Universidad de Alcalá
Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Aplicación web para la monitorización continua del nivel de congestión del tráfico

Autor: Alejandro Véliz Fernández

Tutor/es: Roberto Javier López-Sastre

2016

Universidad de Alcalá

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

Aplicación web para la monitorización continua del nivel de congestión del tráfico

Autor: Alejandro Véliz Fernández

Director: Roberto Javier López-Sastre

TRIBUNAL:

Presidente: *D. Pedro Gil Jiménez*

Vocal 1º: *D. Donato Rodríguez Alonso*

Vocal 2º: *D. Roberto Javier López-Sastre*

FECHA:.....

Aplicación web para la monitorización continua del nivel de congestión del tráfico

Alejandro Véliz Fernández

10 de mayo de 2016

Si lo oigo, lo olvido; si lo veo, lo recuerdo; si lo hago, lo sé.

Proverbio chino

Agradecimientos

A mi tutor Roberto por todo su esfuerzo y dedicación, por toda la paciencia y el compromiso que me ha brindado. Al grupo de investigación GRAM por acogerme como uno más y darme la posibilidad de aprender todo lo que me han aportado durante este tiempo.

Y en especial a mi familia quienes siempre me han apoyado en los buenos y malos momentos, sin ellos nada de esto sería posible.

Índice general

Lista de figuras

Lista de tablas

Resumen

En la actualidad existen diferentes plataformas que muestran el estado del tráfico en las carreteras, aunque ninguna de ellas ofrece la posibilidad de conocer el grado de ocupación de las carreteras mediante la información obtenida directamente desde las cámaras de videovigilancia desplegadas por la Dirección General de Tráfico (DGT).

Este trabajo se plantea como objetivo, crear una aplicación web que posibilite realizar una monitorización continua y precisa del nivel de congestión del tráfico de vehículos en tiempo real. Esto se realiza empleando técnicas de visión artificial, que trabajan directamente con las imágenes que proporcionan las cámaras de la DGT. La aplicación permite visualizar de forma dinámica, sobre un mapa donde han sido geolocalizadas las cámaras, el número medio de vehículos que circulan por las vías, así como una predicción.

Palabras clave: Tráfico, Pyton, Web, Aplicación.

Abstract

Nowadays there exist several platforms able to show the traffic state of the roads, but none of them has the capability of showing this information directly using the images taken from the video surveillance cameras of the DGT.

This work therefore seeks to create a web application that lets the user check, in an accurate and continuous way, the traffic congestion level, using computer vision techniques, working directly with the images provided by the DGT cameras. The application also allows the user to dynamically visualize over a map where the cameras have been geolocated, the average number of vehicles over a road, as well as a prediction of the level of congestion.

Keywords: Traffic, Python, Web, Application.

Resumen Extendido

Hasta ahora, el método empleado por la DGT y otros organismos, para realizar el conteo de vehículos y poder conocer así el grado de congestión de una vía determinada, ha consistido en la utilización de espiras electromagnéticas como se muestra en la Figura 1. Cabe destacar que esta solución supone un elevado coste tanto de instalación, debido a su disposición bajo el asfalto, como de mantenimiento, dado el desgaste por el tránsito de vehículos y condiciones meteorológicas al que se ven sometidas estas espiras.

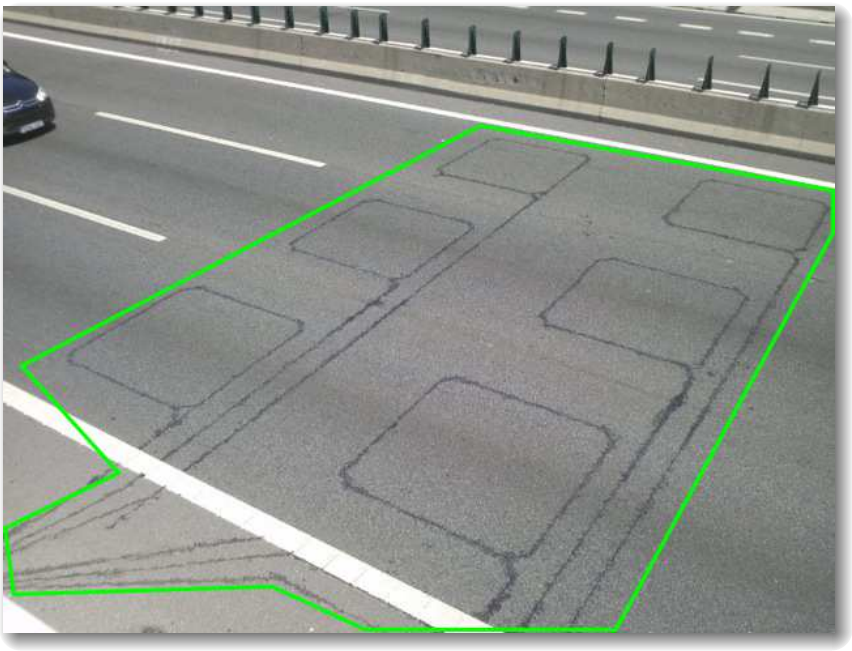

Figura 1: Espiras electromagnéticas.

Con el propósito de solventar algunos de estos problemas surge este proyecto, en el que se ofrece una plataforma online capaz de mostrar el grado de congestión de vehículos por medio de resultados obtenidos a partir de imágenes ofrecidas por la red de cámaras de videovigilancia desplegadas por la DGT.

De este modo se pretende aprovechar la infraestructura actual de cámaras desplegadas para tener una visión más específica del estado de las carreteras, reduciendo el coste asociado a esta tarea y mejorando otros aspectos teniendo en cuenta el modelo de presentación de resultados y de obtención de los mismos.

Se trata, por tanto, de implementar una solución de estimación del número de vehículos que permita prescindir de las espiras, y que obtenga toda la información a partir de las imágenes que proporcionan las cámaras.

En concreto, este proyecto consiste en la creación de una aplicación web que permite al usuario activar y desactivar las cámaras geolocalizadas de la DGT. Como se puede observar en la Figura 2, cada cámara activa es analizada por un módulo de visión artificial que recibe la imagen y es capaz de estimar el número de vehículos presentes. Esta información es volcada en una base de datos, que es utilizada a su vez por la aplicación web para presentar la información al usuario y para realizar predicciones asociadas.

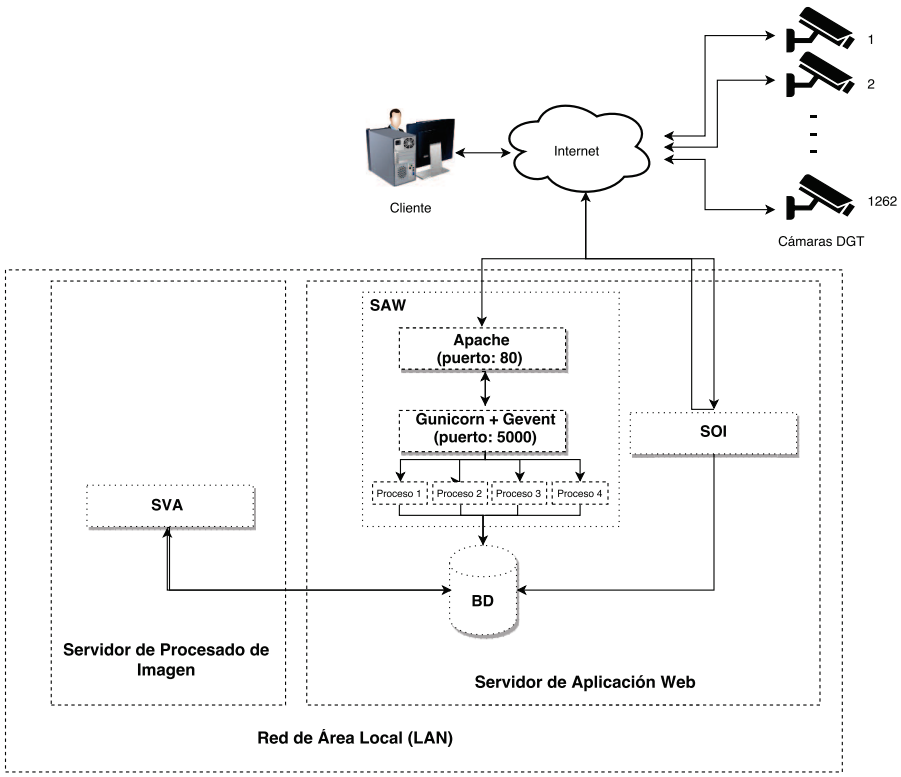

Figura 2: Arquitectura del sistema propuesto desde un punto de vista detallado.

Cabe destacar el método de estimación, ya que se emplea un módulo de visión artificial, el cual realiza un procesado de la imagen obtenida de la cámara mediante Deep Learning [25]. De esta forma se consiguen resultados que difícilmente pueden ser obtenidos con técnicas tradicionales de visión por computador. Estos resultados son alcanzados gracias a la integración de la programación orientada a la GPU (CUDA [7]) y a la existencia de grandes bases de datos anotadas[1]. La Figura 3 se puede observar como realiza el módulo de visión artificial la estimación [1].

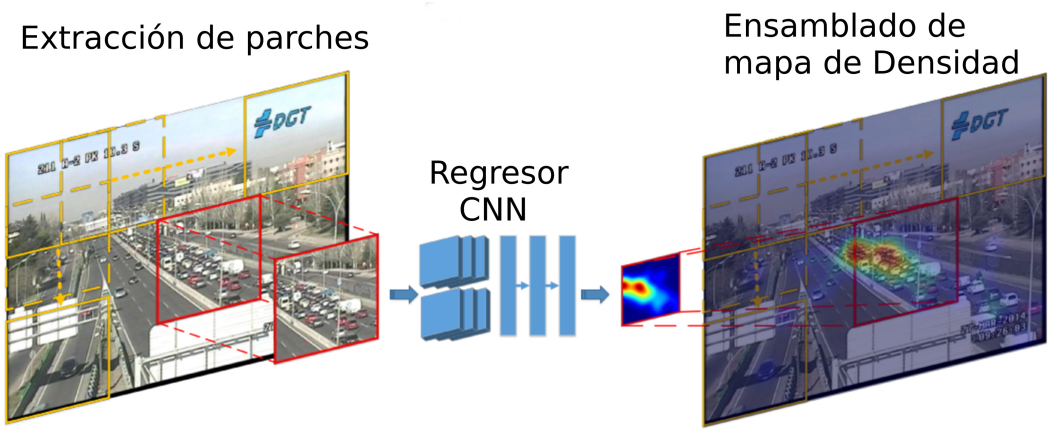

Figura 3: Módulo de visión artificial.

A la izquierda de la figura se encuentra la imagen de entrada en el módulo. En primer lugar se realiza una extracción de parches descomponiendo la imagen en un conjunto de imágenes más pequeñas. Cada uno de los parches de la imagen se utiliza para alimentar un regresor [26] encargado de realizar la estimación del mapa de densidad que crea a partir de cada parche de entrada. Por último se vuelven a ensamblar los parches construyendo el mapa de densidad final del mismo tamaño que la imagen de entrada.

La Figura 4 muestra una imagen de la aplicación web donde se pueden identificar múltiples elementos.

[1]Bases de datos compuestas por imágenes que previamente han sido anotadas por personas, identificando los elementos necesarios para un sistema de detección

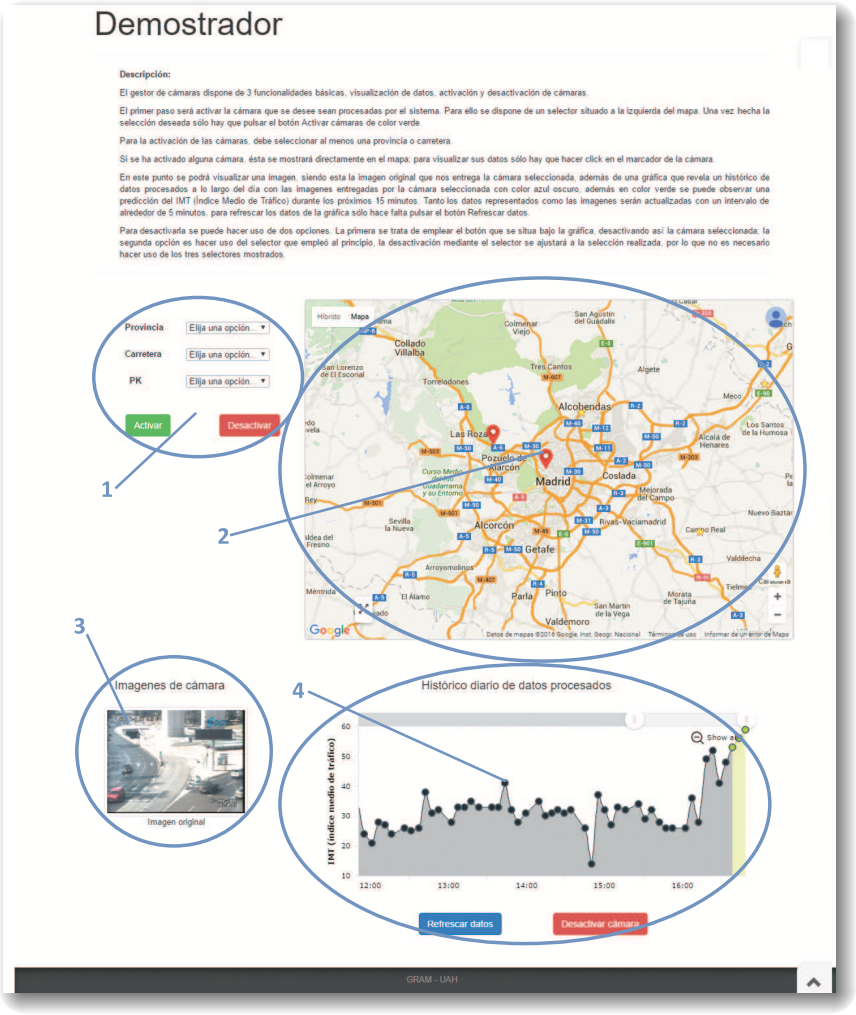

Figura 4: Interfaz principal de la aplicación web.

1. **Gestión de cámaras activas en el sistema**: Se trata de un conjunto de menús desplegables que posibilitan realizar la selección de una cámara la cual se desea activar para conocer el estado del tráfico en el punto que se desee o desactivarla si ya no es de nuestro interés. La selección se realiza mediante tres parámetros que permiten acotar la provincia, carretera y punto kilométrico exacto en el que se encuentra localizada la cámara.

2. **Geolocalización de cámaras activas**: El mapa muestra un marcador geolocalizando el punto exacto en el que se encuentran las cámaras activas en el sistema. Para visualizar los datos correspondientes a alguna cámara, sólo es necesario hacer clic sobre el marcador que geolocaliza la cámara que se desee situado en el mapa.

3. **Visualización de imagen obtenida por la cámara**: Muestra la última imagen obtenida por la cámara de la DGT, la cual es procesada para realizar la estimación del tráfico.

4. **Histórico diario de datos procesados**: Gráfica que muestra el valor del Indice Medio de Tráfico (IMT) obtenidos a lo largo del día, en color azul oscuro, por el sistema en la cámara seleccionada, así como la predicción del IMT durante los 15 minutos posteriores al instante de la última estimación realizada por el SVA, en color verde claro. Permite, además, realizar un zoom con la barra superior acotando el rango horario del cual se desea ver los resultados, de esta forma permite ver los resultados con mayor comodidad.

El histórico de datos recopilados y la geolocalización de las cámaras, da lugar a la generación detallada de informes sobre el grado de ocupación de las vías, permitiendo mejorar las tareas de gestión de las carreteras, e incluso mostrando los puntos en los que la necesidad de inversión en nuevas infraestructuras puede ser prioritario. Estas son dos utilidades claras de la aplicación desarrollada.

Además, el modo de presentación de los datos recogidos por el sistema permite acercar de una forma simple e inmediata los resultados a los usuarios de la web de la DGT, permitiendo emplear esta información para planificar sus trayectos, de modo que presenten el menor nivel de congestión.

Glosario

TFG Trabajo Fin de Grado

DGT Dirección General de Tráfico

IMT Indice Medio de Tráfico

BD Base de Datos

SOI Sistema de Obtención de Imágenes

SVA Sistema de Visión Artificial

SAW Servicio de Aplicación Web

WGSI Web Server Gateway Interface

PK Punto Kilométrico

SSE Server Sent Events

Capítulo 1

Introducción

El Trabajo Fin de Grado (TFG) "Aplicación web para la monitorización continua del nivel de congestión del tráfico", nace ante la necesidad de ofrecer una plataforma online en la que mostrar una monitorización continua y precisa del nivel de congestión del tráfico de vehículos en tiempo real, obtenido a partir de la red de cámaras de videovigilancia desplegadas por la DGT, mediante técnicas de visión por computador. La Figura 1.1 muestra una imagen de la aplicación web desarrollada.

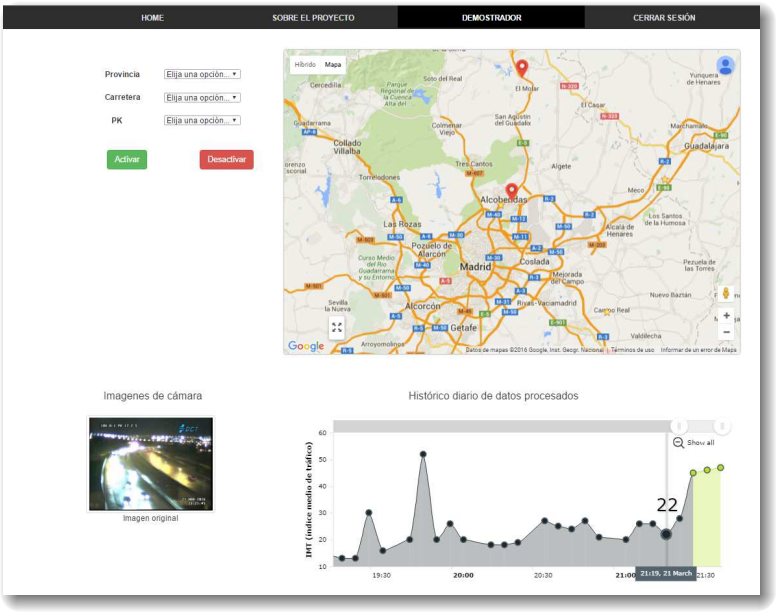

Figura 1.1: Interfaz principal de la aplicación web.

En este capítulo se abordan tres puntos que dan sentido al trabajo realizado, siendo estos: (a) la motivación que lleva a realizar este proyecto, (b) las posibles aplicaciones a las que da lugar el trabajo y (c) el estado actual de las aplicaciones y plataformas de las que se parte como base.

1.1. Motivación

En la actualidad existen aplicaciones móviles encargadas de mostrar el estado del tráfico. El problema que presentan este tipo de aplicaciones es que la información que obtienen para realizar esta tarea, en algunos casos, requiere de la interacción del usuario (Waze [39]), siendo él quien informa sobre el estado del tráfico. Este modelo de uso es peligroso ya que en ocasiones es el propio conductor del vehículo quien informa acerca de la congestión del tráfico en la aplicación, desviando así su atención de la carretera. También hay otros modelos que no necesitan de la intervención del usuario, no siendo así peligroso su uso; el problema que plantean es que rompen la privacidad de los usuarios, pues los dispositivos comparten de forma automática su localización con compañías encargadas de procesar los datos y de determinar el grado de ocupación de las vías (TomTom [36], Sygic [35], etc).

Para solucionar dicho problema, este proyecto se plantea un modelo alternativo que consiste en utilizar técnicas de visión artificial. De este modo se consigue evitar la interacción con la aplicación sin violar la privacidad del usuario, pues el sistema no necesita conocer su posición para proporcionar dicho servicio. Esto se consigue gracias al uso de técnicas de Deep Learning [25], sobre las imágenes que recogen las cámaras de videovigilancia desplegadas por la DGT, para realizar una monitorización continua y precisa del nivel de congestión del tráfico de vehículos en tiempo real, siendo capaz de informar acerca del estado del tráfico. Empleando este enfoque, como se puede ver en la Figura 1.2, se consigue un amplio espectro de fuentes de información ya que se puede acceder a 1262 cámaras desplegadas por la DGT, actualmente.

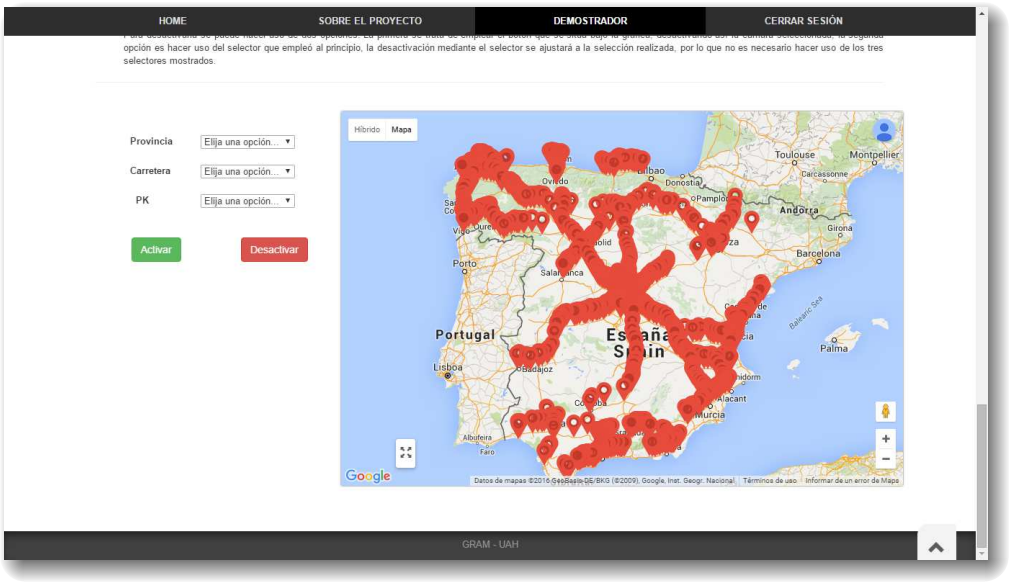

Figura 1.2: Rango de cámaras disponibles en el sistema.

1.2. Campos de aplicación

El método empleado por la DGT y otros organismos, para realizar el conteo de vehículos y poder conocer así el grado de congestión de una vía determinada, consiste en la utilización de espiras electromagnéticas como se muestra en la Figura 1.3. Cabe destacar que esta solución supone un elevado coste tanto de instalación, debido a su disposición bajo el asfalto, como de mantenimiento, dado el desgaste por el tránsito de vehículos y condiciones meteorológicas al que se ven sometidas estas espiras.

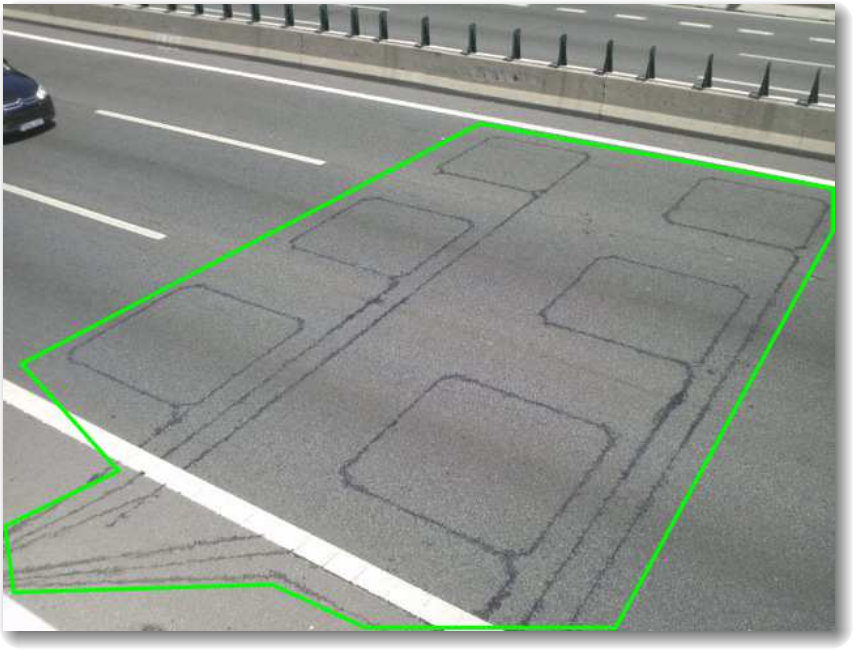

Figura 1.3: Espiras electromagnéticas.

Gracias a la aplicación web con una interfaz gráfica, un posible campo de aplicación podría ser mostrar y contabilizar el flujo de vehículos, sustituyendo así el actual sistema formado por espiras electromagnéticas. De este modo se podrían reducir ampliamente los costes de gestión y mantenimiento de las mismas. Esto podría ser posible gracias a las técnicas de visión artificial empleadas para procesar las imágenes recogidas por las cámaras de la DGT, así es posible conocer el flujo de vehículos que atraviesan cada tramo de carretera en la que la DGT disponga de cámaras instaladas. Se puede decir además que el sistema es escalable, pues se puede ampliar la capacidad del sistema únicamente dando de alta nuevas cámaras en la base de datos que recoge las cámaras disponibles.

Además, el procesado en tiempo real realizado por el módulo de visión artificial y la predicción que el sistema realiza en base al histórico de datos procesados, permiten acercar estos resultados de una forma más simple e intuitiva a los usuarios. Esto permite a dichos usuarios conocer el estado del tráfico en tiempo real e incluso permite utilizar esta información para sus sistemas de navegación.

Otro gran campo de aplicación consiste en que esta tecnología posibilita la generación y consulta de una gran cantidad de datos históricos sobre los niveles de congestión de las vías monitorizadas. Ello daría lugar a la generación de informes detallados sobre los grados de ocupación de las vías, pudiéndose analizar con la resolución temporal que se desee, de modo que se puedan mejorar las tareas de gestión de las carreteras, o planificar dónde se necesita invertir en nuevas infraestructuras.

1.3. Estado del arte

En el marco actual existen múltiples plataformas enfocadas al tráfico y navegación, en diferentes formatos. En al ámbito móvil, cabe remarcar dos aplicaciones que gozan de gran popularidad:

- **Waze** [39]: Trata el método de navegación desde un enfoque de red social, de este modo permite a los usuarios conseguir puntos en función de los kilómetros recorridos y al emitir alertas sobre el estado del tráfico (puntos donde se encuentra la policía, accidentes, zonas de peligro, errores en el mapa, lugares o cámaras).

  Esta plataforma permite, por tanto, conocer el estado de las carreteras mediante interacción de los usuarios, quienes emiten las alertas acerca del estado del tráfico. La figura 1.4 muestra una captura de la aplicación.

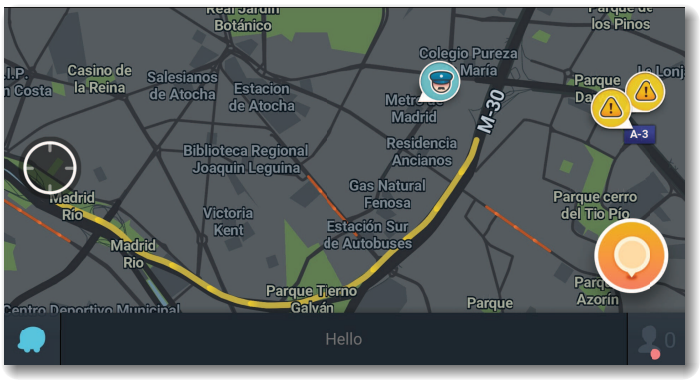

Figura 1.4: Aplicación Waze.

- **Google Transit** [15]: El conocido servicio de mapas de Google, desde Noviembre del 2013 también es capaz de mostrar una capa con la información acerca del estado del tráfico en las carreteras. Los datos que emplean para mostrar esta información los obtienen mediante múltiples fuentes, ya que existe un programa de socios de Google Transit que permite a toda agencia pública encargada de supervisar transportes públicos en su ciudad, incluir sus datos [17]. Una de las fuentes que emplea se trata de Waze la cual, actualmente, es propiedad de Google [16]. La figura 1.5 muestra una captura de la aplicación.

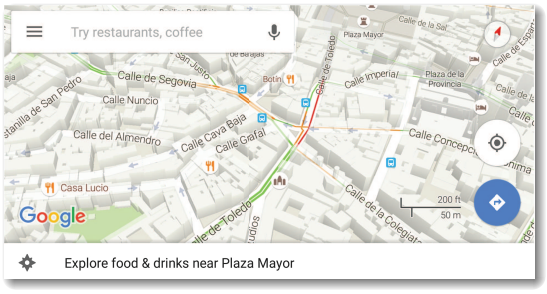

Figura 1.5: Aplicación Google Maps.

Además, se pueden encontrar aplicaciones web dedicadas a informar acerca de obras, accidentes, alertas, eventos, previsiones, estado del tráfico, etc. Entre las que se pueden destacar, en el ámbito nacional, encontramos las siguientes:

- **DGT** [10]: La página web oficial de la Dirección General de Tráfico contiene un mapa que ofrece información sobre las imágenes recogidas por las cámaras desplegadas por ellos. Esta información recoge incidencias, entre las que se puede obtener información del tráfico, localización de radares fijos e información meteorológica. La figura 1.6 muestra una captura de la aplicación.

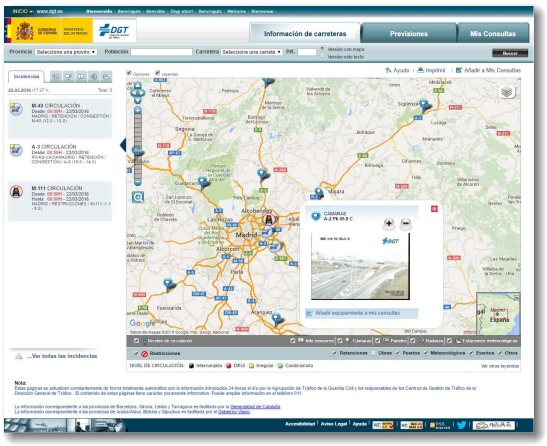

Figura 1.6: Mapa del tráfico de la DGT.

- **Informo Madrid** [8]: Para consultar la información relacionada con el tráfico de Madrid, se pone a disposición la web *Informomadrid*, la cual se trata de otra aplicación web en la que se muestra un mapa con la información de incidentes, imágenes obtenidas por las cámaras, radares de semáforo, densidad de vehículos por hora y grado de ocupación de las vías. La figura 1.7 muestra una captura de la aplicación.

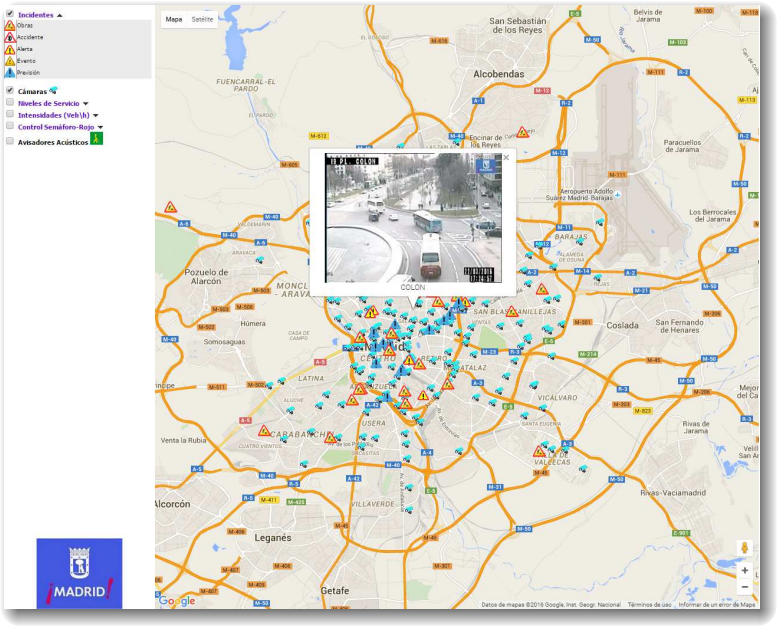

Figura 1.7: Mapa del tráfico de la Comunidad de Madrid.

Cabe destacar que ninguna de las plataformas citadas, muestra el nivel de congestión del tráfico a partir de técnicas de visión por computador, aplicadas a las imágenes obtenidas de las cámaras implantadas, del mismo modo que tampoco muestran una estimación del número de vehículos, ni un histórico de datos obtenidos en el procesado. Este proyecto, permite crear un sistema automatizado de coste reducido, ya que aprovecha las infraestructuras ya desplegadas, capaz de estimar el número de vehículos localizados en un tramo de vía visible en las cámaras soportadas por la DGT, permitiendo conocer el grado de congestión del tráfico y prediciendo el estado del tráfico, gracias al histórico de datos almacenados, el cual también se pone a disposición del usuario para que pueda conocer no solo el estado actual y futuro, sino también el progreso del nivel de congestión al que se ha visto sometido cierto tramo.

Capítulo 2

Diseño de la arquitectura web

Este capítulo se plantea con el objetivo de describir la arquitectura basada en servicios web donde se integran los sistemas de estimación y predicción del número de vehículos para mostrar un uso real del sistema construido.

Para realizar esta tarea se va a adoptar un modelo de vista descendente, centrándose, en primer lugar, en una descripción general de la arquitectura para, a continuación, describir cada una de las partes que componen el sistema de forma más detallada.

2.1. Descripción general de la arquitectura

Para realizar este proyecto se ha empleado un patrón de arquitectura en 4 niveles [4], como se puede observar en la figura 2.1, descentralizando así la aplicación a nivel del servidor, es decir, cada servidor se centra en una tarea específica. Esta arquitectura dota al sistema de:

- Mayor seguridad, dado que se definen las políticas de seguridad para cada servicio y nivel de forma específica.

- Mayor flexibilidad debido a la descentralización de los servicios implicados en el sistema.

- Mejor rendimiento dado que la carga necesaria para poner en marcha el sistema se reparte.

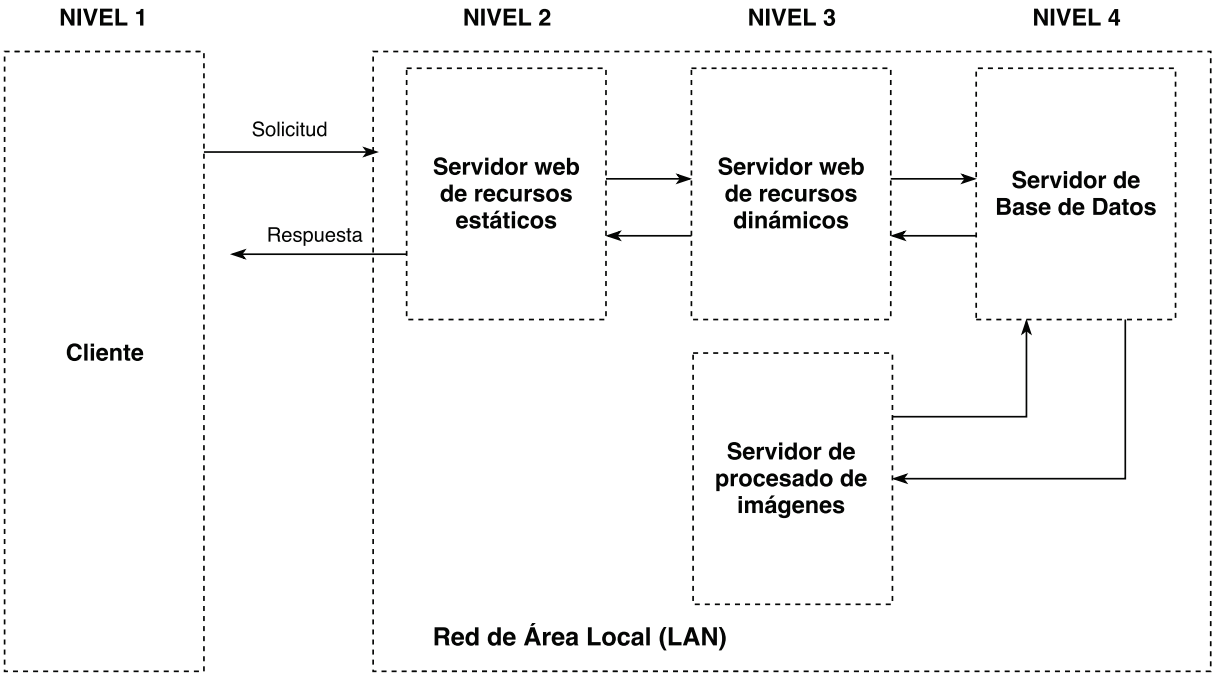

Figura 2.1: Modelo de Arquitectura para el sistema propuesto.

En la Figura 2.2 se muestra el sistema propuesto que se compone de un clúster [42] formado por 2 servidores con Ubuntu 12.04 [37] como sistema operativo. Uno de ellos alojará por un lado una base de datos encargada de almacenar tanto las imágenes obtenidas directamente desde las cámaras de la DGT, como el registro de las cámaras que se vayan a manejar para el proyecto, incluyendo el estado y localización de dichas cámaras. Por otro lado, este mismo servidor, será el encargado de alojar una aplicación web destinada al control del servicio, de tal modo que sólo será necesario un acceso de tipo *Front-end* para controlar el sistema.

Además, en esta arquitectura nos encontramos un segundo servidor en el que se integra el sistema de conteo de vehículos basado en técnicas de visión artificial, implementado con Deep learning y técnicas de cuenta mediante regresión [26]. Este servidor es el encargado de procesar las imágenes que obtiene desde la base de datos alojada en el primer servidor, y de actualizarla con el índice medio de tráfico (IMT) y la imagen resultante del procesado.

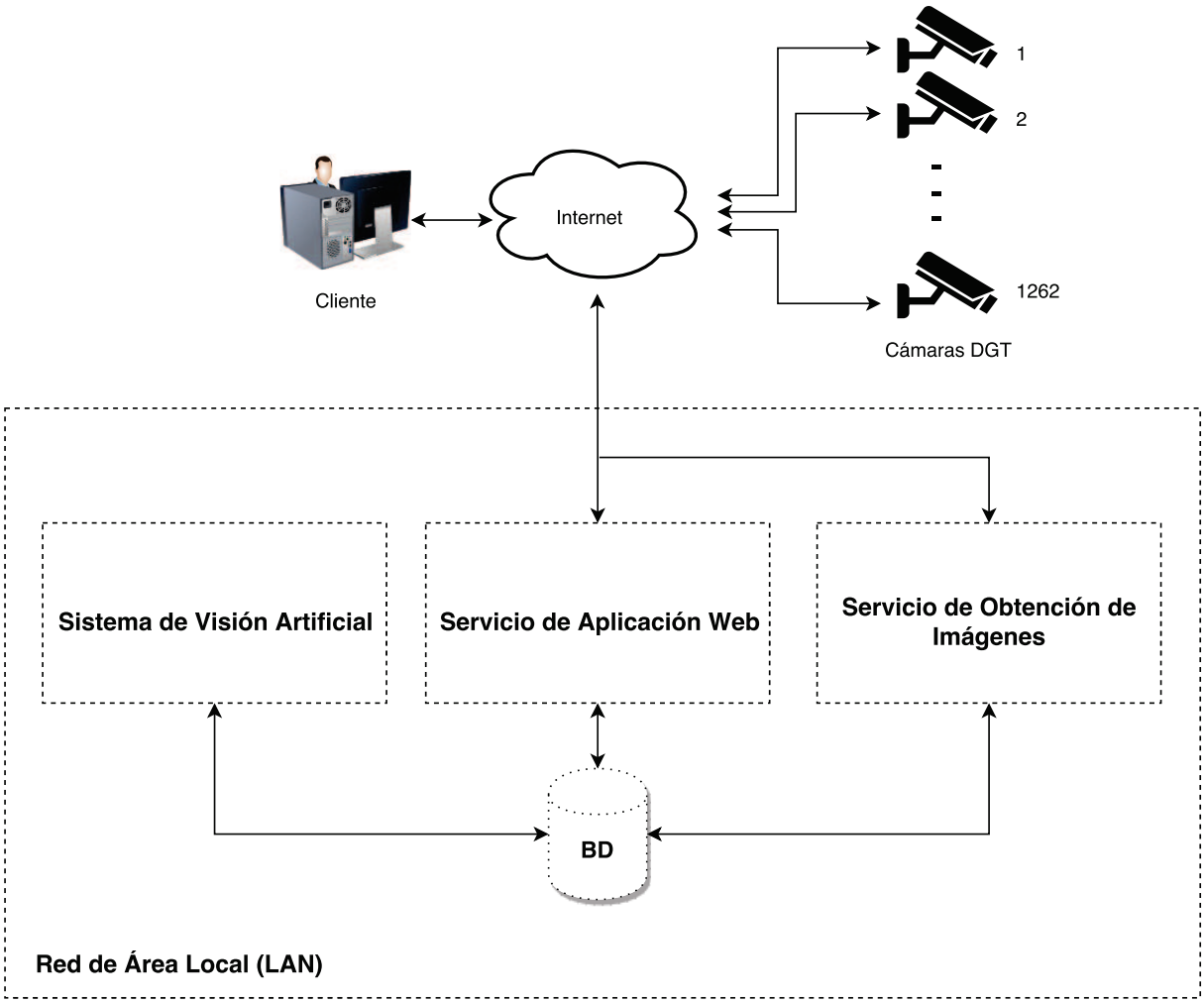

Figura 2.2: Arquitectura del sistema propuesto desde un punto de vista general.

2.2. Descripción detallada de la arquitectura

En este apartado se procede a detallar de forma más precisa cada uno de los componentes que forman el sistema, tratando de aclarar el papel que desempeñan cada uno de ellos. En la Figura 2.3 se muestra la arquitectura de forma más detallada en la que se encuentran los siguientes elementos:

- **Base de Datos (BD).**
- **Sistema de Obtención de Imágenes (SOI).**
- **Sistema de Visión Artificial (SVA).**
- **Servicio de Aplicación Web (SAW).**

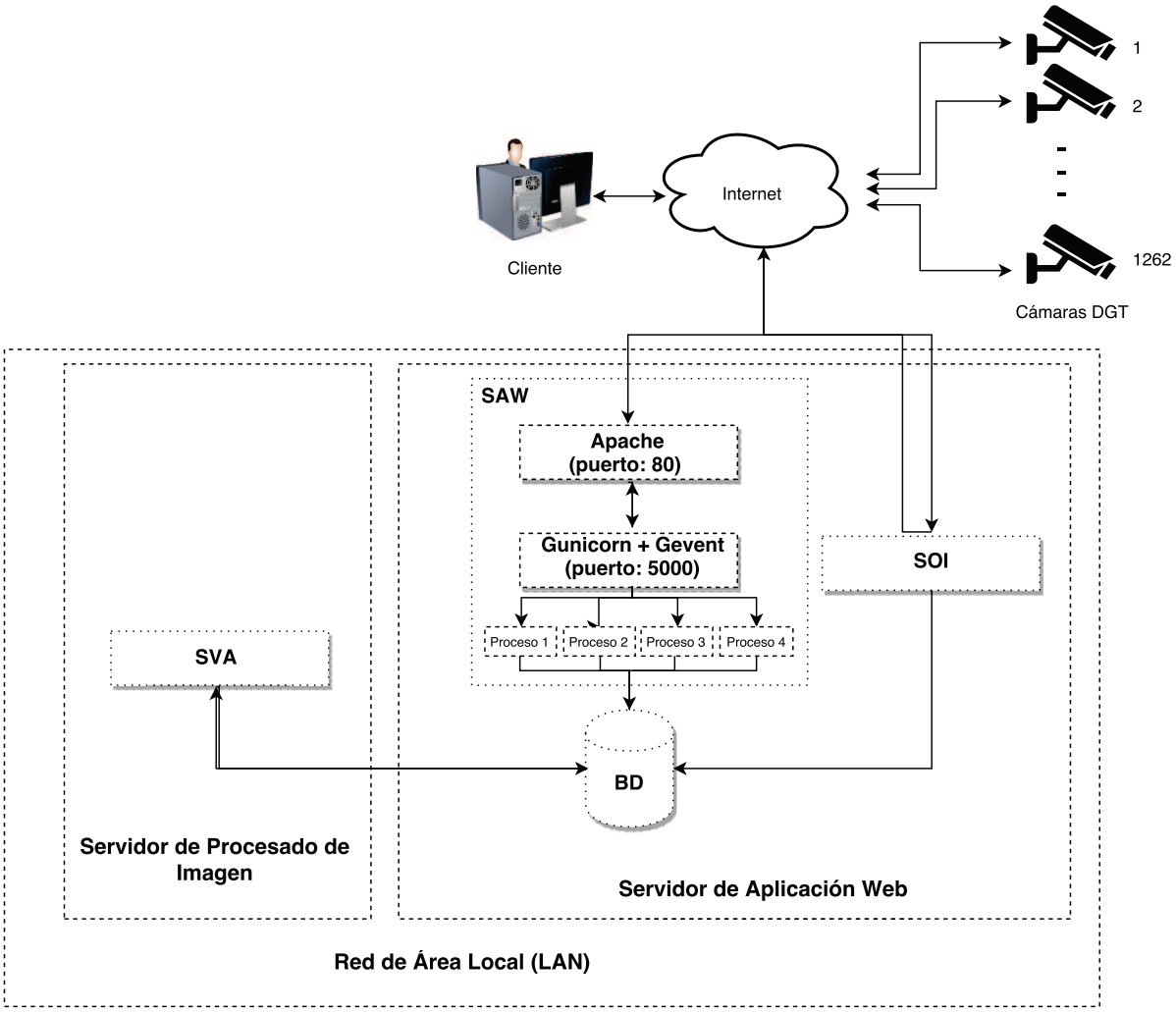

Figura 2.3: Arquitectura del sistema propuesto desde un punto de vista detallado.

2.2.1. Base de Datos

Para la base de datos se ha empleado el sistema de gestión de base de datos MySQL [29] que proporciona un modelo de base de datos relacional. Esta elección es debida a que en esta aplicación web la concurrencia de modificación de datos va a ser baja, mientras que la lectura de datos será intensiva, lo que hace que el motor no transaccional MyISAM [28] que emplea MySQL sea la mejor elección para este proyecto.

Se ha creado una base de datos llamada **dgt** encargada de almacenar toda la información necesaria para el sistema en las dos tablas que se muestran a continuación. La Figura 2.4 muestra un diagrama de flujo que representa a grandes rasgos el funcionamiento del sistema, de modo que resulte más fácil entender el uso que se le da a cada una de las tablas y campos de la base de datos.

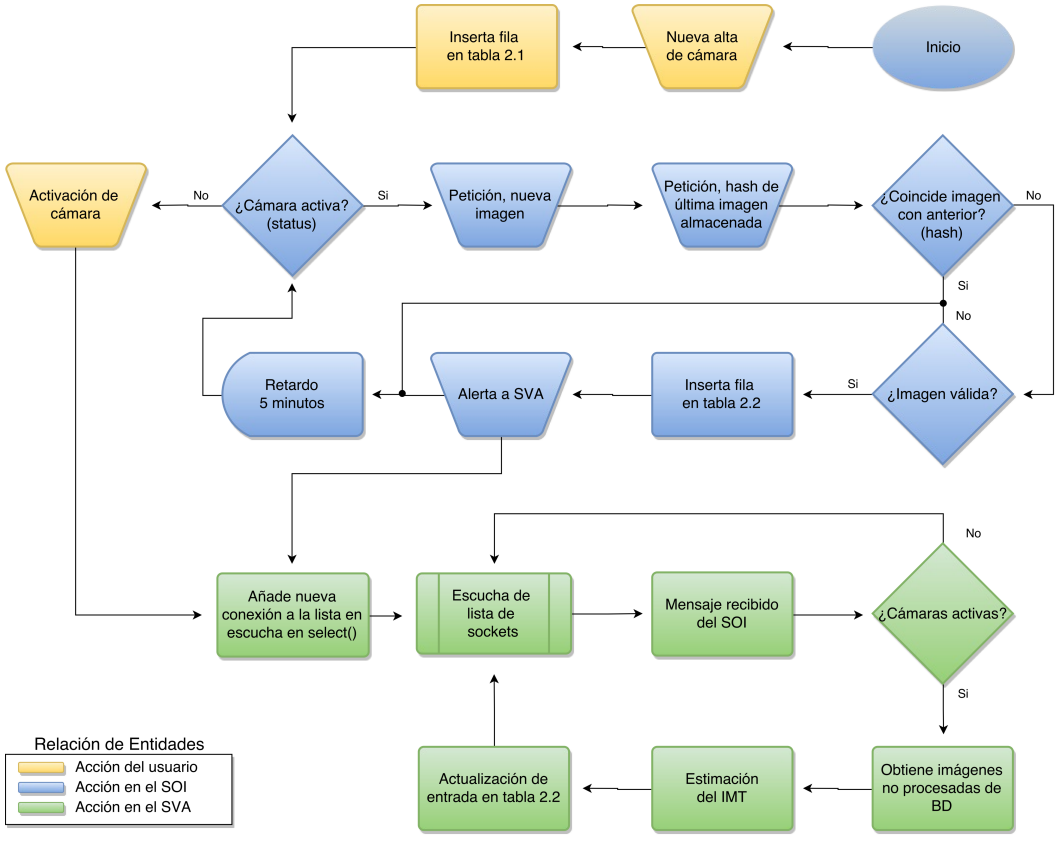

Figura 2.4: Diagrama de flujo general del sistema.

En primer lugar se encuentra la tabla 2.1, la cual es la encargada de guardar toda la información relacionada con las cámaras disponibles en el sistema, por lo que el valor de sus campos sólo se modifica cuando se añaden o se quiere dar de baja alguna cámara en el sistema.

Campo	Valor
id	Identificador interno de la cámara.
status	Estado de la cámara. Sirve para activar o desactivar la cámara.
province	Provincia en la que se encuentra la cámara.
road	Carretera en la que se encuentra la cámara.
pk	Punto kilométrico en la que se encuentra la cámara.
way	Sentido ascendente o descendente en el que se encuentra la cámara.
codEle	Código identificador para obtener las imágenes de la cámara.
lat	Latitud, información para situar la cámara geográficamente.
lng	Longitud, información para situar la cámara geográficamente.
timestamp	Marca de tiempo en la que agregó la cámara al sistema.

Tabla 2.1: Tabla *cameras_ dgt* que contiene información de las cámaras.

Por un lado se encuentra el campo *id* que hace referencia a un código que usa el sistema para identificar la cámara, el cual es asignado en orden ascendente al dar de alta una nueva cámara en el sistema. El campo *status* indica el estado de la cámara y puede tener valor *1* o *0*, dependiendo de si la cámara se encuentra activa para que el sistema procese sus imágenes o no.

Para poder activar o desactivar una cámara, se provee a la interfaz web de un selector, de este modo el proceso de localización de una cámara en el sistema se realiza de una forma más intuitiva, teniendo que seleccionar la provincia, la carretera y el punto kilométrico en el que se encuentra dicha cámara. Este es el propósito de los campos *province*, *road* y *pk*. Además el campo *way* informa si la cámara se sitúa en el arcén que está pegado a las vías en las que los vehículos circulan en sentido ascendente (parten desde Madrid), descendente (viajan hacia Madrid) o en el arcén central.

Además, el campo *codEle* almacena el identificador que la cámara tiene asignada por parte de la DGT. Este identificador lo emplea el sistema para obtener la imagen directamente de cada cámara. Los campos *lat* y *lng*, almacenan los valores de latitud y longitud exacto en que la cámara se localiza. Esto permite al sistema geolocalizar la cámara en el mapa de la interfaz web.

La base de datos diseñada contiene, además, la tabla 2.2, la cual almacena información relacionada con el procesado de imágenes llevado a cabo por todo el sistema.

Campo	Valor
id	Identificador interno de la cámara.
codEle	Código identificador para obtener las imágenes de la cámara.
hash	Hash de la imagen obtenida de la cámara.
processed	Estado de procesado de la imagen. Sirve para saber si la imagen ha sido procesada o no.
imt	Índice medio de tráfico. Valor estimado por el sistema de conteo de vehículos.
img_original	Imagen obtenida de la cámara.
img_processed	Imagen resultado del procesado.
timestamp	Marca de tiempo en la que se obtuvo una nueva imagen.

Tabla 2.2: Tabla *processing_ dgt* que contiene información del procesado de imágenes.

Esta segunda tabla guarda, en cada entrada o fila, la información necesaria para llevar a cabo la estimación del tráfico en un momento determinado y el resultado que se obtiene del procesado. En este proceso participan de forma activa el Sistema de Obtención de Imágenes y el SVA, por lo que es importante que todas las entradas en esta tabla queden bien identificadas para conocer la cámara que recoge la imagen y el tiempo exacto en el que se almacena el resultado del procesado.

Para ello, en primer lugar, se encuentra el campo *id*, que al igual que en la tabla anterior, almacena un identificador de la cámara en este sistema. El campo *codEle* contiene el valor del identificador que la DGT ha asignado a la cámara. El último identificador de una entrada en la tabla es el campo *timestamp*, el cual guarda la marca de tiempo del momento en que se añade la entrada en la tabla.

El SOI es el encargado de insertar una nueva entrada en esta tabla, esto se produce cuando una cámara se activa o transcurren 5 minutos a partir de la anterior adquisición de una imagen por parte de la misma cámara. En la petición que realiza el SOI a la base de datos, se almacenan, además de los identificadores ya citados, la imagen original obtenida directamente de la cámara, la cual se almacena en formato binario en el campo *img_ original*. Además, en el campo *hash*, se guarda el hash de la imagen original, esto se hace para saber si la imagen obtenida es o no válida, bien porque es igual que la imagen recogida en los 5 minutos previos o porque es una imagen que indica que la cámara no funciona correctamente. Esto último se conoce realizando una comparación con el hash de imágenes conocidas que reportan las cámaras cuando se encuentran fuera de servicio. Por último se almacena el campo *processed* con valor *0*, indicando de esta forma que la

imagen de esa entrada no ha sido aún procesada.

El método de funcionamiento del SVA, aunque más adelante se trata en profundidad, pasa por realizar una petición del campo de identificación del sistema a la base de datos de forma continua con la condición de que el campo *processed* tenga valor *0*. De este modo la respuesta a la petición por parte de la base de datos será un diccionario[1] vacío si no hay nuevas imágenes que procesar, y si por el contrario hay imágenes que procesar, el SVA obtendrá la imagen original almacenada en la base de datos gracias al identificador de la cámara, y una vez estimado el número de vehículos que tiene la imagen, actualizará la entrada en la tabla con la imagen resultado del procesado en el campo *img_processed*, y el valor de la estimación de vehículos en el campo *imt*.

2.2.2. Sistema de obtención de imágenes

El SOI se trata de un módulo dedicado a la obtención de imágenes. Para este propósito se ha creado una aplicación en lenguaje Python [32]. Python es un lenguaje de programación interpretado capaz de soportar programación imperativa, orientada a objetos y en menor medida, programación funcional. La versión empleada para este proyecto ha sido la 2.7. La Figura 2.5 muestra un diagrama de flujo que permite entender el funcionamiento del SOI.

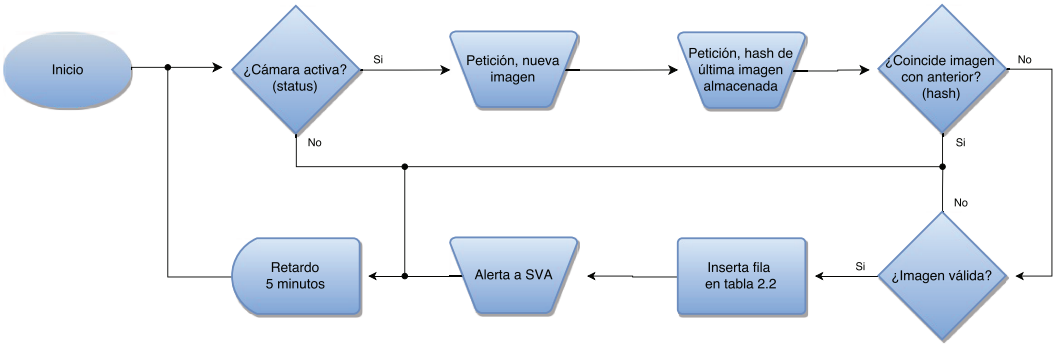

Figura 2.5: Diagrama de flujo del SOI.

El objetivo del SOI es obtener las imágenes del tráfico que ofrecen las cámaras de la DGT. Gracias al identificador *codEle*, se realiza una petición a cada una de las cámaras activas en el sistema para obtener una imagen cuando se arranca la aplicación citada previamente o cuando transcurren 5 minutos a partir de la anterior adquisición de una imagen por parte de la misma cámara. Este intervalo es debido al límite establecido por la DGT, ya que si se realiza una petición con un retardo menor de 5 minutos la imagen recogida no habrá sido actualizada.

[1]Se define diccionario como un conjunto no ordenado de pares clave: valor, con el requerimiento de que las claves sean únicas [33].

Una vez recibida una nueva imagen, el sistema realiza dos comprobaciones antes de insertar la nueva imagen en la tabla 2.2. La primera comprobación se hace para saber si la imagen obtenida no coincide con la imagen adquirida en la última petición por la misma cámara, ya que en este caso se duplicaría la imagen sin aportar información relevante al sistema y, además, requeriría una carga innecesaria en el sistema. Para ello se calcula el hash MD5 de la imagen obtenida, y se realiza una petición a la base de datos del hash MD5 de la última imagen que ha sido procesada perteneciente a la misma cámara. Con una comparación de los hashes el sistema es capaz de conocer si la imagen ha superado la primera comprobación.

La segunda comparación que realiza el sistema emplea de nuevo el hash MD5 calculado de la imagen obtenida, pero en este caso lo compara con un array de hashes MD5 almacenado que contiene el hash de imágenes conocidas que las cámaras reportan cuando se encuentran fuera de servicio. Algunas de las cuales se pueden ver en la Figura 2.6. Así el sistema es capaz de conocer si la imagen obtenida, además de no estar duplicada, no indica que la cámara se encuentra fuera de servicio.

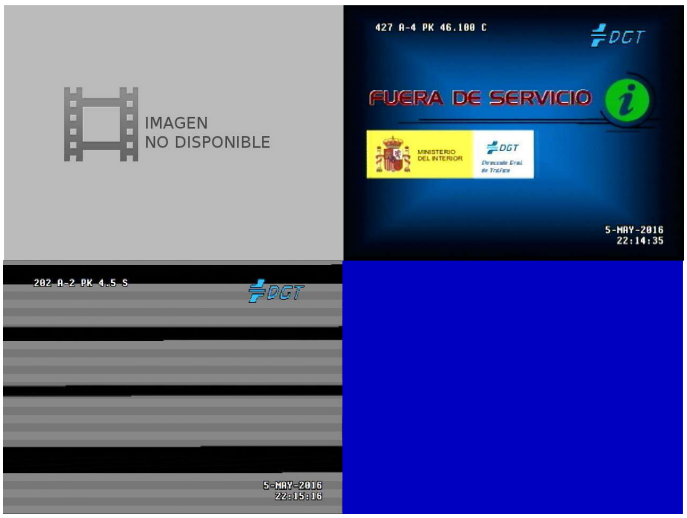

Figura 2.6: Ejemplo de imágenes de cámara fuera de servicio.

Cabe destacar que cada vez que una imagen es recogida de una cámara, esté duplicada o no, e incluso se trate de una imagen de fuera de servicio, el sistema la almacena en una carpeta temporal para mostrarla en la interfaz web y así el usuario pueda saber en todo momento las imágenes entregadas por la cámara seleccionada y el estado de ésta.

Por último el SOI envía un mensaje al SVA indicándole que la base de datos ha sido actualizada. De éste modo el SVA sabe que debe revisar la base de datos en búsqueda de las nuevas imágenes que debe procesar.

2.2.3. Sistema de visión artificial

Por otro lado, se encuentra el SVA, el cual tiene como propósito la integración del sistema de conteo de vehículos con el resto de la arquitectura gracias de nuevo a una aplicación creado en Python. El diagrama de la Figura 2.7 muestra el ciclo de trabajo que emplea el SVA.

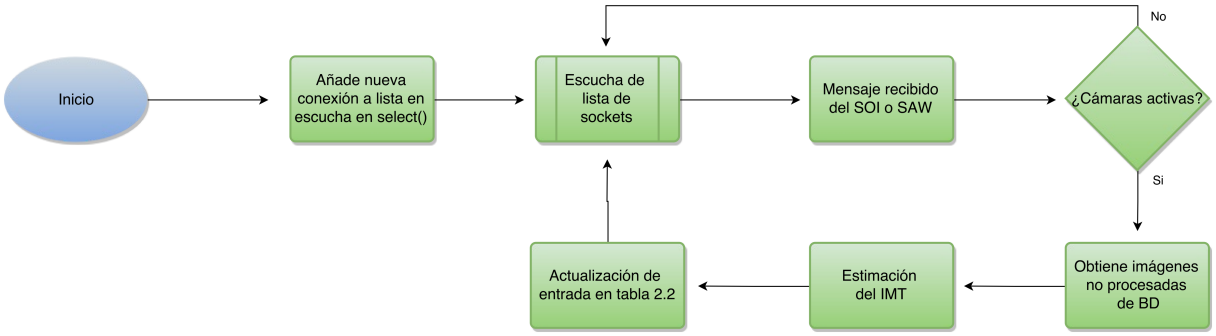

Figura 2.7: Diagrama de flujo del SVA.

La integración del SVA se lleva a cabo con la base de datos como puente de enlace de todo el sistema, como muestra la Figura 2.3. Para ello se ha creado una aplicación, escrita en python, en modo servidor TCP que realiza la escucha a través de la función *select* [13].

El SVA puede recibir tanto peticiones del SOI cada 5 minutos, o del SAW. Las peticiones que puede recibir del SAW sucederán cada vez que se active una nueva cámara y se seleccione para ver sus datos pero la base de datos no contenga entradas pertenecientes a dicha cámara a lo largo del día. Por ello el SVA debe estar preparado para manejar múltiples conexiones de socket, gracias al uso de hilos o mediante la técnica del sondeo. En esta aplicación se ha empleado la función *select* para realizar esta tarea mediante sondeo. En la técnica del sondeo, continuamente se comprueba una lista de sockets, que contiene las conexiones de los clientes, en búsqueda de nuevos eventos. Cuando sucede algún evento nuevo la función *select* devuelve el descriptor del socket que genera dicho evento.

Cuando el SVA recibe un mensaje indicando que se ha insertado una nueva imagen en la base de datos, realiza una petición en la que solicita la última imagen original, almacenada en el campo *img_ original*, de todas las cámaras activas (esto se comprueba con el campo *status* de la tabla 2.1) que contengan el valor del campo *processed* con valor 0 de la tabla 2.2.

De este modo, si no hay datos que procesar en la base de datos, la respuesta será inmediata con una tupla vacía, siendo el tráfico de red generado insignificante respecto a la capacidad de la red y la carga ejercida en el servidor mínima.

Si, por el contrario, hay imágenes que procesar, el SVA almacenará las imágenes recibidas por parte de la base de datos en una carpeta temporal y serán procesadas por el módulo de visión artificial. Una vez se obtenga la estimación del IMT, la aplicación actualizará la entrada de la tabla 2.2 perteneciente a la imagen que ha sido procesada, insertando la imagen resultado del procesado en formato binario en el campo *img_processed* y el índice medio de tráfico estimado.

Cabe destacar que el módulo de visión artificial [1] no se ha desarrollado en este TFG, sino que se ha integrado en el sistema. Este módulo realiza un procesado de la imagen obtenida de la cámara mediante Deep Learning; de esta forma se consiguen resultados que difícilmente pueden ser obtenidos con técnicas tradicionales de visión por computador. Estos resultados son alcanzados gracias a la integración de la programación orientada a la GPU (CUDA [7]) y a la existencia de grandes bases de datos anotadas[2].

En la Figura 2.8 se puede observar como realiza el módulo de visión artificial la estimación. A la izquierda de la figura se encuentra la imagen de entrada en el módulo. En primer lugar se realiza una extracción de parches descomponiendo la imagen en un conjunto de imágenes más pequeñas. Cada uno de los parches de la imagen se utiliza para alimentar un regresor [26] encargado de realizar la estimación del mapa de densidad que crea a partir de cada parche de entrada en el regresor. Por último se vuelven a ensamblar los parches construyendo el mapa de densidad final del mismo tamaño que la imagen de entrada.

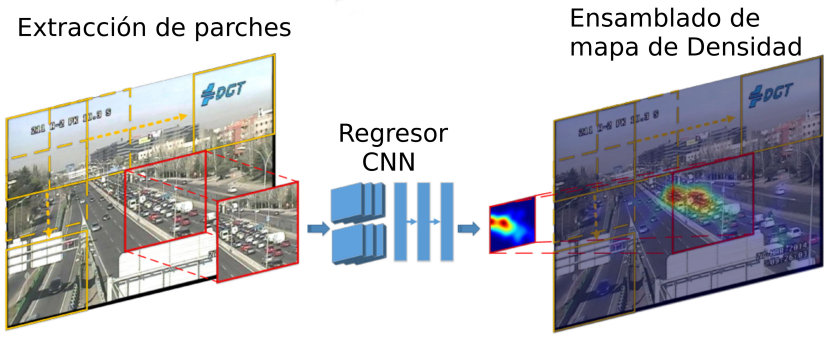

Figura 2.8: Módulo de visión artificial.

[2]Bases de datos compuestas por imágenes que previamente han sido anotadas por personas, identificando los elementos necesarios para un sistema de detección

2.2.4. Servicio de aplicación web

En última instancia cabe destacar el SAW. Éste se encarga de ofrecer una plataforma online capaz de monitorizar y geolocalizar cámaras de videovigilancia de la DGT, mostrando el nivel de congestión que presentan en tiempo real. La aplicación no sólo muestra los datos procesados en tiempo real, sino que también realiza una predicción del nivel de tráfico basándose en los datos recogidos durante un periodo determinado. La Figura 2.9 muestra la localización del SAW en el sistema.

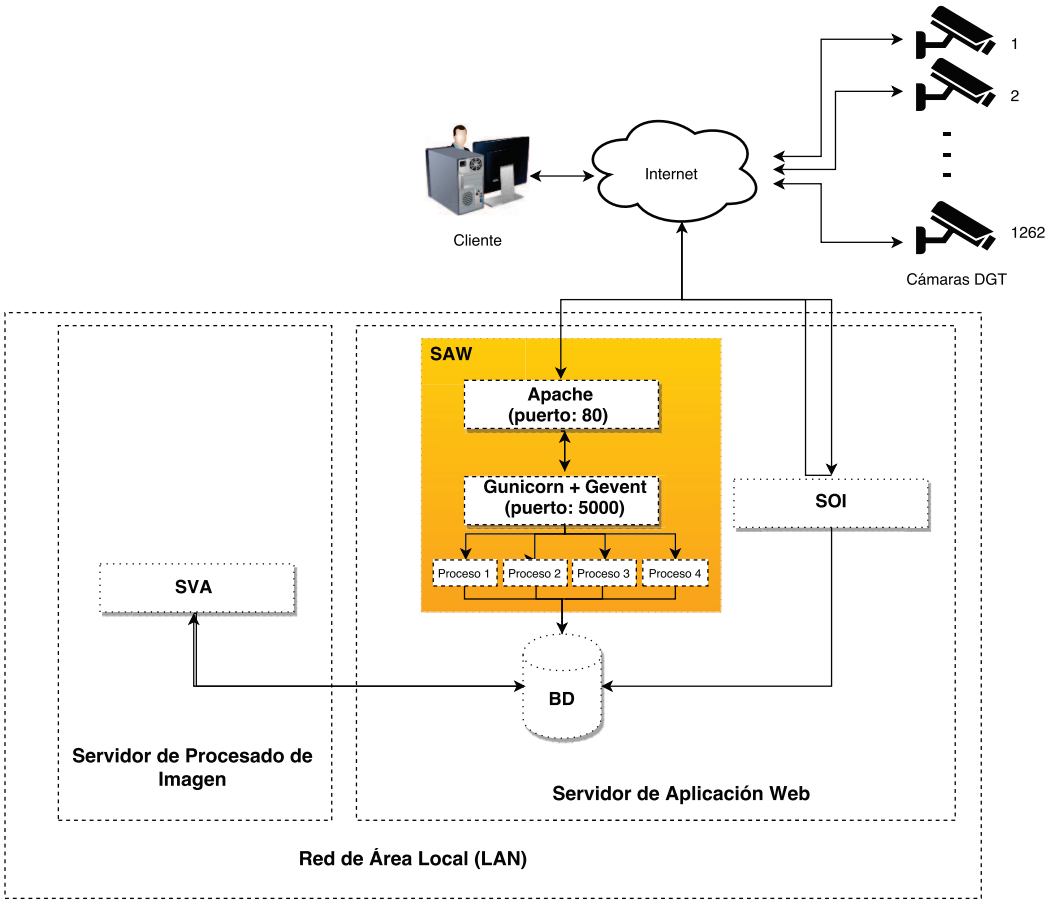

Figura 2.9: Servidor de aplicación web.

Para este propósito se ha empleado un servidor LAMP [44] (Linux, Apache [2], MySQL, PHP [31]), enriquecido con las siguientes tecnologías:

- **Gunicorn** [20]: También llamado 'Green Unicorn', se trata de un servidor web Web Server Gateway Interface (WGSI) [46] para Python, o dicho de otro modo, es una interfaz entre aplicaciones web o frameworks escritas en Python y servidores web, destinado para plataformas UNIX. De entre sus principales características, en este proyecto se ha aprovechado la gestión automática de procesos que a su vez se encargarán de los hilos necesarios.

- **Gevent** [14]: Para que el sistema sea capaz de absorber la carga generada por múltiples usuarios de forma simultánea es necesario dotar a la aplicación web de varios procesos. Esto se realiza al lanzar la aplicación con Gunicorn como servidor, pero para que estos procesos sean gestionados de una forma más eficiente, se hace uso de una librería de red basada en corrutinas destinada a aplicaciones escritas en Python, llamada Gevent. De este modo, Gevent proporciona una API síncrona de alto nivel en la parte superior del bucle de eventos de libevent [27].

  Libevent proporciona un mecanismo basado en una API para ejecutar una función de llamada cada vez que ocurre un evento específico en un descriptor de archivo o después de haber alcanzado un tiempo de espera.

  El comando para que Gunicorn sirva la aplicación *webApp.py* en el puerto 5000 usando 4 procesos gestionados por Gevent, se haría de la siguiente manera,

```
$ gunicorn -k gevent -w 4 -b '0.0.0.0:5000'  webApp:app
```

- **Jinja2** [23]:

  La aplicación web emplea un motor de plantillas llamado Jinja2, que separa la lógica de presentación de la lógica de control. El motor de plantillas permite preparar un bloque de cabecera y otro para el cuerpo en una página escrita en HTML5 [21] como base. A partir de esta base se preparan diferentes plantillas en las que se especifica que porción de código debe asignarse a cada bloque preparado. Con esto se consigue que en el funcionamiento de la aplicación web, dependiendo de donde se encuentre el usuario, se carguen una plantilla u otra en los bloques que se dejan preparados en la plantilla base. Emplear una cabecera distinta para cada plantilla permite agilizar la web, ya que sólo se cargan los recursos necesarios en cada momento (hojas de estilo en CSS3 [6], scripts en JavaScript [22], jQuery [24] y AJAX [41]). Además, la carga dinámica del cuerpo de la página da fluidez a la página y posibilita ajustar el contenido a cada ocasión, eliminando cargas innecesarias de imágenes o datos.

  Además de poder preparar bloques con el contenido de cabecera y cuerpo, se pueden realizar otras acciones como mostrar un contenido u otro dependiendo de si el usuario ha iniciado sesión, dejar preparadas variables como por ejemplo, para mostrar el error que comete el usuario en el inicio de sesión, en este caso. Para más información conviene consultar la documentación principal.

Las pautas seguidas para crear las plantillas han sido las siguientes:

- {% ... %} para declaraciones.
- {{ ... }} para expresiones.
- {# ... #} para comentarios.

De este modo quedaría así el fichero *layout.html*, que se emplea como plantilla base para las páginas necesarias en la aplicación.

```
<!DOCTYPE html>
<html lang="es">
  <meta http-equiv="content-type" content="text/html; charset=
     UTF-8">
  <meta charset="utf-8">
  <link rel="icon" href="../static/images/favicon.ico" type="
     image/png">
  <title>Titulo del sitio</title>
  <meta name="viewport" content="width=device-width, initial-
     scale=1">

      <!---------- STYLES AND SCRIPTS ---------->
  <link rel=stylesheet href="{{ url_for('static', filename='css
     /bootstrap.min.css?121720151') }}">
  <link rel=stylesheet href="{{ url_for('static', filename='css
     /style.css') }}">
  <script src="{{ url_for('static', filename='js/jquery.min.js
     ') }}"></script>
  <script src="{{ url_for('static', filename='js/bootstrap.min.
     js') }}"></script>

  {% block head %}{% endblock %}

  <script type=text/javascript>
    $SCRIPT_ROOT = {{ request.script_root|tojson|safe }};
  </script>

  <!-- Fixed navbar -->
  <div class="navbar navbar-custom navbar-inverse navbar-static
     -top" id="nav">

  {% block body %}{% endblock %}
</html>
```

En ella se pueden observar algunas pautas citadas previamente. La localización de los bloques de cabecera y cuerpo se gestionan con las etiquetas *{ %block head %}{ %endblock %}* y *{ %block body %}{ %endblock %}*, respectivamente. También las llamadas a las hojas de estilo y scripts se hacen también mediante Jinja2 como se puede ver en las llamadas *{{ url_for('RUTA', filename='ARCHIVO')}}*. Además, *$SCRIPT_ROOT* se declara para indicar la ruta base de acceso a los archivos necesarios para la aplicación.

- **Flask** [12]: Como ya se ha comentado, la aplicación web se ha creado en el lenguaje Python, para ello se ha echo uso de este micro framework, enfocado al desarrollo de aplicaciones web, basado en Werkzeug [40] (librería de utilidades para WSGI de Python), y Jinja2.

  El siguiente fragmento de código muestra un ejemplo de uso, en este caso se muestra la ruta del inicio de sesión que emplea la aplicación web. En ella se puede ver como se define la ruta */login* que permite los métodos *GET* y *POST*. Cuando un usuario ingresa en la página de la aplicación web se le muestra un formulario en el que debe ingresar un usuario y contraseña. Al pulsar el botón de iniciar sesión, la página realiza una petición a la ruta */login* enviando la información del usuario y contraseña introducidos.

```
# login route
@app.route('/login', methods=['GET', 'POST'])
def login():
  global db
  global cursor
  error = None

  if request.method == 'POST':
    if request.form['username'] != app.config['USERNAME']:
      error = 'Nombre de usuario incorrecto'
    elif request.form['password'] != app.config['PASSWORD']:
      error = 'Contrasena incorrecta'
    else:
      session['logged_in'] = True
      try:
        return redirect(url_for('demonstrator'))
      except:
        print "Could not connect to database on login"
  return render_template('index.html', error=error.decode('utf
     -8'))
```

Al acceder a esta ruta la aplicación primero comprueba si coincide el usuario y la contraseña con las almacenadas en el archivo de configuración. Si coinciden, marca la sesión como iniciada, de este modo si el usuario cierra la página y vuelve a abrirla no se le pedirá de nuevo las credenciales de acceso, y redirige al usuario a la página de la aplicación web.

Gracias al servicio de plantillas empleado con Jinja2, si el usuario o contraseña introducidos no son correctos, se muestra un mensaje de error instantáneo sin necesidad de recargar la página o emplear Ajax de forma directa.

El servicio SAW, tal y como se muestra en la Figura 2.10, funciona de la siguiente manera. En primer lugar el servidor web recibe las peticiones realizadas por el usuario, de este modo Apache se encarga de gestionar la demanda de recursos estáticos como pueden ser los ficheros de scripts necesarios, las hojas de estilo, imágenes, etc. Además, Apache ejerce la función de servidor proxy inverso, delegando en Gunicorn la tarea de atender la demanda de recursos dinámicos, como pueden ser la obtención de variables con las que se puede trabajar en la aplicación en Python o la generación del entorno web (o página web) a mostrar, en función de la pestaña seleccionada, gracias al uso de plantillas, para lo que se apoya en Jinja2.

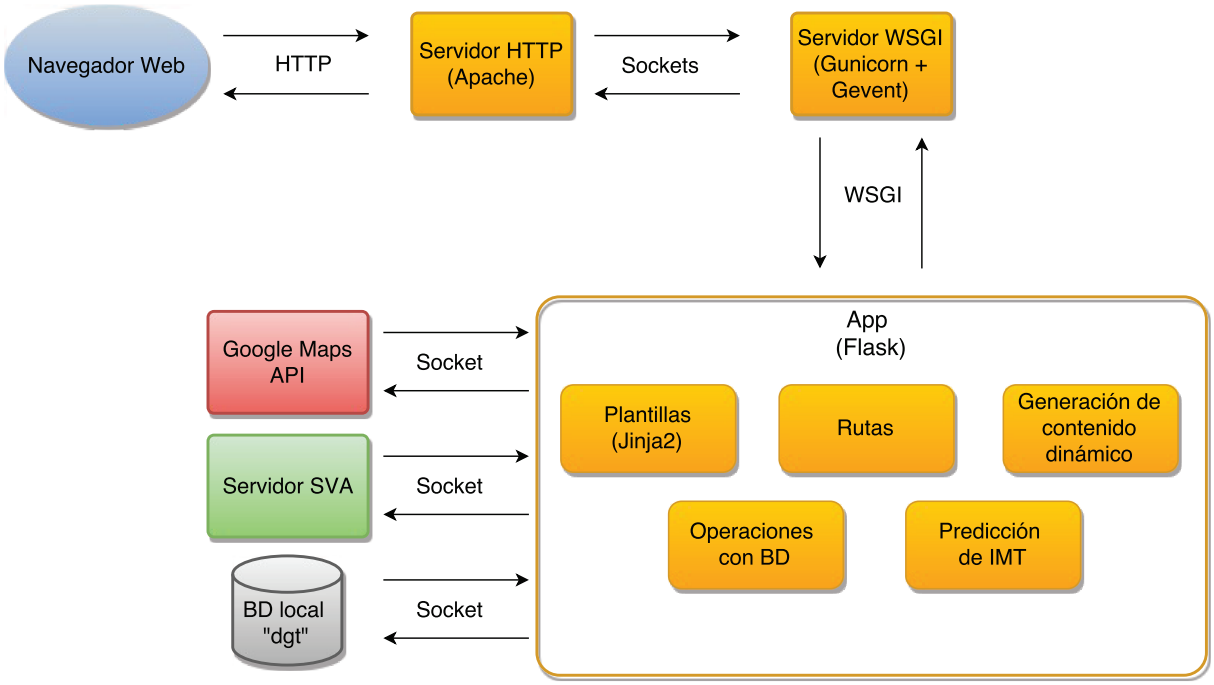

Figura 2.10: Diagrama de flujo del SAW.

2.2.4.1. Predicción del IMT

Cuando la aplicación recibe la solicitud del histórico diario del IMT estimado para una cámara, se realiza una predicción del IMT para los 15 minutos posteriores al instante de la última estimación realizada por el SVA. Esta predicción se ha realizado mediante el modelo de regresión sesgada o *Ridge Regression* [26].

El modelo de regresión sesgada se trata de un estimador capaz de resolver una regresión multi variable, empleando una función lineal de mínimos cuadrados como función de coste, una regularización del tipo L2 [45].

Dada una función de hipótesis, la cual será usada para realizar la predicción:

$$h_\theta(x^{(i)}) = \sum_{j=0}^{m}(x_j^{(i)}\theta_j) \ , \tag{2.1}$$

donde $x_j^{(i)}$ representa la matriz de características de entrada y θ_j simboliza los pesos que se aplican a las características de entrada. Además, m representa el número de muestras que se usan para realizar la estimación, siendo m igual a 6. Esto es debido a que se tienen en cuenta las 6 últimas muestras obtenidas del histórico de datos procesados.

A la función de coste empleada en una regresión lineal, la cual debe ser minimizada, se le añade un termino de regularización que consta de la suma de los parámetros de optimización θ_j al cuadrado penalizados por un coeficiente de regularización λ. De este modo se consigue equilibrar el peso de los parámetros de optimización empleados en el regresor:

$$J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{i=1}^{n}{\theta_j}^2] \ . \tag{2.2}$$

La implementación del regressor sesgado se ha realizado gracias al módulo *sklearn* de python 2.7. El siguiente fragmento de código muestra como se ha implementado el método de predicción. Cabe destacar que el parámetro *yTrain* que se le pasa como argumento contiene las 6 muestras tomadas del histórico de datos procesados.

```
from sklearn import linear_model
# Predict imt for the next 15 minutes (3 values)
def get_Prediction(yTrain, numOfPredictions):
    xTrain = []
    xTest = []

    # set xTrain range depending on Ytrain inputs
    for i in range(len(yTrain)):
        xTrain.append([i+1])
    # get xTest range depending of number of desired predictions
```

```
    for i in range(numOfPredictions):
        xTest.append([len(yTrain)+i+1])

    regr = linear_model.Ridge(alpha = .5)

    regr.fit(xTrain, yTrain)

    prediction = regr.predict(xTest)
    return prediction
```

La figura 2.11 muestra una gráfica del histórico diario de datos procesados para una cámara, donde aparece en color verde, los valores predichos por el regresor.

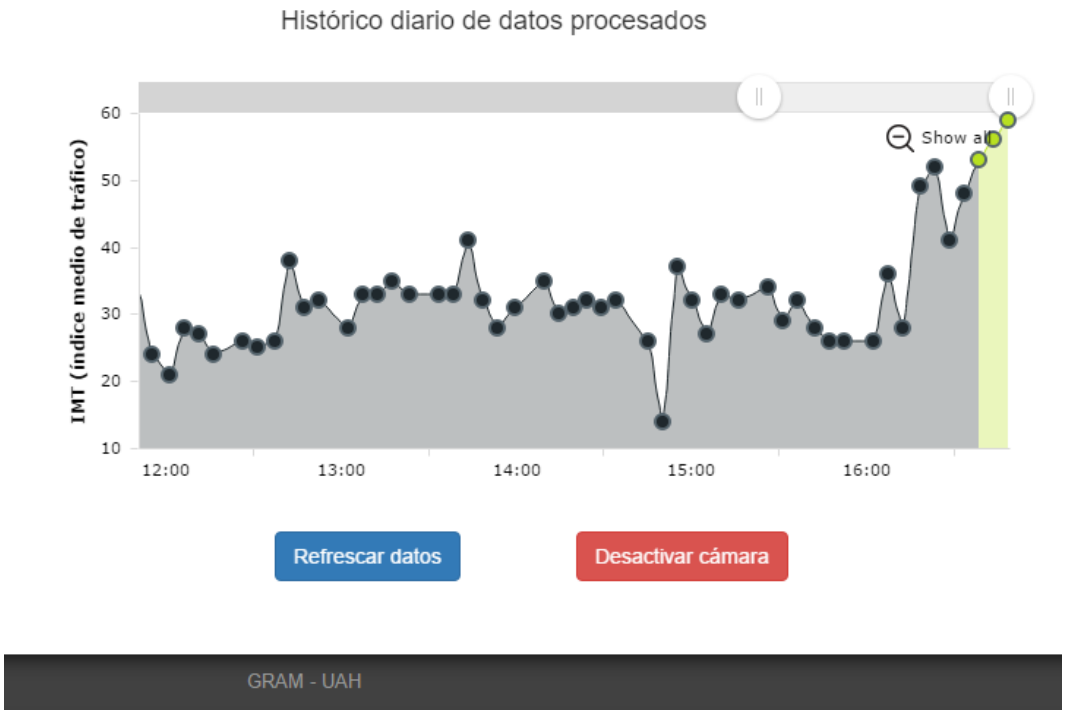

Figura 2.11: Histórico diario de datos procesados y predichos para una cámara.

2.3. Análisis de rendimiento del sistema

En primer lugar, cabe destacar que el sistema desarrollado en este TFG, se trata de un demostrador realizado para un proyecto de investigación llevado a cabo por el grupo de investigación GRAM de la Universidad de Alcalá. Debido a esto, el sistema se encuentra alojado dentro de un entorno de investigación con servidores compartidos, por lo que no se ha exprimido al máximo el potencial del sistema, dejando recursos disponibles al resto de proyectos y experimentos empleados por el grupo de investigación.

Para analizar el rendimiento del sistema completo, se va proceder ha realizar un estudio de los tiempos empleados por las secciones más críticas dentro del sistema, diferenciando cada uno de los servicios desarrollados. Este estudio realiza un análisis de los datos obtenidos con 24 cámaras activas en el sistema durante un periodo de 2 días. La Figura 2.12 muestra el ciclo natural que va desde la activación de una cámara, hasta la visualización de los datos estimados y predichos en la aplicación web.

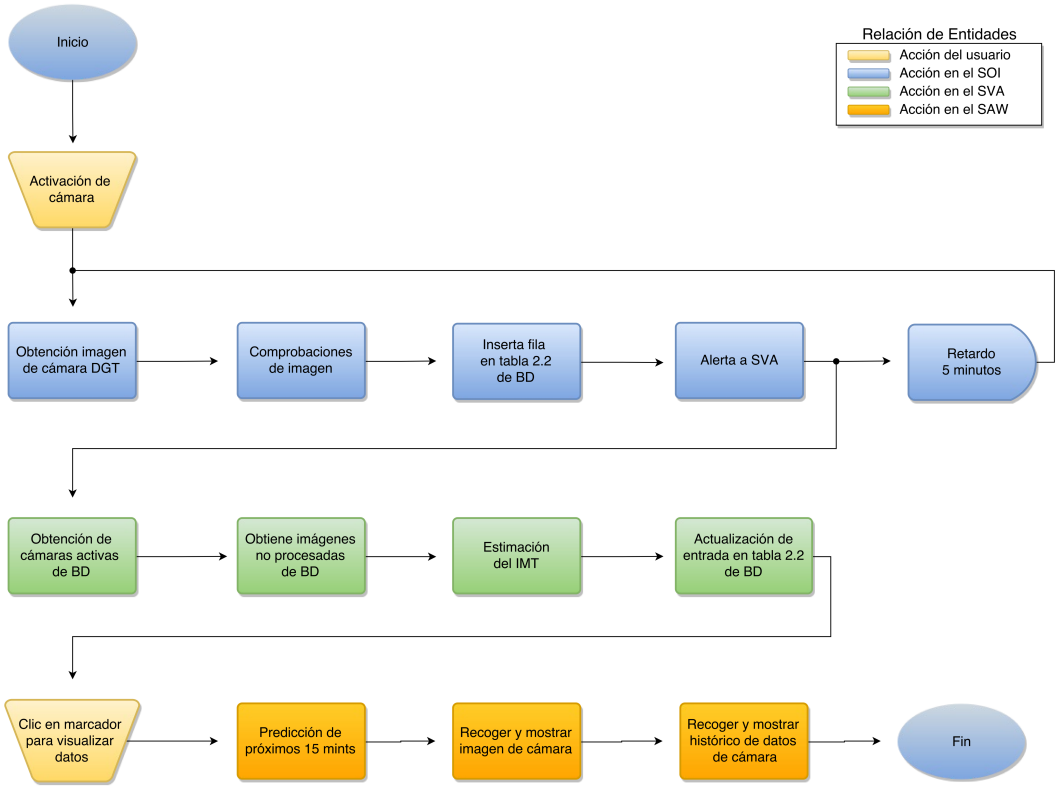

Figura 2.12: Diagrama de flujo del estudio para el análisis de rendimiento.

En el análisis realizado, se ha observado que los tiempos de la mayoría de las cámaras testadas han sido similares en todos los servicios, por lo que los resultados se entregan de forma generalizada mostrando los tiempos obtenidos con la misma cámara en todos los servicios.

2.3.1. Sistema de Obtención de Imágenes

Las posibles secciones críticas del SOI que pueden afectar al rendimiento del sistema, son:

- Obtención de una nueva imagen de la cámara de la DGT.
- Conjunto de comprobaciones para determinar si la imagen es válida.
- Inserción de la nueva entrada en la base de datos.

Cabe destacar que debido a que se obtienen las imágenes directamente de las cámaras, el tiempo empleado en la obtención de una nueva imagen varia dependiendo de la geolocalización de la cámara. La tabla 2.3, muestra el tiempo empleado por las diferentes secciones para una cámara.

Tramo	Tiempo(ms)
Obtención de imagen de la cámara	251,83
Comprobaciones de imagen	326,005
Inserción fila en BD	1,172
Total:	579,007

Tabla 2.3: Tiempo empleado por las secciones críticas del SOI.

Analizando los resultados obtenidos, se puede sacar la conclusión de que el tiempo empleado por el SOI desde la adquisición de una nueva imagen hasta su inserción en la base de datos es mínimo, ya que el tiempo total empleado por el SOI en este proceso de media, se sitúa aproximadamente en 579 ms. La Figura 2.13 muestra el porcentaje de tiempo empleado por las diferentes secciones analizadas del SOI.

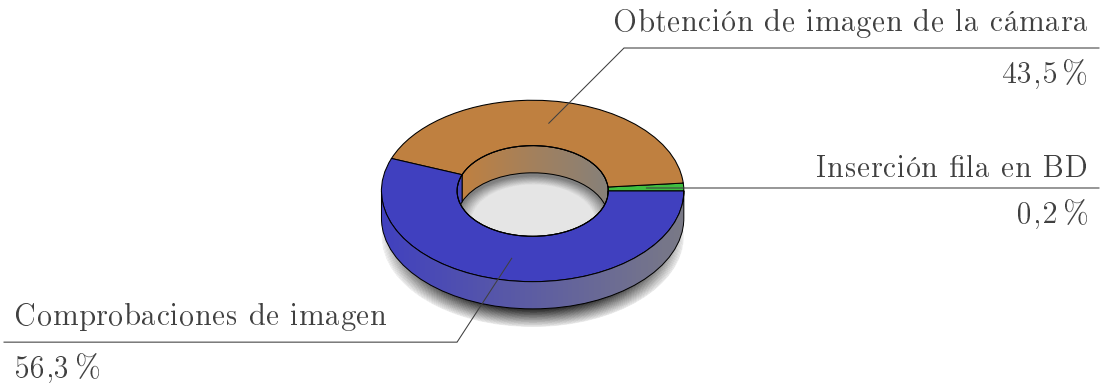

Figura 2.13: Porcentaje de tiempo empleado por las secciones críticas del SOI.

Teniendo en cuenta estos resultados y que la obtención de nuevas imágenes se realizan cada 5 minutos, se puede determinar que el máximo número de cámaras que el SOI es capaz de gestionar sin llegar a saturarse son, de forma aproximada, 518 cámaras. Para poder aumentar la capacidad, una posible solución pasaría por incrementar el tiempo entre la adquisición de nuevas imágenes, o emplear un servidor dedicado con mejores características para poder implementar múltiples hilos en el SOI.

2.3.2. Sistema de Visión Artificial

Las posibles secciones críticas del SVA que pueden afectar al rendimiento del sistema, son:

- Obtención de la última imagen original sin procesar de la base de datos.
- Estimación del IMT, realizado por el módulo de visión artificial.
- Actualización de la entrada en la base de datos.

Como ya se ha comentado, el método empleado para la estimación del IMT se realiza mediante el uso de Deep Learning, por lo que el tiempo empleado en este punto es determinante en la capacidad del sistema para evitar su sobrecarga. Los siguientes resultados de la tabla 2.4, representan de forma generalizada el tiempo empleado por las diferentes secciones para una cámara.

Tramo	Tiempo(s)
Obtención de última imagen original	0,0889
Estimación del IMT	5,3035
Actualización de entrada en BD	0,0715
Total:	5,4639

Tabla 2.4: Tiempo empleado por las secciones críticas del SVA.

A partir de los resultados obtenidos, se puede determinar que el SVA puede crear un cuello de botella en el sistema. Aún así el tiempo total empleado por el SVA para tratar una imagen no resulta excesivo, más aún teniendo en cuenta el método de estimación empleado. La Figura 2.14 muestra el porcentaje de tiempo empleado por las diferentes secciones analizadas del SVA.

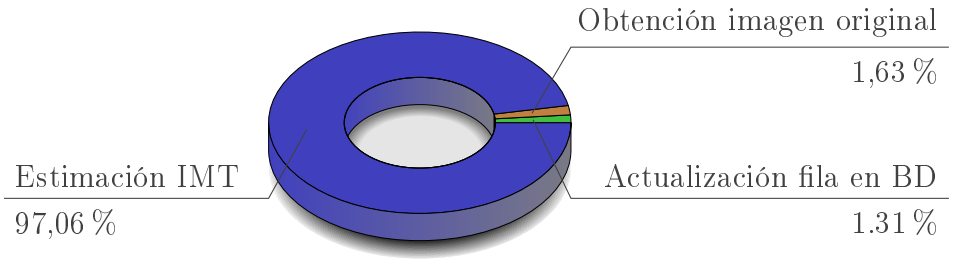

Figura 2.14: Porcentaje de tiempo empleado por las secciones críticas del SVA.

Teniendo en cuenta estos resultados y que la obtención de nuevas imágenes se realizan cada 5 minutos, se puede determinar que el máximo número de cámaras que el SVA es capaz de procesar sin llegar a saturarse son, de forma aproximada, 54 cámaras. Para poder aumentar la capacidad, de igual modo que en el SOI, una posible solución pasaría por incrementar el tiempo entre la adquisición de nuevas imágenes, o emplear un servidor dedicado con mejores características para poder implementar múltiples hilos en el SVA.

2.3.3. Servicio de Aplicación Web

El estudio realizado en el SAW, se centra en conocer el tiempo necesario desde que un usuario hace clic en el marcador de una cámara hasta que se muestran los datos en la página web. Para ello se ha obtenido el tiempo necesario por las siguientes secciones críticas:

- Predicción del IMT durante los 15 minutos posteriores a la última estimación.
- Mostrar la imagen original de la cámara actual.
- Mostrar el histórico diario de datos obtenidos de la cámara actual.

Aunque el SVA y el SOI forman una parte esencial en el sistema, los tiempos empleados por el SAW deben ser mínimos ya que son los que realmente el usuario es capaz de apreciar de forma directa. Mientras que la medición del tiempo empleado en la predicción del IMT durante los 15 minutos posteriores a la última estimación ha sido realizada en la propia aplicación web, el resto de mediciones realizadas en el SAW se ha llevado a cabo con la herramienta para desarrolladores disponible en el navegador Chrome. Por ello, cabe destacar que las mediciones han sido realizadas en una red con una velocidad de descarga de 5 Mbps de bajada y 0,5 Mbps de subida. Los resultados de la tabla 2.5, muestran el tiempo empleado por las diferentes secciones.

Tramo	Tiempo(ms)
Predicción próximos 15 mints.	1,078
Mostrar imagen original.	179
Mostrar histórico diario.	596
Total:	776,078

Tabla 2.5: Tiempo empleado por las secciones críticas del SAW.

La carga del histórico de datos es la responsable de la mayoría de tiempo empleado por el SAW para mostrar los datos en la página web. Cabe destacar que esta prueba ha sido realizada por la noche, de modo que el histórico de datos recoge los datos de un día completo. En la Figura 2.15 se puede observar el porcentaje de tiempo empleado por las diferentes secciones analizadas del SAW.

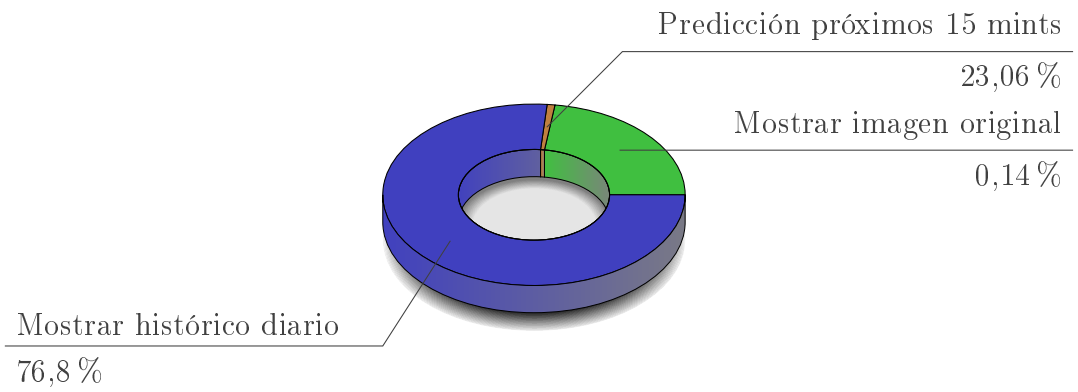

Figura 2.15: Porcentaje de tiempo empleado por las secciones críticas del SAW.

Por otro lado para optimizar el rendimiento de la aplicación web se ha configurado Gunicorn para que trabaje con 4 procesos que se generan mientras que Gunicorn está sirviendo nuestra aplicación web. Esto unido a la compresión de recursos mediante gzip [18], evita ofrecer una calidad de experiencia pobre al usuario, ya que la aplicación debe ocuparse de la obtención de datos así como su visualización, además de gestionar las peticiones y respuestas de los usuario.

Además, se ha realizado un test de rendimiento con la herramienta para desarrolladores de Google, PageSpeed Insight [9]. En las Figuras 2.16 y 2.17 se puede apreciar la puntuación obtenida para la versión para ordenador y móvil, teniendo en cuenta que la penalización en ambos casos, se debe a la carga de hojas de estilo de gran tamaño. Esto es debido al uso de Bootstrap [3] y los estilos creados de forma específica para este fin.

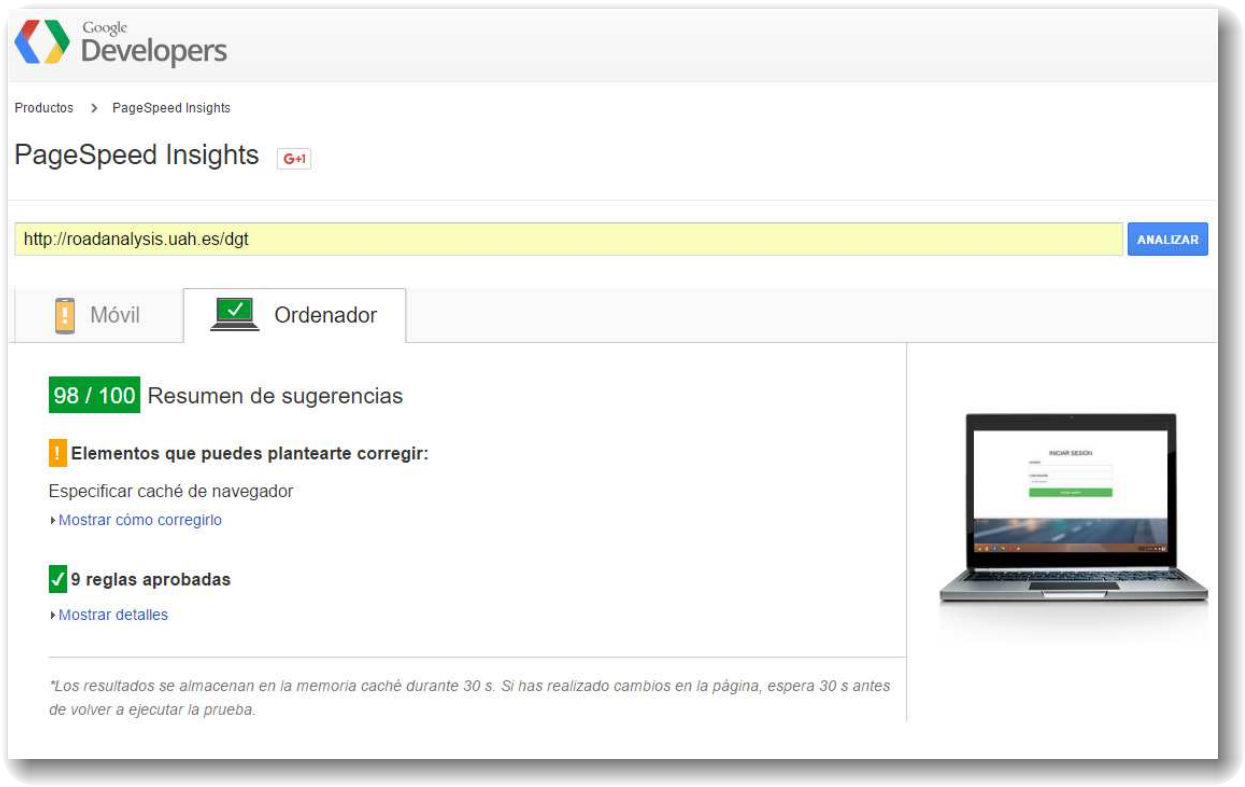

Figura 2.16: Rendimiento en PC.

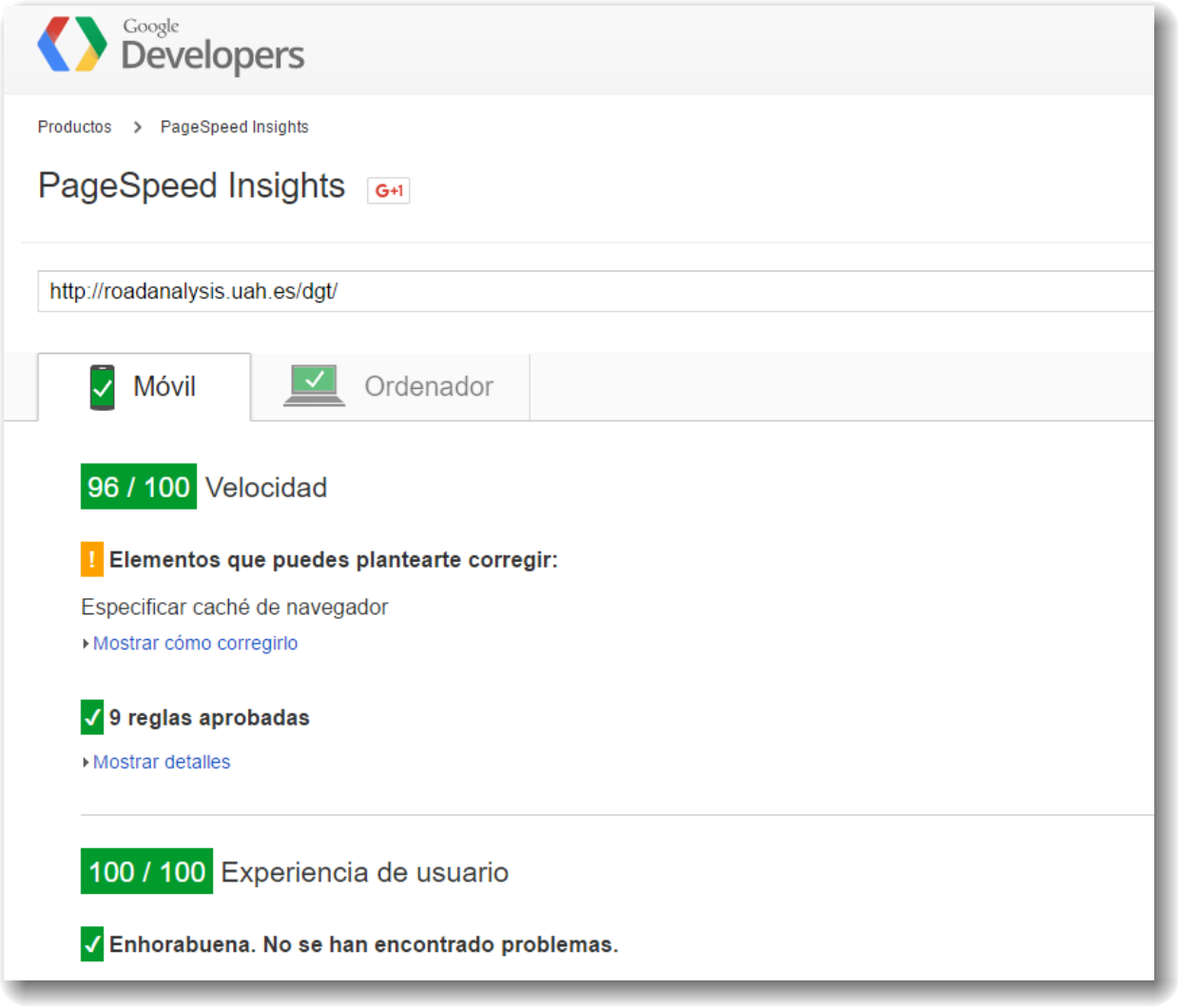

Figura 2.17: Rendimiento en móvil.

2.4. Valoración de precisión del sistema

Para valorar la precisión del sistema, en cuanto a cuenta de vehículos se refiere, también se estudian los resultados obtenidos por el módulo de visión artificial a lo largo de una jornada completa. Para ello las Figuras 2.18 y 2.19 recogen las imágenes obtenidas por una cámara con sus correspondientes imágenes resultado de la estimación del mapa de densidad. Estas imágenes muestran en orden de izquierda a derecha y de arriba a bajo, con un periodo de 20 minutos, las imágenes obtenidas a lo largo de todo el día, divididas en dos conjuntos. Por cada imagen original se muestra debajo de ella la imagen del mapa de densidad resultado del procesado correspondiente a ella.

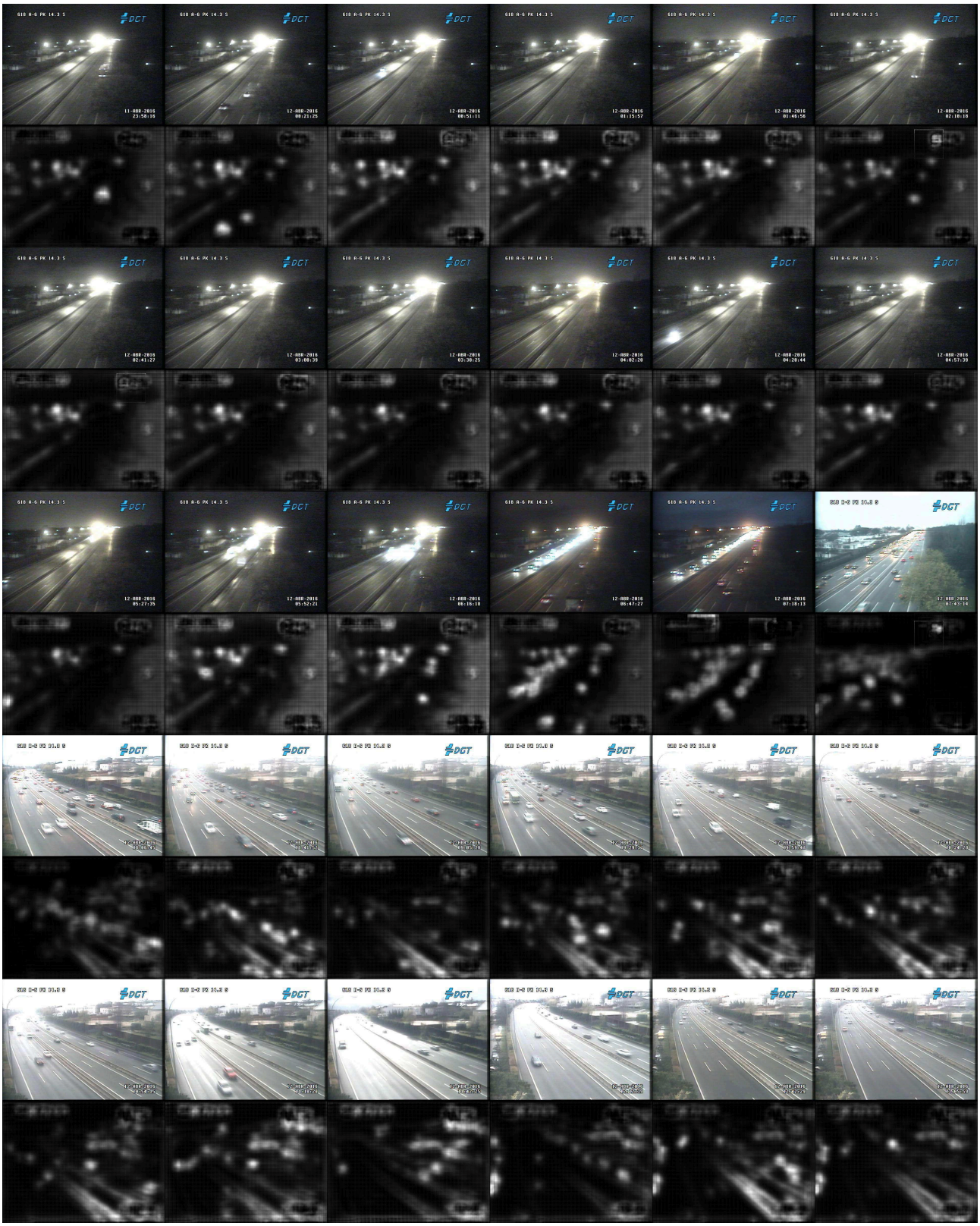

Figura 2.18: Conjunto de imágenes obtenidas en el sistema (00:04-13:12).

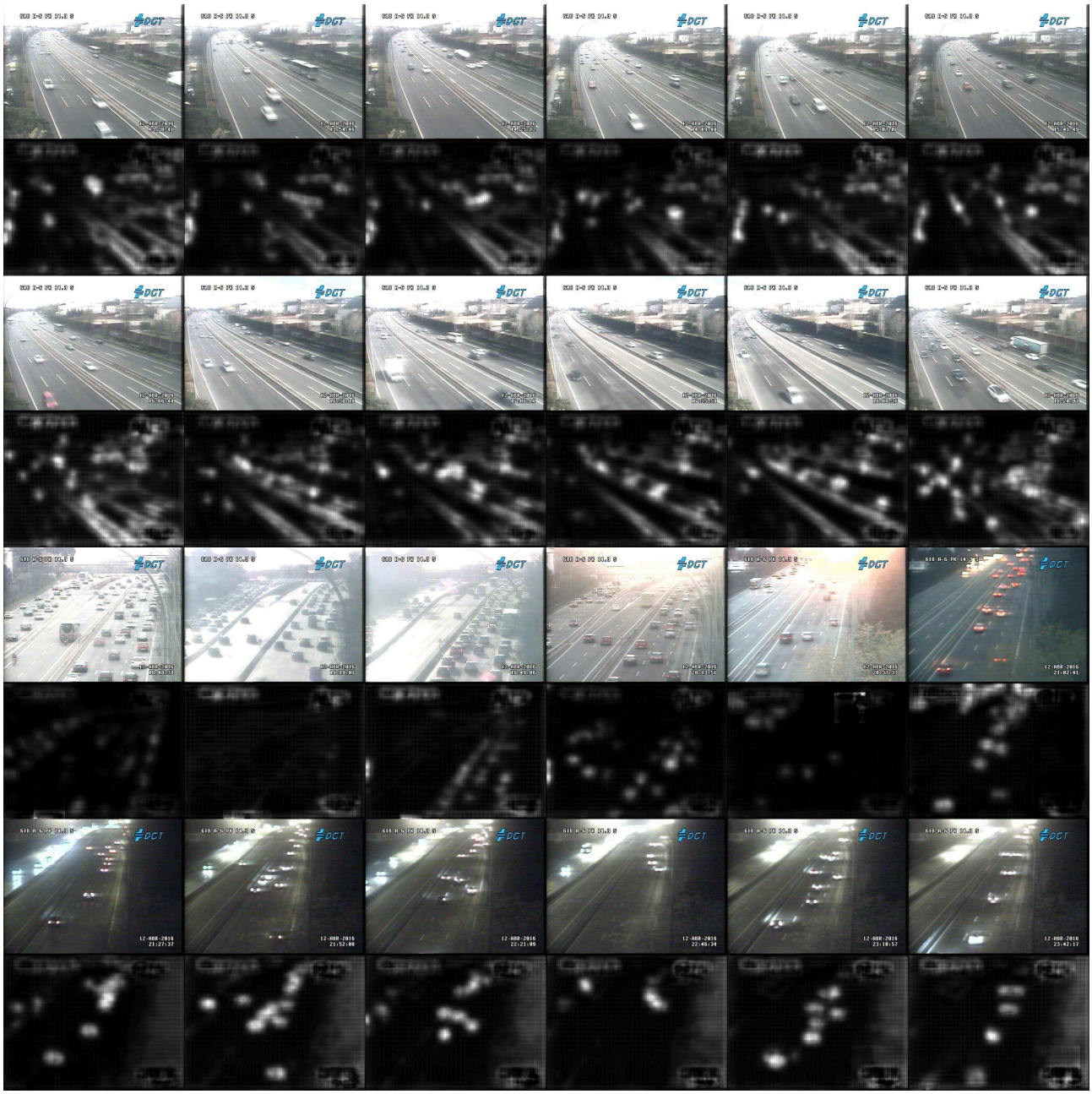

Figura 2.19: Conjunto de imágenes obtenidas en el sistema (13:37-23:49).

Además, para valorar mejor los resultados obtenidos, la Figura 2.20 muestra una gráfica con los índices medios del tráfico obtenidos para las imágenes de las Figuras 2.18 y 2.19, indicando la hora en la que se obtiene la imagen en el eje x y el IMT en el eje y.

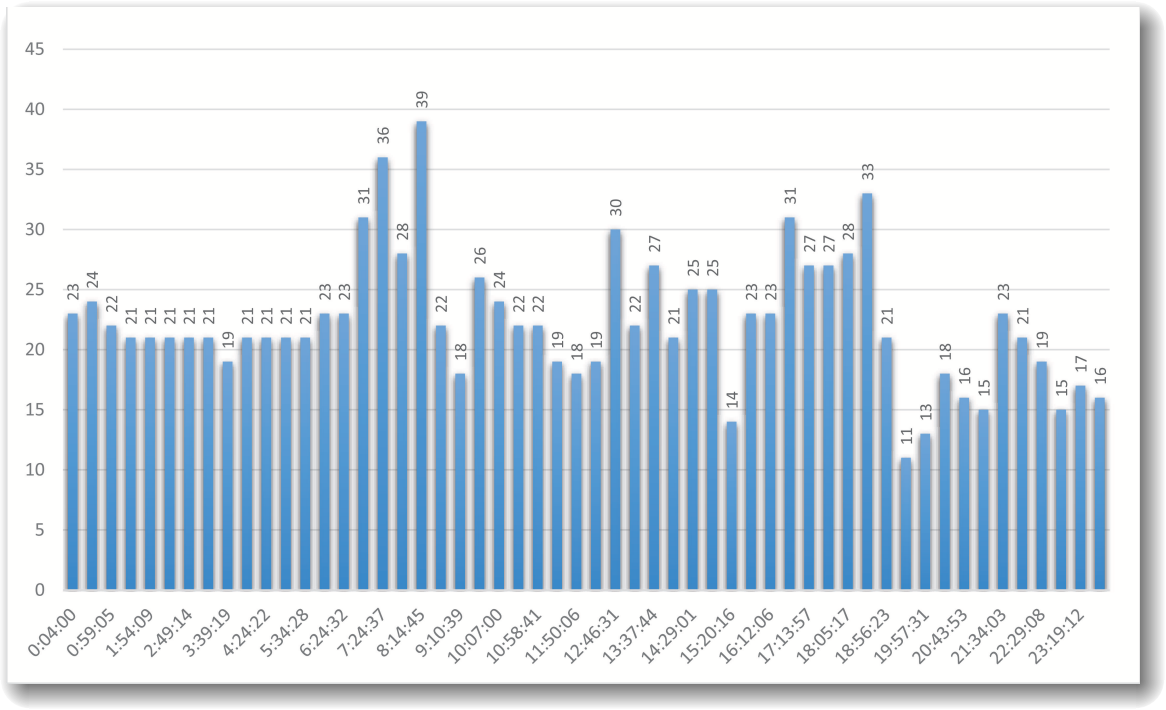

Figura 2.20: Índices medios del tráfico de las imágenes de las Figuras 2.18 y 2.19.

Como se puede observar, por la noche el reflejo de las farolas hace mella en el procesado ya que lo confunde con vehículos y el sistema parece establecer un mínimo de 21 vehículos por hora en la estimación. Aún así, el módulo reconoce la presencia de vehículos en la carretera. A medida que avanza el día y se apagan las farolas, deja de existir tal problema, realizando una buena estimación incluso con la existencia de un alto tránsito de vehículos. También existen problemas dados por fuentes externas como los reflejos causados por el sol o a consecuencia de la lluvia, los cuales influyen en el resultado estimado. Además, existen problemas asociados al hardware empleado en la adquisición de fotogramas, ya que la resolución de las imágenes obtenidas por las cámaras es de 640x480 píxeles. En última instancia, cabe destacar que la capa de información implementada por la DGT sobre las imágenes obtenidas por las cámaras, siguen una distribución variable, como muestra la Figura 2.21, por lo que esto también influye en el tratado de la imagen.

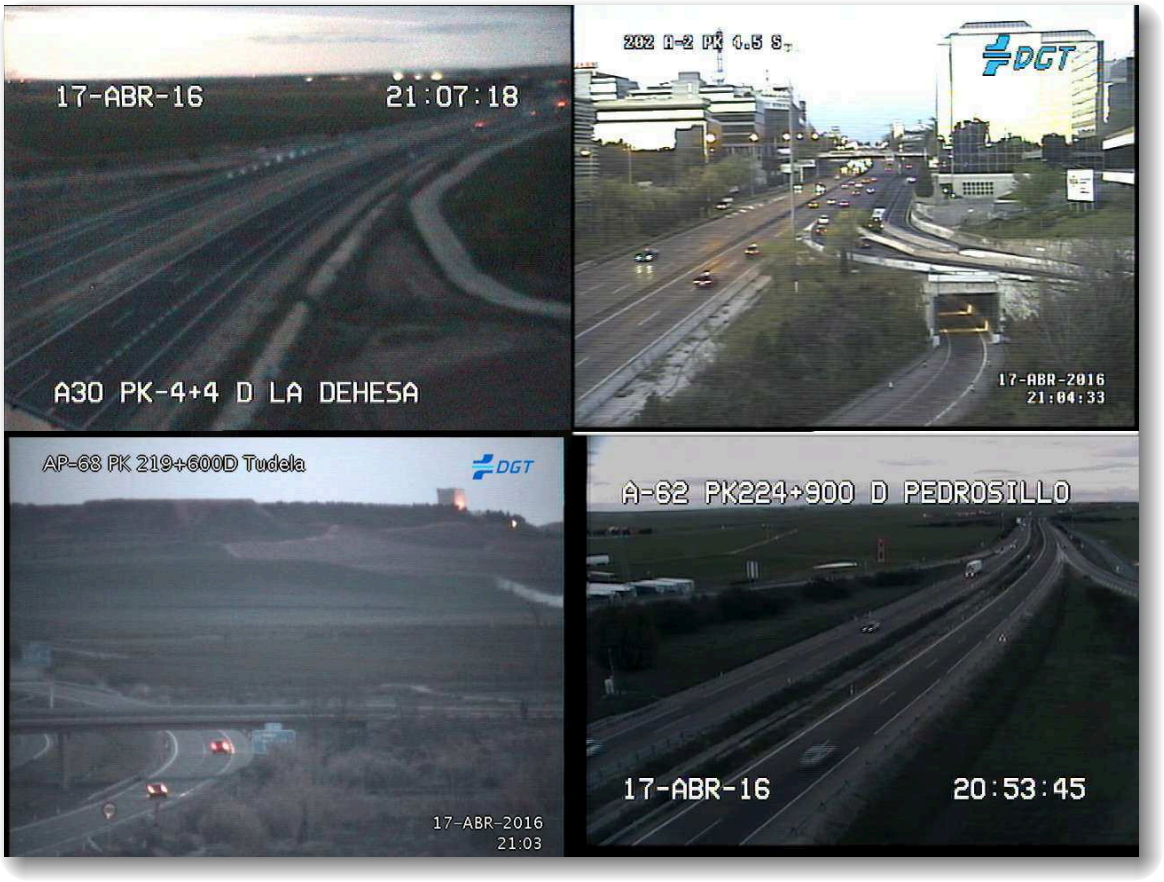

Figura 2.21: Diferencia de capas de información en imágenes de cámara.

Capítulo 3

Aplicación web

La aplicación web desarrollada permite configurar las cámaras con las que se desea trabajar, dando un margen de 1262 cámaras desplegadas por la DGT, localizadas a lo largo de toda España, como se puede ver en la Figura 3.1. Además muestra los resultados que se obtienen de cada una de las cámaras que hayan sido activadas, así como un histórico diario de éstas.

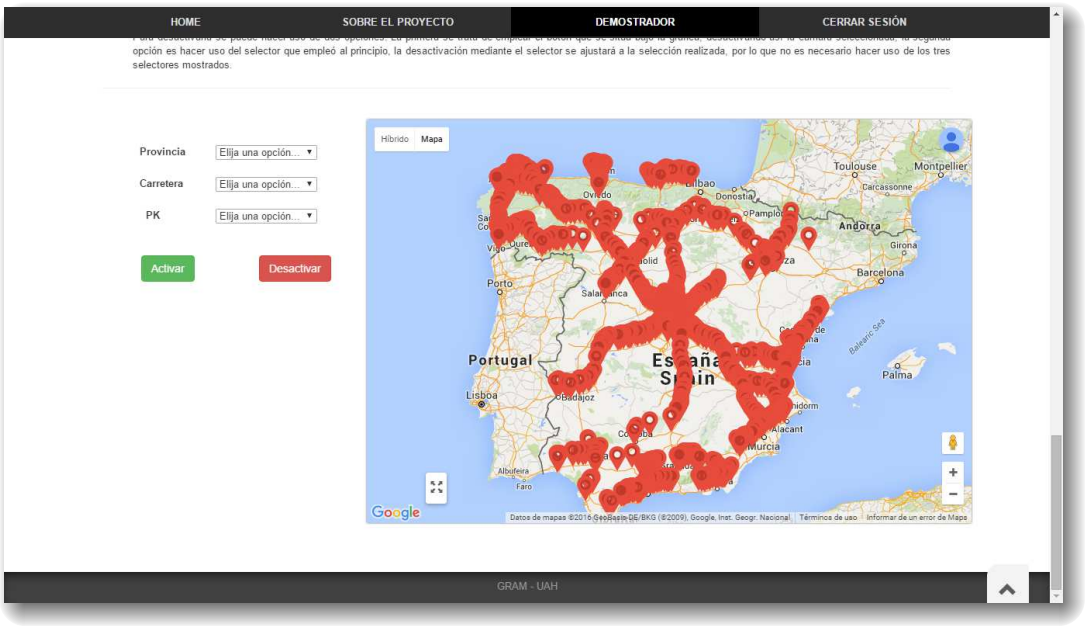

Figura 3.1: Rango de cámaras disponibles en el sistema.

Este capítulo aborda el modo de uso de la aplicación web desde un punto de vista *Front-end*, de modo que el usuario sea capaz de activar y desactivar las cámaras que desee, así como ver los resultados obtenidos por el sistema.

3.1. Descripción de la Interfaz Web

Para acceder a la interfaz web se provee una dirección en la que se encuentra alojada: `http://roadanalysis.uah.es/dgt`.

En la Figura 3.2 se muestra un mapa de la web. De esta forma se pretende aclarar la estructura de la interfaz web antes de pasar a comentar cada una de las partes de ésta y explicar el modo de uso del demostrador.

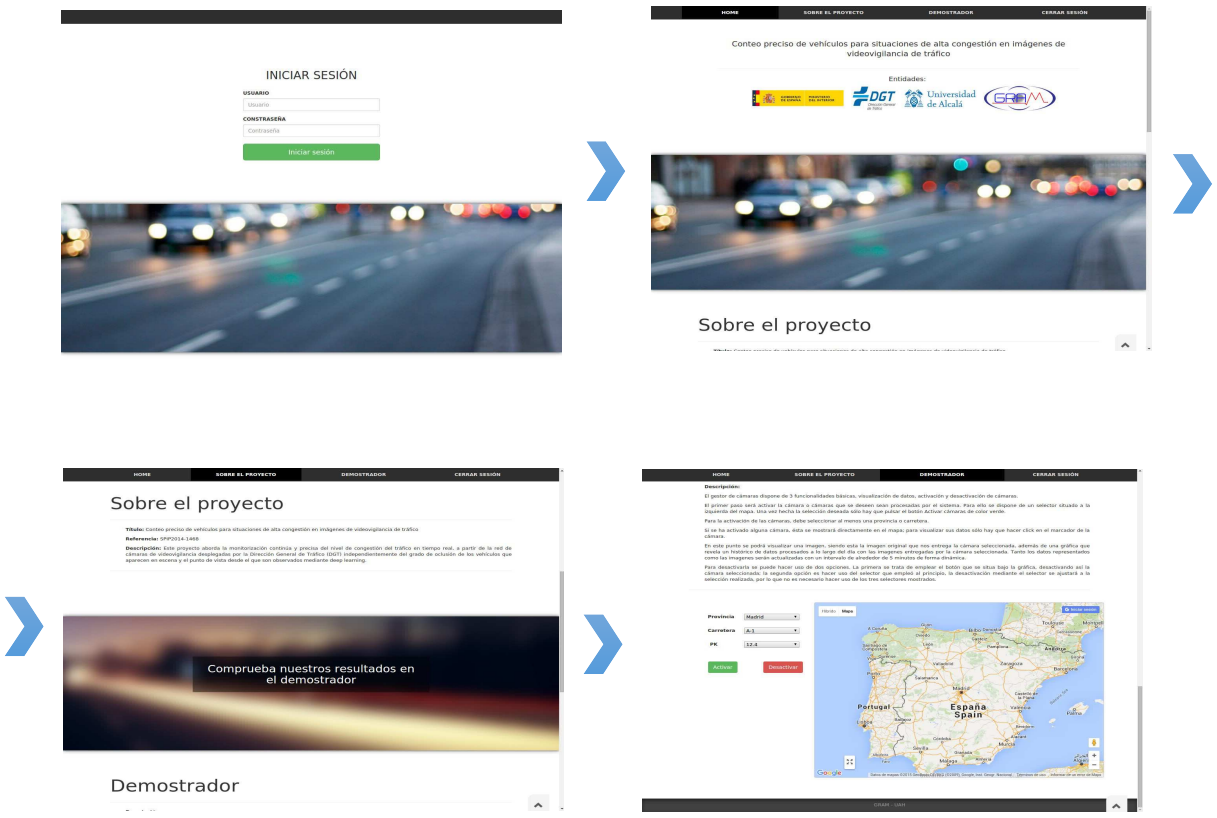

Figura 3.2: Mapa del sitio web.

Para comenzar, es necesario iniciar sesión en el sistema. Para hacerlo se deben introducir las credenciales de autenticación que se encuentran en el Apéndice A. Como la petición de inicio de sesión para acceder a la aplicación es únicamente para tener un control del acceso, la conexión se realiza mediante el protocolo HTTP (puerto 80), de tal modo que la conexión no viaja cifrada y la contraseña se envía como texto en plano.

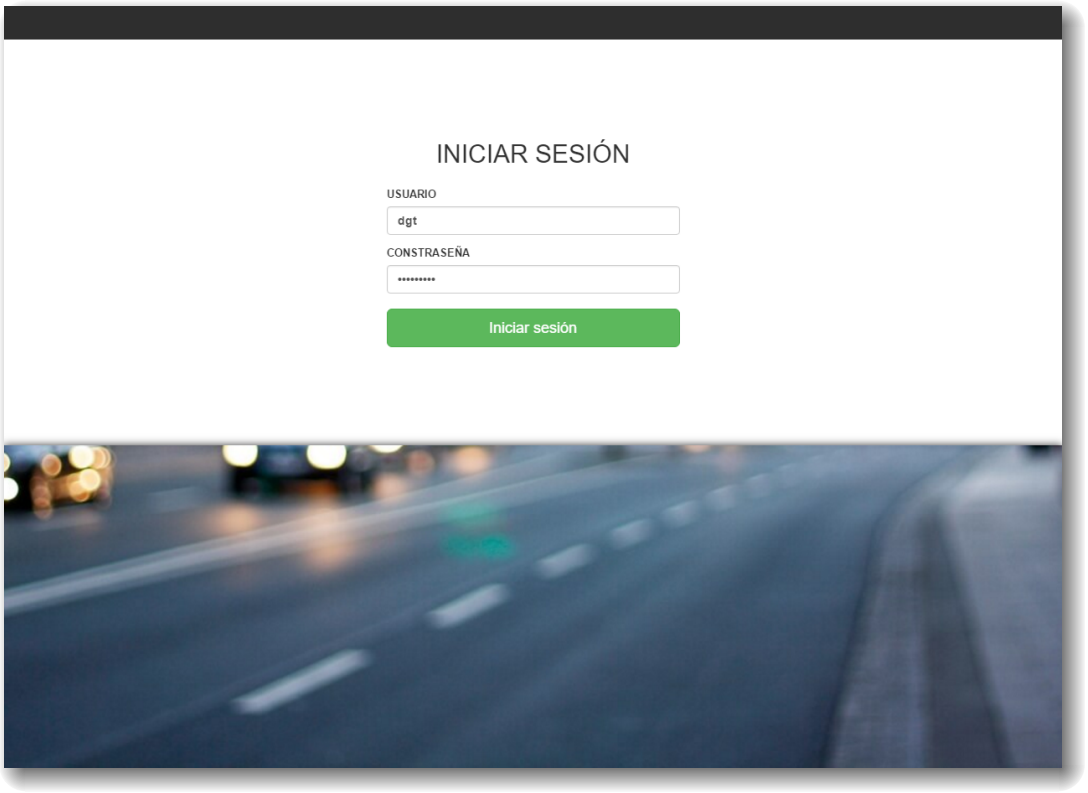

Figura 3.3: Login de la aplicación web.

Una vez hayamos entrado, nos encontraremos en la pantalla de inicio, reflejada en la Figura 3.4. Ya que el trabajo realizado en este TFG se enmarca dentro de un proyecto de investigación dentro del marco nacional, llevado a cabo por el grupo de investigación GRAM de la Universidad de Alcalá, además de tener una sección en la que pueden comprobar los resultados del sistema propuesto, se han añadido dos secciones más en las que se hace referencia a datos acerca del proyecto desarrollado.

En la pantalla de inicio se realiza una introducción a la interfaz de la aplicación web indicando el título del proyecto para el que ha sido desarrollada y las diferentes entidades involucradas (Ministerio del Interior, DGT, Universidad de Alcalá y Grupo de investigación GRAM).

Figura 3.4: Inicio de la aplicación web.

Para movernos por el sitio web, podemos hacerlo mediante la barra lateral de desplazamiento o las pestañas situadas en la barra superior. Si se desea salir de la página, por motivos de seguridad, se recomienda hacer uso del botón de *Cerrar Sesión*. Si no se sale de la sesión, aunque se cierre el navegador, cuando se vuelva a acceder a la aplicación web no será necesario volver a introducir los credenciales por lo que el acceso será directo.

En la sección llamada *Sobre el Proyecto*, como se puede ver en la Figura 3.5, se muestra un fragmento de información acerca del proyecto. En él destaca la referencia administrativa así como una breve descripción acerca del proyecto.

Figura 3.5: Pestaña, sobre el proyecto.

En última instancia se encuentra la sección del demostrador donde vamos a comprobar el funcionamiento del sistema propuesto. La finalidad de esta sección se centra en mostrar una aplicación que permita mostrar el potencial del sistema desarrollado.

Para ello se ha creado una interfaz sencilla e intuitiva, como presenta la Figura 3.6, que permite visualizar un histórico diario de los datos estimados mediante técnicas de visión por computador sobre las imágenes obtenidas de forma directa de las cámaras desplegadas por la DGT.

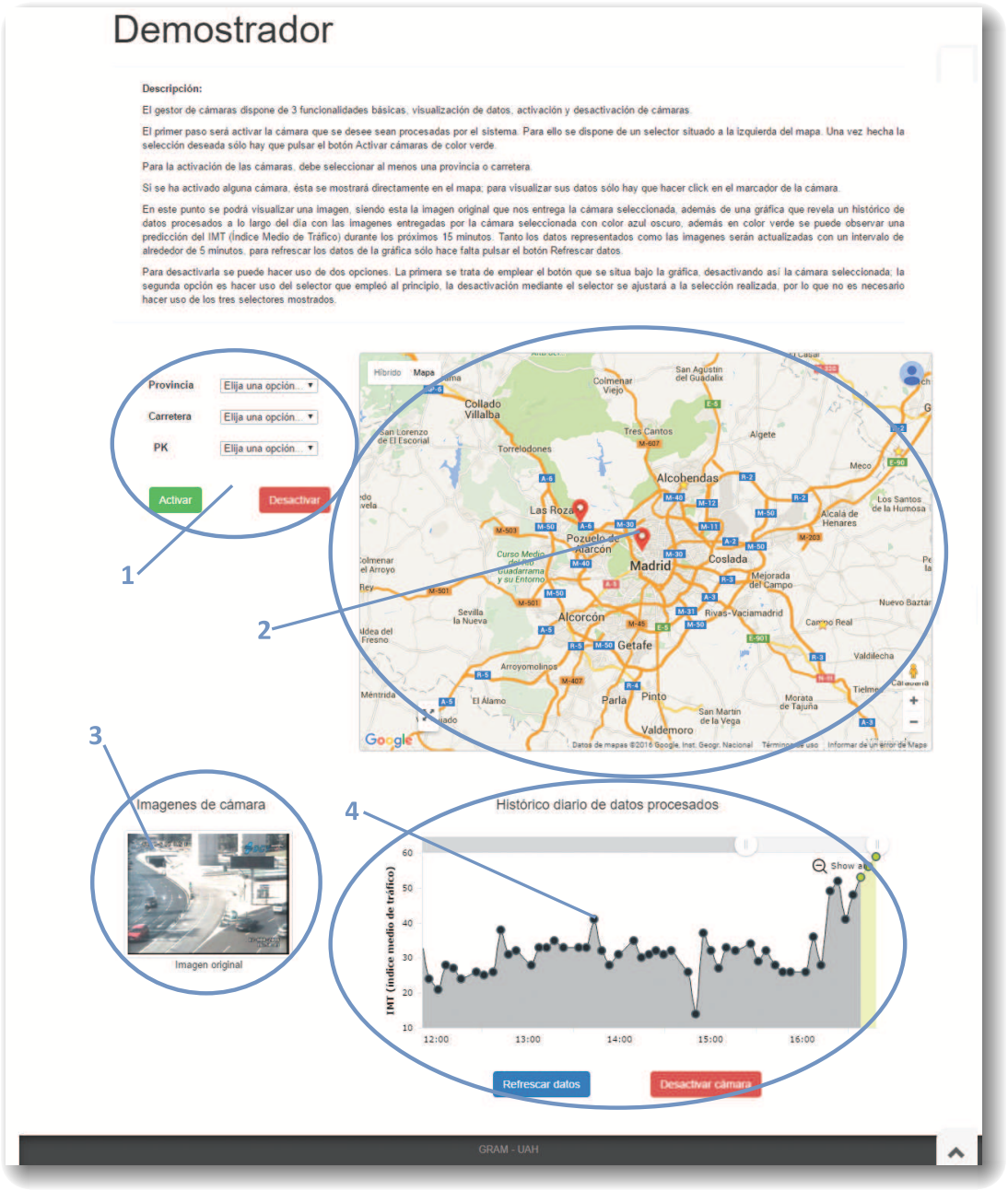

Figura 3.6: Interfaz principal de la aplicación web.

El demostrador ofrece las siguientes funcionalidades:

1. Gestión de cámaras activas en el sistema.
2. Geolocalización de cámaras activas.
3. Visualización de imagen obtenida por la cámara
4. Histórico diario del IMT estimado.

3.1.1. Gestión de cámaras activas en el sistema

Aunque el sistema permite trabajar con 1262 cámaras desplegadas por toda la península, debido al consumo de recursos que supondría realizar una estimación del índice medio del tráfico de 1262 imágenes, con un margen de 5 minutos entre cada ráfaga de procesado, sólo algunas de ellas se encuentran activas en el sistema.

Por tanto, para habilitar o deshabilitar el procesado de datos de alguna cámara disponible en el sistema, se puede hacer uso del selector de cámaras indicado en la Figura 3.7. Debido al empleo de AJAX [41], las opciones de carretera y Punto Kilométrico (PK), no son rellenadas hasta que no se haga uso del selector previo, siendo la provincia y carretera, respectivamente. Una vez hecha la selección, sólo hay que pulsar el botón *Activar* o *Desactivar* para habilitar o deshabilitar la cámara deseada.

Figura 3.7: Gestión de cámaras activas en el sistema.

Cabe destacar que debido al alto consumo de recursos(tanto RAM, como CPU y GPU) que supone realizar la estimación del IMT mediante visión artificial, y teniendo en cuenta que este escenario trata de demostrar el sistema y no de crear un entorno en producción, sólo es posible activar una cámara en cada acción. De este modo no se permite activar todas las cámaras presentes en una carretera o en toda una provincia. Por el contrario, el sistema si permite desactivar todas las cámaras localizadas a lo largo de una carretera o provincia.

3.1.2. Geolocalización de cámaras activas

El mapa muestra un marcador geolocalizando el punto exacto en el que se encuentran las cámaras activas en el sistema, véase Figura 3.8. Por tanto, cuando se activa alguna cámara, en el mapa aparecerá un nuevo marcador indicando la posición de ésta. Del mismo modo al desactivar cualquier cámara, el marcador correspondiente desaparecerá del mapa.

Esto es posible al almacenar la latitud y longitud correspondiente a cada una de las cámaras disponibles en el sistema en la tabla 2.1 de la base de datos.

Para visualizar los datos correspondientes a alguna cámara, sólo es necesario hacer clic sobre el marcador que geolocaliza la cámara situado en el mapa. En este punto se deslizará la página mostrando la imagen obtenida por la cámara seleccionada y un histórico diario de datos procesados.

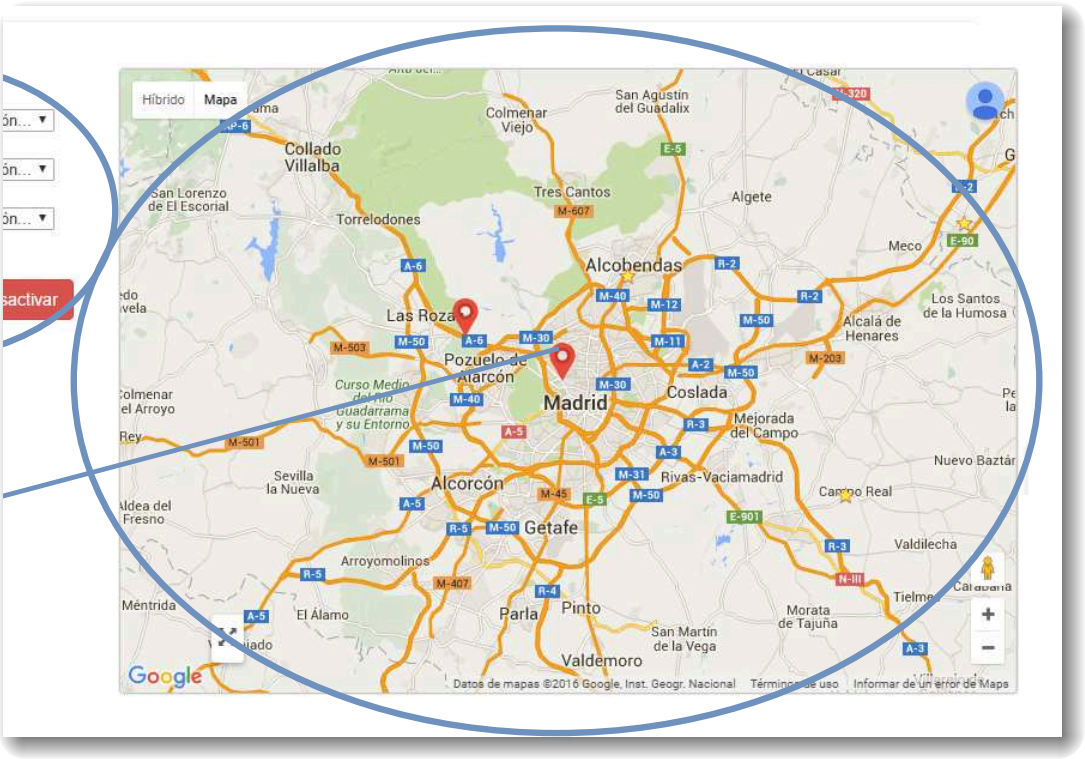

Figura 3.8: Geolocalización de cámaras activas en el sistema.

3.1.3. Visualización de imagen obtenida por la cámara

Para tener una noción más visual del estado del tráfico, la sección mostrada en la figura 3.9 muestra la última imagen obtenida por la cámara de la DGT, la cual es procesada para realizar la estimación del tráfico.

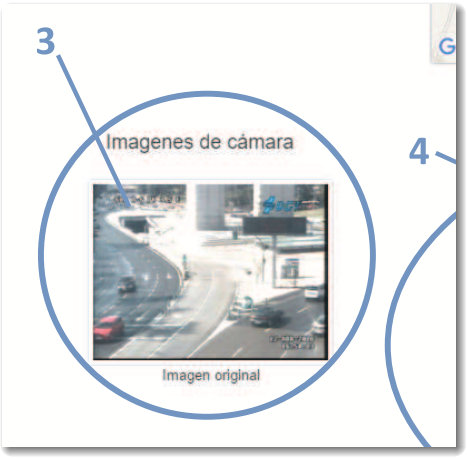

Figura 3.9: Visualización de imagen obtenida por la cámara.

3.1.4. Histórico diario del IMT estimado

Como se puede ver en la Figura 3.10, el histórico de datos procesados representa, con marcadores de color azul marino, el IMT estimado por el módulo de visión artificial mediante Deep Learning, en relación a la hora en la que la imagen original ha sido obtenida, a lo largo de todo el día, véase el elemento 1 de la Figura 3.10. Además, los marcadores de color verde claro, muestran una predicción del IMT durante los 15 minutos posteriores al instante de la última estimación realizada por el SVA, mediante la técnica de regresión sesgada o *Ridge Regression*, véase el elemento 2 de la Figura 3.10.

La gráfica permite un uso interactivo gracias a la barra superior. Esta permite realizar un zoom, ajustando el intervalo horario a mostrar, para ver mejor los datos representados y desplazarla para moverse a lo largo del eje x de la gráfica, véase el elemento 3 de la Figura 3.10.

Tanto los datos representados, como las imágenes, serán actualizadas con un intervalo de 5 minutos debido al límite establecido por la DGT para la adquisición de imágenes. Para refrescar los datos de la gráfica es necesario hacer uso del botón *Refrescar datos* con fondo azul, localizado bajo la gráfica, véase el elemento 4 de la Figura 3.10.

Además, el botón de *Desactivar cámara* en color rojo, desactiva la cámara seleccionada en el sistema, véase el elemento 5 de la Figura 3.10.

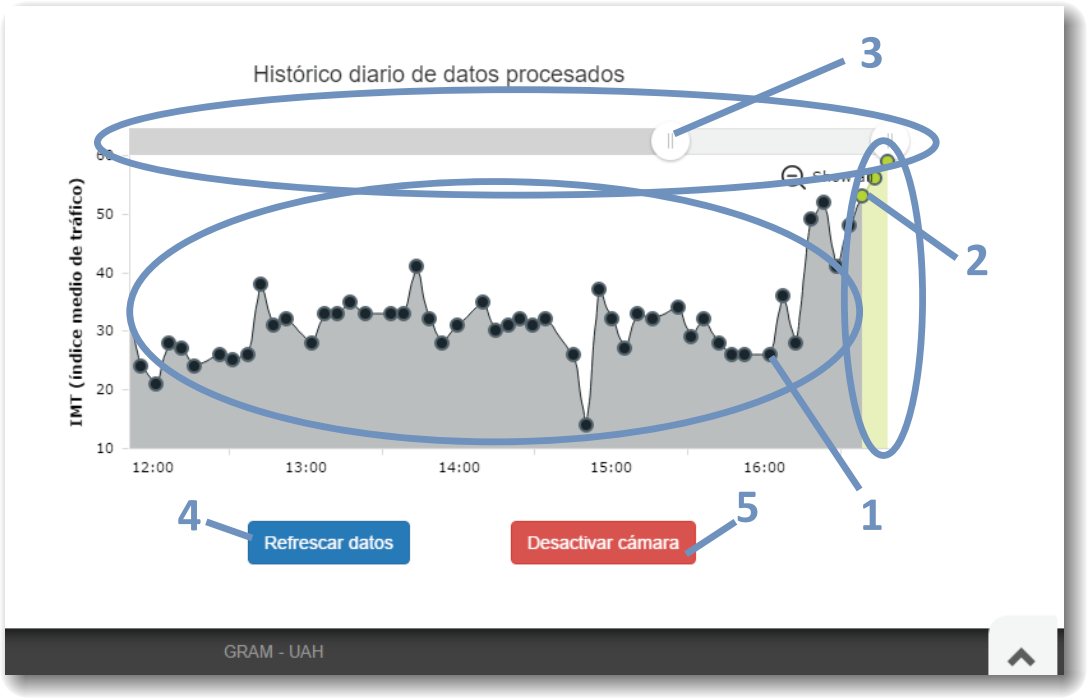

Figura 3.10: Histórico diario del IMT estimado.

Además, dado que la aplicación web se ha realizado con Bootstrap[3], que se trata de un framework HTML, CSS y JS orientado al desarrollo de diseños *sensibles* o *responsive*, la interfaz se adapta a cualquier dispositivo que se emplee, como se puede ver en la Figura 3.11.

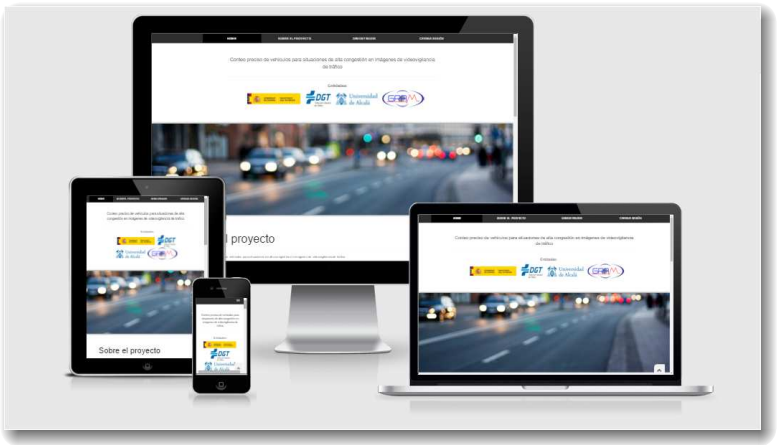

Figura 3.11: Diseño responsive.

Capítulo 4

Conclusiones y futuras líneas de trabajo

4.1. Conclusiones

Con la realización de este TFG se han conseguido los siguientes objetivos:

1. Presentación de una alternativa al actual sistema de conteo de vehículos formado por espiras electromagnéticas. Esta alternativa permite reducir ampliamente los costes de gestión y mantenimiento de las mismas. Se puede decir además que el sistema es escalable, pues se puede ampliar la capacidad del mismo únicamente dando de alta nuevas cámaras en la base de datos que recoge las cámaras disponibles.

2. Desarrollo de una aplicación web de estilo adaptable y contenido dinámico con el motor de base de datos MySQL, capaz de geolocalizar y monitorizar las cámaras de videovigilancia de la DGT, mostrando el nivel de congestión que presentan en tiempo real.

3. Incorporación de un sistema de predicción de la estimación de vehículos.

4. Diseño de una arquitectura distribuida con la integración de múltiples servidores, una base de datos, y una aplicación encargada del procesado de imágenes mediante visión artificial empleando la GPU en su proceso.

Al inicio del proyecto se planteó el uso del micro framework escrito en python, Flask frente al framework Django. Tras desarrollar el proyecto y debido al carácter de éste, la potencia y sencillez que ofrece flask resultó ser suficiente.

Durante el desarrollo del proyecto se ha encontrado principalmente un inconveniente que finalmente ha sido resuelto. La conexión de la aplicación web con la base de datos se cerraba al cabo de unos días y debido a esto, las operaciones que dependen de esta conexión en la interfaz web retornaban un error del tipo 500 (error interno del servidor). Esto se solucionó comprobando la conexión con la base de datos cada vez que un usuario

ingresa en la página web, y reiniciando la conexión en caso de error.

En cuanto al rendimiento del sistema, debido al entorno en el que se encuentra, que sea capaz de soportar 50 cámaras en funcionamiento de manera simultánea resulta sorprendente. Este límite se establece debido a la capacidad del SVA ya que ejerce de cuello de botella en el sistema. Por último, la gráfica que muestra el histórico del IMT asociado a una cámara se podría haber mejorado implementando un refresco automático.

4.2. Futuras lineas de trabajo

Existen dos importantes vías en las que centrarse para mejorar este TFG. La primera sería mejorar la experiencia de uso de la aplicación web, implementando un refresco automático de los datos mostrados, en la gráfica del histórico diario del IMT asociado a una cámara. Esto se puede conseguir empleando un protocolo que permita al servidor generar eventos automáticos como Server Sent Events (SSE) [38]. De este modo cuando el SVA realice el procesado de nuevas imágenes, envía una señal a la aplicación web que permite obtener los IMT de la cámara activa y compararlos para enviar al cliente el valor del nuevo IMT estimado junto a las nuevas predicciones. El cliente recibiría estos datos mediante AJAX y actualizaría la gráfica, eliminando las predicciones previas y añadiendo el nuevo IMT estimado junto a las nuevas predicciones. Los archivos *webApp.py* y *scripts.js* contienen parte del código necesario para llevar a cabo esta mejora y posibilitan continuar con su desarrollo.

La segunda línea de trabajo futuro pasa por generar una aplicación móvil con Apache Córdoba [5] o PhoneGap [30] para aprovechar la aplicación web. Esta aplicación podría, gracias a la localización GPS del dispositivo, emitir alertas acerca del estado del tráfico a partir de una ruta establecida previamente, o incluso a partir de una nueva ruta si el vehículo se desvía de la trayectoria preestablecida.

Capítulo 5

Pliego de condiciones

Para la correcta utilización del sistema desarrollado en este trabajo, se debe disponer de un hardware y un software que cumpla unos requisitos mínimos.

5.1. Requisitos de Hardware

5.1.1. Servidor de Aplicación Web

- PC con procesador 32/64 bits.
- Interfaz de red Ethernet/WIFI.
- 2GB de memoria RAM o superior.
- Al menos 1GB de memoria libre en HDD.

5.1.2. Servidor de Procesado de Imagen

- PC con procesador 32/64 bits.
- Interfaz de red Ethernet/WIFI.
- 8GB de memoria RAM o superior.
- Al menos 3GB de memoria libre en HDD.
- Tarjeta gráfica con soporte CUDA [7] de al menos 2GB de memoria dedicada.

5.2. Requisitios de Software

5.2.1. Servidor de Aplicación Web

- Ubuntu 12.04 o superior.
- Apache2
- MySQL
- Python
- pip
- Flask
- urllib3
- mysqldb
- flask-mysql
- flask-compress

Se recomienda emplear el gestor de paquetes *pip* para realizar la instalación de las dos últimas dependencias (flask-mysql, flask-compress).

5.2.2. Servidor de Procesado de Imagen

- Ubuntu 12.04 o superior.
- Python
- pip
- cython
- numpy
- h5py
- scipy
- ibus
- libibus-1.0-5
- vigra

- joblib
- skimage
- opencv
- mysqldb
- scikit-learn

Es importante que el paquete *scikit-learn* sea la versión 0.16.0 o superior, por ello se recomienda también su instalación mediante el gestor de paquetes *pip*.

Para más información refiérase al archivo *README.md* localizado en la raíz del CD adjunto.

Apéndice A

Datos de acceso

Por motivos de seguridad para acceder a la aplicación web es necesario disponer de un usuario y contraseña.

Se recomienda revisar los ficheros de configuración de las aplicaciones donde se detalla el usuario y contraseña de la aplicación web y la base de datos.

- dgt/webApp/config.py
- gram-counting/config.py

Los datos de acceso establecidos por defecto para la aplicación web son los siguientes.

- **Usuario**: dgt
- **Contraseña**: tute.fod9

Apéndice B

Manual de instalación

Para continuar con la instalación del sistema cabe destacar que el sistema operativo, según se indica en el pliego de condiciones, debe ser Ubuntu versión 12.04 o superior.

B.1. Contenido del paquete comprimido

En los contenidos adjuntos, se encuentra el fichero README.md (escrito en lenguaje Markdown, por lo que se recomienda usar un visor destinado para su lectura aunque no es necesario). En él se detalla cómo poner en funcionamiento el sistema, sus dependencias y todos los ficheros de los que se compone el demostrador.

Este paquete se compone de dos aplicaciones diferenciadas en dos carpetas distintas.

- gram-counting/
- dgt/

La primera, *gram-counting/*, realiza el procesado de las imágenes; la segunda se almacena en el directorio *dgt/*, ésta es la encargada de la aplicación web.

A continuación se detallan las dependencias necesarias para cada aplicación.

Aplicación web:

```
$ sudo apt-get install python-pip python-flask python-urllib3 python-
   mysqldb

$ sudo pip install flask-mysql flask-compress
```

Aplicación para el procesado de imágenes:

```
$ sudo apt-get install python-pip cython python-numpy python-h5py python
    -scipy python-ibus libibus-1.0-5 python-vigra python-joblib python-
    skimage python-opencv python-mysqldb

$ sudo pip install scikit-learn
```

Además será necesario disponer de MySQL para almacenar las bases de datos. Por otro lado, es importante que el paquete scikit-learn sea la versión 0.16.0 o superior. Es por esto que se recomienda el uso sel gestor de paquetes pip para su instalación.

En el directorio *dgt/webApp/developing_tmp/* se dispone de un script para bash llamado *makerdb.sh* que es el encargado de crear la base de datos con las tablas necesarias. Sólo se necesita tener instalado un usuario con el nombre **dgt** en MySQL; para ello se recomienda hacer uso de la herramienta gratuita destinada a la gestión de MySQL a través de internet, phpMyAdmin, y que se disponga de permisos para poder crear bases de datos. Opcionalmente se puede crear la base de datos dgt desde phpMyAdmin y el script generará automáticamente las tablas asociadas.

Para acceder al gestor de MySQL, phpMyAdmin, se puede hacer desde el siguiente enlace, `http://localhost/phpmyadmin`. Para entrar por primera vez se debe emplear el usuario root y la contraseña del administrador de su equipo.

Una vez creadas, sólo queda importar la tabla cameras_dgt.sql contenida en la carpeta developing_tmp/.

B.2. Puesta en marcha del sistema

Antes de poner en marcha el sistema se debe comprobar los archivos de configuración localizados en la carpeta *other/conf/* en ambas aplicaciones, dentro de ellos se explica para que sirve cada parámetro. La aplicación web contiene tres archivos de configuración:

- *app_config.py*: Contiene toda la configuración asociada a la aplicación encargada de obtener las imágenes de las cámaras de la dgt y la aplicación web.
- *gunicorn_config.py*: Almacena la configuración relacionada con el servidor de la aplicación web gunicorn.
- *logging.conf*: Contiene la configuración para gestionar los registros o logs. Se recomienda modificar no modificar este archivo.

La aplicación de procesado de imágenes contiene los siguientes archivos de configuración:

- *config.py*: Almacena la configuración relacionada con la aplicación de procesado de imágenes.
- *logging.conf*: Contiene la configuración para gestionar los registros o logs. Se recomienda modificar no modificar este archivo.

Para arrancar la aplicación web se han creado dos scripts que demonizan, empleando Upstart, el sistema de obtención de imágenes y el servicio de aplicación web, por lo que sólo es necesario mover los archivos *dgtWebApp.conf* y *dgt2db.conf* localizados en la carpeta *developing_tmp/* al directorio */etc/init/*:

```
$ sudo mv developing_tmp/dgt*.conf /etc/init/
```

Para arrancar los servicios sin tener que reiniciar el servidor se puede hacer del mismo modo que cualquier otro servicio:

```
$ sudo service dgtWebApp start
$ sudo service dgt2db start
```

La puesta en marcha de la aplicación de procesado de imágenes pasa por acceder del mismo modo a su directorio,

```
$ cd gram-counting/
```

Unicamente en su instalación debemos ejecutar el script compile.sh, una vez ejecutado no será necesario volverlo a correr.

```
$ sh compile.sh
```

Y siempre que queramos lanzar la aplicación de procesado (por primera vez, cortes de luz, etc) ejecutaremos el script run.sh

```
$ chmod 755 run.sh && ./run.sh
```

Las carpetas *other/logs* en ambas aplicaciones almacenan la información de registro de cada aplicación usando el sufijo *_error* si almacena la información de error y *_debug* si contiene información de debug.

Bibliografía

[1] Project DGT 2014. Accurate count of vehicles for situations of high congestion in traffic surveillance images. http://agamenon.tsc.uah.es/Investigacion/gram/projects/dgt2014/index.html. XIX, 17

[2] Apache. https://httpd.apache.org. 18

[3] Bootstrap. A sleek, intuitive, and powerful mobile first front-end framework for faster and easier web development. http://getbootstrap.com. 30, 46

[4] Bulma. Desarrollo web extremo. https://web.archive.org/web/20140823154111/http://bulma.net/body.phtml?nIdNoticia=734. 7

[5] Apache Cordova. Mobile apps with html, css & js. https://cordova.apache.org. 48

[6] CSS3. cascading style sheets, version 3. http://www.w3.org/Style/CSS. 19

[7] NVIDIA CUDA. http://www.nvidia.es/object/cuda-parallel-computing-es.html. XIX, 17, 49

[8] Ayuntamiento de Madrid. Tráfico madrid. http://informo.munimadrid.es/informo/tmadrid/tmadrid.php. 6

[9] Google Developers. Pagespeed insights. https://developers.google.com/speed/pagespeed/insights. 30

[10] DGT. Dirección general de tŕafico. http://infocar.dgt.es/etraffic. 5

[11] Django. The web framework for perfectionists with deadlines. https://www.djangoproject.com.

[12] Flask. Web developement, one drop at a time. http://flask.pocoo.org. 21

[13] Python Software Foundation. https://docs.python.org/2/library/select.html. 16

[14] Gevent. http://www.gevent.org. 19

[15] Google. Google maps. https://www.google.es/maps. 5

[16] Google. Google maps and waze, outsmarting traffic together. https://googleblog.blogspot.com.es/2013/06/google-maps-and-waze-outsmarting.html. 5

[17] Google. Google maps transit. https://www.google.com/landing/transit/index.html. 5

[18] Google. Optimizing encoding and transfer size of text-based assets. https://developers.google.com/web/fundamentals/performance/optimizing-content-efficiency/optimize-encoding-and,transfer?hl=en. 29

[19] GRAM. Multisensorial analysis and recognition group. http://agamenon.tsc.uah.es/Investigacion/gram/.

[20] Gunicorn. http://gunicorn.org. 18

[21] HTML5. Hypertext markup language, version 5. http://www.w3.org/TR/html5. 19

[22] Javascript. Js. https://www.javascript.com. 19

[23] Jinja2. http://jinja.pocoo.org/docs/latest. 19

[24] jQuery. The write less, do more, javascript library. https://jquery.com. 19

[25] Deep Learning. deep machine learning. https://en.wikipedia.org/wiki/Deep_learning. XIX, 2

[26] Zisserman A Lempitsky, V. Learning to count objects in images. in: Nips. (2010). https://www.robots.ox.ac.uk/~vgg/publications/2010/Lempitsky10b/lempitsky10b.pdf. XIX, 8, 17, 23

[27] Libevent. an event notification library. http://libevent.org/. 19

[28] MyISAM. http://www.ecured.cu/MyISAM. 11

[29] MySQL. https://www.mysql.com. 11

[30] Adobe PhoneGap. http://phonegap.com. 48

[31] PHP. https://secure.php.net/manual/es/index.php. 18

[32] Python. https://www.python.org. 14

[33] Python. Estructuras de datos. http://docs.python.org.ar/tutorial/2/datastructures.html. 14

[34] Scikit-learn. Sklearn linear_model ridge. http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[35] Sygic. Sygic gps navigation. http://www.sygic.com/. 2

[36] TomTom. https://www.tomtom.com. 2

[37] Ubuntu. An open source software platform. http://www.ubuntu.com. 8

[38] W3Schools. Html5 server-sent events. http://www.w3schools.com/html/html5_serversentevents.asp. 48

[39] Waze. Waze la aplicación de tráfico y navegación basada en la comunidad. https://wiki.waze.com/wiki/Main_Page. 2, 4

[40] Werkzeug. http://werkzeug.pocoo.org. 21

[41] Wikipedia. Asynchronous javascript and xml. https://en.wikipedia.org/wiki/AJAX. 19, 43

[42] Wikipedia. Cluster de computadores. https://en.wikipedia.org/wiki/Computer_cluster. 8

[43] Wikipedia. Front and back ends. https://en.wikipedia.org/wiki/Front_and_back_ends.

[44] Wikipedia. Lamp(software bundle. https://en.wikipedia.org/wiki/LAMP_(software_bundle). 18

[45] Wikipedia. Regularization_(mathematics). https://en.wikipedia.org/wiki/Regularization_(mathematics). 23

[46] Wikipedia. Web server gateway interface. https://en.wikipedia.org/wiki/Web_Server_Gateway_Interface. 18

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá