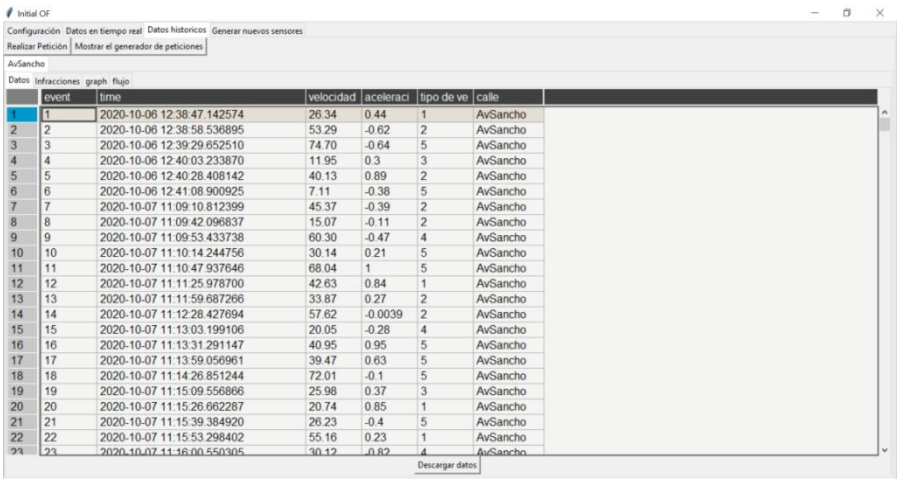


Figura 64: Diagrama de flujo de la ocultación del generado de peticiones de datos históricos.



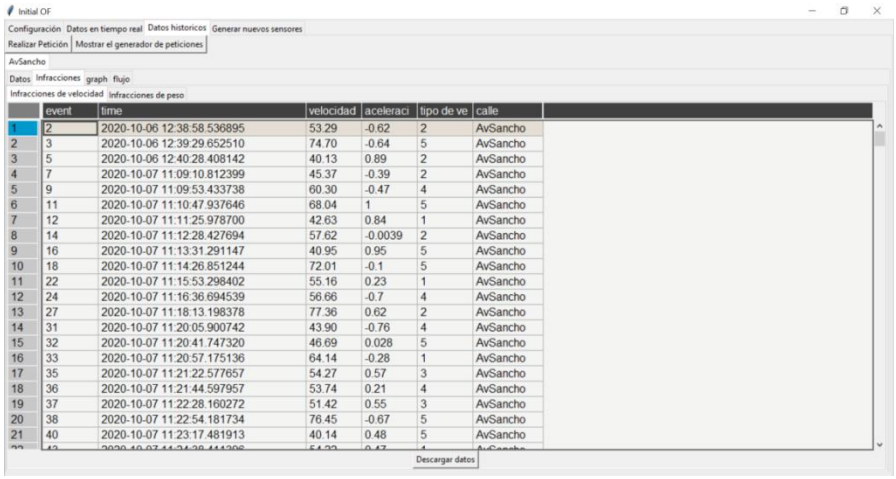
	event	time	velocidad	aceleraci	tipo de ve	calle
1	1	2020-10-06 12:38:47.142574	26.34	0.44	1	AvSancho
2	2	2020-10-06 12:38:58.536895	53.29	-0.62	2	AvSancho
3	3	2020-10-06 12:39:29.652510	74.70	-0.64	5	AvSancho
4	4	2020-10-06 12:40:03.233870	11.95	0.3	3	AvSancho
5	5	2020-10-06 12:40:28.408142	40.13	0.89	2	AvSancho
6	6	2020-10-06 12:41:08.900925	7.11	-0.38	5	AvSancho
7	7	2020-10-07 11:09:10.812399	45.37	-0.39	2	AvSancho
8	8	2020-10-07 11:09:42.096837	15.07	-0.11	2	AvSancho
9	9	2020-10-07 11:09:53.433738	60.30	-0.47	4	AvSancho
10	10	2020-10-07 11:10:14.244756	30.14	0.21	5	AvSancho
11	11	2020-10-07 11:10:47.937646	68.04	1	5	AvSancho
12	12	2020-10-07 11:11:25.978700	42.63	0.84	1	AvSancho
13	13	2020-10-07 11:11:59.687266	33.87	0.27	2	AvSancho
14	14	2020-10-07 11:12:28.427694	57.62	-0.0039	2	AvSancho
15	15	2020-10-07 11:13:03.199106	20.05	-0.28	4	AvSancho
16	16	2020-10-07 11:13:31.291147	40.95	0.95	5	AvSancho
17	17	2020-10-07 11:13:59.056961	39.47	0.63	5	AvSancho
18	18	2020-10-07 11:14:26.851244	72.01	-0.1	5	AvSancho
19	19	2020-10-07 11:15:09.556866	25.98	0.37	3	AvSancho
20	20	2020-10-07 11:15:26.662287	20.74	0.85	1	AvSancho
21	21	2020-10-07 11:15:39.384920	26.23	-0.4	5	AvSancho
22	22	2020-10-07 11:15:53.298402	55.16	0.23	1	AvSancho
23	23	2020-10-07 11:16:00.550305	30.12	-0.82	4	AvSancho

Figura 65: Pestaña de Datos históricos tras ocultar el generador de peticiones.

Para tomar la decisión de cuál sería la mejor manera de visualizar los datos históricos es necesario tener en cuenta cuáles son las necesidades del cliente. En este caso tenemos dos clientes que satisfacer: UPNA y Ayuntamiento de Pamplona. Para conocer las necesidades de la UPNA se han empleado varias publicaciones sobre los sensores de tráfico[9][13] .Mientras que para conocer las necesidades del ayuntamiento se ha consultado un informe de análisis de tráfico[51].

Para visualizar los datos creamos un objeto de la clase *display* por cada uno de los sensores que el usuario haya solicitado. La clase *display* sirve de contenedor de 4 clases diseñadas para la visualización de datos:

- MyTk: Su objetivo es mostrar los datos solicitados dentro de una tabla y hacer que dichos datos sean descargables en formato CVS.
  Ver Figura 65
- Infract: Contiene dos pestañas cada una de las cuales contiene un objeto MyTk que muestran las infracciones de velocidad y las infracciones de tipo de vehículo que hay entre los datos obtenidos.



	event	time	velocidad	aceleraci	tipo de ve	calle
1	2	2020-10-06 12:38:58.536895	53.29	-0.62	2	AvSancho
2	3	2020-10-06 12:39:29.652510	74.70	-0.64	5	AvSancho
3	5	2020-10-06 12:40:28.408142	40.13	0.89	2	AvSancho
4	7	2020-10-07 11:09:10.812399	45.37	-0.39	2	AvSancho
5	9	2020-10-07 11:09:53.433738	60.30	-0.47	4	AvSancho
6	11	2020-10-07 11:10:47.937646	68.04	1	5	AvSancho
7	12	2020-10-07 11:11:25.978700	42.63	0.84	1	AvSancho
8	14	2020-10-07 11:12:28.427694	57.62	-0.0039	2	AvSancho
9	16	2020-10-07 11:13:31.291147	40.95	0.95	5	AvSancho
10	18	2020-10-07 11:14:26.851244	72.01	-0.1	5	AvSancho
11	22	2020-10-07 11:15:53.298402	55.16	0.23	1	AvSancho
12	24	2020-10-07 11:16:36.694539	56.66	-0.7	4	AvSancho
13	27	2020-10-07 11:18:13.198378	77.36	0.62	2	AvSancho
14	31	2020-10-07 11:20:05.900742	43.90	-0.76	4	AvSancho
15	32	2020-10-07 11:20:41.747320	46.69	0.028	5	AvSancho
16	33	2020-10-07 11:20:57.175136	64.14	-0.28	1	AvSancho
17	35	2020-10-07 11:21:22.577657	54.27	0.57	3	AvSancho
18	36	2020-10-07 11:21:44.597957	53.74	0.21	4	AvSancho
19	37	2020-10-07 11:22:28.160272	51.42	0.55	3	AvSancho
20	38	2020-10-07 11:22:54.181734	76.45	-0.67	5	AvSancho
21	40	2020-10-07 11:23:17.481913	40.14	0.48	5	AvSancho

Figura 66: Datos históricos, pestaña de Infracciones.

- *Graphs*: clase heredera de *Graph*. Su objetivo es visualizar una serie de gráficos que representan los datos obtenidos: histograma de velocidad, histograma de aceleración, un gráfico de barras indicando el número de vehículos de cada tipo, un gráfico mostrando cada incidencia mediante su velocidad y el instante en el que se produce y un gráfico mostrando cada incidencia mediante su aceleración y el instante en el que se produce. Todos estos gráficos son descargables.

Figura 67: Datos históricos, pestaña de gráficos.

- *Flux*: clase heredera de *Graph*. Muestra el flujo de tráfico con una ventana de 8 horas en varios gráficos interactivos: el flujo total, flujo divido por tipo de vehículo, flujo dividido en 4 intervalos de velocidad y flujo divido en aceleración y frenado. Estos gráficos son interactivos y permiten al usuario seleccionar aquellas líneas de flujo que deseen ocultar o volver a visualizar. Todos estos gráficos son descargables y son implementados con *Interplot.*

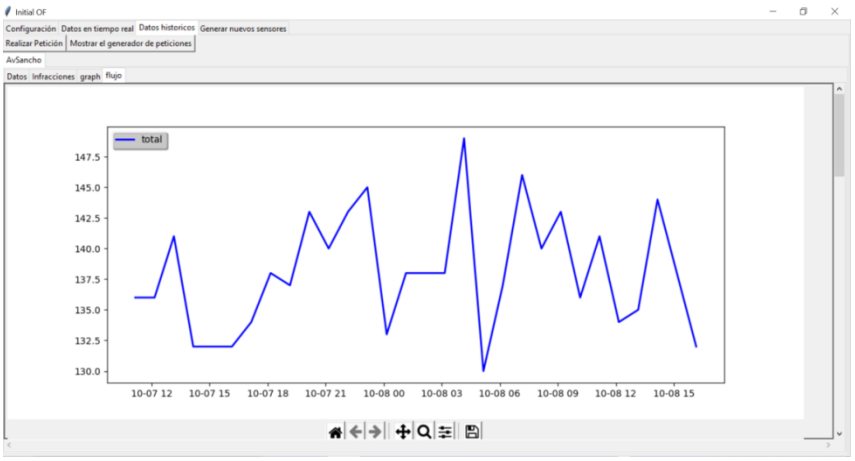


Figura 68: Datos históricos, pestaña de flujo de tráfico.

La clase *Graph* es un *Frame* que tiene la capacidad de hacer *scroll* de manera que si se usa de contenedor de objetos que se salen de los límites de la pantalla, el usuario es capaz de verlos. El motivo por el cual sólo se implementa para *Flux* y *Graphs* es que *MyTk* posee la característica de hacer *scroll* por defecto debido al uso de la clase *Table.*

Por cuestiones de eficiencia, cada vez que el usuario realiza una petición los objetos generados por la petición anterior se destruyen. De esta manera se libera espacio dentro de la RAM.

5.2.3.4 Nuevos Sensores

Clases de librerías externas:

ax	Obtenida de matplotlib. Contiene la mayoría de los elementos de *Figure* y fija el sistema de coordenadas. Actúa de intermediario entre *Figure* y el gráfico[41].
CheckBar	Clase *open source*. Genera una serie de *checkboxes* en una única fila y devuelve su estado mediante un *array*[52].
entry	Obtenida de tkinter. Genera un espacio donde el usuario puede escribir [31].
Figure, FigureCanvasTkAgg	Obtenida de matplotlib. *Figure* permite definir un espacio donde se va a colocar un gráfico, mientras que *FigureCanvasTkAgg* permite colocar Figure dentro del entorno definido por tkinter [33].
Frame	Obtenida de tkinter. Representa un espacio rectangular dentro de la ventana de la interfaz[31].
imread	Obtenida de matplotlib. Permite leer una imagen y colocarla en un gráfico[41].
json	Obtenida del módulo json. Permite leer y escribir objetos de

	formato json [43].
requests	Obtenida de *requests*. Permite comunicarse con http[38].
Slider	Obtenida de matplotlib. Se trata de una barra deslizante que representa un punto flotante entre un rango de valores[41].

Tabla 22: Clases de librerías externas empleadas en la creación de nuevos sensores.

Clases creadas para esta función:

Gen_Control	Contiene todo lo necesario para permitir al usuario definir el nuevo sensor. Se encarga de generar la petición, enviarla al API y procesar la respuesta
Gen_Map	Mapa interactivo que permite al usuario definir la localización del sensor en el mapa.

Tabla 23: Clases creadas para la generación de nuevos sensores.

El objetivo de esta clase es permitir al usuario escoger un punto en el mapa donde deseen colocar un nuevo sensor, así como las características de este nuevo sensor y enviar los datos al API para realizar las modificaciones adecuadas a la base de datos.

Para ello, empleamos una serie de elementos:

- Un objeto de la clase *Gen_Map* para crear un mapa interactivo en la que el usuario puede indicar la posición del sensor haciendo clic sobre él
- Dos *entries* donde el usuario puede especificar el nombre y el límite de velocidad
- Un *CheckBar* donde el usuario puede seleccionar los vehículos que no son admisibles en esa localización.
- Un botón para indicar que se han introducido todos los datos

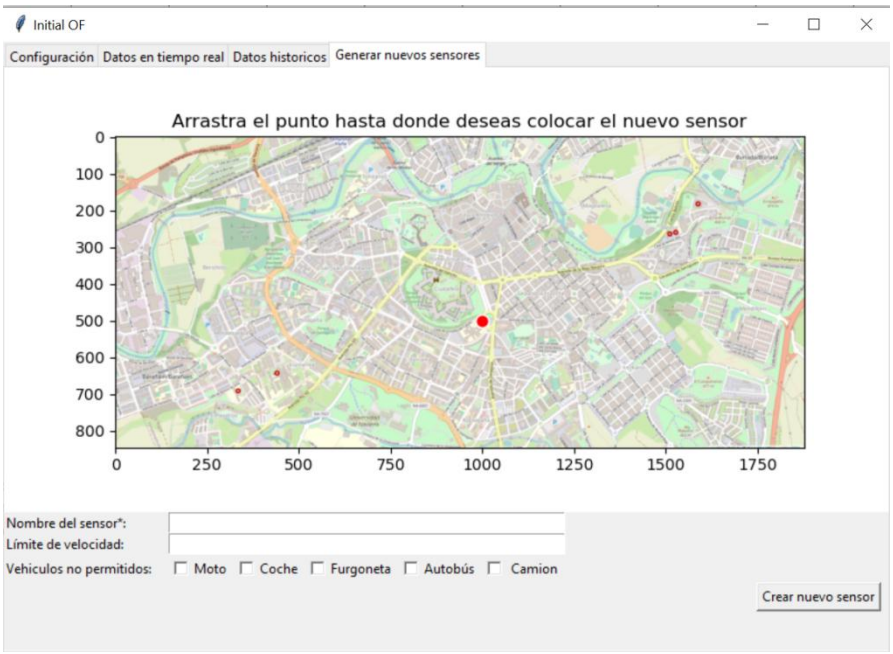


Figura 69: Pestaña de creación de nuevos sensores.

Todos estos elementos están contenidos en un objeto de la clase *Gen_Control* que, además, se encarga de generar la petición, enviarla al API y procesar su respuesta.

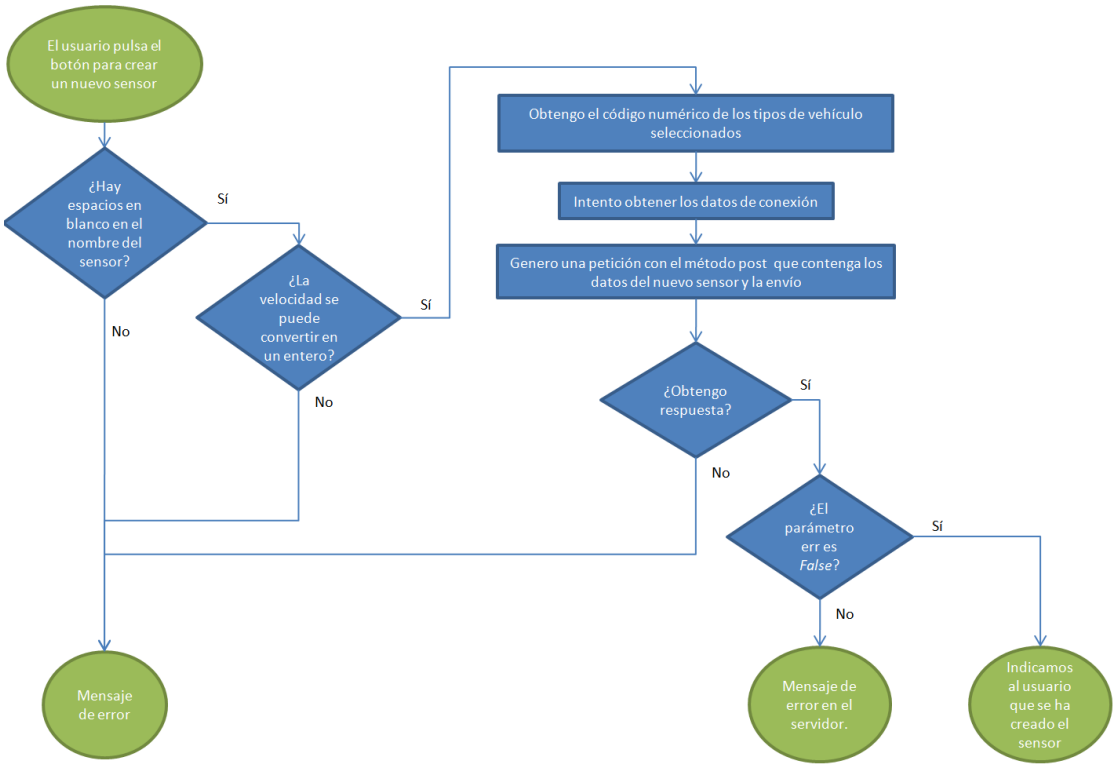


Figura 70: Diagrama de flujo de la creación de nuevos sensores.

Al igual que *MQTT_Map* y *Map* en las secciones anteriores, *Gen_Map* tampoco es un mapa interactivo sino un gráfico con un mapa de pamplona de fondo. En este caso, el gráfico tiene dos objetos de la clase slider ocultos que usamos para localizar la posición del clic y generar un punto en esa posición.

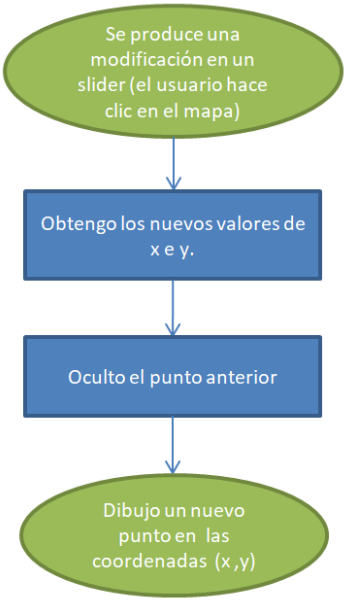


Figura 71: Diagrama de flujo del funcionamiento del mapa para crear nuevos sensores.

6 Conclusión, obstáculos y líneas futuras

6.1 Conclusión:

En este proyecto se ha demostrado la viabilidad de una interfaz de usuario para el acceso y análisis de datos obtenidos de un sensor basado en redes de difracción de Bragg para la monitorización del tráfico.

Se ha resaltado la importancia de la adaptabilidad en proyectos de ingeniería y la capacidad de amoldarse a situaciones inesperadas.

Otro aspecto que estos proyectos han sacado a relucir es la relevancia de las tecnologías de largo alcance y del trabajo a distancia.

6.2 Obstáculos encontrados durante la implementación del TFG

6.2.1 Emergencia Sanitaria

Para poder conectar el ordenador con el interrogador si155 es necesario conectarlo a la IP 10.0.0.121 con la máscara 255.255.255.0. [24]. Esto hace que el ordenador no pueda estar simultáneamente conectado al internet y al interrogador lo que imposibilita el uso de un escritorio remoto.

La orden de confinamiento de marzo de 2020 y la consecuente pérdida de acceso al laboratorio provocó el bloqueo del Proyecto 1. Esta supuso que el TFG quedara bloqueado hasta abril del mismo año cuando por fin se inició el desarrollo del proyecto 2.

6.2.2 Barreras de formación

El inicio del proyecto no fue rápido y directo ya que requiso el uso de tecnologías que no había usado hasta ahora:

- Acceso a servidores remotos
- Programación en *Python*
- Protocolo SSH
- Linux
- *Raspberry Pi*
- Protocolo MQTT
- Empleo de proxy para acceder a los contenidos on-line de la biblioteca

6.2.3 Hilos

Durante la implementación del sistema de escucha y visualización de mensajes enviados mediante el protocolo MQTT se encontraron varios problemas con respecto al uso de varios hilos.

Cosas que se intentaron	Problema	Solución
Usar el hilo que maneja la escucha de MQTT para ordenar a *MQTT_Display* que se actualice llamando a su función *update*.	Si dos hilos intentan modificar el mismo objeto a la vez el programa se bloquea.	Implementar una cola de mensajes, de forma que la actualización la implemente el hilo principal cuando este reciba un mensaje. Ver Figura 56.
Ordenar a un hilo que se pare directamente.	Al contrario que en java, en *Python* los hilos no tienen una función de parada.	Implementar una clase *open Source*, que hace que el hilo se bloquee lanzando una excepción. Ver Figura 54.
Una vez parado el hilo volver a reanimarlo.	En *Python* no se pueden reanimar hilos.	Crear un hilo nuevo cada vez que se inicie la escucha de MQTT. Ver Figura 53.

Tabla 24: Obstáculos encontrados en la implementación de hilos y su solución.

6.2.4 Mapas interactivos

Python posee librerías capaces de implementar mapas interactivos (por ejemplo, folium). El plan original era emplear una de estas librerías para los mapas empleados en las pestañas de 'datos históricos', 'datos en tiempo real' y 'Crear nuevos sensores'. Sin embargo, esto no fue posible ya los mapas generados por estas librerías son archivos HTML y tkinter no soporta este tipo de archivos.

Con el fin de solventar este obstáculo, se decidió emplear gráficos generados con la librería matplotlib y procesar las interacciones del usuario a través de eventos. Matplotlib fue la librería seleccionada ya que, es compatible con tkinter y era la librería programada para usar en los gráficos tradicionales del sistema.

Cada gráfico empleado como mapa posee un sistema de procesado de eventos distinto:

6.2.4.1 Mapa selector de sensores (Map)

Evento: Clic sobre uno de los sensores.

Se detecta sobre qué sensor se ha producido el clic y se modifica su estado (variable interna) y su color (para indicar al usuario que su orden ha sido detectada). Más detalles en Figura 61.

6.2.4.2 Mapa para mostrar el tráfico detectado (MQTT_Map)

Evento: Se recibe un mensaje.

La clase *MQTT_Parser* se encarga no sólo de poner el mensaje en el estado adecuado para añadirse a la tabla de *MyTk* si no que también analiza los datos recibidos y determina si se ha producido una infracción de velocidad y peso. De ser así se comprueba si las infracciones de velocidad entran dentro del margen de error del 15%. Una vez echas estas comprobaciones se genera un objeto diccionario donde se indican las conclusiones de dicho análisis. Dicho objeto se entrega junto con el mensaje procesado dentro de un segundo diccionario que es el que va a parar a la cola de mensajes (ver Figura 55). El diccionario generado por *MQTT_Parser* es lo que es entregado al objeto *MQTT_Map* para que este se actualice.

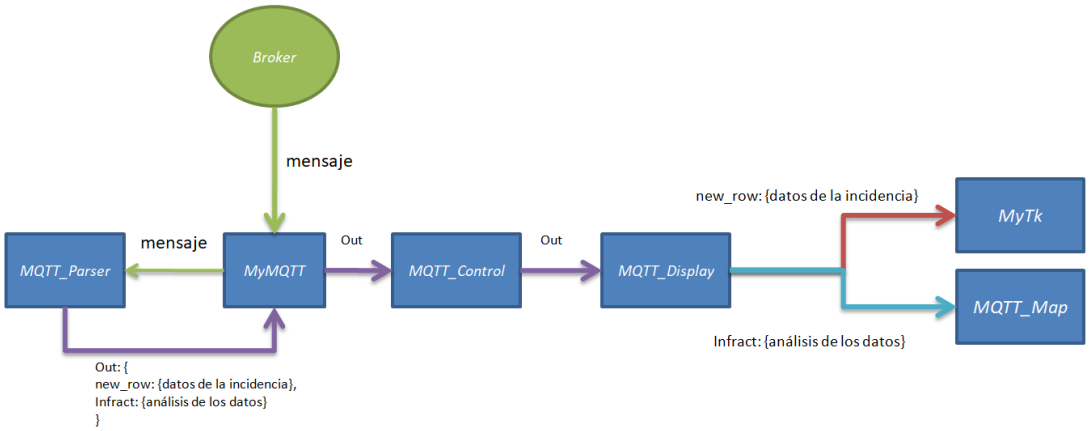


Figura 72: Diagrama de la transmisión de mensajes de datos en tiempo real.

6.2.4.3 *Mapa para la creación de nuevos sensores (Gen_Map)*

Evento: Clic sobre el mapa en el punto donde se desea colocar un sensor.

La clase *Gen_Map* tiene dos *Sliders* (clase propia de Matplotlib que permite escoger un valor dentro de un rango) ocultos que cubren respectivamente el eje x e y. Al hacer un clic en realidad lo que se hace es modificar la posición de los sliders lo que genera un evento.

Figura 73: Mapa para la creación de nuevos sensores con los *Sliders* visibles.

Al producirse un evento se toma la posición de los dos sliders (que corresponde con la posición x e y de la localización del clic), borramos el punto que representa la localización anterior del punto y creamos un nuevo punto en la nueva localización

La imagen del mapa de Pamplona empleado como fondo en los tres gráficos fue obtenido durante las pruebas con la librería 'folium'. La decisión de emplear esta imagen fue debido a la ausencia de marcadores y anotaciones que podían dificultar la fácil detección visual de los sensores.

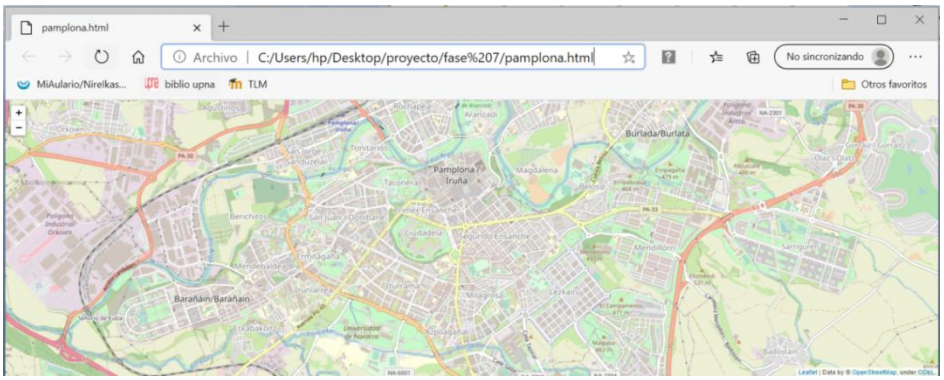


Figura 74: archivo HTML del mapa de pamplona obtenido con folium.

6.2.5 *Glitches* visuales

Durante la fase de pruebas se localizaron varios problemas visuales:

Al restablecer la configuración por defecto, se producía un efecto parpadeo sobre la ventana que resultaba muy molesto. Esto se debía a que entre la destrucción de la representación de la configuración antigua y la colocación en pantalla de la configuración por defecto se produce un breve intervalo de tiempo en el que la ventana está vacía. Para solucionar esto, se decidió modificar el proceso de actualización de la ventana colocando la representación de la nueva configuración sobre la antigua y posteriormente destruyendo la antigua. Ver Figura 47.

Al crear las pestañas de datos históricos, datos de tiempo real y creación de nuevos sensores después de pulsar el botón de 'guardar cambios' dentro de configuración, se producía un efecto de parpadeo similar al descrito en el párrafo anterior. Esto estaba causado por la diferencia de tamaño mínimo entre las distintas pestañas: configuración requiere un espacio menor para representar todos sus elementos que las pestañas de datos históricos y datos en tiempo real.

Para evitar esto la ventana tiene establecido un tamaño mínimo equivalente al espacio requerido por la pestaña de mayor tamaño. Este tamaño mínimo es también el tamaño por defecto de la ventana.

6.2.6 Problemas de eficiencia

Durante las fases de pruebas se localizaron varios problemas de eficiencia:

Originalmente todas las pruebas que realizan búsquedas de infracciones, así como aquellas que necesitan conocer la localización de los sensores debían pedir los datos de límites y localización de forma independiente unas de otras. Esto provocaba serios problemas de eficiencia ya que, simplemente iniciar el programa y pedir los datos correspondientes a un único sensor exigía realizar tres peticiones para obtener la localización de sensores (las cuales se almacenaban en tres variables distintas) más la petición de incidencias detectadas por el sensor.

Para evitar esto, nada más iniciar el programa *Central_Hub* se encarga de solicitar la tabla de límites y localizaciones de los sensores y esa única variable se entrega por inyección al resto de objetos que la necesitas. Adicionalmente esto permite realizar una comprobación de la

conexión antes de crear aquellas clases que requieran de una conexión para funcionar. Ver Figura 44.

Originalmente en al realizar varias peticiones de datos históricos los objetos *Display* generados por cada una de ellas no se destruían, se ocultaban sus pestañas, pero su carga computacional seguía en la RAM. Por cuestiones de eficiencia se decidió destruir los objetos *Display* almacenados cada vez que se solicitan nuevos datos históricos.

6.2.7 Fallos del servidor

Como hemos descrito anteriormente el sistema opera empleando *una raspberry pi* simultáneamente como *broker* y servidor. Esta ha fallado varias veces durante el desarrollo del proyecto. Los fallos han tenido dos causas principales:

- Refrigeración inadecuada: En los días más calurosos del verano, especialmente en julio y agosto, se han producido varios incidentes en los que la *raspberry* ha dejado de responder. Es probable que esto se debiera a las altas temperaturas.
- Tarjeta SD: La *raspberry* emplea como memoria una tarjeta SD. Esto supone el almacenaje y operación del sistema operativo dentro de la misma, algo para lo que no está preparada. Se han producido varios incidentes en los que la tarjeta SD, se ha corrompido o bloqueado haciendo fallar a todo el sistema.

Figura 75: Raspberry Pi empleada como servidor.

6.3 Líneas Futuras

Si se completara el desarrollo del proyecto 1, este software podría ser de especial interés en aplicaciones de laboratorio.

El procesado de datos realizado en el proyecto 2 es muy sencillo y si bien cumple los objetivos planteados, este sistema podría beneficiarse de la aplicación de tecnologías más avanzadas. El uso de *machine learning* y otras técnicas de inteligencia artificial para el análisis de datos podrían ser de gran utilidad. La implementación de estas tecnologías podría ser especialmente beneficiosa si se desea aplicar este tipo de sensores a sistemas de *big data*.

7 Bibliografía

1. A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, & M. Ayyash, Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, **17** (2015) 2347–2376. https://doi.org/10.1109/COMST.2015.2444095.

2. The Fiber Optic Association, FOA Reference Guide To Fiber Optics. (n.d.). https://www.thefoa.org/ESP/Fibra_optica.htm (accessed October 10, 2020).

3. A. Ghatak & K. Thyagarajan, *An Introduction to Fiber Optics* (Cambridge University Press, 1998).

4. J. M. Senior, Optical fiber waveguides. *Opt. Fiber Commun. Princ. Pract.* (Prentice Hall, 2009), pp. 12–82.

5. J. M. Senior, Advantages of optical fiber communication. *Opt. Fiber Commun. Princ. Pract.* (Prentice Hall, 2009), pp. 7–10.

6. U. Sharma & X. Wei, Fiber Optic Interferometric Devices. *Fiber Opt. Sens. Imaging* (Springer, 2013), pp. 29–55.

7. M. Bravo Acha, Contribution to the development of new photonic systems for fiber optic sensing applications, 2015.

8. H. E. Joe, H. Yun, S. H. Jo, M. B. G. Jun, & B. K. Min, A review on optical fiber sensors for environmental monitoring. *International Journal of Precision Engineering and Manufacturing - Green Technology*, **5** (2018) 173–191. https://doi.org/10.1007/s40684-018-0017-6.

9. M. Bravo, D. Leandro, A. Bravo-acha, M. Bravo-navas, J. R. Mitxelena, J. J. Martinez-mazo, E. Camarero, & M. López-amo, Sensores de fibra óptica en asfalto para la monitorización de tráfico rodado. *11º Renunión Española de Optoelectrónica*, (2019) 1–5.

10. Y.-G. Han, Optical Fiber Gratings for Mechanical and Bio-sensing. *Fiber Opt. Sens. Imaging* (Springer, 2013), pp. 91–119.

11. G. Patil, A seminar on fiber bragg grating(FBG). (2017). https://www.slideshare.net/Poornimagugale/fbg-ppt.

12. A. D. Kersey, M. A. Davis, H. J. Patrick, M. LeBlanc, K. P. Koo, C. G. Askins, M. A. Putnam, & E. J. Friebele, Fiber grating sensors. *Journal of Lightwave Technology*, **15** (1997) 1442–1462. https://doi.org/10.1109/50.618377.

13. M. Bravo, D. Leandro, A. Bravo-acha, M. Bravo-navas, J. Ramon, J. J. Martinez-mazo, E. Camarero, & M. López-amo, Optical Fiber Sensors in Asphalt for Smart Cities Traffic Monitoring. (n.d.) 5–8.

14. R. Denenberg, Open Systems Interconnection. *Library Hi Tech*, **3** (1985) 15–26. https://doi.org/10.1108/eb047578.

15. Cisco Systems Inc, ¿Qué es el modelo OSI? (2002). https://sites.google.com/site/redesbasico150/protocolos-de-red/-que-es-el-modelo-

osi.

16. S. Potts & M. Kopack, Understanding How Web Services Communicate. *SAMS Teach Yours. webservices 24 hours* (SAMS, 2003), pp. 109–125.

17. C. Moyano, David, Modelo TCP IP. (2020). https://www.docsity.com/es/escrito-sobre-modelo-tcp-ip-ieee/5544891/.

18. I. J. Taylor & A. Harrison, Web Services. *From P2P Grids to Serv. Web*, 2nd ed, (Springer, 2008), pp. 127–138.

19. P. Adamczyk, P. H. Smith, R. E. Johnson, & M. Hafiz, REST and Web Services: In Theory and in Practice. *REST From Res. to Pract.* (Springer, 2011), pp. 35–61.

20. R. Elmasri & S. B. Navathe, Bases de datos y usuarios de bases de datos. *Fundam. Sist. bases datos*, 5th ed, (Pearson-Addison Wesley, 2007), pp. 3–25.

21. R. Elmasri & S. B. Navathe, El modelo de datos relacional y las restricciones de una base de datos racional. *Fundam. Sist. bases datos*, 5th ed, (2007), pp. 123–144.

22. R. Elmasri & S. B. Navathe, SQL-99: definición del esquema, restricciones, consultas y vistas. *Fundam. Sist. bases datos*, 5th ed, (Pearson-Addison Wesley, 2007), pp. 205–244.

23. K. Haritos-Shea & C. McCathieNevile, WAI Glossary. *W3C*, (2002). https://www.w3.org/WAI/GL/Glossary/.

24. Micron Optics Inc., *Optical Sensing Instrumentation and Software. User Guide* (2017).

25. The MathWorks Inc, tcpip documentation. (n.d.).

26. B. Kjell, Big Endian and Little Endian. (n.d.). https://chortle.ccsu.edu/AssemblyTutorial/Chapter-15/ass15_3.html.

27. P. Fernandez, “Through the looking glass: envisioning new library technologies” how inexpensive computers can transform information access and literacy. *Library Hi Tech News*, **32** (2015) 4–7. https://doi.org/10.1108/LHTN-07-2015-0046.

28. Python Software Fundation, General Python FAQ. (n.d.). https://docs.python.org/3/faq/general.html#what-is-python.

29. M. Eric, *Python Crash Course, 2nd Edition*, 2nd ed (No Starch Press, 2019).

30. Paho-mqtt documentation. (n.d.). https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php.

31. Python Software Fundation, tkinter — Python interface to Tcl/Tk. (n.d.). https://docs.python.org/3/library/tkinter.html.

32. D. Farrell, DataExplore: An Application for General Data Analysis in Research and Education. (2016). https://pandastable.readthedocs.io/.

33. J. Hunter, D. Dale, E. Firing, M. Droettboom, & the Matplotlib development Team, Embedding Matplotlib in Tk. (n.d.). https://matplotlib.org/gallery/user_interfaces/embedding_in_tk_sgskip.html?highlight=tkinter.

34. W. McKinney, *Python For Data Analysis* (O'Reilly, 2012).

35. The pandas development team, Pandas documentation. (2020). https://pandas.pydata.org/pandas-docs/stable/index.html.

36. Python Software Fundation, queue — A synchronized queue class. (n.d.). https://docs.python.org/3/library/queue.html.

37. Pallets, Flask documentation. (n.d.). https://flask.palletsprojects.com/en/1.1.x/.

38. K. Reitz, Requests documentation. (n.d.). https://es.python-requests.org/es/latest/index.html.

39. Oracle Corporation, MySQL Connector/Python Developer Guide. (2020). https://dev.mysql.com/doc/connector-python/en/.

40. Python Software Fundation, PEP 249 -- Python Database API Specification v2.0. (2017). https://www.python.org/dev/peps/pep-0249/.

41. J. Hunter, D. Dale, E. Firing, M. Droettboom, & the M. development Team, Matplotlib: Visualization with Python. (n.d.). https://matplotlib.org/.

42. J. Monsel, tkcalendar. (2018). https://tkcalendar.readthedocs.io/en/stable/.

43. Python Software Foundation, json — JSON encoder and decoder. *Python 3.8.5 documentation*, (2019). https://docs.python.org/3/library/json.html.

44. MQTT.org. (n.d.). https://mqtt.org/.

45. The HiveMQ Team, HIVEMQ: MQTT ESSENTIALS. (2019). https://www.hivemq.com/blog/mqtt-essentials-part-3-client-broker-connection-establishment/.

46. D. PARDO, ¿Para qué sirve una API? Solventa por fin esta duda de primerizo. (n.d.). https://web.archive.org/web/20190215165536/https://blog.pandorafms.org/es/para-que-sirve-una-api/.

47. G. Motroc, The State of API Integration: SOAP vs. REST, public APIs and more. (2017). https://jaxenter.com/state-of-api-integration-report-136342.html.

48. IBM, Structure of a SOAP message. (2020). https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.4.0/fundamentals/web-services/dfhws_message.html.

49. R. Blackbourn, Enhancing JSON Serialization In Python With Typing. (n.d.). https://medium.com/@rob.blackbourn/enhancing-json-serialization-in-python-with-typing-a481017934c5%0A.

50. Manthanchauhan, Python | Different ways to kill a Thread. (n.d.). https://www.geeksforgeeks.org/python-different-ways-to-kill-a-thread/.

51. POLICÍA MUNICIPAL UDALTZAINGOA, *Informe de tráfico del tráfico entre las calles Señorío de Amocáin y Señorío de Egulbati* (Pamplona).

52. K. B. Bernd, Checkboxes. (n.d.). https://www.python-course.eu/tkinter_checkboxes.php.

8 ANEXO I: Códigos del Proyecto 1

8.1 Conectar

```
function [C]=conn(RHOST,RPORT)
    %Bsize=128108;
    %RPORT=51971;%puerto por el que se conecta

    C=tcpip(RHOST,RPORT,'NetworkRole','client','InputBufferSize',32768)
    %C.ByteOrder='littleEndian'
    fopen(C);

end
```

8.2 Petición

```
function [ask]=peticion(instrumento,comando,argumento,opcion)
%necesito: %(con)el instrumeneto a donde la quiero enviar -el tcpip de conexion-
           %cabecera :%request option -u8- nos indica como se devuelve la
                                        %respuesta
                                        %   0 - None
                                        %   1 - Suppress Message
                                        %   2 - Suppress Content
                                        %   3 - Compress
                      %byte reservado -NA-
                      %comand length -u16-
                      %argument length -u32-
            %el mensaje:%comand -ascii-
                        %argument -ascii-
%tanto comand length como argument length deben estar en little endian
  %% cabecera
  %request option
  header=uint8(opcion);
  %Reservado
  %comand length
  commandLength = uint16(length(comando));
  header(3:4) = LilEndian(commandLength,2)
  %argument length
  argumentLength=uint32(length(argumento));
  header(5:8)=LilEndian(argumentLength,4);
  %% mensaje
  com=uint8(comando);
  arg=uint8(argumento);
  %% creamos la peticion
  ask=[header,com,arg];
  %% %realizamos la peticion
  fwrite(instrumento,ask);

end
```

8.3 Formato *Little Endian*

```
function [byteseq]=LilEndian(num,bytecount)
%convierte el num a formato Little Endian
%num:numero a combertir
%bytecount:numero de bytes que tiene la clase de num
    bytes=bytecount:-1:1;
    index=1;
    for i=num
        for j=bytes
            shift=(bytecount-j)*8; %numero de bits que tenemos que desplazar
            swap=bitshift(i,-shift); %shift negativo porque queremos desplazar a la der
            byte=bitand(swap,255);%convertimos los huecos del desplazamiento en ceros
            byteseq(index)=uint8(byte);
            index=index+1;
        end
    end

end
```

8.4 Respuesta

```
function Answr=Response(C, bytes)
    %funcion que lee las respuestas enviadas por el objeto TCPIP C
    %bytes-> numero de bytes que deseamos leer
    bufferAvailable=0;%numero de bytes que caben en el buffer
    bytesread=0; %numero de bytes que hemos leído
    Answr = uint8([]); %almacenamos la respuesta en formato uint8
    %
   while(bytesread<bytes)
       %primero calculamos el numero de bytes que tenemos que leer
       bytes2read=bytes-bytesread
       %sacamos el tamaño del buffer disponible
       bufferAvailable=C.BytesAvailable
       %si el numero de bytes que tenemos que leer no
       %caben en el buffer, leemos lo que nos permita el buffer
       if (bytes2read>bufferAvailable)
           bytes2read=bufferAvailable;
       end
       %leemos los bytes que podemos
       if (bytes2read>0)
            data=fread(C,double(bytes2read));
            %sumamos al contador el numero de bytes que acabamos de leer
            bytesread=bytesread+bytes2read;
            %añadimos los bytes que acabamos de leer a la respuesta
            Answr=[Answr;data];
       end
   end

  %%
%    while(totalBytesRead < numBytes)


%     bytesToRead=numBytes-totalBytesRead;
%     numBytesAvailable = instrument.BytesAvailable;

% if(bytesToRead > numBytesAvailable)
%     bytesToRead = numBytesAvailable;
% end
%     if bytesToRead > 0
%         newData = fread(instrument, double(bytesToRead));
%         totalBytesRead = totalBytesRead + bytesToRead;
%         dataOut = [dataOut; newData];
%     end
end
```

8.5 Desconectar

```
function disc(C)
    global C;
    fclose (C); %poner el estado de la conexion como desconectado
    delete (C); %eliminar el objeto de interfaz
    clear global C; %eliminar la variable global

end
```

9 ANEXO II: Códigos del Proyecto 2

9.1 API

```
import json
from flask import Flask,request
import mysql.connector
from mysql.connector import Error
from datetime import datetime
from decimal import Decimal
import json
#REF:https://medium.com/@rob.blackbourn/enhancing-json-serialization-in-python-with-typing-a481017934c5
class MyEncoder(json.JSONEncoder):
#codificador de json para datetime y decimal
    def default(self, obj):
        if isinstance(obj, datetime):
            return obj.isoformat()
        if isinstance(obj, Decimal):
            return float(obj)


def get_data(fetch, query):
    #esta funcion procesa la query y devuelve un json con lo contenido en la tabla
    try:
        #intenta abrir una conexion
        connection = mysql.connector.connect(host='localhost',
                                             database='Pamplona',
                                             user='Sensores',
                                             password='upna2017')
        #si se puede abrir la conexion
        if connection.is_connected():
            print("Conectado")
            print("voy a ejecutar "+query)
            #ejecutamos la query
            cursor=connection.cursor()
            cursor.execute(query)
            #recojo los datos
            if fetch == True:
                data = cursor.fetchall()
                for row in data:
                    print(row)
            else :
                connection.commit()
                data = 'Sensor creado.'
            #rawData=cursor.fetchall()
            #combierto los datetimes y decimal en formatos adecuados
            #data=json.dumps(rawData, cls=MyEncoder, indent=4)
            err=False
    except Error as e:
        err = True
        #si no se puede abrir la conexión
        #imprimo un mensaje de error y devuelvo data=error
        print("Error while connecting to MySQL", e)
        data = "Error while connecting to MySQL" + str(e)
        #print(data)

    finally:
        #tanto si puedo abrir la conexión como si no
        #la cerramos y devuelvo el valor de data
        if (connection.is_connected()):
            cursor.close()
            connection.close()
            print("MySQL connection is closed")

            out={'err':err,
                'data':data}
            #print(data)
```

```
64              j_out = json.dumps(out, cls = MyEncoder, indent = 4)
65              return j_out
66
67
68  ###def de flask
69  #ruta inicial
70  app = Flask(__name__)
71  @app.route('/',methods=['GET'])
72  def home():
73      #esta funcion recoge los parametros y genera la query apartir de ellos
74      #query='SELECT * FROM '+id+' WHERE time >='+TS1+' AND time<='+TS2
75      TS1 = request.args.get('TS1')
76      TS2 = request.args.get('TS2')
77      ident = request.args.get('id')
78      if TS1 == None or ident==None:
79          out = {'err' : True ,'data' : 'missing values'}
80          j_out = json.dumps(out)
81      else:
82          if TS2==None:
83              end=''
84          else:
85              end=" AND tme <='"+TS2+"'"
86          query="SELECT * FROM "+ident+" WHERE tme >='"+TS1+"'"+end
87          print(query)
88          j_out=get_data(True, query)
89      #print(query)
90      return j_out
91  ###
92  @app.route('/limites',methods=['GET'])
93  def get_config():
94      #cuando config recibe una funcion GET devuelve un json que contiene la base de ↵
        datos de
95      #limitaciones, podemos especficar una calle en concreto o pedir todas
96      calle = request.args.get('calle')
97      if calle == None:
98          query = "SELECT * FROM Limite"
99      else:
100         query = "SELECT * FROM Limite WHERE calle = '"+calle+"'"
101     print(query)
102     j_out = get_data(True, query)
103     return j_out
104
105
106 @app.route('/new',methods=['POST'])
107 def update():
108     print('estoy en update')
109     j_out= None
110     print(request.form)
111     #creamos una nueva base de datos para el nuevo sensor y una nueva fila en limites
112     #comprovamos que todos los  parametros sean correctos
113     print('voy a sacar los datos')
114     d_rec = request.form.to_dict()
115     print(d_rec)
116     calle = d_rec['sensor']
117     vel = d_rec['velocidad']
118     vehiculos = d_rec['vehiculos']
119     x = d_rec['x']
120     y = d_rec['y']
121     print('tengo los datos')
122     if calle == None or vel == None or vehiculos == None or x == None or y == None:
123         #es obligatorio especificar todos los parametros
124         out = ({'err':True,'data':'missing variables'})
125         j_out = json.dumps(out)
126     else:
```

```
127         try:
128
129             #comprovamos que todos los parametros sean del tipo adecuado
130             if ' ' in calle:
131                 print('espacio en blanco en calle')
132                 #no puede haber espacios en blanco en el nombre
133                 raise Exception
134             vel = Decimal( vel ) #velocidad debe ser convertible en decimal
135             print('vel dec')
136             x = int (x)#x e y deben ser convertibles a enteros
137             y = int (y)
138             print(x)
139             print(x)
140             #vehiculos debe ser una lista que o bien esté vacía o sólo contenga
                numeros
141             #creamos la tabla
142             query = 'CREATE TABLE '+ calle+' ( id INT(20) PRIMARY KEY
                AUTO_INCREMENT NOT NULL, tme DATETIME(6) NOT NULL, vel DECIMAL(7,4),
                acc DECIMAL(7,4), tipo INT(11))'
143             j_crear = get_data(False, query)
144             crear = json.loads(j_crear)
145             print(crear)
146             if crear ['err'] == False:
147                 print(vehiculos)
148                 print(vehiculos == 'NA')
149                 if vehiculos == 'NA':
150                     print('estoy en NA')
151                     query=("INSERT INTO Limite (calle, vel, x, y) VALUES ('"+
                        calle +"', "+str(vel)+", "+str(x)+", "+str(y)+")")
152
153                 else:
154                 #actualizamos la tabla de limites
155                     query=("INSERT INTO Limite (calle, vel, vehiculos, x, y)
                        VALUES ('"+ calle +"', "+str(vel)+", '"+ str (vehiculos)+"',
                        "+str(x)+", "+str(y)+")")
156                 j_out = get_data(False,query)
157                 #out = json.load (j_out)
158             else:
159                 j_out = j_crear
160         except Exception as e:
161             print (e)
162             err = True
163             data = 'Incompatible values'
164             out = {'err' : err, 'data' : data}
165             j_out = json.dumps( out )
166
167     #j_out = json.dumps( out )
168     return j_out
169
170
171         #   b = 'vehiculos='+vehiculos
172         #query="UPDATE limites SET "+a+b+" WHERE calle="+calle
173
174 host = '0.0.0.0'
175
176 app.run(host=host, port=23040)
177
```

9.2 *Central_Hub*

```
import tkinter as tk
from tkinter import ttk

from config import Config
from mqtt_control import MQTT_Control
from control2 import Control
from gen_control import Gen_Control
import json
import requests

class Central_Hub(tk.Frame):
    def __init__(self, master, filename):
        super().__init__(master)
        #creo los atributos basicos
        self.tab={'mqtt': None, 'hist_data': None, 'config': None, 'new' : None }
        self.filename = filename

        #nos colocamos en master
        self.grid (row = 0, column = 0, sticky ='NSWE')

        #creamos el notebook
        self.nb = ttk.Notebook(self)
        self.nb.grid(row = 1, column = 0, sticky ='NSWE')
        self.rowconfigure(1, weight=  1)
        self.columnconfigure(0,weight = 1)

        #creamos el objeto config, de este modo nos aseguramos que settings exista
        #y tenga los indices correctos

        self.config = Config (self.nb, filename = filename)
        self.tab['config'] = self.config
        self.nb.add (self.config, text = 'Configuración')
        print('hola')
        self.try_conn()



    def get_settings (self) :
        #obtenemos los datos de conxexion de filename
        try:
            #intentamos abrir el archivo de configuraciones
            with open (self.filename) as j_settings:
                settings = json.load(j_settings)
                url = settings['data']['url_data']
                port = settings['data']['port_data']
                lim = settings['data']['lim']
                self.url = url + ':' + str(port) + lim
                print(self.url)
            return False

        except Exception as e:
            #si no se puede usamos la cola para pedir a mqtt_control que saque un mensaje de error
            message = 'Se ha producido un Error de configuración. Revise la configuración.'
            tk.messagebox.showerror(title='Error', message = message)
            print(e)
            print('error en get_setting')
            return True

    def get_data (self):
        #obtengo la lista de limites y las coord del mapa
        try:
```

```
63              #solicito los datos
64              print(self.url)
65              r = requests.get(self.url)
66              response = r.json()
67              print('esto es response')
68              print(response)
69
70              #los pongo en el formato adecuado
71              self.lim_list = dict()
72              self.map_coord = dict ()
73              for data in response['data']:
74                  vehiculos = []
75                  for veh in data[3].split(','):
76                      #convertimos la lista de vehiculos de str a list
77                      vehiculos.append(int(veh))
78                  self.lim_list.update( { data[1] : dict(
                    zip(['velocidad','vehiculos'] , [data[2] ,vehiculos]) )} )
79                  self.map_coord.update ({data[1]: dict (zip (['x', 'y', 'state'],
                    [[data[4]], [data[5]], 0]) ) })
80              return False
81
82          except Exception as e:#si se produce un error al conectarme con url
83              #si no se puede usamos la cola para pedir a mqtt_control que saque un
                mensaje de error
84              message = 'Se ha producido un error al solicitar al servidor la
                localización de los sensores. Revise la configuración y su conexión a
                internet.'
85              tk.messagebox.showerror(title='Error', message = message)
86              print(e)
87              return True
88
89
90      def try_conn (self) :
91          #si existen pestañas antiguas las eliminamos
92          for tab in self.tab.keys():
93              if self.tab[tab] != None and tab !='config':
94                  self.nb.forget (self.tab[tab])
95                  self.tab[tab] = None
96
97          #intentamos obtener los datos de conexión
98          err = self.get_settings()
99          print(err)
100         if err == False:
101             #si los conseguimos intentamos sacar los limites y las coordenadas del
                mapa
102             err2 = self.get_data()
103
104             if err2 == False:
105                 #una vez tengamos la conexión creamos el resto de pestañas
106                 #mqtt
107                 print('listo')
108                 self.tab['mqtt'] = MQTT_Control(master = self.nb, filename =
                    self.filename, map_coord = self.map_coord,  lim_list =
                    self.lim_list)
109                 self.nb.add (self.tab['mqtt'], text = 'Datos en tiempo real')
110                 #datos historicos
111                 self.tab['hist_data'] = Control(self.filename, self.lim_list ,
                    self.map_coord, master = self.nb)
112                 self.nb.add(self.tab['hist_data'], text ='Datos historicos')
113                 #generar nuevos sensores
114                 self.tab['new']= Gen_Control(master = self.nb, filename =
                    self.filename)
115                 self.nb.add(self.tab['new'], text='Generar nuevos sensores')
116
```

```
117 if __name__ == "__main__":
118
119     def quit_me():
120         #print('quit')
121         root.quit()
122         root.destroy()
123     root = tk.Tk()
124     #(X,Y)
125     root.minsize (800,550)  #tamaño minimo
126     root.geometry('800x550') #tamaño inicial
127     root.title ('Initial OF')
128     root.protocol("WM_DELETE_WINDOW", quit_me)
129     root.columnconfigure(0, weight = 1)
130     root.rowconfigure(0, weight = 1)
131
132     p = Central_Hub(root, 'settings.json')
133     root.mainloop()
134
```

9.3 Configuración

```
import json
import tkinter as tk
from tkinter import ttk
import tkinter.messagebox


class Config(tk.Frame):
    #frame que muestra la configuracion y permite modificarla
    def __init__(self, master, filename):
        super().__init__(master)
        self.grid(row = 0, column = 0, sticky = 'NSWE')
        self.rowconfigure (0, weight =1)
        self.columnconfigure(0,weight= 1)

        #datos basicos: nombre del archivo e indices que son necesarios
        self.filename = filename
        self.mqtt = {'addr':'Dirección:','port_mqtt':'Puerto:','user':'Usuario:','password':'Contraseña:','topic':'Tema:'}
        self.data = {'url_data':'Dirección:','port_data':'Puerto:','lim':'Subdirectorio:'}
        self.new = {'url_new':'Dirección:','port_new':'Puerto:','sub':'Subdirectorio:'}
        self.full ={'mqtt' : self.mqtt, 'data': self.data, 'new': self.new}
        self.container = None
        #leemos el archivo, si no se puede, los indices son incorrectos o los puertos no son enteros
        # reescribimos el archivo con los parametros por defecto
        try:
            #intenta leer el archivo
            with open(self.filename) as f:
                self.settings = json.load(f)


            #comprobar que tenga los indices correctos
            for key in self.mqtt.keys():
                #usar in es mas rapido ref:https://stackoverflow.com/questions/1602934/check-if-a-given-key-already-exists-in-a-dictionary
                if key not in self.settings['mqtt']:
                    raise Exception("WrongKeys")
            for key in self.data.keys():
                if key not in self.settings['data']:
                    raise Exception("WrongKeys")

            for key in self.new.keys():
                if key not in self.settings['new']:
                    raise Exception("WrongKeys")

            #comprobamos que los puertos sean numeros enteror
            if isinstance(self.settings['mqtt']['port_mqtt'],int) == False or isinstance(self.settings['data']['port_data'],int) == False or isinstance(self.settings['new']['port_new'],int) == False:
                raise Exception("PortIsNotInt")

        except Exception as e:
            #si se produce algún problema
            #reescribimos el archivo
            print(e)
            self.reset_file()


        #nos colocamos dentro de master y mostramos los datos de configuracion
```

```
56          self.row = 0
57          self.grid(column = 0, row = 0, sticky = 'NSWE')
58          self.fill_container()
59
60      def fill_container(self):
61          #rellenamos nuestro frame con un subframe llamado container en el cual
            introducimos
62          #los datos dentro de labels y  de entries
63
64          #creamos un container nuevo y procedemos a rellenarlo
65          new_container = tk.Frame(self)
66          self.entries = dict ()
67          self.row = 1
68
69          for section in ['mqtt', 'data', 'new']:
70              #si es la primera etiqueta creamos las secciones
71              #print(self.settings[section].items())
72              if section == 'mqtt':
73                  text = 'Configuración de la solicitud de datos en tiempo real:'
74              elif section == 'data':
75                  text = 'Configuración de la solicitud de datos historicos:'
76              elif section == 'new':
77                  text = 'Configuración para la creación de nuevos sensores:'
78              label = ttk.Label (new_container, text = text)
79              label.grid (column = 0, row = self.row)
80              self.row = self.row + 1
81              for key, value in self.settings[section].items():
82                  #creamos una etiqueta
83                  #print(self.full[section][key])
84                  label = ttk.Label (new_container,text = self.full[section][key])
85                  label.grid (column = 1, row = self.row, sticky ='WE')
86                  #creamos un entry con los valores
87                  entry = tk.Entry (new_container, width = 50)
88                  entry.insert(0, value)
89                  entry.grid (column = 2, row = self.row, sticky = 'WE')
90                  #almacenamos las entries en un diccionario usando key de indice
91                  #new_container.rowconfigure(self.row, weight= 1)
92                  self.entries.update ({key : entry})
93                  self.row = self.row + 1
94
95
96          #linea en blanco
97          self.row = self.row +1
98          label11 = ttk.Label(new_container,text = '')
99          label11.grid(column = 0, row = self.row, sticky='w')
100
101         #boton para guardar los cambios
102         self.row = self.row +1
103         self.button1 = tk.Button(new_container, text = 'Guardar los cambios',
            command = self.save_changes)
104         self.button1.grid(row = self.row, column = 1, sticky = 'e')
105
106         #boton para volver a los valores por defecto
107         button2 = tk.Button(new_container, text = 'Restablecer los valores por
            defecto', command = self.reset)
108         button2.grid(row = self.row, column = 2, sticky = 'w')
109         #new_container.columnconfigure(0,weight= 1)
110         #new_container.columnconfigure(1,weight= 1)
111         #colocamos el nuevo container, destruimos el anterior y almacenamos el nuevo
112         new_container.grid(row = 0, column = 0, sticky = 'NSWE')
113         if self.container != None:
114             self.container.destroy()
115         self.container = new_container
116
```

```
117     def reset(self):
118         #rellenamos el archivo con los datos por defecto
119         self.reset_file()
120         #modificamos el frame
121         #rellenamos container
122         self.fill_container()
123         #self.button1['state'] = 'disabled'
124 
125 
126 
127     def save_changes(self):
128         #comprovamos los datos escritos en las entries
129         #y si todo el correcto modificamos el archivo settings.json
130         try:
131             for key,entry in self.entries.items():
132                 #¿hay algo escrito en todos los entrys?
133                 if len(entry.get()) == 0:
134                     raise EmptyEntry
135                 #¿los puertos son enteros?
136                 if key == 'port_mqtt' or key == 'port_data' or key == 'port_new':
137                     int(entry.get())
138 
139             #si todo es correcto
140             #modificamos settings y el archivo
141             count = 1
142             print(self.entries.items())
143             for key,entry in self.entries.items():
144                 if count <= 5:
145                     value = entry.get()
146                     if key == 'port_mqtt':
147                         value = int(entry.get())
148                     self.settings["mqtt"][key] = value
149                 elif count <=8:
150                     value = entry.get()
151                     if key == 'port_data':
152                         value = int(entry.get())
153                     self.settings["data"][key] = value
154                 else:
155                     value = entry.get()
156                     if key == 'port_new':
157                         value = int(entry.get())
158                     self.settings["new"][key] = value
159 
160                 count = count + 1
161 
162             with open(self.filename,'w') as out:
163                 out.write(json.dumps(self.settings))
164             self.master.master.try_conn()
165 
166         except EmptyEntry:#hay alguna  entry vacia
167             tk.messagebox.showerror(title='Error', message='Faltan datos por rellenar.')
168 
169         except ValueError:#los puertos no son enteros
170             tk.messagebox.showerror(title='Error', message='Los puertos deben ser números enteros.')
171         with open(self.filename) as f:
172             self.settings = json.load(f)
173             print(self.settings)
174 
175     def reset_file(self):
176         #reescribimos el archivo los valores por defecto
177         default = self.get_default()
178         with open(self.filename,'w') as out:
```

```
179             out.write(default)
180
181         #lo leemos
182         with open(self.filename) as f:
183             self.settings = json.load(f)
184             print(self.settings)
185
186
187
188     def get_default(self):
189         #devuelve la configuracion por defecto
190         data='{"mqtt" : {"addr" : "keltxo.no-ip.org","port_mqtt" : 1883,"user" :
            "root","password" : "upna2017","topic" : "Pamplona/#"},"data" :
            {"url_data" : "http://keltxo.no-ip.org","port_data" : 23040,"lim" :
            "/limites"},"new" : {"url_new" : "http://keltxo.no-ip.org","port_new" :
            23040,"sub" : "/new"}}'
191         return data
192
193
194 class EmptyEntry(Exception):
195     #hay una entrada vacia
196     pass
197
198 if __name__ == "__main__":
199     def quit_me():
200         #print('quit')
201         root.quit()
202         root.destroy()
203     root=tk.Tk()
204     root.protocol("WM_DELETE_WINDOW", quit_me)
205     root.columnconfigure(0, weight=1)
206     root.rowconfigure(0, weight=1)
207
208     nb=ttk.Notebook(root)
209
210     p=Config(nb, 'settings.json')
211     nb.add(p,text='Configuracion')
212     nb.grid(row=0,column=0,sticky='NSWE')
213     root.mainloop()
214
```

9.4 Datos en tiempo real

9.4.1 *Thread_with_exception*

```
1  import threading
2  import ctypes
3  import time
4
5  #ref:https://www.geeksforgeeks.org/python-different-ways-to-kill-a-thread/
6  class thread_with_exception(threading.Thread):
7      def get_id(self):
8          if hasattr(self, '_thread_id'):
9              return self._thread_id
10         for id, thread in threading._active.items():
11             if thread is self:
12                 return id
13
14     def raise_exception(self):
15         thread_id = self.get_id()
16         res = ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id,
17             ctypes.py_object(SystemExit))
18         if res > 1:
19             ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
20             print('Exception raise failure')
21
```

9.4.2 *MQTT_Control*

```
from mqtt_display import MQTT_Display
from mymqtt import MyMQTT
import tkinter as tk
from t_e import thread_with_exception
import queue

#para que prueba funcione
import requests

class MQTT_Control(tk.Frame):
    def __init__(self, master, filename, map_coord,  lim_list):
        super().__init__(master)
        #generamos el estado inicial apartir de las coord del mapa
        for calle in map_coord.keys():
            map_coord[calle].update
            ({'last_vel':'--','last_type':'--','vel_infract':4,'type_infract':4})
        initial_state = map_coord
        #me coloco en master
        self.grid(row = 0, column = 0, sticky= 'NSWE')
        self.rowconfigure(1, weight = 1)
        self.columnconfigure(0, weight = 1)

        #atributos basicos
        self. queue = queue. Queue()
        self.mqtt = MyMQTT(filename, self.queue, lim_list)

        #elementos del frame
        self.b =  tk.Button(self, text = 'Recopilar datos en tiempo real', command
        = self.activate_mqtt)
        self.b.grid(row = 0, column = 0, sticky= 'WE')
        self.display_frame = tk.Frame(self)
        self.display_frame.grid (row = 1, column = 0, sticky= 'NSWE')
        self.display_frame.rowconfigure(0, weight = 1)
        self.display_frame.columnconfigure(0, weight = 1)
        self.display = MQTT_Display (self.display_frame, initial_state)

        #activamos el proceso de comprovación de mensajes
        self.periodicCall(  )


    def periodicCall(self):
        #hacemos que la cola se compruebe cada 200
        self.processIncoming(  )
        self.master.after(200, self.periodicCall)


    def processIncoming(self):
        #procesado de la cola de mensajes
        while self.queue.qsize(  ):
            try:
                msg = self.queue.get(0)
                if msg['err'] == False:
                    #si no hay error, enviamos los datos a display para actualizar
                    self.display.update(msg['data'])
                else:
                    if msg['data']['conn_error'] == True:
                        #si se ha producido un error de conexion matamos el hilo
                        de mqtt y mostramos un mensaje de error
                        self.deactivate_mqtt()
                    #mostrarmos una ventana indicando al usuario lo que debe hacer
                    self.error_message(msg['data']['message'])


            except queue.Empty:
```

```
                pass


    def activate_mqtt(self):
        #Hago que se empiece a recopilar
        #siempre creo un hilo nuevo porque los hilos no pueden reinciarse una vez se matan
        self.hilo = thread_with_exception(target = self.mqtt.activate, daemon = True)
        self.hilo.start()

        #modifica el botón
        self.b['text'] = 'Deactivar la recopilación de datos en tiempo real'
        self.b['command'] = self.deactivate_mqtt

    def deactivate_mqtt(self):
        #deactivo el hilo que maneja el bucle de mqtt
        self.hilo.raise_exception()

        #modifico el botón
        self.b['text'] = 'Recopilar datos en tiempo real'
        self.b['command'] = self.activate_mqtt


    def error_message (self, error):
        #mostramos una alerta de que se ha producido un error
        tk.messagebox.showerror(title='Error', message = error)

if __name__ == "__main__":
    import pandas as pd
    #prueba
    plot_list={'AvCat1':{'x':[1100],'y':[800],'state':0,'last_vel':'--','last_type':'--','vel_infract':4,'type_infract':4},
        'AvCat2':{'x':[1225],'y':[745],'state':0,'last_vel':'--','last_type':'--','vel_infract':4,'type_infract':4},
        'AvSancho':{'x':[850],'y':[530],'state':0,'last_vel':'--','last_type':'--','vel_infract':4,'type_infract':4},
        'Labrit':{'x':[1150],'y':[250],'state':0,'last_vel':'--','last_type':'--','vel_infract':4,'type_infract':4}
    }
    df = pd.DataFrame(columns=['event','Instante','Velocidad','Aceleracion','Vehiculo','Calle'])

    def quit_me():
        #print('quit')
        root.quit()
        root.destroy()

    root = tk.Tk()
    root.protocol("WM_DELETE_WINDOW", quit_me)
    root.columnconfigure(0, weight = 1)
    root.rowconfigure(0, weight = 1)

    #creo la lista de limites que necesito:
    #el formato de la lista de limites debe ser
    #{calle1:{'velocidad': int, 'vehiculos':[_lista de vehiculos no permitidos_]},
    #   calle2:{'velocidad': int, 'vehiculos':[_lista de vehiculos no permitidos_]},
    #   ..
    #   }
    #obtengo los datos
```

```
115     try:
116         url = 'http://keltxo.no-ip.org:23040/'
117         r = requests.get(url+'/limites')
118         response = r.json()
119 
120         #los pongo en el formato adecuado
121         lim_list = dict()
122         map_coord = dict()
123         for data in response['data']:
124             vehiculos = []
125             for veh in data[3].split(','):
126                 #convertimos la lista de vehiculos de str a list
127                 vehiculos.append(int(veh))
128             lim_list.update( { data[1] : dict( zip(['velocidad','vehiculos'] ,
                [data[2] ,vehiculos]) )} )
129             map_coord.update ({data[1]: dict (zip (['x', 'y', 'state'],
                [[data[4]], [data[5]], 0]) ) })
130     except:
131         lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2':
            {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0,
            'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}
132 
133     print(lim_list)
134 
135 
136     m = MQTT_Control(master = root, filename = 'settings.json',  map_coord =
        map_coord ,  lim_list = lim_list)
137     root.mainloop()
138 
```

9.4.3 *MyMQTT*

```
#Almacena todos los mensajes recibidos através de mqtt en un dataframe
import paho.mqtt.client as mqtt
import pandas as pd
from pandastable import Table, TableModel

import tkinter as tk
from mytk import MyTk


#para que prueba funcione
import json #para procesar lo almacenado en settings
import queue
import requests

from mqtt_parser import MQTT_Parser as parser
#procesa los mensajes recibidos y los mete en la cola como diccionarios
#tambien los almacena en un df

class MyMQTT(mqtt.Client):
    def __init__(self, filename, queue, lim_list):
        super().__init__()
        #almacenamos la lista de limites
        self.p = parser (lim_list)

        #recibimos la cola sobre la que ponemos los mensajes por inyeccion
        self.queue = queue

        #obtenemos la configuración de la conexión de filename
        self.get_settings(filename)




    def get_settings(self,filename):
        #obtener los datos del archivo de configuraciones
        try:
            #intentamos abrir el archivo de configuraciones
            with open (filename) as j_settings:
                settings = json.load(j_settings)
                self.addr = settings['mqtt']['addr']
                self.port = settings['mqtt']['port_mqtt']
                self.user = settings['mqtt']['user']
                self.password = settings['mqtt']['password']
                self.topic = settings['mqtt']['topic']
        except:
            #si no se puede usamos la cola para pedir a mqtt_control que saque un mensaje de error
            message = 'Se ha producido un error de configuración. Revise la configuración'
            self.queue.put({'err':True, 'data': {'conn_error' : False, 'message': message}})


    def on_connect(self,client, userdata, flags, rc):
        #al conectarnos
        print("Connected with result code "+str(rc))
        client.subscribe(self.topic)


    def on_message(self,client, userdata, msg):
        #cuando recibo un mensaje lo proceso y lo meto en la cola
        #proceso el mensaje
```

```
62          text=str(msg.payload)
63          data= self.p.parse_text (text,str(msg.topic))
64
65          #lo pongo en la cola
66          self.queue.put({'err':False,'data': data})
67
68
69
70
71      def activate(self):
72          #inicia el bucle de mqtt
73          try:
74              #intentamos abrir una conexion
75              self.username_pw_set(self.user,password=self.password)
76              self.connect(self.addr, self.port, 60)
77
78          except:
79              #si no se puede usamos la cola para hacer que mqtt_control muestre un mensaje de error
80              print('error de conexión')
81              message = 'Se ha producido un error de conexión. Revise la configuración de los datos en tiempo real y su conexión a internet.'
82              self.queue.put({'err':True, 'data': {'conn_error' : True, 'message': message}})
83
84          self.loop_forever()
85
86
87
88      def deactivate(self):
89          #para el bucle de mqtt
90          self.loop_stop()
91
92
93
94  if __name__ == "__main__":
95      #prueba
96      #creo la cola que que neceisto por inyeccion
97      queue = queue.Queue(  )
98
99      #creo la lista de limites que necesito:
100     #el formato de la lista de limites debe ser
101     #{calle1:{'velocidad': int, 'vehiculos':[_lista de vehiculos no permitidos_]},
102     #   calle2:{'velocidad': int, 'vehiculos':[_lista de vehiculos no permitidos_]},
103     #   ..
104     #   }
105     #esta puesto en un try para que pueda seguir haciendo pruebas sin internet
106     try:#obtengo los datos
107         url = 'http://keltxo.no-ip.org:23040/'
108         r = requests.get(url+'/limites')
109         response = r.json()
110
111         #los pongo en el formato adecuado
112         lim_list = dict()
113         for data in response['data']:
114             vehiculos = []
115             for veh in data[3].split(','):
116                 #convertimos la lista de vehiculos de str a list
117                 vehiculos.append(int(veh))
118             lim_list.update( { data[1] : dict( zip(['velocidad','vehiculos'] , [data[2] ,vehiculos]) )} )
119     except:
120         lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2': {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0,
            'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}
121
122
123
124     #creo el objeto MyMQTT y lo activo
125     client=MyMQTT(filename = 'settings.json', queue = queue, lim_list = lim_list)
126     client.activate()
127
128
129
```

9.4.4 *MQTT_Parser*

```
#convierte los mensajes de mqtt al formato correcto
class MQTT_Parser():
    def __init__ (self ,lim_list):
        #almacenamos los datos necesarios
        print(lim_list)
        self.lim_list = lim_list
        self.margin = 0.15 #margen de infraccion para ver cual es amarillo y cual es rojo

    def parse_text (self,text, topic):
        print(text)
        #devuelvo el mensaje en el formato correcto
        #{new_row: _dict con los datos de la incidencia_, infract: {_diccinario con los datos del color del indicador-}}
        #procesamos el mensaje para convertirlo en un diccionario
        data= text.split ("'")
        data = data[1].split(';')
        topic = topic.split('/')
        vehiculos = {'1' : 'Moto',
            '2' : 'Coche',
            '3' : 'Furgoneta',
            '4' : 'Autobús',
            '5' : 'Camión',
            '666.0' : 'No identificado'}
        #convertimos el tipo de vehiculo en una palabra
        #la velocidad en un float de 2 decimales
        new_row = {'event' : 0,
                'Instante' : data[0],
                'Velocidad' : float("{:.2f}".format(float(data[1]))),
                'Aceleracion' : float("{:.2f}".format(float(data[2]))),
                'Vehiculo' : vehiculos[data[3]], #data[3],
                'Calle' : topic[1]}

        #procesamos  new_row para comprovar si hay errores en el mensaje que hemos recibido
        #errores de vehiculo
        vehiculos = self.lim_list[new_row['Calle']]['vehiculos']
        for vehiculo in vehiculos :
            if str(data[3]) == str(vehiculo):
                #vehiculo no permitido
                type_infract = 3 #red
                break #salimos del bucle
            else:
                #vehiculo permitido
                type_infract = 1 #green

        #errores de velocidad
        if new_row['Velocidad'] >= self.lim_list[new_row['Calle']]['velocidad']:
            #velocidad incorrecta
            if new_row['Velocidad'] > (self.lim_list[new_row['Calle']]['velocidad']*self.margin):
                #fuera del margen
                vel_infract = 3 #red
            else:
                #dentro del margen
                vel_infract = 2 #yellow
        else:
            #velocidad correcta
            vel_infract = 1 #green

        #devolvemos los datos
        infract ={'vel_infract' : vel_infract, 'type_infract' : type_infract}
        out = {'new_row': new_row, 'infract' : infract}
        print(out)
        return out


```

9.4.5 *MQTT_Display*

```
from mqtt_map import MQTT_Map  as Map
import tkinter as tk
from mytk import MyTk
import pandas as  pd

#wrapper class que mete mqtt_map y mytk en el mismo frame
class MQTT_Display(tk.Frame):
    def __init__(self, master, initial_state):
        super().__init__(master)
        #definimos el dataframe y nos colocamos en master
        self.df = pd.DataFrame(columns=['event','Instante','Velocidad','Aceleracion','Vehiculo','Calle'])
        self.grid(row = 0, column = 0, sticky= 'NSWE')
        self.rowconfigure(0, weight = 0)
        self.rowconfigure(21,weight=1)
        self.columnconfigure(0, weight = 1)

        #tiene un frame que contiene un mapa
        self.map_frame = tk.Frame(self)
        self.map_frame.grid(row = 0,column = 0,sticky = 'NSWE', rowspan= 20)
        self.map_frame.rowconfigure(0, weight = 0)
        self.map_frame.columnconfigure(0, weight = 1)

        #coloco el mapa
        self.mf = tk.Frame(self.map_frame)
        self.mf.columnconfigure(0, weight = 1)
        self.mf.rowconfigure(0, weight = 1)
        self.mf.grid(row = 0 ,column = 0, rowspan = 20, columnspan=20)
        self.map = Map(master = self.mf, initial_state = initial_state)

        #tiene un mytk
        self.table_frame = tk.Frame(self)
        self.table_frame.grid(row = 21, column = 0, sticky = 'NSWE')
        self.table_frame.rowconfigure(0, weight = 1)
        self.table_frame.columnconfigure(0, weight = 1)

        self.table=MyTk(dataframe = self.df, master = self.table_frame)

    def update(self, data):
        print('estoy en update de display')
        new_row = data['new_row']
        infract = data['infract']
        #el formato de data es
        #{new_row: dict, infract {'vel_infract' : int, 'type_infract' : int}}
        #actualiza el modelo de la tabla
        #hacer append en dataframe devuelve un nuevo objeto, por eso hay que volver a reasignar
        #ref:https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.append.html
        self.df = self.df.append(new_row, ignore_index = True)
        self.table.update_model(self.df)

        #actualiza el mapa
        self.map.update(calle = new_row['Calle'], vel = new_row['Velocidad'],
        vel_infract = infract['vel_infract'], tipo = new_row['Vehiculo'],
        tipo_infract = infract['type_infract'])


    def prueba (self,msg):
        print('display imprimiendo')
        print(msg)

```

```
58  if __name__ == "__main__":
59      #prueba
60
        plot_list={'AvCat1':{'x':[1100],'y':[800],'state':0,'last_vel':30,'last_type':'m
        oto','vel_infract':1,'type_infract':1},
61
            'AvCat2':{'x':[1225],'y':[745],'state':0,'last_vel':70,'last_type':'camión',
            'vel_infract':3,'type_infract':3},
62
            'AvSancho':{'x':[850],'y':[530],'state':0,'last_vel':60,'last_type':'coche',
            'vel_infract':2,'type_infract':1},
63
            'Labrit':{'x':[1150],'y':[250],'state':0,'last_vel':80,'last_type':'moto','v
            el_infract':3,'type_infract':1}
64      }
65      df =
        pd.DataFrame(columns=['event','Instante','Velocidad','Aceleracion','Vehiculo','C
        alle'])
66
67      def quit_me():
68          #print('quit')
69          root.quit()
70          root.destroy()
71
72      root = tk.Tk()
73      root.protocol("WM_DELETE_WINDOW", quit_me)
74      root.columnconfigure(0, weight = 1)
75      root.rowconfigure(0, weight = 1)
76
77      frame = tk.Frame(root)
78      frame.grid(row = 0, column = 0)
79      m = MQTT_Display(master = root, initial_state = plot_list , df = df)
80
81
82
83      #b.grid(row = 1, column = 0)
84      root.mainloop()
85
```

9.4.6 *MyTk*

```
import tkinter as tk
from tkinter import ttk
from tkinter import filedialog
from pandastable import Table, TableModel
import pandas as pd
import requests

from threading import Thread

class MyTk(tk.Frame):
    #frame que muestra un dataframe como una tabla
    def __init__(self, dataframe, master = None):
        super().__init__(master)
        #parametros que recibo por inyeccion
        self.df = dataframe
        self.master = master

        #colocaccion basica del frame
        self.grid(row = 0, column = 0, sticky = 'NSWE')
        self.rowconfigure(0, weight = 1)
        self.columnconfigure(0, weight = 1)


        self.tableFrame = tk.Frame(self)
        self.tableFrame.grid(row = 0, column = 0, sticky = 'NSWE')
        self.model()

        #creo el botón de descargar datos
        self.downloadbutton = tk.Button(self, text = 'Descargar datos', command = ↵
        self.download_cvs)
        self.downloadbutton.grid(row=1,column=0)

    def create_conditional_style(self, df):
        #ref:https://github.com/plotly/dash-table/issues/432
        style=[]
        for col in df.columns:
            PIXEL_FOR_CHAR = 5
            name_length = len(col)
            pixel = 50 + round(name_length*PIXEL_FOR_CHAR)
            pixel = str(pixel) + "px"
            style.append({'if': {'column_id': col}, 'minWidth': pixel})
        return style

    def model(self):
        #crea la tabla apartir del dataframe
        self.pt = Table(self.tableFrame,dataframe = self.df)
        self.pt.show()
        #print('he hecho show')

    def update_model(self,df):
        #recibo una nueva df por inyeccion y actualizo la tabla con ese nuevo df
        self.df = df
        self.pt.updateModel(TableModel(df))
        self.pt.redraw()

    def download_cvs(self):
        #para descargar df como cvs
        export_file_path = filedialog.asksaveasfilename(defaultextension = '.csv', ↵
        filetypes =[('archivo csv', '*.csv')])
        self.df.to_csv (export_file_path, index = None, header = True, sep = ';')

    def error_message(self,message):
        #recibo un mensaje de error por inyeccion y lo muestro
        tk.messagebox.showerror(title = 'Error', message = message)
```

```
63
64  #prueba
65  if __name__ == "__main__":
66      def quit_me():
67          #print('quit')
68          root.quit()
69          root.destroy()
70
71      root=tk.Tk()
72      root.protocol("WM_DELETE_WINDOW", quit_me)
73      root.columnconfigure(0, weight = 1)
74      root.rowconfigure(0, weight = 1)
75
76      url = 'http://keltxo.no-ip.org:23040/'
77      payload = {'TS1':'2020-7-14-14 2013:19:55' , 'id':'AvCat2'}#, 'TS2':'2020-7-14
        15:00:00'}
78      r = requests.get(url, params = payload)
79      response = r.json()
80      df=pd.DataFrame(response['data'], columns =
        ['event','Instante','Velocidad','Aceleracion','Vehiculo'])
81
82
83
84      app = MyTk(dataframe = df, master = root)
85      app.mainloop()
86
87  #print('primer hilo iniciado)')
88  #app.mainloop()
89  #control_thread.join()
90
91  #app.start()
92  #app.loop_forever()
93
```

9.4.7 *MQTT_Map*

```
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

#esto es un plot que permite visualizar los mensajes de mqtt en un mapa
class MQTT_Map(tk.Frame):
    def __init__(self,master,initial_state):
        super().__init__(master)
        #parametros basicos

        self.colors={1:'green', 2:'yellow', 3:'red', 4:'black'}

        #me coloco dentro del master
        self.grid(row = 0, column = 0,sticky ='NSWE')
        #self.rowconfigure(0, weight = 1)
        #self.columnconfigure(0, weight = 1)


        #creo la figura y el ax donde va el plot
        f = Figure(figsize = (8,4), dpi = 100, )
        self.ax = f.add_subplot(111)

        #añado la imagen al fondo
        img = plt.imread("pamplona1.png")

        self.ax.imshow(img)
        #creo el diccionario donde voy a almacenar los indicadores
        self.points = dict()

        #creo los indicadores y los guardo en el diccionario
        for sensor in initial_state.keys():

            #ref:https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.axes.Axes.scatter.html#matplotlib.axes.Axes.scatter

            #calculo la posicion de los dos puntos ene el eje y
            #porque s=70, del centro del marker al borde hay 35, por tanto para que estén pegados debemos subir
            #un centro 35/2=17 y bajar el otro la misma distancia
            y1 = initial_state[sensor]['y'][0]-17
            y2 = initial_state[sensor]['y'][0]+17

            #creamos los dos indicdores, uno para velocidad y otro para tamaño del vehículo
            #scatter(self, x=_valores de x_ , y = _valores de y_, c=_color_, s=_tamaño_, edgecolors=__color del borde__)
            #point es el indicador -el punto- y ann es la anotacion -el texto que tiene al lado-

            #indicador de velocidad
            point_vel = self.ax.scatter(initial_state[sensor]['x'], [y1], c = self.colors[initial_state[sensor]['vel_infract']], s = 70, edgecolors = 'white', marker = 's')
            ann_vel = self.ax.annotate(initial_state[sensor]['last_vel'], xy = (initial_state[sensor]['x'][0]+20, y1+15))

            #indicador del tipo de vehículo
            point_type = self.ax.scatter(initial_state[sensor]['x'], [y2], c = self.colors[initial_state[sensor]['type_infract']], s = 70, edgecolors = 'white', marker = 's')
            ann_type = self.ax.annotate(initial_state[sensor]['last_type'], xy = (initial_state[sensor]['x'][0]+20,y2+15))

```

```
53              #una vez que he creado los indicadores los almaceno en el diccionario points
54              self.points.update({  sensor : [point_vel, point_type, ann_vel, ann_type]})
55
56          #oculto los ejes, coloco el titulo y fijo los margenes
57          self.ax.axis('off')
58          self.ax.set_title('Sensores en tiempo real')
59          plt.tight_layout()
60
61          #lo meto todo en un canvas
62          self.canvas = FigureCanvasTkAgg(f, master = self)
63          self.canvas.get_tk_widget().grid(row = 0, column = 0, sticky = 'NSWE')#.pack()
64
65      def update(self, calle, vel, vel_infract, tipo, tipo_infract):
66          #funcion para actualiar el indicador calle
67
68          #cambio los colores de los indicadores
69          self.points[calle][0].set_facecolor(self.colors[vel_infract])
70          self.points[calle][1].set_facecolor(self.colors[tipo_infract])
71
72          #cambio las anotaciones
73          self.points[calle][2].set_text(vel)
74          self.points[calle][3].set_text(tipo)
75
76          #redibujo el canvas
77          print('he modificado map')
78          self.canvas.draw()
79
80          print('voy a redibujar el canvas')
81
82
83
84
85
86  if __name__ == "__main__":
87      #prueba
88      plot_list={'AvCat1':{'x':[1100],'y':[800],'last_vel':'no disponible','last_type':'no disponible','vel_infract':4,'type_infract':4},
89          'AvCat2':{'x':[1225],'y':[745],'last_vel':'no disponible','last_type':'no disponible','vel_infract':4,'type_infract':4},
90          'AvSancho':{'x':[850],'y':[530],'last_vel':'no disponible','last_type':'no disponible','vel_infract':4,'type_infract':4},
91          'Labrit':{'x':[1150],'y':[250],'last_vel':'no disponible','last_type':'no disponible','vel_infract':4,'type_infract':4}
92      }
93
94      def quit_me():
95          #print('quit')
96          root.quit()
97          root.destroy()
98
99      root = tk.Tk()
100     root.protocol("WM_DELETE_WINDOW", quit_me)
101     root.columnconfigure(0, weight = 1)
102     root.rowconfigure(0, weight = 1)
103
104     frame = tk.Frame(root)
105     frame.grid(row = 0, column = 0)
106     m = MQTT_Map(frame, initial_state = plot_list)
107
108     #probamos la funcion actualizar
109     m.update(calle = 'AvCat1', vel = 666, vel_infract = 3, tipo = 'ESTO ES UNA PRUEBA', tipo_infract = 3)
110
111     #b.grid(row = 1, column = 0)
112     root.mainloop()
113
```

9.5 Datos históricos

9.5.1 *Map*

```
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg


class Map(tk.Frame):
    def __init__(self, master, plot_list):
        super().__init__(master)
        #me colcoco en master y guardo los datos que necesito
        self.grid(row = 0, column = 0, sticky = 'NSWE')
        self.plot_list = plot_list

        #frame donde voy  a guardar todo
        plotFrame = tk.Frame(self)
        plotFrame.grid(row = 0, column = 0,sticky = 'NSWE')

        #creo la figura  donde va el plot
        f = Figure(figsize = (8,4), dpi = 100, )
        self.ax = f.add_subplot(111)

        #añado la imagen al fondo
        img = plt.imread("pamplona1.png")
        self.ax.imshow(img)

        #añado los puntos
        self.points = dict()

        for sensor in plot_list.keys():
            #ref:https://matplotlib.org/3.3.1/api/_as_gen/matplotlib.axes.Axes.scatter.html#matplotlib.axes.Axes.scatter
            #scatter(self,x,y,c=_color_,s=_tamaño_,edgecolors=__color del borde__)
            point = self.ax.scatter(self.plot_list[sensor]['x'], self.plot_list[sensor]['y'], c = 'red',s = 70, edgecolors = 'white')
            self.points.update({point : sensor})
            point.set_picker(True) #para que sea clicable
            self.ax.text(self.plot_list[sensor]['x'][0]-40, self.plot_list[sensor]['y'][0]-17, sensor, size = 8)

        #oculto los ejes
        #self.ax.axis('off')
        self.ax.set_title('Seleciona los sensores que desea activar')
        plt.tight_layout()

        #lo meto todo en un canvas
        self.canvas = FigureCanvasTkAgg(f, master = plotFrame)
        self.canvas.get_tk_widget().grid(row = 0, column = 0, sticky = 'NSWE')

        #asigno la función para el evento de clickar
        f.canvas.mpl_connect('pick_event', self.onpick)



    def onpick(self, event):
        #evento de clickar sobre el plot
        #ref: https://matplotlib.org/3.1.1/users/event_handling.html
        #donde se ha producido el evento,¿a que sensor corresponde?
        point = event.artist
        sensor = self.points[point]

        #acutalizar el color y el estado
        if self.plot_list[sensor]['state'] == 0:
            #si está desactivado lo activamos  y lo pintamos verde
```

```
61                  self.plot_list[sensor]['state'] = 1
62                  point.set_facecolor('green')
63              else:
64                  #si está activado lo desactivamos  y lo pintamos rojo
65                  self.plot_list[sensor]['state'] = 0
66                  point.set_facecolor('red')
67
68              #volver a dibujar el canvas
69              self.canvas.draw()
70
71
72  if __name__ == "__main__":
73      plot_list={'AvCat1':{'x':[1100],'y':[800],'state':0},
74          'AvCat2':{'x':[1225],'y':[745],'state':0},
75          'AvSancho':{'x':[850],'y':[530],'state':0},
76          'Labrit':{'x':[1150],'y':[250],'state':0}
77      }
78
79      def quit_me():
80          #print('quit')
81          root.quit()
82          root.destroy()
83      root = tk.Tk()
84      root.protocol("WM_DELETE_WINDOW", quit_me)
85      root.columnconfigure(0, weight = 1)
86      root.rowconfigure(0, weight = 1)
87      frame=tk.Frame(root)
88      frame.grid(row = 0, column = 0)
89      m=Map(frame, plot_list)
90      root.mainloop()
91
```

9.5.2 *Control*

```
import tkinter as tk
from mytk import MyTk
import requests
import pandas as pd
from tkinter import ttk, Entry
from map import Map
from tkcalendar import Calendar, DateEntry

#from t_e import thread_with_exception
from display import Display
import tkinter.messagebox
import json


class Control(tk.Frame):
    def __init__(self, filename, lim_list, map_coord, master=None):
        super().__init__(master)
        #almaceno los datos que necesito como atributos
        self.filename = filename
        self.limites = lim_list
        self.map_coord = map_coord


        #me pongo en master
        self.grid(row = 0,column = 0,sticky = 'NSWE')

        #configuro la expansion de las filas
        self.rowconfigure(0,weight=0)
        self.rowconfigure(26,weight=1)
        self.columnconfigure(0,weight=1)

        #
        self.get_Sensor()
        self.basic_window()

    def basic_window(self):
        #frame donde voy a meter el mapa
        self.control_frame = tk.Frame(self)
        self.control_frame.grid(row = 0,column = 0,sticky = 'NSWE')
        self.control_frame.rowconfigure(0, weight = 0)
        self.control_frame.columnconfigure(0, weight = 1)
        self.control_frame.rowconfigure(3, weight = 0)
        self.control_frame_vis = True

        #controles para escoger el timestamp
        self.tscontrols = tk.Frame(self.control_frame)
        self.tscontrols.grid(column = 0, row = 0, rowspan = 2, sticky = 'NSWE',
        columnspan = 4)
        self.rowconfigure(0, weight = 0)
        self.rowconfigure(1, weight = 0)
        self.lab1=ttk.Label(self.tscontrols, text = 'Escoje la fecha de inicio')
        self.lab1.grid(column = 0, row = 0)
        self.lab2 = ttk.Label(self.tscontrols, text = 'Escoje la fecha fin')
        self.lab2.grid(column = 0, row = 1)
        #datenetry para escoger el día
        #x defecto DateEntry toma la fecha actual
        self.ts1 = DateEntry(self.tscontrols, width = 12, background = 'darkblue',
        foreground = 'white', borderwidth = 2)
        self.ts1.grid(column = 1, row = 0)
        self.ts2 = DateEntry(self.tscontrols, width = 12, background = 'darkblue',
        foreground = 'white', borderwidth = 2)
        self.ts2.grid(column = 1, row = 1)
        #listbox para escoger la hora y el minuto
        self.lab3 = ttk.Label(self.tscontrols, text = 'Escoje la hora de inicio:')
```

```
62          self.lab3.grid(column = 2, row = 0)
63          self.h_ts1 = Entry(self.tscontrols, width = 7)
64          self.h_ts1.insert (0,'00:00')
65          self.h_ts1.grid (column = 5, row = 0)
66          self.lab4 = ttk.Label(self.tscontrols, text = 'Escoje la hora fin:')
67          self.lab4.grid(column = 2, row = 1)
68          self.h_ts2 = Entry(self.tscontrols, width = 7)
69          self.h_ts2.insert (0,'23:59')
70          self.h_ts2.grid (column = 5, row = 1)
71
72          #creo y colco el mapa
73          self.mf = tk.Frame(self.control_frame)
74          self.mf.columnconfigure(0, weight = 1)
75          self.mf.rowconfigure(0, weight = 1)
76          self.mf.grid(row = 3 ,column = 0, rowspan = 20, columnspan=20)
77          m = Map(self.mf,self.map_coord)
78
79          #botones de peticion y mqtt y para ocultar las peticiones
80          self.b_frame = tk.Frame(self)
81          self.b_frame.grid(row = 24, column = 0, sticky = 'NSWE')
82          self.OK = tk.Button(self.b_frame)
83          self.OK["text"] = "Realizar Petición"
84          self.OK["command"] = self.update
85          self.OK.grid(column = 1, row = 24)
86
87          self.vis_button = tk.Button(self.b_frame, text = 'Ocultar generador de
            petciones', command = self.toggle_control_frame_vis)
88          self.vis_button.grid(column = 2, row = 24)
89
90          #creamos el notebook donde vamos a almacenar display
91          self.nb = ttk.Notebook(self)
92          self.nb.grid(row = 26, column = 0, sticky = 'NSWE')
93
94          self.frame = tk.Frame(self.nb)
95          self.frame.grid(row = 0, column = 0, sticky = 'NSWE')
96          self.frame.columnconfigure(0, weight = 1)
97          self.frame.rowconfigure(0, weight = 1)
98
99
100
101     def toggle_control_frame_vis(self):
102         #cambia la visibilidad de frame
103
104         if self.control_frame_vis == True:
105             #si está visible lo ocultamos
106             self.control_frame_vis = False
107             self.control_frame.grid_remove()
108             self.vis_button["text"] = "Mostrar el generador de peticiones"
109
110         else:
111             #si no es visible la mostramos
112             self.control_frame_vis = True
113             self.control_frame.grid(row = 0, column = 0, sticky = 'NSWE')
114             self.vis_button["text"] = 'Ocultar generador de petciones'
115
116
117     def get_data(self, ts1, ts2, calle):
118         #obtener el df de datos
119         payload = {'TS1':ts1 , 'id':calle, 'TS2':ts2}
120         print(payload)
121         try:
122             print('voy a hacer la peticion')
123             r = requests.get(self.url, params = payload)
124             response = r.json()
```

```
125                 print(response)
126                 if response['err'] == True:
127                     message = 'Se ha producido un error en el servidor.'
128                     tk.messagebox.showerror(title='Error', message = message)
129                 else:
130                     df = pd.DataFrame(response['data'], columns = ['event', 'time',
                        'velocidad', 'aceleracion', 'tipo de vehiculo'])
131                     if df.empty == False:
132                         print('df no vacio')
133                         #ponemos el time en el formato adecuado
134                         df['time'] = pd.to_datetime(df['time'],
                            format="%Y-%m-%d""T""%X"".""%f")
135                         df['calle'] = calle
136                         print(df)
137                     return df
138             except:
139                 #si no podemos contactar con url sacamos un mensaje de error y
                    devolvemos un df null
140                 message = 'Se ha producido un error al realizar la petición. Compruebe
                    la configuración y la conexión a internet.'
141                 tk.messagebox.showerror(title='Error', message = message)
142
143             #devolvemos el dataframe con los datos
144
145
146
147
148         def get_Sensor(self):
149             #obtener la lista de coordenadas y de sensores
150             #obtenemos los parametros de peticion del archivo
151             try:
152                 with open (self.filename) as j_settings:
153                     settings = json.load(j_settings)
154                     self.url = settings['data']['url_data'] + ":" +
                        str(settings['data']['port_data'])
155                     url_lim = self.url + settings['data']['lim']
156             except:
157                 message = 'Se ha producido un error. Compruebe la configuración.'
158                 tk.messagebox.showerror(title='Error', message = message)
159
160             #hago la peticion
161             self.limites = dict()#diccionario de los limites de peso y velocidad de
                cada sensor
162             self.map_coord = dict()#diccionario de las coordenadas de los sensores
163             self.sensor_tab = dict() #diccionario de los frames donde metemos los display
164             self.sensor_list = []#lista con los sensores
165             r = requests.get(url_lim)
166             response = r.json()
167             print(response)
168             #pongo los resultados en el formato adecuado
169             for data in response['data']:
170                 vehiculos=[]
171                 for veh in data[3].split(','):
172                     vehiculos.append(int(veh))
173                 self.limites.update({data[1] :
                    dict(zip(['id','calle','velocidad','vehiculos'],[data[0],data[1],data[2]
                    ,vehiculos]))})
174
175                 self.map_coord.update({data[1] :
                    {'x':[int(data[4])],'y':[int(data[5])],'state':0}})
176                 self.sensor_list.append(data[1])
177                 self.sensor_tab.update({data[1] : None})
178
179
```

```
180         def update(self):
181             #compruebo que hay un sensor seleccionado y si es así pido los datos
                correspondientes
182             #y genero un objeto display para mostrar los resultados
183             one_sensor_selected = False
184             ts1 = str(self.ts1.get_date()) + ' ' + str(self.h_ts1.get())
185             ts2 = str(self.ts2.get_date())+ ' ' + str(self.h_ts2.get())
186 
187             #comprueba el estado de cada sensor
188             for sensor in self.map_coord.keys():
189                 if self.sensor_tab[sensor] != None :
190                         #si existe un display de este sensor, lo oculto y lo elimino
191                         self.nb.forget(self.sensor_tab[sensor])
192                         self.sensor_tab[sensor] = None
193 
194                 if self.map_coord[sensor]['state'] == 1:
195                     #si el sensor está activado
196                     one_sensor_selected = True
197                     #esta seleccionado, pido sus datos
198                     calle = sensor
199                     df = self.get_data(ts1 ,ts2, calle)
200                     try:
201                         if df.empty == True :
202                             #si el dataframe esta vacio ->mensaje de error
203                             tkinter.messagebox.showinfo(title = None, message = 'No
                                hay nada registrado para los datos seleccionados')
204 
205                         else:
206                             #si hay datoscreo el frame donde voy a meter el display
207                             self.sensor_tab[sensor] = tk.Frame(self.nb)
208                             self.sensor_tab[sensor].rowconfigure(0, weight = 1)
209                             self.sensor_tab[sensor].columnconfigure(0, weight = 1)
210 
211                             #creo el display
212                             Display(self.sensor_tab[sensor], df, self.limites[sensor])
213 
214 
215                             #añado la nueva
216                             self.nb.add(self.sensor_tab[sensor], text = sensor)
217                     except:
218                         #si df no tiene  atributo empty es porque es None
219                         #se ha producido un error de conexión y lo hemos indicado
220                         pass
221 
222             if one_sensor_selected == False :
223                 #si no hay ningún sensor seleccionado saco un mensaje de error
224                 tkinter.messagebox.showinfo(title = None, message = 'No ha selecionado
                    ningún sensor')
225 
226 
227 
228 
229 if __name__ == "__main__":
230 
231     def quit_me():
232         #print('quit')
233         root.quit()
234         root.destroy()
235     root=tk.Tk()
236     root.title ('Initial OF')
237     root.protocol("WM_DELETE_WINDOW", quit_me)
238     root.columnconfigure(0, weight = 1)
239     root.rowconfigure(0, weight = 1)
240 
```

```
241     try:
242         url = 'http://keltxo.no-ip.org:23040/'
243         r = requests.get(url+'/limites')
244         response = r.json()
245 
246         #los pongo en el formato adecuado
247         lim_list = dict()
248         for data in response['data']:
249             vehiculos = []
250             for veh in data[3].split(','):
251                 #convertimos la lista de vehiculos de str a list
252                 vehiculos.append(int(veh))
253             lim_list.update( { data[1] : dict( zip(['velocidad','vehiculos'] ,
                [data[2] ,vehiculos]) )} )
254             map_coord.update ({data[1]: dict (zip (['x', 'y', 'state'],
                [[data[4]], [data[5]], 0]) ) })
255     except:
256         lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2':
            {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0,
            'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}
257         map_coord ={'AvCat1':{'x':[1100],'y':[800],'state':0},
258         'AvCat2':{'x':[1225],'y':[745],'state':0},
259         'AvSancho':{'x':[850],'y':[530],'state':0},
260         'Labrit':{'x':[1150],'y':[250],'state':0}
261     }
262 
263 
264 
265 
266     d = Prueba(master = root, filename = 'settings.json', lim_list = lim_list,
        map_coord = map_coord)
267     root.mainloop()
268 
```

9.5.3 *Display*

```
#display: es un frame que contiene un notebook
#recibe por inyeccion df y limites de una calle
import tkinter as tk
from tkinter import ttk
from flux import Flux
from graphs import Graphs
from infract import Infract
from mytk import MyTk
import requests
import pandas as pd

class Display(tk.Frame):
    def __init__(self, master, df, list_lim):
        super().__init__(master)
        #me coloco dentro de master
        self.grid(row = 0, column = 0, sticky = 'NSWE')
        self.rowconfigure(0, weight= 1)
        self.columnconfigure(0, weight = 1)

        #creo mi notebook
        self.notebook = ttk.Notebook(self)
        self.notebook.grid(row = 0, column = 0, sticky = 'NSWE')

        #añado la pestaña de todos los datos
            #para que se ajuste a los cambios de la pantalla necesito un frame intermediario
        self.dframe = tk.Frame(self.notebook)
        self.dframe.grid(row = 0, column = 0, sticky = 'NSWE')
        self.dframe.columnconfigure(0, weight = 1)
        self.dframe.rowconfigure(0, weight = 1)

        self.mytk = MyTk(df, self.dframe)
        self.notebook.add(self.dframe, text = 'Datos')


        #añado la pestaña de infracciones
        frame2 = tk.Frame(self.notebook)
        frame2.columnconfigure(0, weight = 1)
        frame2.rowconfigure(0, weight = 1)
        self.infract = Infract(frame2, df, list_lim)
        self.notebook.add(frame2, text = 'Infracciones')


        #añado la pestaña de graficos
        self.gs = Graphs(self.notebook ,df)
        self.notebook.add(self.gs,text='graph')

        #añado la pestaña de flujo
        self.flux = Flux(self.notebook, df)
        self.notebook.add(self.flux, text = 'flujo')

if __name__ == "__main__":
    url = 'http://keltxo.no-ip.org:23040/'
    TS1='2020-9-8 10:23:06'
    TS2 ='2020-9-20 15:00:00'
    calle = 'AvCat1'
    headers = {'Accept': 'application/vnd.github.v3+json'}
    sensor_list=['AvCat1', 'AvCat2', 'Labrit', 'AvSancho']
    payload = {'TS1':TS1 , 'id':calle, 'TS2':TS2}
    r = requests.get(url, params=payload)
    response = r.json()
    df=pd.DataFrame(response['data'], columns=['event','time','velocidad','aceleracion','tipo de vehiculo'])
    df['time']=pd.to_datetime(df['time'],format="%Y-%m-%d""T""%X"".""%f")
```

```
63
64
65      url='http://keltxo.no-ip.org:23040/'
66      calle='AvCat1'
67
68
69      payload={'calle':calle}
70      r = requests.get(url+'/limites', params=payload)
71      response = r.json()
72
73
74      root=tk.Tk()
75      root.columnconfigure(0, weight=1)
76      root.rowconfigure(0, weight=1)
77      #obtengo los LIMITES
78      try:
79          url = 'http://keltxo.no-ip.org:23040/'
80          r = requests.get(url+'/limites')
81          response = r.json()
82
83          #los pongo en el formato adecuado
84          lim_list = dict()
85          sensor_list = dict()
86          for data in response['data']:
87              vehiculos = []
88              for veh in data[3].split(','):
89                  #convertimos la lista de vehiculos de str a list
90                  vehiculos.append(int(veh))
91              lim_list.update( { data[1] : dict( zip(['velocidad','vehiculos'] ,
                [data[2] ,vehiculos]) )} )
92      except:
93          lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2':
            {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0,
            'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}
94
95
96
97      d = Display(root, df, lim_list[calle])
98      #d.grid(row=0,column=0,sticky='NSWE')
99      root.mainloop()
100
```

9.5.4 *MyTk*

Ver sección 9.4.6 *MyTk*

9.5.5 *Infract*

```
import tkinter as tk
from tkinter import *
from tkinter import ttk

import pandas as pd
import requests

from mytk import MyTk
#recordatorio:
#lim_list ahora tiene el formato
#lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2': {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0, 'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}
#del cual le pasaremos solo el correspondiente a su calle
class Infract(tk.Frame):
    def __init__(self, master, df, lim_list ):#,url,calle,):#es un frame que contiene un notebook
        super().__init__(master)
        #nos colocamos dentro de master
        self.grid(row = 0, column = 0,sticky = 'NSWE')
        self.rowconfigure(0, weight = 1)
        self.columnconfigure(0, weight = 1)

        #guardamos la lista de limites
        self.limites = lim_list

        #creo mi notebook
        self.nb = ttk.Notebook(self)
        self.nb.grid(row = 0, column = 0, sticky = 'NSWE')

        #creo los dataframes donde vamos a almacenar las infracciones
        self.peso=pd.DataFrame(columns=['event', 'time', 'velocidad', 'aceleracion', 'tipo de vehiculo'])
        self.vel=pd.DataFrame(columns=['event', 'time', 'velocidad', 'aceleracion', 'tipo de vehiculo'])

        #compruebo el df y almaceno sus infracciones
        self.check_df(df)

        #inicializar notebook
        #infracciones por velocidad
        self.vel_tab = MyTk(dataframe = self.vel, master = self.nb)
        self.nb.add(self.vel_tab,text = 'Infracciones de velocidad')
        #infracciones por peso
        self.peso_tab = MyTk(dataframe = self.peso ,master = self.nb)
        self.nb.add(self.peso_tab, text = 'Infracciones de peso')


    def check_df(self, df):
        #genera un df qut contiene las infracciones que se han detectado
        #df:[id,time,sensor,vel,aceleracion,tipo]
        #limites:{'velocidad': -float-, 'tipos':[int,int,..]}

        #obtengo las infraciones de velocidad
        self.vel = df.loc[(df['velocidad'] > self.limites['velocidad'])]

        #obtengo las infracciones de peso
        vehiculos = {'1' : 'Moto',
            '2' : 'Coche',
            '3' : 'Furgoneta',
            '4' : 'Autobús',
            '5' : 'Camión',
            '666.0' : 'No identificado'}
        for vehiculo in self.limites['vehiculos']:
```

```
            dataframe = df.loc[df['tipo de vehiculo'] == vehiculo ]
            self.peso = self.peso.append(dataframe, ignore_index = True)




    def update(self,new_df):
        #volver a rellenar las tablas con las infracciones de un nuevo df
        self.check_df(new_df)
        self.peso_tab.update_model(self.peso)
        self.vel_tab.update_model(self.vel)




if __name__ == "__main__":
    url = 'http://keltxo.no-ip.org:23040/'
    TS1 = '2020-9-13 13:23:06'
    TS2 = '2020-9-14 15:00:00'
    calle = 'AvCat1'
    headers = {'Accept': 'application/vnd.github.v3+json'}
    sensor_list=['AvCat1', 'AvCat2', 'Labrit', 'AvSancho']
    payload = {'TS1':TS1 , 'id':calle, 'TS2':TS2}
    r = requests.get(url, params=payload)
    response = r.json()
    df=pd.DataFrame(response['data'],
    columns=['event','time','velocidad','aceleracion','tipo de vehiculo'])
    df['velocidad'] = pd.to_numeric(df['velocidad'], downcast="float")


    url='http://keltxo.no-ip.org:23040/'
    calle='AvCat1'


    payload={'calle':calle}
    r = requests.get(url+'/limites', params=payload)
    response = r.json()
    #obtengo los datos
    try:
        url = 'http://keltxo.no-ip.org:23040/'
        r = requests.get(url+'/limites')
        response = r.json()

        #los pongo en el formato adecuado
        lim_list = dict()
        for data in response['data']:
            vehiculos = []
            for veh in data[3].split(','):
                #convertimos la lista de vehiculos de str a list
                vehiculos.append(int(veh))
            lim_list.update( { data[1] : dict( zip(['velocidad','vehiculos'] ,
            [data[2] ,vehiculos]) )} )
    except:
        lim_list = {'AvCat1': {'velocidad': 30.0, 'vehiculos': [4, 5]}, 'AvCat2':
        {'velocidad': 50.0, 'vehiculos': [5]}, 'Labrit': {'velocidad': 30.0,
        'vehiculos': [2, 5]}, 'AvSancho': {'velocidad': 40.0, 'vehiculos': [4, 5]}}

    root=tk.Tk()
    root.columnconfigure(0, weight=1)
    root.rowconfigure(0, weight=1)




    infract=Infract(root, df, lim_list= lim_list[calle])
    infract.mainloop()
```

9.5.6 *Graph*

```
import tkinter
from tkinter import *
from tkinter import ttk

import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import pandas as pd
#ref:https://stackoverflow.com/questions/26435349/tkinter-canvas-refuses-to-expand-or-contract-when-window-resized
#root = Tk()
#root.columnconfigure(0, weight=1)
#root.rowconfigure(0, weight=1)
class Graph(Frame):
    def __init__(self,master,extra=0):
        super().__init__(master)


        #metodo al que hay que llamar para  mostrar lo que hay en fill_canvas
        #ajusto mi posicion
        self.grid(column=0, row=0, sticky=(N, W, E, S))
        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)




        #necesito un canvas para poder hacer scroll
        self.canvas = Canvas(self,      scrollregion=(0, 0, 1000, 2150+extra ))


        self.canvas.grid(column=0, row=0, sticky=(N, W, E, S))



        ##creo las scrollbar y les asigno el control del canvas

        self.scrollx = Scrollbar(self, orient="horizontal", command=self.canvas.xview)
        self.scrolly = Scrollbar(self, orient="vertical", command=self.canvas.yview)
        self.scrollx.grid(sticky='NSEW',row=1,column=0)
        self.scrolly.grid(sticky='NSWE',row=0,column=1)

        self.canvas.configure(xscrollcommand=self.scrollx.set,yscrollcommand=self.scrolly.set)
        self.viewPort = Frame(self.canvas)
        self.canvas_window = self.canvas.create_window((4,4), window=self.viewPort, anchor="nw",            #add view port frame to canvas
                                  tags="self.viewPort")
        #relleno el canvas
        #self.fill_canvas()

    #def fill_canvas(self):

        #importante meter los plots en viewPort

        #f1 = plt.Figure( dpi=100)
        #ax1 = f1.add_subplot(111)
        #line1 = FigureCanvasTkAgg(f1, self.viewPort)
        #line1.get_tk_widget().grid(column=0,row=0)
        #self.df.plot.scatter(x='time',y='velocidad',ax=ax1,legend=True)
        #ax1.set_title('Scatter plot de velocidad')


        #f2 = plt.Figure( dpi=100)
        #ax2 = f2.add_subplot(111)
        #line2 = FigureCanvasTkAgg(f2, self.viewPort)
        #line2.get_tk_widget().grid(column=0,row=1)
        #self.df.hist(column='velocidad',ax=ax2)
        #ax2.set_title('Histograma de velocidad')







        # canvas.create_line(10, 10, 200, 50)
#t=Graph(root)
#root.mainloop()

```

9.5.7 *Graphs*

```
import tkinter
from tkinter import *
from tkinter import ttk
from matplotlib.dates import (DAILY,HOURLY,YEARLY,MONTHLY, DateFormatter,
                              rrulewrapper, RRuleLocator, drange)
from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
import matplotlib.pyplot as plt

import matplotlib.dates as mdates# para el formato del eje
#para la herencia
from graph import Graph

#para que funcione la prueba
import pandas as pd
import requests


class Graphs(Graph):
    def __init__(self, master, df):
        super().__init__(master,extra=550)
        #usamos el constructor heredado de graph para convertirnos en un
        #frame con scroll


        #rellenamos el canvas
        #scatter plot e histograma de velocidad
        self.get_scatter(df['velocidad'],df['time'],0,0,'velocidad')
        self.get_hist(df['velocidad'],0,1,'velocidad')

        #scatter plot e histograma de aceleracion
        self.get_scatter(df['aceleracion'],df['time'],0,2,'aceleracion')
        self.get_hist(df['aceleracion'],0,3,'aceleracion')

        print (df.loc[df['velocidad']>50])

        #creamos un grafico de barras para ver el trafico dividido por vehiculos
        #para ello creamos la figura y el canvas y definimos los tipos de vehiculos
        frame1 = Frame(self.viewPort)
        f3 = plt.Figure( dpi = 100, figsize = (12,5))
        ax3 = f3.add_subplot(111)
        canvas3 = FigureCanvasTkAgg(f3, frame1)
        tipos={1:'motos', 2:'coches', 3:'furgonetas', 4:'autobuses', 5:'camiones'}
        #por cada tipo de vehiculo  creamos una barra
        for tipo in range (1,5):
            x = len(df.loc[df['tipo de vehiculo'] == tipo])
            #solo creamos barra para los que aparezcan
            if x != 0:
                ax3.bar(tipo, x, label = tipos[tipo])
        #creamos una ultima barra para los no identificados
        x = len(df.loc[df['tipo de vehiculo'] == 666.0])
        if x!=0:
            ax3.bar(tipo, x, label = 'no identificado')
        #colocamos el titulo, la leyenda y ocultamos los ejes
        ax3.set_title('tipos de vehiculos')
        ax3.xaxis.set_visible(False)
        ax3.legend(loc = 'upper left')

        #añadimos la barra de tareas
        toolbarFrame = Frame(frame1)
        toolbarFrame.grid(row = 1, column = 0)
        toolbar = NavigationToolbar2Tk(canvas3, toolbarFrame)

        #colocamos el canvas y el frame dentro del scrollable
```

```
65          canvas3.get_tk_widget().grid(column = 0, row = 0)
66          frame1.grid(row = 4, column = 0)
67
68
69      def get_scatter(self, x, y, i, j, var):
70          #creamos un scatterplot con las variables que nos han pasado
71          #x , y-> listas que contiene los valores de x e y que vamos a graficar
72          #i , j ->columna y fila que ocupamos dentro del grid
73          #var-> nombre de la variable que estamos graficando
74
75
76          #creamos el frame donde vamos a colocar el grafico
77          frame = Frame(self.viewPort)
78
79          #creamos la figura y el canvas del grafico
80          f1 = plt.Figure(dpi = 100, figsize = (12,5))
81          ax1 = f1.add_subplot(111)
82          #ax1.xaxis.set_major_formatter(mdates.DateFormatter("%Y-%m"))
83          canvas1 = FigureCanvasTkAgg(f1, frame)
84
85          #creamos el scatterplot
86          ax1.scatter(y, x)
87          ax1.set_title('Scatter plot de '+var)
88
89          #añadimos la barra de tareas
90          toolbarFrame = Frame(frame)
91          toolbarFrame.grid(row=1,column=0)
92          toolbar = NavigationToolbar2Tk(canvas1, toolbarFrame)
93
94          #colocamos el canvas dentro del frame
95          canvas1.get_tk_widget().grid(column=0,row=0)
96          frame.grid(row = j, column = i)
97
98
99      def get_hist(self ,x, i, j, var):
100         #creamos un histograma con las variables que nos han pasado
101         #x -> lista que contiene los valores que vamos a graficar
102         #i , j ->columna y fila que ocupamos dentro del grid
103         #var-> nombre de la variable que estamos graficando
104
105         #creamos el frame donde vamos a almacenar el gráfico
106         frame = Frame(self.viewPort)
107
108         #creamos el canvas  y la figura del grafico
109         f2 = plt.Figure( dpi = 100,figsize = (12,5))
110         ax2 = f2.add_subplot(111)
111         canvas2 = FigureCanvasTkAgg(f2, frame)
112
113         #creamos el histogramos
114         ax2.hist(x)
115         ax2.set_title('Histograma de '+var)
116
117         #creamos la barra de herramientas
118         toolbarFrame = Frame(frame)
119         toolbarFrame.grid(row = 1, column = 0)
120         toolbar = NavigationToolbar2Tk(canvas2, toolbarFrame)
121
122         #colocamos el canvas dento del grafico
123         canvas2.get_tk_widget().grid(column = 0, row = 0)
124         frame.grid(row = j, column = i)
125
126
127 class nb(Frame):
128     #para hacer una prueba
```

```
129     def __init__(self,master):
130         super().__init__(master)
131         #sacamos unos datos del sistema
132         df = self.get_data()
133 
134         #nos colocamos dentro del master
135         self.grid(column = 0, row = 0, sticky = 'NSWE')
136 
137         #creamos una pestaña
138         self.nOb = ttk.Notebook(master)
139         self.nOb.grid(column = 0, row = 0, sticky='NSWE')
140 
141         #creamos un objeto graphs y lo metemos dentro de la pestaña
142         self.g = Graphs(self.nOb,df)
143         #self.g.fill_canvas(df)
144         self.nOb.add(self.g, text = 'graph')
145 
146     def get_data(self):
147         #obtenemos unos datos del sistema
148         url='http://keltxo.no-ip.org:23040/'
149         #2020-9-13 13:19:55
150         ts1 = '2020-9-13 13:19:55'
151         ts2 ='2020-9-14 13:19:55'
152         sensor = 'AvCat1'
153         payload = {'TS1':ts1 , 'id':sensor, 'TS2':ts2}
154         r = requests.get(url, params = payload)
155         response = r.json()
156         print(response)
157         #los ponemos en formato dataframe
158         df = pd.DataFrame(response['data'], columns =
            ['event','time','velocidad','aceleracion','tipo de vehiculo'])
159         df['time']= pd.to_datetime(df['time'], format = "%Y-%m-%d""T""%X"".""%f")
160         print (df.loc[df['velocidad'] > 300])
161         return df
162 
163 
164 if __name__ == "__main__":
165     root = Tk()
166     root.columnconfigure(0, weight = 1)
167     root.rowconfigure(0, weight = 1)
168     t = nb(root)
169     root.mainloop()
170 
```

9.5.8 *Flux*

```
#recibe df y genera un frame scrollable que contienen los graficos interactuables
#flux corresponde a un solo sensor
from graph import Graph
from interactive_plot import InterPlot
import tkinter as tk
import requests
import pandas as pd
import datetime

class Flux(Graph):
    def __init__(self,master,df):
        #usamos el constructor heredado de graph para convertirlo en scrollable
        super().__init__(master)
        #vamos a empezar con una ventana fija de 8 horas
        self.window = 1

        self.df = df
        #necesitamos conocer el máximo y el minimo del tiempo para poder calcular la ventana
        self.ul = max(df['time'])
        self.start = min(df['time'])
        self.fill_canvas()



    def get_total(self,df):
        #obtener una ventana con el fujo total
        #calculamos el inicio y el final de la ventana inicial
        inicio = self.start
        sigue = True
        fin = self.start + datetime.timedelta(hours = self.window)
        y=[]
        x=[]

        #empezamos a sacar los datos
        while sigue == True:
            #contamos el numero de incidencias que abarca la ventan y lo almacenamos en y
            #x es el inicio de la ventana
            h = df.loc[(df['time']>inicio) & (df['time']<fin) ]
            y.append(len(h.index))
            x.append(inicio)

            #recalculamos el inicio y el fin de la ventana
            inicio = fin
            fin = inicio + datetime.timedelta(hours = self.window)

            #vemos si la siguiente ventana se pasa del final hemos termiando
            sigue = fin < self.ul

        #colocamos los datos en el formato correcto
        data={'y':{'total':y},
            'x':x}
        return data



    def get_tipos(self, df):
        #funciona pero es feisimo
        #obtenemos el fujo divido por tipo de vehiculos

        #caculamos el inicio y final de la ventana e inicializamos las listas donde van los datos
        inicio= self.start
```

```
62          sigue = True
63          fin = inicio+datetime.timedelta(hours = self.window)
64          y1=[]
65          y2=[]
66          y3=[]
67          y4=[]
68          y5=[]
69          y6=[]
70          x=[]
71          while sigue==True:
72              #saco los datos de cada tipo de vehiculo
73              h1 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 1)]
74              h2 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 2)]
75              h3 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 3)]
76              h4 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 4)]
77              h5 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 5)]
78              h6 = df.loc[(df['time']>inicio) & (df['time']<fin) & (df['tipo de
                vehiculo']== 666.0)]
79
80              #los cuantifico y los alamceno
81              y1.append(len(h1.index))
82              y2.append(len(h2.index))
83              y3.append(len(h3.index))
84              y4.append(len(h4.index))
85              y5.append(len(h5.index))
86              y6.append(len(h6.index))
87              x.append(inicio)
88
89              #recalculamos el inicio y el fin de la ventana
90              inicio = fin
91              fin = inicio+datetime.timedelta(hours=self.window)
92
93              #vemos si tenemos que seguir
94              sigue= fin<self.ul
95
96          data={'y':{'motos':y1, 'cohes':y2 ,'furgonetas':y3, 'autobuses':y4,
            'camiones':y5, 'no identificado': y6},
97              'x':x}
98          return data
99
100
101     def get_acc(self,df):
102         #obtenemos el flujo divido por aceleracion positiva o negativa
103         frenar = []
104         acelerar =[]
105         x = []
106         #calculo el inicio y el final de la ventana inicial
107         inicio =    min(df['time'])
108         sigue = True
109         fin = inicio + datetime.timedelta(hours = self.window)
110
111         while sigue == True:
112             #saco los datos
113             a=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['aceleracion']>0)]
114             f=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['aceleracion']<0)]
115
116             #los cuantifico y almaceno
117             acelerar.append(len(a.index))
118             frenar.append(len(f.index))
```

```
119             x.append(inicio)
120
121             #recalculo el inicio y final de la ventan y veo si tengo que seguir
122             inicio=fin
123             fin=inicio+datetime.timedelta(hours=self.window)
124             sigue= fin<self.ul
125
126         #pongo los datos en el formato adecuado
127         data={'y':{'acelerando':acelerar,'frenando':frenar},
128             'x':x}
129         return data
130
131
132     def get_vel(self,df):
133         #obtengo el flujo divido en 5 grupos de velocidades
134         #calculo la franja de velocidades que van en cada grupo
135         max_vel=max(df['velocidad'])
136         min_vel=min(df['velocidad'])
137         step=(max_vel-min_vel)/5
138
139         #incializo la lsita donde van los datos y calculo el inicio y final de la ventana inicial
140         y1=[]
141         y2=[]
142         y3=[]
143         y4=[]
144         y5=[]
145         x=[]
146         inicio= self.start
147         sigue=True
148         fin=inicio+datetime.timedelta(hours=self.window)
149
150         while sigue == True: #por cada ventana
151             #obtengo los datos
152             h1=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['velocidad']<min_vel+step)]
153             h2=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['velocidad']<min_vel+(2*step))]
154             h3=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['velocidad']<min_vel+(3*step))]
155             h4=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['velocidad']<min_vel+(4*step))]
156             h5=df.loc[(df['time']>inicio) & (df['time']<fin) & (df['velocidad']<min_vel+(5*step))]
157
158             #los cuantifico y almaceno
159             y1.append(len(h1.index))
160             y2.append(len(h2.index))
161             y3.append(len(h3.index))
162             y4.append(len(h4.index))
163             y5.append(len(h5.index))
164             x.append(inicio)
165
166             # recalculo el inicio y final de la ventana y veo si tengo que seguir
167             inicio=fin
168             fin=inicio+datetime.timedelta(hours=self.window)
169             sigue= fin<self.ul
170
171         #almaceno los datos en el formato adecuado
172         data={'y':{
173             'por debajo de'+str(round(min_vel+step))+'km/h':y1,
174             'por debajo de'+str(round(min_vel+(2*step)))+'km/h':y2,
175             'por debajo de'+str(round(min_vel+(3*step)))+'km/h':y3,
176             'por debajo de'+str(round(min_vel+(4*step)))+'km/h':y4,
```

9.5.9 *Interactive plot*

```
import tkinter as tk
from tkinter import ttk
#from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure

from matplotlib.backends.backend_tkagg import (
    FigureCanvasTkAgg, NavigationToolbar2Tk)
import numpy as np
#interactive plot, es un grafico que permite hacer visibles e invisibles a los plots que contiene
#clicando en la leyenda
#referencias que he usado para guiarme
#https://matplotlib.org/gallery/event_handling/legend_picking.html
#https://stackoverflow.com/questions/46727512/matplotlib-tkinter-event-picking-on-the-legend
class InterPlot(tk.Frame):
    def __init__(self, master, data, row):
        super().__init__(master)
        #nos colocamos dentro de master
        self.grid(row = row, column = 0, sticky = 'NSWE')

        #lista de posibles colores
        color = ['b','r','g','c','k','m','y']

        #creo la figura y el canvas donde va el plot
        f = Figure(figsize = (12,5), dpi=100, )
        self.ax = f.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(f, master = self)
        self.canvas.get_tk_widget().grid(row = 0, column = 0)
        self.canvas.mpl_connect('pick_event', self.onpick)

        #empiezo a dibujar los graficos de lineas
        i = 0 #contador
        lines = [] #lista donde almacenamos las lineas
        for label in data['y'].keys():
            #creo un grafico por cada
            #la coma es necesaria para deshacer el tupple
            line, = self.ax.plot(data['x'], data['y'][label], lw = 2, color = color[i], label = label)
            lines.append(line)
            #por cada linea que queremos dibujar sacaremos un color de la lista,
            #si llegamos al final de la lista volvemos a empezar
            i = i + 1
            if i>len(color):
                i = 0

        #modificamos los parametros de la leyenda
        leg = self.ax.legend(loc = 'upper left', fancybox = True, shadow = True)
        leg.get_frame().set_alpha(0.4)

        #añado la barra de herramientas
        toolbarFrame = tk.Frame(self)
        toolbarFrame.grid(row = 2, column = 0)
        toolbar = NavigationToolbar2Tk(self.canvas, toolbarFrame)
        self.canvas.get_tk_widget().grid(row = 0, column = 0)

        #legline-> linea en la leyenda
        #origline->linea en el grafico

        #hacemos que las lineas de la leyenda sean clicables
        #y alamacenamos en un diccionario -lined- que origline corresponde a que legline
        self.lined = dict()

```

```
61
62          for legline, origline in zip(leg.get_lines(), lines):
63              #legline.set_picker(5)  # 5 pts tolerance#metodo desfasado
64              #usar mejor este
65              legline.set_picker(True)
66              self.lined[legline] = origline
67
68      def onpick(self, event):
69          #vemos sobre que lineas que ha clicado
70          legline = event.artist
71          origline = self.lined[legline]
72
73          #modifcamos la visibilidad
74          vis = not origline.get_visible()
75          origline.set_visible(vis)
76
77          # Cambiamos la opcacidad en la leyenda
78          if vis:
79              legline.set_alpha(1.0)
80          else:
81              legline.set_alpha(0.2)
82          self.canvas.draw()
83
84
85
86  if __name__ == "__main__":
87      t = np.arange(0.0, 0.2, 0.1)
88      y1 = 2*np.sin(2*np.pi*t)
89      y2 = 4*np.sin(2*np.pi*2*t)
90      y3 = 3*np.sin(2*np.pi*2*t)
91      y4 = 5*np.sin(2*np.pi*2*t)
92      y5 = 6*np.sin(2*np.pi*2*t)
93      data={'y':{'1 HZ':y1,'2 HZ':y2,'amp3':y3,'amp5':y4,'amp6':y5},
94          'x':t,
95          }
96      root=tk.Tk()
97      nb=ttk.Notebook(root)
98      nb.grid(row = 0, column = 0, sticky = 'NSWE')
99
100     i = InterPlot(nb, data, row = 0)
101     nb.add(i, text='gráfico interactivo')
102     root.mainloop()
103
```

9.6 Nuevos Sensores

9.6.1 *Gen_Control*

```
from gen_map import Gen_map
from checkbar import CheckBar
import tkinter as tk
from tkinter import ttk
import json
import requests

class Gen_Control(tk.Frame):
    def __init__(self, master, filename):
        super().__init__(master)
        #nos colocamos dentro de master
        self.grid(row = 0, column = 0, sticky ='NSWE')

        #almacenamos el nombre del archivo
        self.filename = filename

        #Colocamos el mapa
        self.map = Gen_map(self)

        #Para introducir el nombre
        self.lab1 = ttk.Label(self, text = 'Nombre del sensor*: ')
        self.lab1.grid(column = 0, row = 1, sticky ='NSWE')
        self.nombre = tk.Entry (self)
        self.nombre.grid (column = 1, row = 1, sticky ='NSWE')

        #Para indicar la maxima velocidad
        self.lab2 = ttk.Label(self, text = 'Límite de velocidad: ')
        self.lab2.grid(column = 0, row = 2, sticky ='NSWE')
        self.vel = tk.Entry (self)
        self.vel.grid (column = 1, row = 2, sticky ='NSWE')

        #vehiculos no permitidos
        self.lab2 = ttk.Label(self, text = 'Vehiculos no permitidos: ')
        self.lab2.grid(column = 0, row = 3, sticky ='NSWE')
        self.types = CheckBar(self, ['Moto', 'Coche', 'Furgoneta', 'Autobús',' ↵
        Camion'] )
        self.types.grid (column = 1, row = 3, sticky ='NSWE')

        #nota acerca de nombre
        #self.lab1 = ttk.Label(self, text = '* no debe contener espacios en blanco')
        #self.lab1.grid(column = 20, row = 20,sticky ='W')

        #botón para crear el nuevo sensor
        self.b = tk.Button (self, text = 'Crear nuevo sensor', command = self.update)
        self.b.grid (column = 19, row = 21, sticky = 'W')

    def update (self):
        #comprobamos que todo esté correcto
        try:
            if ' ' in self.nombre.get():
                #no puede haber espacios en blanco en el nombre
                raise Exception(0)
            else:
                sensor = self.nombre.get()
            vel = int( self.vel.get() ) #velocidad debe ser convertible en float
            estados = list ( self.types.state() )
            typos = ['Moto', 'Coche', 'Furgoneta', 'Autobús',' Camion']
            vehiculos = []
            for i in range (0,5):
                if estados[i] == 1:
                    vehiculos.append( i + 1 )
            err = False
        except Exception as e:
            if e.args[0] == 0:
```

```
64                  message = 'El nombre del sensor no puede contener espacios en
                    blanco.'
65              else:
66                  message = 'El limite de velocidad debe ser un número (se ignoran
                    los decimales)'
67              tk.messagebox.showerror(title = 'Error', message = message)
68              err = True
69          #si no se ha producido ningún error
70          if err == False:
71              #preguntamos a ver si todo es correcto
72              message = '¿Está seguro de que quiere crear un nuevo sensor con estos
                parametros?'
73              MsgBox = tk.messagebox.askquestion ('Comprovación', message)
74              if MsgBox == 'yes':
75                  print('voy a send')
76                  self.send_data( sensor, vel, vehiculos )
77
78              else:
79                  tk.messagebox.showinfo(message = 'Se ha cancelado la operación')
80
81
82      def send_data(self, sensor, vel , vehiculos):
83          print('estoy en send')
84          try:
85              #obtenemos los datos de la conexion
86              with open(self.filename) as f:
87                  print('estoy en filename')
88                  settings = json.load(f)
89                  url = settings['new']['url_new']
90                  print(url)
91                  port = settings ['new'] ['port_new']
92                  subdir = settings ['new'] ['sub']
93
94              #realizamos la petición
95              url = url + ':'+str(port)+ subdir
96              print(url)
97              if not vehiculos:
98                  print('vacio')
99                  vehiculos = 'NA'
100             payload  = {'sensor' : sensor, 'velocidad': int(vel), 'vehiculos':
                vehiculos, 'x': int(self.map.x ), 'y':int(self.map.y)}
101             print(payload)
102             r = requests.post(url, data = payload)
103             print('he hecho requests')
104             response = r.json()
105             print(response)
106
107             #comprobamos la respuesta
108             if response['err'] == True:
109                 message = 'Se ha producido un error en el servidor al intentar
                    crear el nuevo sensor.'
110                 tk.messagebox.showerror(title = 'Error', message = message)
111             else:
112                 tk.messagebox.showinfo(title = 'Listo',message = 'Se ha creado un
                    nuevo sensor con los parametros indicados. Para ver esta
                    modificación es necesario')
113         except Exception as e:
114             print('estoy en exception')
115             print(e)
116             #si no podemos contactar con url o hay un problema con filename
                mostramos un mensaje
117             message = 'Se ha producido un error al realizar la petición. Compruebe
                la configuración y la conexión a internet.'
118             tk.messagebox.showerror(title = 'Error', message = message)
119
120
121 if __name__ == "__main__":
122
123     #defino root
124     def quit_me():
125         #print('quit')
126         root.quit()
127         root.destroy()
128     root = tk.Tk()
129     root.protocol("WM_DELETE_WINDOW", quit_me)
130     root.columnconfigure(0, weight = 1)
131     root.rowconfigure(0, weight = 1)
132
133     #creo el mapa
134     g = Gen_Control(root, 'settings.json')
135
136     #activo el bucle de root
137     root.mainloop()
138
```

9.6.2 *Gen_Map*

```
#ref:https://stackoverflow.com/questions/40325321/python-embed-a-matplotlib-plot-with-slider-in-tkinter-properly
import random
import matplotlib
import tkinter as tk
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import weakref
class Gen_map (tk.Frame):
    def __init__ (self, master):
        super().__init__(master)
        #me coloco dentro de master
        self.grid (row = 0, column = 0, sticky='NSWE', columnspan = 20)

        #genero la posición inicial del punto
        self.y = [500]
        self.x = [1000]

        #creo la figura y el canvas donde va el mapa
        f = plt.Figure(figsize = (8,4), dpi = 100, )
        self.canvas = FigureCanvasTkAgg(f, master = self)
        self.canvas.get_tk_widget().grid(row = 0, column = 0, sticky = 'NSWE')
        self.rowconfigure (0, weight=1)
        #creo el axes donde voy a trabajar y coloco el titulo
        self.ax = f.add_subplot(111)


        #creo los sliders y les asigno la gestión de eventos
        self.s_x = Slider(self.ax, '', 0, 1880, valinit = 1000)
        self.s_y = Slider(self.ax, '', 0, 850, valinit = 500, orientation = 'vertical')
        self.s_x.on_changed(self.update)
        self.s_y.on_changed(self.update)

        #oculto los sliders
        self.ax.cla()

        #coloco el punto y la imagen de fondo
        self.basic()

    def update(self, val):
        #actualizo los valores de x e y
        self.x = self.s_x.val
        self.y = self.s_y.val

        self.ax.cla()#oculto el punto anterior (no lo elimina, poco eficiente)

        #coloco el punto y la imagen de fondo
        self.basic()


    def basic (self):
        #coloco el punto
        self.point = self.ax.scatter(self.x, self.y, c = 'red',s = 70, edgecolors = 'white')

        #pongo la imagen de fondo
        img = plt.imread("pamplona1.png")
        self.ax.imshow(img)

        #pongo el titulo
        self.ax.set_title('Arrastra el punto hasta donde deseas colocar el nuevo sensor')
```

```
61
62
63
64  if __name__ == "__main__":
65
66      #defino root
67      def quit_me():
68          #print('quit')
69          root.quit()
70          root.destroy()
71      root = tk.Tk()
72      root.protocol("WM_DELETE_WINDOW", quit_me)
73      root.columnconfigure(0, weight = 1)
74      root.rowconfigure(0, weight = 1)
75
76      #creo el mapa
77      g = Gen_map(root)
78
79      #activo el bucle de root
80      root.mainloop()
81
```