

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE INFORMÁTICA

MÁSTER EN DATA SCIENCE

TRABAJO FIN DE MÁSTER

**VISUALIZACIÓN STREAMING DE MODELO PREDICTIVO DE DATOS DE RED**

Autor: Ignacio Arias Barra

Tutor: José Felipe Ortega Soto

Curso académico: 2016/2017

**RESUMEN**

El trabajo presentado a continuación recoge el diseño, implementación y pruebas de un sistema de visualización en *streaming*. El principal objeto de visualización consistirá en la evaluación de un modelo de predicción de tráfico cursado a través de la red.

Con vistas en el desarrollo en el mundo del Big Data [17], poco a poco van siendo más demandados sistemas de análisis de grandes cantidades de datos que tomen decisiones o calculen resultados en tiempo real.

Estos modelos tienen como objetivo la optimización de recursos, así como la posibilidad de encontrar nuevas formas de explotación de los propios.

**AGRADECIMIENTOS**

*El presente trabajo de fin de máster ha sido realizado bajo la supervisión del profesor José Felipe Ortega Soto al que le quisiera agradecer el interés mostrado y todo el tiempo empleado en motivarme y hacer que todo lo sufrido durante el desarrollo y las pruebas hayan merecido la pena.*

*A Leticia Martín Martín, por darme todo su apoyo en los días en los que la jornada laboral se unía al desarrollo del trabajo.*

*A mi hermana Lucía Arias Barra, por mostrar su afecto e interés por el trabajo que he realizado.*

*A mis padres José Ignacio Arias Castro y María Jesús Barra Bermejo, por hacerme la vida más fácil durante el proyecto y mostrar su interés en que lo finalizara.*

*A mis amigos, que me han ayudado a evadirme y a disfrutar en los momentos difíciles.*

**ÍNDICE GENERAL**

[Introducción 1](#_Toc490234919)

[1.1 Estructura de la memoria 2](#_Toc490234920)

[Descripción del dataset 9](#_Toc490234921)

[2.1 Descripción general 9](#_Toc490234922)

[2.2 Elementos del sistema 10](#_Toc490234923)

[2.2.1 Sistemas embarcados en el vehículo 10](#_Toc490234924)

[2.2.2 Sistemas embarcados en el semáforo 11](#_Toc490234925)

[2.3 Integración del sistema 12](#_Toc490234926)

[2.4 Implementación del prototipo 13](#_Toc490234927)

[2.5 Tecnologías empleadas 14](#_Toc490234928)

[2.5.1 Vehículo 14](#_Toc490234929)

[2.5.2 Semáforo 21](#_Toc490234930)

[2.6 OBU y RSU 21](#_Toc490234931)

[2.7 Sistema radiante 24](#_Toc490234932)

[Arquitectura general 26](#_Toc490234933)

[20.1 Introducción 26](#_Toc490234934)

[20.2 Modelo de conducción 27](#_Toc490234935)

[20.3 Red de comunicación Ad-Hoc 32](#_Toc490234936)

[20.3.1 Topología de la red Ad-Hoc 32](#_Toc490234937)

[20.3.2 Preparación y evaluación de parámetros 33](#_Toc490234938)

[20.3.3 Configuración de Red 34](#_Toc490234939)

[20.3.4 Nivel de aplicación 36](#_Toc490234940)

[20.3.5 Capa física 37](#_Toc490234941)

[20.3.6 Parámetros de aplicación 38](#_Toc490234942)

[20.4 Implementación APP v-Road 40](#_Toc490234943)

[20.4.1 Functions.py 40](#_Toc490234944)

[20.4.2 Distance\_Converter.py 41](#_Toc490234945)

[20.4.3 Deg2utm.py 45](#_Toc490234946)

[20.4.4 Coords\_URJC.txt 45](#_Toc490234947)

[20.4.5 Proc\_data\_map.py 46](#_Toc490234948)

[20.4.6 Map\_converer.py 46](#_Toc490234949)

[20.4.7 Data\_semaphore 46](#_Toc490234950)

[20.4.8 Data\_car.py 46](#_Toc490234951)

[20.4.9 Controller.py 47](#_Toc490234952)

[20.4.10 Interfaz.py 51](#_Toc490234953)

[20.4.11 Car.py 52](#_Toc490234954)

[20.5 Implementación APP Semáforo 55](#_Toc490234955)

[20.5.1 Data\_semaphore.py 55](#_Toc490234956)

[20.5.2 Semaphore.py 56](#_Toc490234957)

[Casos de estudio 58](#_Toc490234958)

[24.1 Realización de pruebas 58](#_Toc490234959)

[24.1.1 Pruebas en estático 58](#_Toc490234960)

[24.1.2 Pruebas en movimiento 61](#_Toc490234961)

[24.2 Análisis información recibida 62](#_Toc490234962)

[24.3 Demostración de sistema 67](#_Toc490234963)

[Conclusiones y líneas futuras 64](#_Toc490234964)

[35.1 Conclusiones finales 64](#_Toc490234965)

[35.2 Base y aprendizaje 65](#_Toc490234966)

[35.2.1 Conocimientos previos 65](#_Toc490234967)

[35.2.2 Conocimientos adquiridos 66](#_Toc490234968)

[35.3 Líneas futuras 67](#_Toc490234969)

[BIBLIOGRAFÍA 70](#_Toc423381750)

[SEGUIMIENTO DE PROYECTO 72](#_Toc423381751)

[VERSIONES 95](#_Toc423381752)

[CÓDIGO DEL PROGRAMA 102](#_Toc423381753)

##### CAPÍTULO I

# Introducción

La principal motivación de este trabajo es el desarrollo de un sistema de visualización de datos en *streaming* [20], en el cual se apliquen técnicas de *data science* [18] que se encuentran a la orden del día.

En el mundo tecnológico actual, la generación de datos por cualquier dispositivo se encuentra en un crecimiento constante, prácticamente exponencial. Esto genera la necesidad de crear nuevas técnicas de procesado y modelado de datos en grandes cantidades, así como nuevas tecnologías para llevarlo a cabo.

El procesado de estos datos se puede llevar a cabo de dos maneras: una con conjuntos estáticos de información y otra con conjuntos de datos dinámicos. La primera forma reúne un conjunto de técnicas en el que se hace más hincapié en el modelo que se quiere obtener. En este caso, podría ser necesitarse tener máquinas potentes con la capacidad de cómputo suficiente para el análisis de grandes cantidades de datos y posiblemente sea necesaria una arquitectura distribuida o clúster de máquinas. Sin embargo, en la segunda forma, se ha de prestar atención además a otro tipo de factores que aumentan la complejidad de la estructura de procesado, ya que dicha estructura tiene que mantener un flujo constante de obtención, transformación, envío y modelado de datos. En este caso el almacenamiento pasa a un segundo plano. Este último entorno será el que se desarrollará en el presente trabajo.

El principal interés en este trabajo reside en juntar la aplicación de técnicas de procesado, modelado y visualización de datos en *streaming* [20]. Como principales puntos se destacan:

* Creación de entorno multi-tecnológico.
* Utilización de modelos de Machine-Learning (ML) [19] en *streaming.*
* Modelo de predicción de datos de navegación por la red.

El funcionamiento básico consiste en un usuario, que genera tráfico por internet de diferentes fuentes de información (diario deportivo, descarga de vídeo y simulación de llamadas sobre VoIP). El tráfico que generan estas fuentes, ha sido analizado con la finalidad de crear un modelo de predicción. Cuando el usuario cursa un tráfico determinado, el modelo tratará de predecir de qué fuente provienen los datos y con la visualización podremos ver las estadísticas de acierto del modelo entre otros datos de interés.

## Estructura de la memoria

* Capítulo І “Introducción”: este capítulo se muestra la principal motivación por la que se ha propuesto el sistema, así como los puntos que se han de tener en cuenta en arquitectura de procesado de datos en *streaming* [20].
* Capítulo ІІ “Descripción del *dataset*”: este apartado se describe cómo se ha obtenido el conjunto de datos, así como algunas características y el análisis realizado en él.
* Capítulo ІІІ “Arquitectura general”: este capítulo detalla el funcionamiento, así como la estructura y los parámetros del sistema.
* Capítulo ІV “Caso de estudio”: en este apartado se explican los experimentos realizados. Además, se muestran los resultados obtenidos durante la puesta en marcha de todos ellos y las dificultades encontradas.
* Capítulo V “Conclusiones y líneas futuras”: aquí se exponen los conocimientos adquiridos durante el desarrollo e implementación del sistema y los conocimientos de los que se partía antes de comenzar. Una vez puesto en marcha, se realiza una evaluación de los resultados obtenidos con los que se extraen las conclusiones finales sobre el funcionamiento del sistema. Para finalizar, se realiza un resumen a modo de visión futura, de lo que podrían ser algunas mejoras del sistema para dar cabida a un mayor rango de casuísticas que tener en cuenta.

##### CAPÍTULO II

# Descripción del dataset

fuente de datos

variables

extension



## Descripción general

En el presente trabajo se propone un sistema como solución para paliar algunos de los problemas actuales del tráfico urbano. Se basa en la mejora del funcionamiento del semáforo tradicional. Para ello se propone una comunicación vehículo-infraestructura entre el semáforo y el vehículo.

Para ello se ha generado un modelo de sistema que contará con diferentes elementos para poder conseguir la comunicación final deseada.

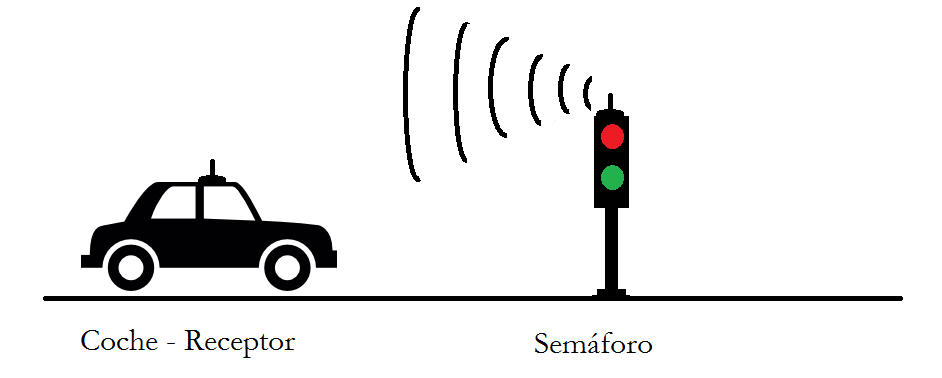


Figura . Escenario modelo de la comunicación

En la presenta figura 3 vemos una representación del modelo de sistema. Por un lado nos encontramos con el vehículo que será el encargado de recibir los mensajes generados por el semáforo y procesarlos. Por el otro, tenemos el semáforo que enviará una sencilla información al vehículo que le será imprescindible en sus cálculos.

## Elementos del sistema

El sistema propuesto es el resultado de la coordinación y cooperación de un conjunto de subsistemas repartidos en dos puntos; el semáforo o emisor y el vehículo o receptor.

A continuación se detallarán qué subsistemas son necesarios, así como la implementación y el prototipo desarrollado para llevar a la práctica dicho modelo.

### Sistemas embarcados en el vehículo

* On Board Unit (OBU)

La OBU es el subsistema cuya tarea es la comunicación con el exterior del vehículo utilizando para ello distintas tecnologías de comunicación. Tiene como objetivo principal la recepción y envío de mensajes entre el vehículo y otros vehículos, así como elementos de la infraestructura de tráfico. La comunicación puede realizarse mediante telefonía celular (3G, LTE, WiMax) o sistemas de comunicación AD-HOC, como 802.11p. Además, el OBU sirve de pasarela entre el exterior del vehículo y el resto de los subsistemas que componen su arquitectura electrónica.

* Application Unit (AU)

La AU se compone un conjunto de subsistemas que realiza el resto de tareas. Se divide en tres principalmente:

* + Human Machine Interface (HMI)

Es el encargado de la representación de datos, una vez recibidos y procesados en el vehículo. Será la interfaz de comunicación entre la máquina y el conductor. La información presentada tiene que tener ser de utilidad para la conducción y estar lo más actualizada posible. Por ello, todo dato que sea poco relevante para el conductor, será transparente para éste. Estos otros datos se usarán en el subsistema de procesado para llevar a cabos los cálculos a realizar.

Además de la utilidad, se debe tener en cuenta en la simplicidad de representación. La información tiene que verse dispuesta de tal manera que sea fácilmente comprendida por el conductor con un mínimo de entrenamiento.

* + GPS

Este sistema es el encargado de la obtención de la posición geográfica en coordenadas Latitud-Longitud-Altura (LLA), así como una medida de velocidad sobre Tierra y otros datos de diagnóstico del propio sistema GPS. Estos datos obtenidos en este subsistema se usarán con posterioridad en el procesado de información.

* + Sistema de procesado

Esta parte es la encargada de obtener la información del bus de datos que interconecta los subsistemas y combinarla. Es indispensable que tenga una computadora (Electronic Control Unit o ECU) que procese todos estos datos. Se recolectará todo lo que los otros subsistemas le proporcionan, lo analizará, lo convertirá en mensajes de ayuda para el conductor que serán enviados al HMI.

### Sistemas embarcados en el semáforo

* Road Side Unit (RSU)

La RSU es la unidad de comunicación implantada en el lado de la carretera, en este caso, en el semáforo. Su cometido es la emisión de balizas cada poco tiempo para que la OBU del vehículo los reciba. La RSU es muy similar al OBU, pero instalada en un armario.

* AU

La parte de la Application Unit del semáforo. En este proyecto, es mucho más sencilla que la del vehículo. Esto se debe a que no tiene que realizar ningún procesado de información. Se trata de una máquina de dos estados, que se alternarán en el tiempo durante toda la vida de la máquina. Estos estados son el rojo y el verde. Además, este subsistema se comunicará con la RSU para el envío de información al vehículo

## Integración del sistema

Para que todo el sistema funcione se necesita la coordinación y comunicación interna entre los diferentes subsistemas.

El modelo de sistema propuesto cuenta con un bus de datos. Este será el responsable del envío de mensajes entre cada subsistema. En él, el GPS depositará las coordenadas para que la unidad de procesado recoja esos datos. Al igual, la OBU dejará la información obtenida del semáforo. Una vez que la unidad procesadora ha hecho su trabajo, se enviarán los mensajes al conductor a través del subsistema HMI.

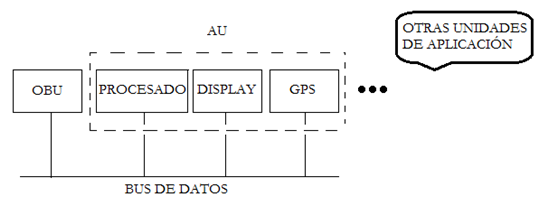


Figura . Subsistemas y bus de datos del vehículo.

En la figura número 4 vemos la unión entre los subsistemas mediante el bus de datos. Este modelo propuesto pretende que el vehículo tenga una sola OBU y una AU. En un sistema real y ya integrado, la idea es que esa OBU pueda ser usada por diferentes unidades de aplicación que realicen otras tareas en el vehículo. En el modelo general, cada subsistema tendrá el acceso al bus de forma independiente.

Centrándonos en la figura, la línea punteada encuadra los tres subsistemas en los que se divide la AU. Se puede observar que todos están conectados al mismo bus de datos que sirve de comunicación entre ellos. Por otro lado, se ve la OBU también conectado al bus pero fuera de la AU. A este bus irían conectadas diferentes AUs que se comunicarían de la misma forma con la parte de envío/recepción de mensajes al exterior del sistema.

## Implementación del prototipo

Una vez que se tienen definidos las partes que componen el sistema propuesto se procede a la implementación del prototipo. Para ello se ha recolectado una serie de tecnologías que son capaces de representar el modelo en la realidad. Este mapeo uno a uno de los diferentes subsistemas del modelo con la realidad es uno de los puntos fuertes del trabajo.

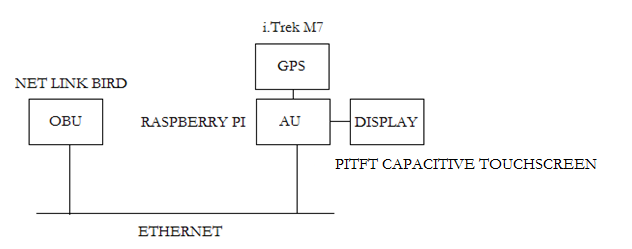


Figura . Esquema de prototipo

En la figura 5 se aprecia la relación modelo – tecnología empleada. Por la parte de la OBU (RSU en el semáforo) se utiliza un router específico para comunicaciones entre vehículos (V2V) o V2I. El modelo concreto es el NEC LINK BIRD. Esta máquina está configurada para soportar el protocolo de comunicaciones empleado, WiFi 802.11p. En la parte del vehículo, la unidad de procesado la relacionamos con la AU ya que es la parte que se comunica con la OBU, y corresponde a una Raspberry Pi Model-B, descrita más adelante (u ordenador en el caso del semáforo). En la parte de comunicación entre OBU-AU, encontramos un cable Ethernet con conexiones RJ-45[[1]](#footnote-1). Esta conexión llevará los datos hacia la AU. Aquí se aprecia la principal diferencia entre el modelo y el cómo se ha llevado a cabo. En el modelo propuesto, cada subsistema tenía acceso independiente al bus de datos. En nuestra integración, hay varios buses de comunicación: el general Ethernet ya descrito y otros dos que conectan el GPS y el display con la unidad de procesado. La conexión entre GPS y parte de procesado se realiza con un cable USB. La conexión con el display con pantalla táctil se realiza mediante un bus interno de la Raspberry Pi. La unidad GPS empleada es el modelo i.Trek-M7. Se ha utilizado este módulo que carece de pantalla porque los datos que obtiene se quieren representar en otra parte del sistema. En cuanto al display, se ha querido usar una pantalla PITFT de 2.8”, específica para Raspberry Pi. Esta pantalla es táctil, lo que permitirá en líneas futuras poder interactuar con la interfaz.

A continuación se entrará más en detalle de los elementos empleados en el prototipo.

## Tecnologías empleadas

Las tecnologías empleadas en la parte del vehículo son más y más complejas, ya que se requiere de un procesado de datos y la ayuda de diferentes subsistemas internamente coordinados y conectados entre sí.

### Vehículo

El sistema propuesto será integrado en un vehículo, ya que sirve de ayuda al conductor mientras lleva su vehículo. Por ello, el vehículo será el primer elemento de importancia a la hora de realizar experimentos con el sistema.

Para las pruebas realizadas, ha sido imprescindible usar un vehículo con puerto USB y conexión de 12V para alimentar la Rasbperry Pi y los encaminadores, necesarios para la comunicación con el semáforo. Además de estos dos elementos, en ambos puntos de la comunicación se con una antena para la recepción/envío de mensajes.

* Raspberry – Pi



Figura . Imagen de un ordenador Raspberry Pi

.

La Raspberry Pi [5] es ordenador que sólo consta de la parte electrónica de la CPU y de los conectores para los periféricos. La memoria en la que se instala el sistema operativo, tiene que ser una SD CARD de al menos 4 GB.

Además, la mini computadora cuenta con puerto HDMI[[2]](#footnote-2), puerto Jack[[3]](#footnote-3), micro usb para la alimentación, dos puertos USB, lector de tarjetas SD[[4]](#footnote-4) y puerto Ethernet para conectarse a la red. Este elemento constituirá la parte hardware de la AU del vehículo.

El sistema operativo elegido es el Raspbian[[5]](#footnote-5). Es una distribución Linux[[6]](#footnote-6), preparada para Raspberry Pi, que nos ha facilitado las cosas a la hora de desarrollar la aplicación.

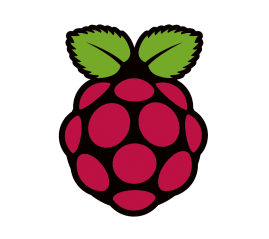


Figura . Logotipo de Raspbian y Raspberry Pi; una frambuesa.

Para el trabajo se ha utilizado una pantalla TFT táctil (capacitiva) [6] de 320x240. Esta pantalla ha sido soldada por sus conectores a los de la Raspberry Pi, de tal forma que quede tapando todo el conjunto electrónico.

Se ha decidido por usar una Raspberry Pi, con esta pantalla específica, ya que simulará perfectamente el tamaño que tienen las pantallas informativas de los vehículos. Con un soporte para sujetarlo al vehículo, y conectado al GPS, proporcionará al conductor la información en su display, que con un ligero giro de cabeza, podrá observar y analizar.

* Python y TKinter

Python es un lenguaje sencillo, de alto nivel, usado para muchas y muy diferentes aplicaciones. Es usado como introducción a la programación, pero también es usado por desarrolladores de software de alto nivel como Google, la NASA y Lucasfilm Ltd [7].

Tiene una gran versatilidad y ha dado mucho juego a la hora de probar diferentes métodos de implementar la aplicación. Esto se debe en parte a las extensas y variadas librerías que posee, así como su fácil manejo y aprendizaje. Las principales han sido:

* + **Time:** ha permitido controlar el flujo del programa, teniendo en cuenta los tiempos necesarios y siendo de gran importancia la hora de la sincronización en todos sus módulos.
  + **Os:** esta librería nos permite introducir comandos de la Shell de Linux en la ejecución de nuestro programa. En concreto, ha sido de gran ayuda a la hora de preparar el puerto para la adquisición de datos del GPS hacia la Raspberry Pi.
  + **Math:** ha proporcionado las funciones necesarias para la transformación de las coordenadas UTC[[7]](#footnote-7) a UTM[[8]](#footnote-8). Funciones como los senos, cosenos, tangentes y raíces cuadradas.

Otro punto por el que ha sido elegido como lenguaje de desarrollo, es por su buena integración en las distribuciones de Linux y en concreto en la Raspberry Pi.

A la hora de mostrarse los datos al conductor, se ha optado por utilizar TKinter [8]. TKinter es un módulo estandarizado de Python. Esta herramienta proporciona las funciones y variables necesarias para generar una interfaz gráfica de manera sencilla.

Gracias a esto se ha desarrollado un diseño sencillo para la lectura del conductor. Con sencillo nos referimos a que el conductor pueda echar un vistazo a la pantalla y rápidamente analizar los datos que necesita para actuar, sin quitar mucho tiempo la vista de su trayectoria en carretera.

Esto último representa el objetivo principal de una interfaz gráfica en un vehículo; sencilla y con los datos exactos que necesita el conductor de un rápido repaso por el salpicadero.

A continuación se muestra el diseño de la interfaz gráfica.



Figura . Interfaz gráfica.

Los colores elegidos son el negro y el naranja. Esto es debido a que se ha intentado mimetizar con el resto de paneles luminosos que tienen los vehículos, con un fondo oscuro sobre unas letras llamativas.

Como se puede observar, lo que primero llama la atención en el panel es el centro. Ahí se mostrará la velocidad recomendada. En este caso se muestran dos rayas, perteneciente al estado en el que se hace caso omiso a los mensajes recibidos. Este será el dato de principal interés por lo que se ha intentado que sea lo primero en verse.En la parte superior se dispondrá la información a leer por el conductor. Serán mensajes cortos y precisos para no tener que reparar mucho en ellos. En las esquinas inferiores se muestran otros datos de interés, la velocidad actual y la distancia hasta el semáforo. Estos datos se despliegan en un tamaño menor ya que es información complementaria que el conductor puede observar en el velocímetro y en su vista al frente durante la conducción respectivamente.

TKinter junto a la pantalla incorporada a la Raspberry Pi formaría el subsistema HMI, ya que es el encargado de presentar la información al conductor.

* GPS

Es un sistema que permite determinar en todo el mundo la posición de un objeto con una precisión de hasta centímetros. Para calcular las coordenadas correspondientes a un punto, utiliza una constelación de 24 satélites y la trilateración (que es el método geométrico para determinar posiciones relativas mediante la geometría de triángulos) [9].

Para el correcto funcionamiento del sistema, se necesita de la posición geográfica del vehículo en todo momento. Para ello usamos un aparato de localización geográfica o GPS, que entienda el estándar desarrollado por la *National Marine Electronics Association* (NMEA). Este subsistema representa el mismo en el modelo de sistema propuesto.

* NMEA

[10] Consiste en una serie de especificaciones que definen la interfaz de comunicación entre varias elementos de la electrónica marina. Esto permite el envío de información entre ordenadores y otros elementos del equipamiento marino.

Dentro de estas normas, se encuentran postuladas las necesarias para las comunicaciones con un receptor GPS. Estos datos generados por el GPS, incluyen el PVT (posición – velocidad - tiempo).

NMEA se basa en el envío de una línea de datos, con un formato específico, de tal manera que sea fácil su entendimiento y la extracción de información por parte de los aparatos electrónicos entre los que se están comunicando con este recurso.

Para este proyecto, incorporaremos un módulo en nuestra programación, llamado GPSd, el cual se encargará del entendimiento con este estándar y por lo tanto de la extracción de información del GPS.

* GPSd

Es un servicio de background o “demonio” que nos ofrece una interfaz para la comunicación entre nuestra programación y el GPS [11].

En dicho módulo vienen incorporadas librerías en el lenguaje de programación Python, que nos facilitarán la comunicación y extracción de datos de posición, velocidad y tiempo.

GPSd entiende el estándar NMEA, por lo que en nuestro programa se han añadido sentencias que capturarán la información proveniente de estas líneas que nuestro dispositivo de geolocalización genera.

* Sistema de coordenadas

El sistema desarrollado tiene procesos que se encargan de la medición de distancias, con la ayuda de un GPS. Estas distancias medidas, son las calculadas para saber cuántos metros quedan desde el vehículo hasta el semáforo.

Para realizar dicho cálculo, se ha de realizar una transformación de sistema de coordenadas, desde el representado por el GPS (LLA) hasta un sistema que nos permita medir distancias en metros.

* + UTC – Coordinated Universal Time

El tiempo universal coordinado, es el estándar de tiempo generalizado para todos los relojes del mundo. Se obtiene a partir de una media ponderada de los mejores 70 relojes atómicos de todo el mundo [12].

Dicho sistema se implantó como solución a los desajustes que se producen en el horario GMT (Greenwich Mean Time), extrapolado a partir del movimiento de la Tierra alrededor del Sol.

En relación con el sistema UTC, se define huso horario como cada una de las veinticuatro áreas en las que se divide la Tierra, y en las que se determina un horario diferente para cada una de ellas [13].

En la figura número 9 podemos observar las divisiones geográficas marcadas por estos husos horarios.

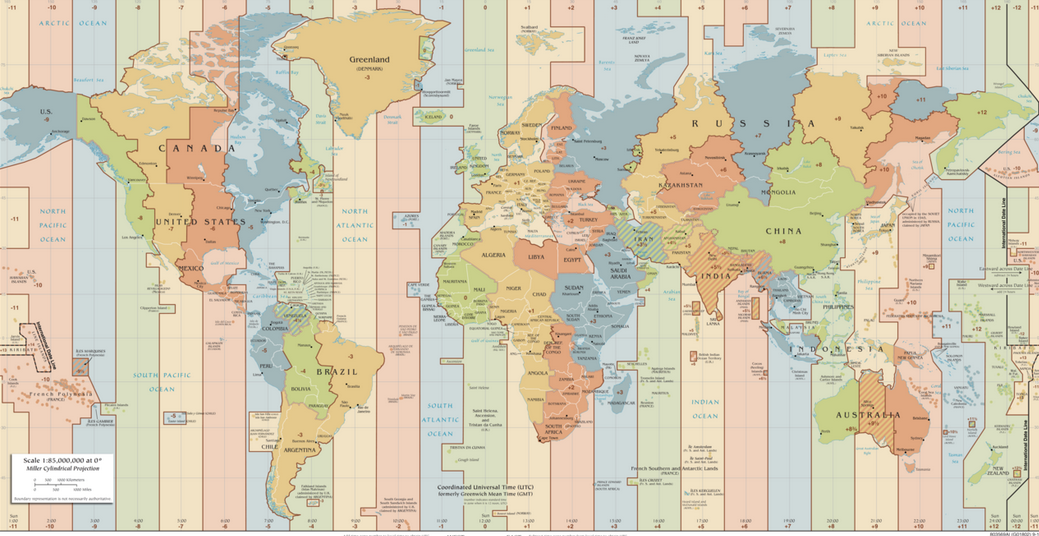


Figura . Husos horarios UTC [13].

Si pasamos la vista de Este a Oeste, tendremos que restar una hora al pasar de un huso horario a otro, ya que la Tierra gira de Oeste a Este.

* + UTM – Universal Transverse Mercator

Consiste en un sistema de coordenadas, apoyado en la proyección cartográfica o proyección geográfica, que tiene como finalidad la representación de la Tierra en otro sistema de referencia diferente al de coordenadas LLA [14].

A esto último se ha de añadir que, el sistema de representación resultante del UTM, considera sus magnitudes en metros.

Para llevar a cabo dicha representación, se divide el mapa de la Tierra en 60 husos de 6º de longitud (a diferencia de los 15º usados en el sistema de coordenadas geográficas). Dichas áreas son nombradas del 1 al 60. En la figura 10 podemos verlos representados en el mapa del planeta.

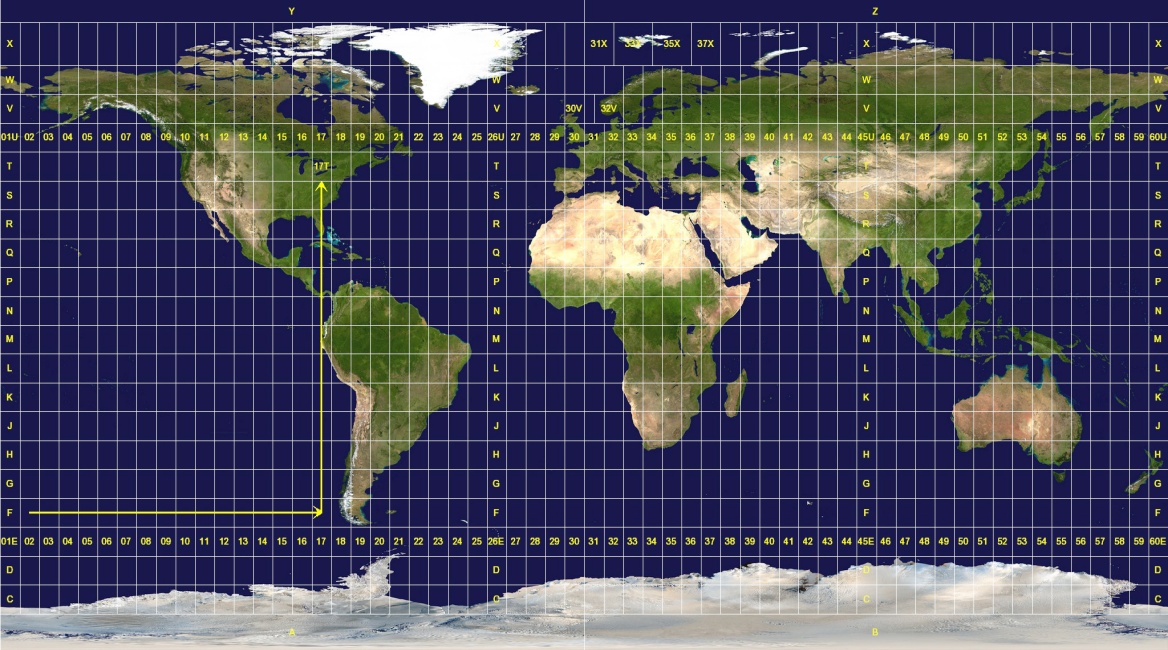


Figura . Husos UTM

UTM se basa en el establecimiento de una relación ordenada entre los puntos de la superficie curva de la Tierra y los puntos de una superficie plana o mapa. Esta relación viene dada por la proyección cilíndrica del globo terráqueo dando como resultado la representación de la superficie terrestre en un plano. En esta forma de dibujar la Tierra, la superficie cilíndrica es tangente al ecuador y mediante el despliegue de dicha área se genera el mapa plano. Ello conlleva a que los meridianos se representen proyectados con una separación proporcional y de forma equidistante. Por otro lado, los paralelos se disponen a distancias cada vez mayores del Ecuador, entre el grado 80º Sur y 84º Norte. En los polos se rigen las zonas por el sistema UPS (Universal Polar Stereographic), puesto que es imposible tomar un sistema de referencia que represente todo el globo de la misma forma ya que posee ciertas deformidades.

En nuestra aplicación existe un módulo completamente dedicado a fórmulas matemáticas que transformarán las coordenadas proporcionadas por el GPS en puntos que podamos tratar de tal forma que, los resultados originados por nuestro cálculos den distancias en metros, legibles y más fáciles de relacionar y entender para el conductor.

### Semáforo

La segunda parte del sistema es el lado del semáforo. Esta parte requiere de menos subsistemas que se conecten entre sí. Esto es debido a que no hay que realizar ninguna recepción de datos que después haya que procesar ni la obtención de coordenadas GPS.

En nuestra realidad, el semáforo será representado por un ordenador (la AU), el cual contendrá el programa que rige el comportamiento del semáforo. No se ha utilizado ningún semáforo como los que se ven normalmente por la calle. Simplemente se ha conectado la AU a la RSU correspondiente y se ha procedido al envío de datos hacia el vehículo. Esta conexión ha sido con un cable Ethernet. Una de las mejoras que se pueden hacer es dotar al semáforo de presencia física de luz, lo que hará más fácil la realización de experimentos.

* Python

La programación que lleva el semáforo en su interior, tiene como protagonista el leguaje Python. Este lenguaje es el ya utilizado para el vehículo. Como se ha mencionado anteriormente, este lenguaje con sus herramientas sencillas e intuitivas, ha permitido desarrollar con facilidad la máquina de estados que había que implementar, así como la conexión a los sockets necesarios para el envío de información.

## OBU y RSU

Los elementos de sistema empleados en las comunicaciones, tanto para el lado del vehículo como para el lado semáforo poseen las mismas máquinas: encaminadores específicos para el protocolo utilizado de comunicaciones inalámbrica, IEEE 802.11p, específico para comunicaciones entre vehículos.

Las máquinas usadas para el enrutamiento de mensajes son los encaminadores modelo NEC LinkBird-MX. A continuación se detallan sus especificaciones técnicas más relevantes:

Hardware:

* 64bits MIPS Microprocesador @ 266MHz
* 512MB NAN-Flash, Nor-Flasd and 128 SDRAM.
* Consumo de poca potencia.
* Modo de programa de test para el diagnóstico del hardware.

Software:

* Linux Kernel 2.6.19.
* U-BOOT Boot Loader.
* NEC C2X-SDK Sistema de comunicación y API.

Entorno y requerimientos de potencia:

* Almacenaje a temperatura entre -40ºC y +85ºC.
* Temperatura de operación entre -20ºC y +65ºC.
* 12VDC @ 400mA

Interfaz de red:

* Fast Ethernet embebido 10/100 Base-T
* Draft versión D3.0, mini-PCI IEEE802.11p embebido
* 1xPCMCIA WLAN 802.11a/b/g opcional.

Dimensiones:

* 152.5mm(W) x 118mm(D) x 43mm(H)

El estándar IEE 802.11 es el que comúnmente se conoce como WiFi[[9]](#footnote-9). Dependiendo de las especificaciones de las aplicaciones y de los requisitos que requieren las comunicaciones con esta tecnología, se han desarrollado diferentes variantes. Cada una de ellas se nombra con una letra; 802.11b, 802.11g, 802.11n, etc.

En entornos vehiculares, estos requerimientos se reducen a comunicaciones cortas y con delays muy pequeños en los envíos de datos. Esto se debe a que la situación en el entorno del vehículo es muy cambiante. Los vehículos necesitan procesar datos cortos y claros para tomar decisiones y hacérselas llegar al conductor, el cual tendrá que analizar y pensar cómo actuar en muy poco tiempo.

La variante 802.11p es la que recoge las especificaciones para las comunicaciones vehiculares. El proceso de estandarización se origina a partir de la asignación de la banda de espectro, en los EE.UU de América, para las comunicaciones dedicadas de corto alcance o DSRC (Dedicated Short Range Communications) [15]. Esta banda de frecuencia se encuentra en la banda de los 5,9 GHz, apartada de la banda usada por el WiFi. La nueva banda DSRC será utilizada en exclusiva por los . El principal objetivo es permitir el desarrollo de aplicaciones que puedan salvar vidas y mejorar el flujo de tráfico.

En la **capa física** de este protocolo encontramos definida una modulación OFDM y un ancho de banda de 10MHz (en vez de los 20MHz usuales en la mayoría de los dispositivos 802.11). Esta medida tiene como finalidad abordar el tema del aumento de la dispersión del retardo en estos entornos vehiculares. Con un ancho de 20 MHz, el intervalo de guarda para evitar la interferencia entre símbolos (OFDM) no es suficiente, por lo que se realiza este escalado, bajando el ancho a 10 MHz.

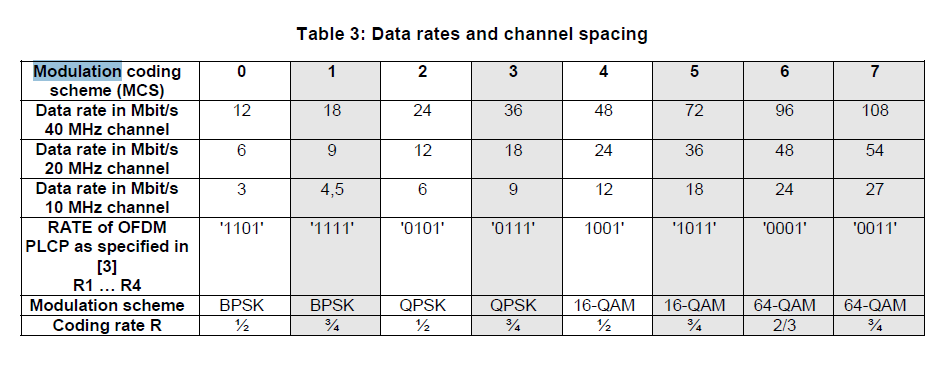


Figura . Modulaciones en función de la tasa y codificación [16]

En la figura 11 se puede apreciar las tasas en función de la codificación y el ancho de banda utilizado. Para nuestro sistema, como se ha explicado, nos tenemos que centrar en la fila de los 10 MHz. Esas son las posibilidades que nos ofrece este protocolo en nuestra comunicación.

## Sistema radiante

Los elementos utilizados en esta parte, son el último punto por el que pasa la información antes de entrar al medio de comunicación, el aire, y el primero al salir de él.

Durante todo el desarrollo del trabajo se han utilizado diferentes sistemas radiantes. El primero de ellos, constaba de una antena WiFi de 15 cm por router. Estas fueron usadas que para las pruebas iniciales estáticas del canal de comunicación, que se explicarán más adelante.



Figura . Antena WiFi.

En la anterior figura se muestra la antena empleada para estas pruebas.



Figura . Antena WiFi conectada al router.

En la anterior figura podemos ver la antena conectada al router. Tiene varios conectores por si se quiere probar comunicaciones variando la diversidad.

El segundo sistema utilizado, se usó para las pruebas en movimiento y estuvo compuesto por dos antenas omnidireccionales, cada una para una parte de la comunicación semáforo-vehículo. Las dimensiones son de 40 cm de alto por 3 cm de diámetro. Tras la realización de diferentes intentos de experimento, nos dimos cuenta que las bases magnéticas empleadas para la sujeción de las antenas encima de los vehículos tenían muchas pérdidas. Esto nos provocaba una pérdida de potencia que con las antenas WiFi no pasaba y por consiguiente un alcance mucho menor del deseado para el correcto funcionamiento del sistema.

Las antenas usadas, que podemos ver en la figura número 14, se corresponden al sistema radiante usado para las primeras pruebas en movimiento. Se anclan a los techos de los vehículos mediante bases magnéticas. Esta decisión se tomó por querer tener una comunicación libre de obstáculos, situación que no tendríamos si usásemos antenas dentro del vehículo.



Figura . Antenas omnidireccionales

Las bases utilizadas para la sujeción, se visualizan en la figura número 15.



Figura . Base magnética

Lo que se decidió finalmente fue usar una antena direccional de gran capacidad en el lado del semáforo y una antena WiFi en el lado del vehículo, en su interior. Con esta configuración, se han conseguido realizar todos los experimentos que se detallarán más adelante.

Para las pruebas realizadas, tanto los equipos pertenecientes al emisor, como los equipos que forman el sistema de comunicación del receptor, han sido alimentados desde vehículos. En una futura implementación, el emisor o semáforo irá alimentado de la Red Eléctrica Pública, como se encuentran actualmente.

En la siguiente figura vemos la antena final utilizada para el semáforo. Debido a que las pruebas finales se realizaron en una recta, la directividad ayudó a la mejorar el alcance de la comunicación.



Figura . Antena directiva del semáforo

En los semáforos reales, las antenas que se empleen irán a una altura mucho mayor, por lo que se podrá conseguir un alcance de comunicación aún mejor, dándose así la oportunidad de mejorar la aplicación con otras funcionalidades.

En el siguiente capítulo se podrá conocer como se ha conseguido que todas estas tecnologías se hayan coordinado para dar como resultado el sistema final.

##### CAPÍTULO III

# Arquitectura general

2. presentacion del esquema
3. tecnologias
4. interaccion entre ellas y el rol que desempeÃ±a cada una
5. plataforma (portatil o cluster de amazon o donde), caracteristicas técnicas
6. - TecnologÃ­as
7. - Python: https://www.python.org/ 2.7 y 3.5
8. - linux 16.04
9. - Shell Bash script: https://es.wikipedia.org/wiki/Bash
10. - Docker: https://www.docker.com/
11. - Kafka: https://kafka.apache.org/
12. - Kibana: https://www.elastic.co/products/kibana
13. - ElasticSearch: https://www.elastic.co/
14. - Spark, Spark Streaming: https://spark.apache.org/, https://spark.apache.org/streaming/
15. - Spark ML y MLlib (RandomForest)https://spark.apache.org/mllib/, https://spark.apache.org/docs/latest/ml-guide.html, https://es.wikipedia.org/wiki/Random\_forest
16. - Jupyter: http://jupyter.org/
17. - TCPDUMP: http://www.tcpdump.org/tcpdump\_man.html
18. - anaconda: https://docs.continuum.io/anaconda/

## Introducción

En esta parte hablaremos de cómo se ha desarrollado la aplicación hasta su estado final así como las premisas que se han seguido y los diferentes módulos que se han desarrollado para dar soporte a todo el procesado y envío-recepción de datos.

El “mundo” sobre el que se ha construido la aplicación y se han realizado los experimentos, se ha simplificado lo máximo. Para llevarse a cabo, sólo se ha tenido en cuenta que existirá un único vehículo. Al igual por la parte emisora, sólo tendremos un semáforo.

El sistema está preparado para que funcione en cualquier recorrido. El elegido para los experimentos ha sido la carretera situada alrededor de todos los edificios de la Universidad Rey Juan Carlos de Fuenlabrada (URJC). Lo que hay que tener en cuenta es que necesitamos un conocimiento previo del circuito a utilizar, ya que, como se explicará más adelante, necesitamos unos puntos de referencia marcados por las coordenadas del trayecto.

Estas tres premisas, han sido la base del diseño de la aplicación desarrollada. Esto se podrá mejorar en futuras mejoras del sistema, consiguiendo una mayor universalización del trabajo.

## Modelo de conducción

La idea principal es que el vehículo aconseje al conductor a que vaya a una velocidad para cruzar el semáforo en verde. Dependiendo de si la velocidad del vehículo es una u otra, se tendrá que reducir o aumentar la velocidad para llegar a la adecuada.

Para conseguir esto, se ha llevado a cabo un modelo sencillo de conducción en el cual, no se tiene en cuenta si la aceleración o deceleración que se debe producir es viable para una conducción confortable. Simplemente se centra en aconsejar velocidades según un modelo de rectas que se detalla a continuación.

Para explicar este modelo primero se va a hace referencia, en la figura 17. Se trata de una representación de la trayectoria de uno o varios vehículos donde el tiempo y el espacio se representan simultáneamente. Recibe el nombre de Campo de Espacio Tiempo Velocidad, o Campo STV.

El eje de abscisas pertenece al tiempo transcurrido en segundos. En el eje de ordenadas se representa el espacio, indicado en metros. El punto Sf, es en el que se encuentra el semáforo situado en el circuito. La situación de éste será fija. Los parámetros t1 y t2, hacen referencia a los tiempos de duración del estado verde y el estado rojo del semáforo, respectivamente. Por ello, se puede observar que, en el punto Sf, a medida que transcurre el tiempo, los estados se alternan entre verde y rojo.

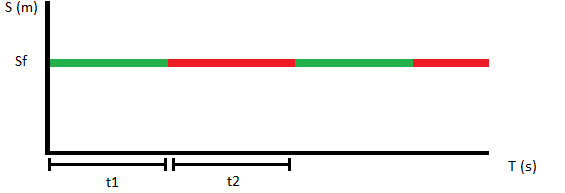


Figura . Tiempos de semáforo.

A partir de aquí, representaremos la velocidad en función del tiempo y el espacio. Por lo tanto la pendiente de la recta que se formará corresponderá a la velocidad actual que lleva el vehículo.

En la figura número 18, Vact hace referencia a este valor. La punta de la flecha señalará al momento exacto en el que el vehículo llegará al semáforo.

Para este caso representado, se observa que el vehículo, con la velocidad actual, llegaría en estado verde y podría cruzar sin problemas.

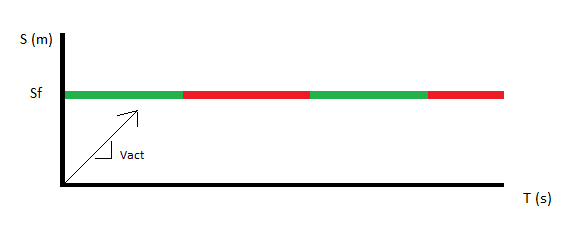


Figura . Situación verde, mantengo velocidad.

Cuando nos encontramos esta situación, se indicará una velocidad mínima a la que se debe ir para que no se cambie a color rojo. Si el conductor hace caso a estas indicaciones, podrá cruzar el semáforo en verde.

En la siguiente figura, la número 19, nos encontramos un estado diferente. Ahora, con la velocidad actual, se llegaría estado rojo, por lo que tendría que parar ante el semáforo. Para este caso el vehículo, con el tiempo restante para el cambio a verde y la distancia restante, calcularía una velocidad a la que el conductor tendría que ir para llegar en el estado que le permita cruzar una vez cambiado de estado. Esto es, Vrec o velocidad recomendable. Es un caso en el que se refleja una deceleración para no tener que parar. Simplemente decelerar y continuar en esta nueva velocidad hasta que el semáforo se vuelva a poner en verde. En caso de que la velocidad calculada sea muy baja, se indicará al conductor que se vaya preparando para frenar.

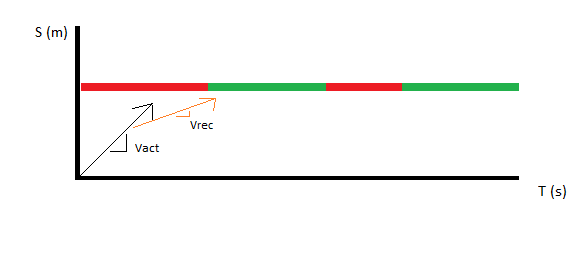


Figura . Estado rojo, disminuimos velocidad.

Se ha de añadir que existen unos límites en las velocidades calculadas. Estos topes han sido cogidos en función de las normas y prohibiciones de exceso de velocidad y pensando en una velocidad mínima que sea cómoda de sostener en el pedal de aceleración. Es decir, velocidad máxima de 50 km/h y velocidad mínima de 12 km/h.

El siguiente gráfico de la figura 20, representa el caso en el que el semáforo se encuentra en verde, pero según podemos observar, si seguimos la línea discontinua, que nos marca la predicción de cuándo llegaremos al semáforo, nos indica que llegaremos en estado rojo y nos tendremos que detener.

En esta situación aparecen dos posibles casos. En el caso de que la predicción sea de no llegar en verde porque la velocidad calculada sea muy alta, o que sea muy baja. En el primer caso se calcula una velocidad alternativa a la que se recomendará ir. Para el cálculo de esta velocidad se ha tenido en cuenta el tiempo que queda para el cambio a verde más el tiempo de duración del estado rojo y la distancia restante. De esta forma se intenta llegar al semáforo cuando haya cambiado a verde sin haber parado.

Para estos casos esa velocidad calculada será baja ya que tenemos que recorrer una distancia a la espera de que el semáforo cambie a rojo y vuelva a cambiar a verde sin detenernos. En caso de que la velocidad sobrepase el límite inferior, habrá que parar.

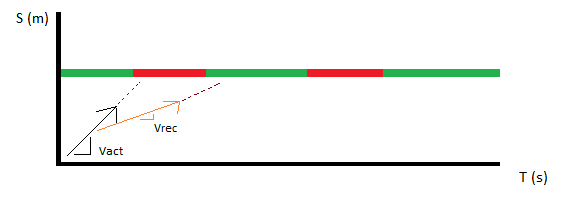


Figura . Situación rojo, deceleración para verde.

El segundo caso tiene el mismo efecto para la aplicación, predicción de que no vas a llegar. Pero esto no es cierto. En realidad lo que pasa es que la velocidad calculada es tan baja que el sistema no nos permite conducir en estas condiciones. Ante esta situación lo que se hará es aumentar la velocidad recomendada hasta un mínimo admisible. De esta forma, si se cumple ese mínimo de velocidad llegaremos en verde, aunque el sistema detectó que se iba a llegar en rojo.

Por último, en la figura 21 se refleja el estado restante. Según nuestra posición, el semáforo se encuentra en rojo. Si seguimos la flecha discontinua, la predicción nos indica que llegaremos a un estado en el que podemos cruzar el semáforo. En este caso, el sistema indicará una velocidad máxima a la que tendrá que ir el conductor. Esta velocidad será fruto de la reducción de una primera velocidad calculada para llegar al cambio de color. Al reducir este valor tendremos una velocidad que nos permita cruzar en verde.

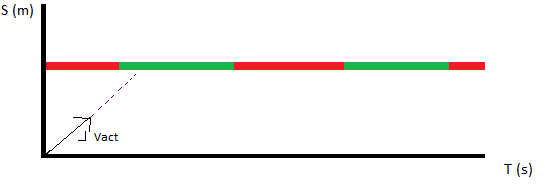


Figura .Situación rojo, llego a verde con Vact.

Para los casos en los que el sistema nos avise que no vamos a llegar a este estado, ocurre la situación contraria que para el estado verde. Esto es debido a que nos interesa pasar en verde. Por ello, si no llegamos por una velocidad calculada muy alta, se calculará una velocidad máxima a la que ir. Este valor calculado nos hará estar en movimiento hasta que se cambie el color a verde. En el caso de que la velocidad calculada sea muy baja, el conductor se ha de preparar para frenar.

Un punto importante, en cuanto a la presentación de información ante el conductor se refiere, es el de cuándo y con qué frecuencia se le muestra al conductor una información. Se debe tener en cuenta ya que el vehículo está continuamente realizando cálculos y recibiendo mensajes del semáforo (cuando se encuentra dentro de su radio de cobertura). No se puede saturar al conductor de datos ni estar variando las decisiones y la información cada poco tiempo.

El criterio elegido para discernir estos tiempos y situaciones en las que se muestra la información, ha sido el de dividir el circuito en tres áreas.

Se debe tener en cuenta que el conductor es una persona y como tal, existen límites a la hora de procesar un número de mensajes muy elevado. Cuando el vehículo entra dentro del rango de cobertura de las balizas enviada por el semáforo, éste recibe una gran cantidad de información en un espacio muy pequeño de tiempo. Se encontrará por tanto en un continuo análisis de los datos recibidos, pudiendo generar gran cantidad de mensajes que mostrar al conductor a través de la pantala.

Debido a esto, se han establecido diferentes áreas del circuito en las que el comportamiento del programa variará según la distancia hasta el semáforo. Esto es, aunque el vehículo empiece a recibir a una distancia de 100 metros del semáforo, no se empezarán a mostrar mensajes de recomendación hasta los 50 metros. Con esto reducimos los posibles cambios repentinos de decisión, ya que a una distancia mucho mayor, el semáforo puede cambiar varias veces de estado y se producirían cálculos poco precisos e información muy variante. Así conseguimos una información más útil a mostrar.

En la figura 22 se observa la división del circuito en tres zonas. Nuestro circuito de la universidad es cíclico, se ha representado la Z3 repetida.

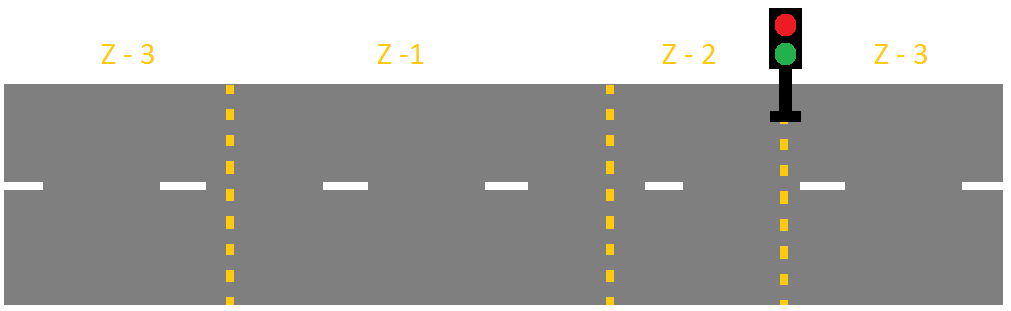


Figura . Zonas del circuito.

La zona denominada como Z - 1, está comprendida entre los 7 y los 50 metros desde el semáforo. En esta área comienzan los mensajes de ayuda a la conducción, en función de los mensajes recibidos por el emisor.

La zona denominada como Z – 2, está comprendida entre el semáforo y los 7 metros. Esta es una zona que dependiendo del estado del semáforo, se mantiene como Z-2 o puede transformarse en parte de la zona Z-1. Esto va a depender del movimiento del vehículo. Es decir, si el vehículo se encuentra parado en esta zona, porque el semáforo se encuentra en rojo y está esperando a que cambie el color para cruzar, los mensajes de la pantalla harán que el conductor se prepare para arrancar de nuevo. En el caso en el que el vehículo se encuentre en movimiento, la inteligencia actuará del mismo modo que para Z – 1, avisando al conductor de la velocidad que debe tomar o si por el contrario, debe frenar.

## Red de comunicación Ad-Hoc

En este apartado se expone en detalle la configuración de la red entre el semáforo y el vehículo, tanto la topología como los niveles físico, enlace y red. También se detallan las pruebas estáticas realizadas.

### Topología de la red Ad-Hoc

En la presente figura 23, viene representado el esquema de la red con la configuración IP seguida. En cada extremo de la comunicación tenemos los dispositivos de inteligencia u OBUs. Para nuestro trabajo vendrán representados por un ordenador y una Raspberry Pi. Al ordenador se le asignará la IP 192.168.2.2 mientras que a la Raspberry Pi, la IP 192.168.1.2.

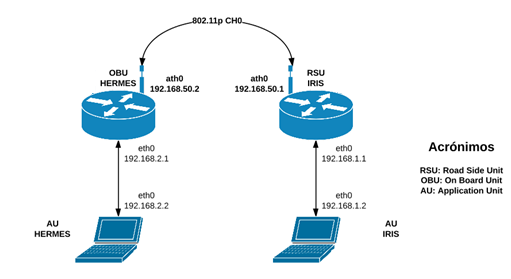


Figura . Esquema conexión

La OBU y la RSU son los routers específicos del protocolo 802.11p. Dichos elementos vienen previamente configurados con la tabla de encaminamiento, de tal forma que se puedan comunicar ambas partes. El vehículo tendrá en su interior al llamado router IRIS, con el gateway 192.168.1.1 mientras que el semáforo tendrá la IP 192.168.2.1 como gateway en el router denominado como HERMES.

Físicamente poseen dos interfaces a las que conectar hasta dos por interfaz, pudiendo jugar con la diversidad en la comunicación. En este trabajo se usará la interfaz ath0, con una sola antena en cada punto de la comunicación (diversidad = 1).

### Preparación y evaluación de parámetros

Antes del comienzo del desarrollo de las aplicaciones Semáforo y Vehículo, se realizaron pruebas estáticas que se detallarán más adelante. En ellas se ha realizado un estudio previo de los parámetros tanto a nivel físico como a nivel de aplicación.

Dichas variaciones tuvieron como objetivo comprobar la variación de las prestaciones en los diferentes modos de funcionamiento. En las pruebas que se realizaron se cambiaron los valores de las modulaciones y tasas de la capa física, así como la potencia de transmisión.

En los siguientes apartados se detalla cómo se ha configurado la red, desde un punto de vista de comandos y cómo se han ajustado los parámetros de la capa física y a nivel de aplicación para los experimentos realizados.

### Configuración de Red

El primer paso, para que exista la comunicación entre ambos puntos, es la configuración de los parámetros de red. Los encaminadores que se usan para el proyecto vienen preconfigurados con un protocolo de WiFi especial para comunicaciones en entorno vehicular (802.11p). Además, en su software se encuentran las rutas y tablas IP necesarias para que se haga posible el encaminamiento de paquetes de datos entre ellos.

Estos valores cargados por defecto en las máquinas nos obligan a configurar los equipos en los que correrán ambas aplicaciones. Necesitamos asignar una IP y una ruta tanto al elemento que proporciona la inteligencia en el vehículo como al que la proporciona en el semáforo.

Para ello, tenemos dos opciones para configurar nuestra red en los extremos de la comunicación.

**Opción 1 - Network Manager**

Esta opción es sólo válida para el lado del semáforo, ya que el sistema operativo de la Raspberry Pi no contiene este programa de gestión de redes.

Los pasos a seguir a la hora de crear la nueva red son:

1. Desactivar red inalámbrica. Por defecto, cuando el gestor de redes detecta una red inalámbrica, desconecta el resto e intenta conectarse a ella. Por ello debemos desactivar el receptor Wireless y centrarnos en las comunicaciones a través del Ethernet.
2. En el menú del Network Manager, buscaremos la opción de editar conexiones. Dentro de esta opción, añadimos una red de tipología cableada.
3. Una vez creada, modificamos el nombre de la red, y le damos a configuración de datos manual. De esta forma, podremos introducir en los ajustes IPv4 tanto la IP de la máquina, como la netmask (255.255.255.0) y la IP del Gateway (la de la puerta de enlace del router correspondiente).
4. Guardamos y salimos.

A partir de aquí, cada vez que conectemos el cable de red que va al router, directamente se configurará la red del nodo.

**Opción 2 – Fichero de las interfaces de red de Ubuntu**

El sistema operativo usado en ambos equipos a configurar, son extensiones del sistema UNIX. Por ello en los dos la estructura y los comandos para gestionar el sistema operativo desde la terminal son los mismos.

En este caso, nos centraremos en la edición del fichero Interfaces, que se encuentra en el directorio home/etc/network/. Para acceder a éste, por comandos en la terminal, con la finalidad de su edición, introduciríamos “sudo nano /etc/network/interfaces”. Así conseguimos abrirlo con el editor Nano, habiéndonos dado privilegios anteriormente con “sudo” y tecleando nuestra contraseña de super-usuario.

Dentro del fichero escribimos:

auto eth0

iface eth0 inet static

address 192.168.x.2

network 192.168.x.0

netmask 255.255.255.0

gateway 192.168.x.1

up route add -net 192.168.x.0 netmask 255.255.255.0 gw 192.168.x.x

En las dos primera líneas instanciamos la interfaz de red que vamos a utilizar. La línea “address” indica la IP de la máquina ordenador o raspberry. La x indica si la máquina se encontrará en la red 1 o la 2. La línea “network” hace referencia a la red en la que estemos configurando el nodo. Por ello acaba en cero, y en la x se introducirá un 1 o un 2 dependiendo de la red seleccionada. La línea “netmask” indica la máscara de red. Es igual en ambas redes. En cuanto al “gateway”, indica la IP de la puerta de enlace del router a la que está conectada la máquina. Debido a esto acaba en uno y la x determinará la red en la que se encuentre el nodo.

La última línea nos indica una ruta por defecto para los paquetes que tenga que enviar. En ambos casos la ruta a generar tendrá como red el uno o el dos, misma máscara de red para cada nodo y como siguiente salto en ambos casos el router al que se encuentran conectados.

Cerramos el editor pulsando “CTRL + X”, pulsamos Y para guardar con el mismo nombre y presionamos la tecla Enter para aceptar y cerrar.

Una vez configurado el fichero, necesitamos cargar esa información en el gestor de redes. Para ello se introduce en una terminal el siguiente comando: “sudo /etc/init.d/networking restart”. Con esto conseguimos reiniciar el gestor y cargar la estructura de red a utilizar en la comunicación.

### Nivel de aplicación

Para las pruebas en estático se tuvieron que variar parámetros a nivel de aplicación. En ellas se variaron parámetros de la comunicación para observar la cantidad de paquetes perdidos que había entre ambos nodos.

Para comprobar el dato de paquetes perdidos, se ha utilizado la herramienta Iperf [17]. Su finalidad básica es establecer un nodo servidor y un nodo cliente en el que el primero se encuentra a la escucha del medio, esperando recibir un flujo de datos del segundo. En la terminal del servidor se pueden apreciar los resultados de la comunicación entre ambos, con los que se pueden realizar informe y gráficas como las presentadas más adelante.

Las posibilidades de configuración de parámetros en cada nodo es la siguiente:

* Instanciación nodo Servidor: Iperf –s
* Instanciación nodo Cliente: Iperf -c <IP Nodo1>
* Conexión UDP (con pérdida de paquetes): -u
* Formato de unidades: -f <[nknK]> ([transferido, ancho de banda]) , bmb, kb, KB, MB
* Tiempo de transmisión total: -t <segundos>
* Tiempo entre presentación de informes de la comunicación: -i <segundos>
* Bidireccional: -r
* Bytes a transmitir: -n
* Ancho de banda: -b <numero><unidad> (sólo en conexiones UDP, recomendable unidad mb)

Uno de los parámetros más importantes y que se ha decidido para el esquema de la comunicación, es el tipo de conexión. Este proyecto cuenta con conexiones UDP. La elección viene determinada por las características de la comunicación; necesitamos que el envío sea rápido, no nos importa que se pierdan paquetes pero sí disminuir el tiempo de retardo en el que le llegan los paquetes. Debido a que en este sistema no existe la retransmisión por paquetes perdidos, nos aseguramos unos mejores resultados en la comunicación que por parte del sistema TCP[[10]](#footnote-10).

### Capa física

Para modificar los parámetros de la capa física, debemos configurar los parámetros del router. Antes de cambiar valores, necesitamos realizar una conexión por ssh desde el ordenador hasta el router, para poder acceder a su configuración. Introducimos en una terminal: ssh root@192.168.x.1. En la x introducimos la red en la que se encuentra el router (uno o dos).

Después de introducir la contraseña (root), podremos acceder a la interfaz del router. Desde aquí, ejecutamos el comando “wlan11p”, el cual nos mostrará la configuración de red inalámbrica que tiene el router cargada, es decir, los parámetros del nivel físico por defecto.

Las opciones siguientes presentan algunos de los valores que se pueden cambiar en la configuración inicial:

**Nombre**: wlan11p <interface> essid <parameter>

Nombre incluido en todos los paquetes para identificarlos como parte de esa red, uno por punto de acceso o router.

**Canal**: wlan11p <interface> channel <parameter>

Canales dentro de la frecuencia de 5.9 Ghz. Valores posibles: 20-70, 145-185.

**BSSID**: wlan11p <interface> bssid <parameter>

Es el nombre que se le ponen a los paquetes para identificarlos de alguna manera como parte de una misma red inalámbrica. A diferencia del essid, este se puede poner en varias BSS (puntos de acceso). Estilo FF:FF:FF:FF:FF:FF.

**Bandwidth**: wlan11p <interface> bandwitdh <parameter>

Nosotros trabajaremos con 10MHz, que así lo dicta el protocolo 802.11p

**Rate**: wlan11p <interface> rate <parameter>

Para los 10 Mhz, pueden ser estos valores => 3, 4'5, 6, 9, 12, 18, 24, 27, todo en Mbps.

**Diversity**: wlan11p <interface> diversity <parameter>

Según en qué valor se encuentre, estamos activando la diversidad al usar 2 antenas.

(0 on=> elección automática entre 1º y 2º antena, 1 se usa la 1º y 2 se usa la 2º)

La interface utilizada será la ath0, donde colocaremos nuestra antena en cada nodo. Antes de escribir cada sentencia en la terminal del router, se ha de escribir el comando “mountrw”, de tal manera que los cambios aplicados se guarden como iniciales para futuros encendidos de la máquina.

En el trabajo presentado, se ha optado por configurar una tasa de envío en el canal de 24Mbps, 21dBm de potencia, alimentados desde los vehículos y diversidad uno (ya que la comunicación se realizará con uno antena en cada punto).

### Parámetros de aplicación

El primer punto que se trata en este apartado será el circuito de la universidad. Como se ha mencionado con anterioridad, para la aplicación es necesario que el trayecto, por el que el vehículo se va a desplazar, esté parametrizado con puntos de referencia. Dichos puntos están dados como longitud y latitud, formando un sistema de coordenadas x – y.



Figura . Posición semáforo e inicio del circuito.

En la figura 24 apreciamos el mapa de la universidad. La flecha negra dibujada sobre él, indica el sentido en el que se han obtenido las coordenadas del trayecto. En este caso se han recogido en el sentido a favor de las agujas del reloj. Este dato ha sido importante tenerlo en cuenta a la hora del cálculo de distancias vehículo-semáforo.

En la parte inferior de la imagen se observa la posición en la que se encuentra el semáforo (Longitud: 3.818465 W, Latitud: 40.279046 N).

El funcionamiento básico del sistema pasa por la circulación del vehículo por el trayecto marcado en blanco, en forma de óvalo vertical. El conductor pasará cerca del semáforo para que se haga posible la comunicación y se vea en funcionamiento el sistema.

Por otro lado tenemos la tecnología **In road**. Consiste en la detección de si el vehículo se encuentra dentro o fuera del carril del circuito. Para ello se fija un ancho de carril máximo para el cual, toda distancia desde mi posición hasta el centro de la carretera superior a este tope, será considerada como fuera del circuito.

Otros parámetros de interés son los delimitadores de zona. Estos se encargan de dividir el trayecto en tres zonas, en la que en cada una de ellas el comportamiento del programa será diferente, mostrando por lo tanto diferentes mensajes de recomendación al conductor. Son los metros desde el semáforo para delimitar la zona dos (7 metros desde el semáforo), zona uno (50 metros) y el resto del circuito es la zona tres.

En cuanto a los límites de velocidad, se ha establecido como máximo los 50 km/h permitidos en zonas urbanas. Se ha visto necesario la imposición de un límite de velocidad mínimo, ya que se trata de ajustarnos a la realidad, y una velocidad menor de 12 km/h se hace difícil e incómoda de llevar y tener controlada al volante.

Por la parte del semáforo, además de las coordenadas en las que se encuentra fijado, encontramos otros hitos que determinar, principalmente en tema de tiempos y temporizaciones. Como se ha explicado, el semáforo es una máquina de estados en la que pasa de verde a rojo y permanece en ambos estados durante un período. Dicho período es fácilmente moldeable en el programa.

El segundo dato de importancia en la parte emisora son los tiempos de frecuencia de las balizas. El semáforo se encarga de enviar mensajes de su estado y el tiempo que queda para cambiar cada cierto intervalo de tiempo fácilmente modificable en el programa. Para el proyecto se ha decidido poner estos tiempos en 0.01 segundos. Es decir, se enviarán un total de 100 mensajes por segundo.

El resto de datos iniciales de programa, son simplemente inicializaciones de variables necesarias para el funcionamiento global del sistema. Estos parámetros tales como direcciones IP, puertos de conexión y otros, no se podrán cambiar, ya que si esto ocurre el sistema no podría funcionar.

## Implementación APP v-Road

A continuación se va a proceder a la explicación de la funcionalidad de cada módulo desarrollado para coordinarse entre sí y configurar la inteligencia del programa por parte del vehículo.

### Functions.py

Debido a la complejidad y extensión del código se ha visto necesaria el desarrollo de este módulo. En él se proveen funciones que se usan con frecuencia en el programa principal. Así se ha conseguido desarrollar una aplicación fácilmente legible y entendible en cuanto a funcionamiento lógico.

Entre las funciones más representativas encontramos:

* “Ms\_to\_kmh”: transformación de metros por segundo (unidades utilizadas por el gps y el semáforo respectivamente) a kilómetros por hora (unidades más representativas a la hora de reflejar velocidades.)
* “Speed\_Limits”: marca los límites de velocidad, tanto el superior como el inferior.
* “Media\_speeds”: realización de la media de velocidades recomendables almacenadas. En ella se realiza un cálculo que vendrá marcado por el número de valores guardados, hasta un límite de 7 números.
* “Data\_extractor”: realiza el Split de información de los mensajes recibidos desde el semáforo en el vehículo.
* Otras: el resto de funciones han servido como inicialización de parámetros, importante a la hora de reiniciar máquinas de estado.

### Distance\_Converter.py

Una de las principales cuestiones a resolver. Debido a que no se han utilizado mapas ni ayudas externas de APIs de geolocalización ni programas del estilo, se necesitaba crear un modelo de medición.

Para la realización de este modelo, se ha llevado a la simplificación total de las mediciones. La forma más fácil de calcular distancias en un sistema de coordenadas es usar el cálculo de la distancia Euclidea. Por esto, la distancia entre dos puntos se define como la raíz de la diferencia entre la resta de sus coordenadas en X al cuadrado y la resta de sus coordenadas en Y al cuadrado. Se ve expresado en la siguiente fórmula:

Pero esta distancia no refleja el recorrido del vehículo por una carretera, que es una trayectoria curvilínea en R2 .

Para ello se ha parametrizado el recorrido a realizar por el vehículo mediante sus coordenadas. De esta forma, para calcular distancias en las que intervienen curvas, asumimos que:

1. El intervalo entre dos puntos de referencia del circuito es un segmento.
2. Los puntos de referencia están lo suficientemente cerca entre sí. Esto nos permite dividir la trayectoria curva en pequeños segmentos.
3. La distancia total en la suma de la longitud de los segmentos.

Así pues, para el cálculo total de distancia entre dos puntos cualesquiera del trayecto, se siguen estos pasos:

* Lectura de posición geográfica del vehículo mediante GPS.
* Identificación del punto de referencia del vehículo.
* Cálculo de distancia Euclídea entre nuestra posición y el punto de referencia.
* Identificación del punto de referencia del semáforo (coordenadas fijas).
* Cálculo de distancia Euclídea entre el semáforo y su punto de referencia.
* Suma de distancias: distancia entre los puntos de referencia comprendidos entre el del vehículo y el del semáforo, distancia entre semáforo y punto de referencia y distancia entre coche y punto de referencia.

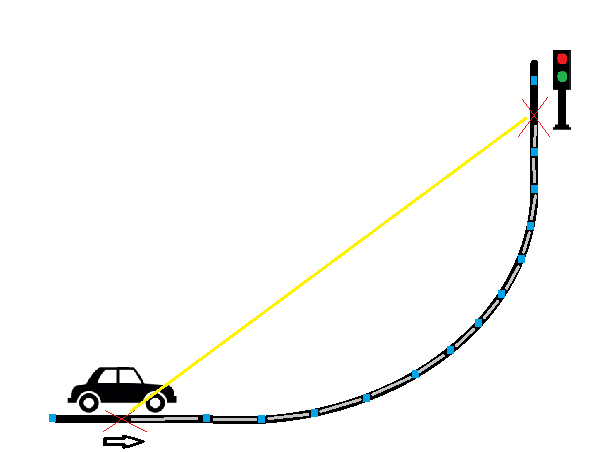


Figura . Distancia Euclídea vs distancia curvilínea

En la figura anterior se puede apreciar el cálculo realizado. Los puntos azules son los puntos de referencia que hemos cogido del circuito. El objetivo es calcular la distancia entre ambas cruces rojas. Si usásemos directamente la distancia Euclídea (línea amarilla), nos aparece el problema de las curvas. En vez de esto, calculamos la distancia entre cada punto de referencia, y las sumamos. Así podemos obtener distancias entre puntos sin importar la geometría del circuito.

Para que esto sea posible, primero se ha de establecer un sentido del recorrido. Dependiendo de para qué sentido se recorra el trayecto, el punto de referencia elegido será uno u otro (de los dos posibles más cercanos a las coordenadas).

En cuanto al cálculo de la distancia entre unas coordenadas (tanto de vehículo como de semáforo) y el punto de referencia elegido, no se realiza directamente. Para ello se usa el teorema de Pitágoras, ejemplificado su uso en la siguiente figura:

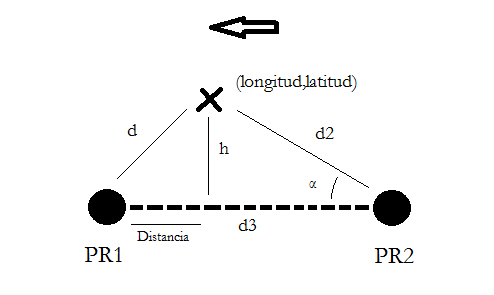


Figura . Ejemplo cálculo Distancia

Poniendo como ejemplo el sentido marcado por la flecha de la figura y siendo la X el punto de coordenadas obtenido por el GPS para el vehículo, vemos como se ha asignado el punto de referencia PR1 al vehículo. Por consiguiente, se asigna como segundo punto de referencia el PR2. De esta forma conseguimos retratar la posición entre dos puntos de referencia. Tanto d como d2, son distancias que podemos calcular mediante la distancia euclídea, ya que tenemos sus coordenadas. En cuanto a d3, se corresponderá con la distancia previamente calculada y almacenada entre esos dos puntos de referencia. Por último se realizan los siguientes cálculos trigonométricos para introducir estas 3 distancias en el teorema y calcular Distancia:

Finalmente:

Lo que hemos conseguido es el cálculo de una distancia entre, la proyección de las coordenadas en la línea imaginaria entre puntos de referencia y el punto de referencia elegido.

Para el cálculo de distancias en metros, se ha de realizar la transformación al sistema UTM con la ayuda de otro módulo. Por último y para agilizar este procesamiento, al comienzo del programa del vehículo, se crea una base de datos en la que se mantienen todas las distancias entre los puntos de referencia consecutivos. Al ser un circuito fijo, con tener las coordenadas de todo el camino, podemos crear esta base y usarla en cualquier momento del cálculo. Así se consigue un mejor rendimiento y una mejora en la eficiencia del programa, ya que es un dato muy recurrido a lo largo de todo el funcionamiento de la aplicación.

Entre las funciones más importantes de este módulo encontramos:

* “dist\_coords”: calcula la distancia euclídea entre dos coordenadas.
* “dist\_proyectada”: mediante la fórmula obtenida desde los cálculos trigonométricos aplicados al teorema de Pitágoras, obtenemos la distancia entre la proyección de las coordenadas y el punto de referencia.
* “basedatos\_distancias”: como se ha explicado, la distancia entre los puntos de referencia serán valores fijos y muy recurridos a lo largo del procesamiento de datos, por lo que esta función las calcula y las almacena.
* “lectura”: se encarga de realizar una lectura de coordenadas, hora y velocidad actual del vehículo a través del GPS.
* “elección\_sig\_coord”: calcula cuáles son las coordenadas del otro punto de referencia, perteneciente a la franja de coordenadas en las que se encuentra tanto el vehículo como el semáforo.
* “caza\_iden”: encargado de obtener el punto de referencia más cercano a unas coordenadas.
* “franja\_coords”: unifica varias funciones para determinar la franja de puntos de referencia entre los que se encuentran unas coordenadas.
* “suma\_distancias”: lleva a cabo el cálculo de las diferentes distancias a tener en cuenta a la hora del cálculo total, ayudado de otras funciones, así como la suma total de cada una de ellas para la obtención del dato de interés.
* “in\_road”: función muy sencilla pero muy útil que nos reportará si nos encontramos dentro del trayecto o fuera.

### Deg2utm.py

Este módulo es el encargado de la transformación de las coordenadas definidas como longitud y latitud al sistema de Mercader UTM, en el que los puntos son metros y por consiguiente la distancia entre dos puntos también.

La principal función convierte un conjunto de coordenadas a la vez. Todo el cálculo se ha realizado a partir de la transformación de scripts de Matlab [18] en los que venía la conversión realizada al lenguaje Python.

Para convertir a este sistema las coordenadas obtenidas por el GPS en cada lectura, se ha tenido que transformar el script entero de tal forma que sólo coja un par de valores para transformar.

Este módulo se ayuda de la librería numpy, encargada de proporcionarnos funciones predefinidas, tales como cosenos y senos, necesarias para el cálculo de la conversión de coordenadas de un sistema a otro.

### Coords\_URJC.txt

La necesidad de tener controlado el trayecto mediante la recogida de las coordenadas que lo forman requiere que se cree este fichero. En él se encuentran cada par de valores, longitud-latitud, separados por una @ y se almacenan aquí. En el arranque del programa se procede a la lectura de todas las coordenadas y a su procesamiento para el cálculo de las distancias. Estos valores contenidos en este archivo formarán las coordenadas de los llamados puntos de referencia.

Un punto a tener en cuenta a la hora del procesamiento de todo el sistema, es el sentido con el que se han cogido las coordenadas. Al ser un circuito cerrado no es relevante dónde se empezó a la toma de coordenadas y dónde acabó, pero sí el sentido. Para este proyecto se ha realizado en sentido horario.

### Proc\_data\_map.py

Consiste en un módulo de poca extensión encargado de dar permisos de lectura al fichero de coordenadas, de abrirlo, extraer cada par de valores en éste presente y su procesado, dando como resultado dos variables en las que se almacenarán por un lado las longitudes y por otro lado las latitudes.

### Map\_converer.py

Será el encargado de recoger las coordenadas del fichero y su transformación al sistema UTM. Como resultado de la única función presente, obtenemos 4 variables, dos para las longitudes y latitudes y dos para sus análogos en el nuevo sistema.

### Data\_semaphore

Para el correcto funcionamiento del semáforo, necesitamos que ciertos parámetros vengan predefinidos. Al igual que estos, se ha visto útil el sacar estos valores fuera del programa principal del emisor, para que sean de fácil acceso y modificación a la hora de realizar los experimentos.

En este apartado encontramos los tiempos de duración de cada color, las coordenadas fijas del semáforo, el tiempo entre intervalo de balizas y otras inicializaciones de variables necesarias para el correcto funcionamiento de la máquina de estados que representa el semáforo.

Estos parámetros se han externalizado fuera del programa principal por la sencilla razón de poder ser llamados tanto por el vehículo como por el semáforo. Para implementaciones futuras, estos datos podrían ser descargados de bases de datos en internet para su descarga y uso en el vehículo.

### Data\_car.py

Como se ha hecho con los datos del semáforo, también para el vehículo se ha visto oportuno extraer los parámetros que podemos querer cambiar, a la hora de realizar pruebas, de forma fácil y rápida.

Entre las funciones que contiene, cabe destacar:

* “datos\_socket”: determinan la IP y el puerto donde va a recibir los mensajes del semáforo.
* “datos\_limites”: establece los límites de velocidad, así como el límite de anchura de carril para la determinación de si nos encontramos en el trayecto o no con la tecnología In road y los metros que dividen el trayecto en tres zonas.

Se ayuda de una tercera función que servirá para la inicialización de variables útiles para el funcionamiento. Estos valores los modificará el programa principal, por lo que, a la hora de instanciarlos, deben comenzar siempre de la misma forma.

### Controller.py

Llegados a este punto, nos encontramos el módulo más importante de la aplicación del vehículo, el controlador. Es el encargado de tomar todas las decisiones en función de todos los datos obtenidos del exterior o producidos por otras partes del programa. Gracias a los mensajes que procese y las velocidades que calcule, el conductor tendrá una mayor probabilidad de cruzar el semáforo en verde, sin tener que detenerse y de una manera confortable.

En primer lugar, diferenciamos las tres zonas del circuito ya que en cada una de ellas el funcionamiento del mismo será diferente.

Como se observa en la figura 27, el programa es sometido a un proceso de detección de zonas. En él, dependiendo de la distancia a la que nos encontremos del semáforo, el vehículo se encontrará en una u otra.

Figura . Detección zonas.

En la zona uno, el vehículo está dentro del área de cobertura del semáforo por lo que estará recibiendo cuando llegue a esta zona. La aplicación tomará una decisión con el primer mensaje que reciba en esta zona y le mostrará al conductor el consejo.

Al cambio de zona, el vehículo se introduce en la zona 2. El comportamiento de ésta será el mismo que al entrar en zona uno con la distinción de que aquí importa si estamos en movimiento o no. Esto quiere decir que, al ser una zona muy próxima al semáforo, si el vehículo detecta que está parado, avisará de si hay que arrancar porque el semáforo esté en verde o de si hay que mantenerse parado porque el semáforo esté en rojo. Esta es una forma de aumentar en comodidad, ya que muchas veces el semáforo nos queda poco visible porque lo tapa alguna parte de nuestro vehículo y tenemos que estar agachándonos o girando el cuello para poder verlo. Así sabemos cuándo va a cambiar de estado mirando nuestro display.

A continuación se desglosan las 3 zonas.

Figura . Zona 1.

En la figura 28 se observa el funcionamiento del programa controlador en la zona uno. Cuando recibimos el primer mensaje en esta zona, dependiendo de si el estado actual del semáforo es verde o rojo se tomarán unas acciones u otras. En caso de estar en verde, tenemos que ver si con los segundos recibidos para el cambio y la distancia a la que nos encontramos podemos llegar a cruzarlo en verde. Esto se llama *decisión a priori*. Sus resultados son OK, llegamos al cambio, NO OK, no llegamos al cambio. Llegar al cambio quiere decir que la velocidad calculada como resultado de la división de segundos para el cambio entre distancia hasta semáforo se encuentra dentro de los límites de velocidad establecidos.

Para el caso OK en verde, necesitamos llevar la velocidad calculada como mínimo, por lo que se intenta aumentar para mantener un cierto margen, siempre que esté dentro de los límites. En caso de que no se pueda aumentar la velocidad recomendada, se mostrará como velocidad a llevar la velocidad calculada.

Para el caso NO OK en verde, tenemos que distinguir si el culpable de dicha decisión es porque la velocidad recomendable es muy alta o muy baja. Cuando está por encima del límite superior (Vup), calculamos lo que en el programa se llama *velocidad de rojo.* Consiste en una velocidad baja, dentro de los límites siempre, que nos permita pasar el tiempo que queda en verde y el tiempo de rojo en movimiento hasta que justo al llegar al semáforo se cambie a verde. De esta forma conseguimos una conducción más confortable ya que no hemos tenido que parar y volver a arrancar, ahorrando también en tiempo y gasolina. Cuando esa velocidad no es viable por ser demasiado baja, avisamos al conductor de que se prepare para frenar. Aquí aparecerían dos rayas horizontales en la parte de la velocidad; “--”. Por otro lado, si el culpable es el límite inferior, tratamos de aumentar la velocidad recomendable hasta que entre dentro de los límites mostrando esa velocidad como mínima.

Cuando el primer mensaje recibido en la zona 1, tiene como estado actual el rojo, se pretende que cambie. Es decir, interesa que cuando se llegar al semáforo con una velocidad que permita pasar cuando el estado haya cambiado a verde.

La situación OK en rojo quiere decir que llegamos al semáforo justo al cambiar de color. Para asegurarnos de que pasamos este estado y cruzamos el semáforo en verde, intentamos disminuir la velocidad calculada, disminuyendo en un segundo el tiempo que queda para cambiar y la presentamos al conductor. Por otro lado, la situación NO OK nos presenta de nuevo dos divisiones. Si el culpable es una velocidad que excede por encima de la permitida, bajamos el cálculo hasta que esté dentro de los límites. De esta forma conseguimos llegar cuando el semáforo ya ha cambiado. Si el culpable es una velocidad que no llega al mínimo, avisamos al conductor de que vaya preparándose para detenerse, ya que no se podrá ir más despacio para que vuelva a cambiar a verde.

La siguiente zona, representa los metros finales antes de llegar al semáforo. Aquí, además de que preocupe si el semáforo está en verde o rojo, la aplicación detecta si el vehículo se encuentra en movimiento. En caso de que haya movimiento, el controlador seguirá el mismo esquema que para la zona uno ya que se puede considerar una prolongación de ésta.

En caso de que el vehículo se halle parado, nos centraremos en la jerarquía de decisiones de la figura 29. Para el estado verde, avisamos al conductor de que tiene que arrancar ya que se encuentra obstaculizando el paso de otros posibles vehículos. Para el estado rojo, avisamos al conductor de que se tiene que preparar, indicando los segundos que quedan para el cambio.

Figura . Zona dos, sin movimiento del vehículo.

Por último, desde que cruzamos el semáforo y hasta que el vehículo vuelve a introducirse en la zona uno, el sujeto se encontrará en la zona tres. En esta zona, no es importante si está recibiendo o no, si el semáforo está en verde o no puesto que ese semáforo ya no existe para los cálculos del vehículo debido a que se ha dejado atrás.

Las funciones que contiene este módulo se dividen en:

* “decisión a priori”: usa los segundos para el cambio de estado de semáforo y la distancia hasta éste para calcular una decisión a priori de si el vehículo llegará a cruzar el semáforo en verde o no.
* “guilty”: nos dice si las velocidades calculadas no son válidas porque nos pasamos el límite inferior o el superior.
* “zona”: dependiendo de la distancia a la que se encuentre el vehículo del semáforo nos proporcionará la zona en la que se encuentre, uno, dos o tres.
* “movimiento”: detecta si el vehículo está en movimiento o no, importante para la zona dos.
* “bajar velocidad” y “subir velocidad”: encargados de modificar la velocidad calculada a priori para llegar al semáforo en verde.
* “controlador”: parte más importante del sistema, ya que se encarga de procesar las decisiones a mostrar por el display y que servirán de ayuda al conductor.

### Interfaz.py

La comunicación vehículo semáforo es la principal que se produce en el sistema. No se debe olvidar otro tipo de comunicación muy importante y es la producida entre el vehículo y el conductor. Ésta se producirá en ambos sentidos, al contrario que la principal, ya que el semáforo es el emisor y el vehículo el receptor. El conductor envía mensajes al vehículo cuando acelera o disminuye su velocidad, ya que la posición GPS variará más rápido o más lento y esto son datos que el vehículo debe procesar. El display debe representar datos sencillos y de una manera que sean fácilmente reconocibles y entendibles.

Por todo esto, se ha decidido crear este módulo el cuál se encargará de mostrar los mensajes de recomendación calculados sobre un fondo negro con letras naranjas fosforitas, muy llamativas y fácilmente visibles dentro del vehículo.

Las funciones que se encuentran en este apartado son:

* “graficos”: encargado de inicializar y centrar los valores a mostrar en las posiciones previamente establecidas dentro del display.
* “imprime\_interfaz”: encargada de mostrar los valores de las decisiones tomadas cada vez que hay un cambio de zona. De esta forma se consigue no aturdir al conductor con cambio de mensajes con cada información recibida. Al final, se encuentra la parte en la que se abre el fichero logfile y se vuelca todos los datos que se ha ido procesando y recibiendo a modo de registro para poder realizar gráficas que muestren los resultados obtenidos. Esta escritura se realiza cada cierto tiempo para no tener un acceso a disco por dato recibido, tarea que ocuparía mucho tiempo de CPU que se debe usar para otros procedimientos.

### Car.py

En este módulo de programación se determina el comportamiento principal de la aplicación. Es la parte de mayor complejidad. Se encargará de recopilar los datos recibidos por el semáforo y produciendo una serie de mensajes y velocidades recomendables para ayudar al conductor a que cruce el semáforo en verde. Además se intenta conseguir una conducción más confortable y un ahorro de tiempo y combustible.

Para llevar a cabo la coordinación de todas las sentencias usadas, se ha optado por crear cuatro procesos, uno principal y tres de ayuda. Con su coordinación, mientras trabajan en paralelo, se conseguirá obtener un mayor rendimiento de la aplicación.

El primero de ellos aborda el tema de la interfaz gráfica. A través de ella se mostrarán los datos y mensajes al conductor, de la manera más sencilla e intuitiva posible.

Una vez lanzada la interfaz, se prepara el puerto por el que se va a conectar el demonio GPSd. Se mata cualquier proceso anterior de GPSd (por si hubiera alguno usando el puerto) y se crea el proceso GPS. Ahora tendremos un proceso que irá actualizando los datos de las coordenadas y velocidad actual del vehículo, el cual será usado por el principal cuando necesite esta información.

El siguiente paso es la inicialización de variables y la obtención de datos del semáforo. Como se ha explicado con anterioridad, obtendremos los parámetros del semáforo de otro módulo. Esto equivaldría a la obtención de los datos de alguna base de datos donde se pudieran tener almacenados todos los datos de los semáforos de la red de carreteras.

Las siguientes sentencias corresponden al desarrollo del proceso de las comunicaciones. Este será el encargado de comunicarse con el semáforo, para obtener los datos recibidos más actualizados posibles y de fácil acceso para el programa principal. En dicho hilo de ejecución, se irán contando el número de mensajes recibidos de tal manera que, comparándolo con el número de mensajes enviados, podemos sacar estadísticas de pérdidas de paquetes.

Para el cálculo de distancias, se necesitaba tener parametrizado el trayecto. A partir de ahí, necesitábamos medir las longitudes entre esos puntos y guardarlas en una base de datos de fácil acceso en cualquier punto del programa.

Acto seguido identificamos entre que dos puntos de referencia hemos puesto el semáforo. Esto nos será útil en las pruebas, ya que se podrá poner el semáforo en cualquier lugar del recorrido que el programa, con las coordenadas, sabrá ubicarlo y tenerlo en cuenta.

A partir de aquí se entra en el proceso principal. Éste se basa en un bucle infinito, marcado por el proceso interfaz y con un tiempo de refresco de 500ms.

Lo primero que hace en cada iteración es buscar un estado actualizado de todo el sistema, recopilando datos de todos sus procesos.

Después, hace una lectura de coordenadas para determinar su posición. Convierte las coordenadas a metros para facilitar los cálculos de distancias y determina entre qué puntos de referencia se encuentran las coordenadas obtenidas.

Acto seguido analiza si estamos dentro de la carretera o no. Para ello, calcula la distancia entre la coordenada del coche y el punto de referencia elegido. Para el caso en el que dicha distancia sea mayor que un umbral, definido como el ancho de un carril, nos encontraremos fuera del trayecto. En este caso nos lanzará un mensaje advirtiéndonos de la distancia que nos queda para llegar al camino.

Cuando nos encontremos en carretera el programa tomará el resto de decisiones. La primera de ellas es el cálculo de la distancia. Puesto que tenemos nuestra posición y la del semáforo se sabe en todo momento, podemos calcular cuántos metros quedan para llegar, dependiendo siempre del sentido del recorrido. Este dato es uno de los que se mostrará al conductor.

Acto seguido, comprobamos si hemos recibido algún mensaje o no. Esto ocurrirá siempre que el vehículo se encuentre dentro del área de cobertura del semáforo.

Para no usar mucho tiempo de CPU en la acción de escritura en el fichero logfile, tenemos un contador de mensajes procesados y al llegar a un tope preestablecido, todos los mensajes se volcarán en el fichero de salida, optimizando así el procesado del vehículo.

El mensaje recibido será analizado y extraído de él los datos de interés; estado y tiempo para el cambio de estado.

Con el tiempo recibido y la distancia hasta el semáforo, calculamos la velocidad a la que se debería ir para llegar justo cuando se cambie de estado. Esta velocidad no es la que se muestra en la interfaz. Este valor calculado tiene un triple procesado. El primero consiste en una ventana de velocidades de tope limitado. En dicha ventana se van introduciendo las velocidades calculadas para un mismo estado, de tal manera que la velocidad calculada para un estado, sea una media de velocidades. Esto nos ayuda a que la rigidez del sistema no nos diga que no vamos a llegar al cambio porque hayamos reducido en algún momento la velocidad debido por ejemplo a un paso de peatones previo o algún otro imprevisto. El segundo procesado consiste en introducir dicho valor en el módulo del controlador. Éste, con la media calculada, toma una decisión a priori. Es decir, nos dice si llegamos o no al cambio. Pero nuestro programa va más allá, ya que dependiendo de si el estado es verde o rojo, si nos encontramos en cada una de las tres zonas y de si nos encontramos en movimiento o parado, el tercer procesado de los datos tratará de darnos un consejo que haga que modifiquemos nuestro comportamiento frente al volante.

Hasta aquí sería el comportamiento del programa mientras estemos dentro del área de cobertura. En el caso de que no recibamos nada del semáforo, pueden darse dos casuísticas:

* Caso en el que salimos de la zona de cobertura y entramos de nuevo en zona tres. Se le presentará de nuevo la interfaz estándar al conductor, en el que el mensaje será el de URJC y la velocidad recomendable “--”, ya que nos encontramos en el área de caso omiso o sin recibir.
* Caso en el que hemos recibido mensajes pero por causas inesperadas, dejamos de recibir, estando en zona uno o dos. En estas se debería estar mostrando consejos al conductor acorde a los mensajes que no se están recibiendo. Para que el funcionamiento sea ininterrumpido hasta que el semáforo vuelva a cambiar, el vehículo utiliza el último estado y los últimos segundos para el cambio recibidos. De esta forma, restándoles un tiempo de espera a conexiones que posee el vehículo, podrá seguir realizando sus cálculos sin estar recibiendo mensajes. Una vez cambiado que la simulación notifica un cambio de estado, todas las variables se vuelven a inicializar y el programa se pone a la espera de posibles mensajes.

Cada vez que el mensaje recibido o simulado ha sido pasado por el controlador para que tome una decisión, se llamará a la subrutina de interfaz. De esta forma se le mantendrá al conductor informado de todo cambio producido en el sistema.

Todos estos procesos se repiten una y otra vez con las variaciones producidas por la posición del vehículo y los mensajes recibidos.

## Implementación APP Semáforo

A continuación se explican los dos módulos que componen el subsistema del semáforo.

### Data\_semaphore.py

Esta parte del programa, va dirigido tanto al vehículo como al semáforo. En ella se encuentran los tiempos de balizas y los tiempos de duración de cada color así como las coordenadas del semáforo y datos de inicialización de la máquina de estados.

Para el correcto funcionamiento, se ha decidido poner una posición fija al semáforo dentro del campus. Esto es debido a que necesitamos su referencia de localización en el mapa para poder medir la distancia que queda entre el vehículo y el semáforo.

Al extraer estos parámetros en otro módulo se consigue que su manipulación sea más fácil a la hora de realizar pruebas.

### Semaphore.py

Este módulo de programación constituye la inteligencia principal de la máquina semáforo.

En el arranque se obtienen los datos de inicialización del semáforo como los tiempos de duración de cada color, los tiempos entre balizas, las coordenadas donde se ubica en el mapa y datos acerca del socket.

Lo primero que realiza es la conexión a un socket para poder enviar los mensajes posteriores. Esto lo realiza mediante un bucle para no parar de intentar conseguir una conexión hasta que se tenga éxito.

El siguiente punto es el inicio de la máquina de estados. En éste se inicia una variable cogiendo la referencia de tiempo actual en la que comienza el sistema a funcionar.

El paso que viene a continuación es la recogida de datos de los argumentos introducidos por terminal. El primero argumento que se debe introducir es la versión del fichero logfile que tendrá el programa como salida de datos generados. De esta forma se hace más fácil y accesible generar diferentes archivos para las diferentes pruebas que se quieran realizar. El segundo argumento y último determinará el modo de actuación del semáforo. Con el modo “–a”, (automático) el semáforo funcionará por sí solo, sin tener nadie capacidad de modificación de los estados. Con el modo “-c” (cambio), se podrá tener control sobre el funcionamiento del semáforo. Si durante la marcha de éste, se pulsa la letra g, forzaremos a que el semáforo se ponga en el inicio del estado verde (ya sea estando en verde o en rojo anteriormente). Por el contrario, si se pulsa la letra r, forzaremos a que el estado sea el rojo (al igual que para el caso anterior, es independiente el estado en el que se encuentre).

Una vez entrado en uno de los dos modos de actuación, el resto es común para ambos. Da comienzo la máquina de estados de la siguiente figura:

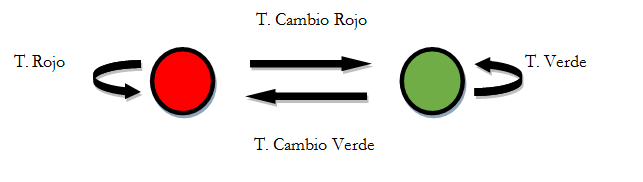


Figura . Máquina de estados del semáforo.

Como podemos observar en la figura 30, la máquina de estados consta de dos estaciones. Por un lado tenemos el color Verde, correspondiente al estado en verde del semáforo y por el otro lado tenemos el color Rojo, correspondiente al color rojo. El T. Cambio Rojo representa el cambio de estado de estación roja a verde. El T. Cambio Verde representa el cambio de estado de estación verde a roja. En cuanto a los T. Rojo y T. Verde, representan la permanencia en la misma estación, es decir, la máquina de estados permanecerá sin realizar ningún cambio mientras el tiempo de duración de cada color no haya llegado a su fin.

Para dar cabida a estos cambios, primero se actualiza el tiempo transcurrido desde el arranque del semáforo hasta el momento actual. Mientras que el tiempo transcurrido se encuentre entre el inicio y el tiempo de duración del estado rojo, el estado será rojo. Aquí se enviará un mensaje de baliza con el color y el tiempo restante para el cambio de estado que será recibido por el vehículo. Cuando el tiempo transcurrido se encuentre entre el tiempo de duración de color rojo y el tiempo de color rojo más el verde, el mensaje enviado será de estado verde, junto con el tiempo para el cambio correspondiente. En el momento en el que el tiempo transcurrido es mayor que este último valor, se reinicia la máquina de nuevo al estado rojo. Por ello, en el arranque del semáforo siempre va a comenzar por rojo y después continuará con el modo que le hayamos indicado.

En cada mensaje enviado el programa grabará en el logfile el estado en el que se encuentra, el tiempo que queda para el cambio y la hora. Ésta última se necesitará para sincronizar los datos obtenidos con el logfile del vehículo y poder obtener gráficas coherentes a la realidad.

##### CAPÍTULO IV

# Casos de estudio

2. analisis que me he planteado
3. (clasificacion analisis, =========>> probar RandomForest, Grading boosting trees, regresion logistica multinomial\* opcional
4. benchmarks y resultados

## Realización de pruebas

Este trabajo buscaba desarrollar una idea para comunicar un vehículo y un semáforo. Pero eso no era lo único, se quería llevar a la práctica y que no se quedara en una simple propuesta de modelo. Los resultados obtenidos son la evidencia de que se ha conseguido crear un prototipo del sistema pensado inicialmente.

Se han realizado dos tipos de pruebas: Pruebas en el laboratorio, previas al desarrollo software, con los equipos de transmisión. Estas pruebas tuvieron como objetivo familiarizarse con el equipamiento y comprobar sus prestaciones. Los otros experimentos son en movimiento mediante el software desarrollado.

### Pruebas en estático

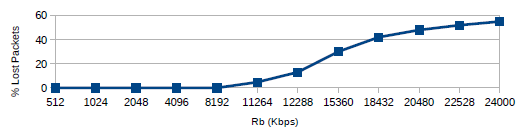
Estas pruebas fueron la primera toma de contacto con los equipos que usaban el protocolo 802.11p. En estas se usó el inyector de tráfico Iperf para evaluar las prestaciones de estos equipos.

Las siguientes gráficas muestran los diferentes experimentos que se realizaron. En ellas ve como se ha variado tanto la modulación como la inyección de tráfico UDP desde nivel de aplicación. Las comunicaciones han tenido una duración de 5 minutos por modulación y esquema de codificación. A partir de ahí, se ha ido variando los datos a transmitir desde el nivel de aplicación, aumentando la tasa.

Para llevar a cabo el cambio de modulación, se ha de cambiar la tasa de la capa física. Cada una de ellas tiene asociada una tasa de transmisión y un esquema de codificación, presentes en la tabla de la figura 11.

SIMULACIONES DE 300 SEGUNDOS MEDIANTE CONEXIÓN UDP

64 QAM (2/3) - PHY 24 Mbps



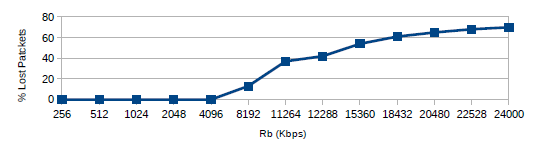
Figura

Comandos de aplicación:

IRIS=> Iperf -s -u -i 1

HERMES => Iperf -c 192.168.1.2 -u -t 300 -b 0.256m -i 1

16 QAM 1/2 - PHY 12 Mbps



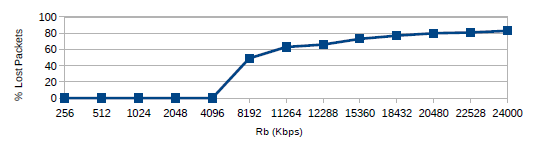
Figura

Comandos de aplicación:

IRIS=> Iperf -s -u -i 5

HERMES => Iperf -c 192.168.1.2 -u -t 300 -b 0.256m -i 5 >> 16 QAM1-2

QPSK ½ - PHY 6Mbps



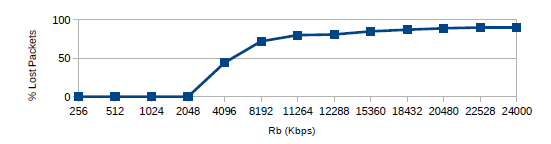
Figura

Comandos de aplicación:

IRIS=> Iperf -s -u -i 5

HERMES => Iperf -c 192.168.1.2 -u -t 300 -b 0.256m -i 5 >> QPSK1-2

BPSK ½ - PHY 3Mbps



Figura

Comandos de aplicación:

IRIS=> Iperf -s -u -i 5

HERMES => Iperf -c 192.168.1.2 -u -t 300 -b 0.256m -i 5 >> BPSK1-2

Las figuras de la 31 a la 34, muestran los diferentes resultados obtenidos de estas conexiones. En el eje de abscisas se presenta el porcentaje de paquetes perdidos frente al eje de ordenadas que hace referencia a la tasa del nivel de aplicación.

En cada una de las gráficas se observa que se ha procedido al envío de datos, manteniendo la codificación y modulación del nivel físico, por lo tanto también la tasa de transmisión de dicho nivel, mientras que se aumentaba cada 5 minutos el número de paquetes inyectados desde nivel de aplicación (aumento de tasa en el envío).

Estos datos recogidos pertenecen a pruebas del medio en estado parado, es decir, sin movimiento de los nodos. En la comunicación del proyecto, necesitamos un ancho de banda de canal en el que la comunicación se produzca de manera robusta, ya que no existe retransmisión de paquetes a nivel de transporte (UDP). Es importante aclarar este detalle: El protocolo de transporte utilizado ha sido UDP, un protocolo no orientado a conexión y sin retransmisiones a nivel de transporte. En cambio, 802.11 sí contempla las retransmisiones a nivel de enlace.

Las antenas usadas para estas pruebas fueron antenas WiFi de 15 cm. Las antenas de los experimentos en movimiento han permitido obtener una buena comunicación y una buena comunicación con la tasa de 24MB, es decir, 64-QAM (2/3). Esta modulación ha sido la elegida para estas siguientes pruebas.

### Pruebas en movimiento

Una vez acabado el desarrollo del sistema se llevó a la práctica en el circuito con los vehículos.

Las primeras pruebas realizadas se tuvieron que repetir durante varios días. Se realizaron pruebas del correcto funcionamiento de las aplicaciones utilizando una red cableada Ethernet, en laboratorio. A pesar de ello surgieron muchos contratiempos durante las pruebas de campo y se tuvieron que repetir las pruebas durante varios días (véase el apéndice A).

Entre los errores más destacados están el de la carga del fichero de coordenadas erróneo y la poca previsión de realizar pruebas con primer sistema radiante usado. La carga del fichero erróneo consistió en que al realizar pruebas en el circuito final (URJC) se dejó cargado el fichero de coordenadas del circuito secundario de pruebas. Evidentemente, las distancias que se calculaban no eran las correctas. El segundo error fue el de no realizar pruebas con el sistema radiante previamente. Las primeras antenas que se usaron en movimiento fueron dos antenas de 40 cm de longitud ancladas con bases magnéticas a los vehículos. Pero veíamos que la distancia que nos proporcionaban no llegaba a más de 30 metros. Transmitiendo a máxima potencia (21dBm), esto no era lógico. Posteriormente se realizaron pruebas en estático con este sistema radiante y un analizador de espectros y se comprobó que la potencia recibida no era la deseada, por lo que decidimos probar con otras configuraciones de antenas. Descubrimos que las bases magnéticas hacían que se perdiese gran parte de la potencia transmitida, por lo que decidimos cambiar las antenas.

Las antenas que se usaron finalmente son, por parte del semáforo, una antena directiva Rohde Schwarz de tipo log periódica, directiva y con gran ancho de banda, y por parte del vehículo, un dipolo de 15cm de 2,4GHz y 5,8 a 6 GHz.

Cada prueba consistía en dar tres vueltas a la carretera perimetral de la URJC mientras se recogían datos de velocidad, posición, paquetes recibidos y marcas temporales.

## Análisis información recibida

Se ha dividido el espacio del recorrido en segmentos de 15m. Los mensajes recibidos en cada uno de estos tramos son una variable aleatoria cuyas muestras son la concatenación del número de mensajes recibidos en cada una de las tres vueltas dadas al circuito por experimento. El problema del cálculo de estadísticos de estas variables, es que cada una tiene un tamaño diferente. Además, el número de muestras por variable es muy pequeño.

Hemos encontrado que las técnicas de remuestreo Bootstrap [19] es una buena aproximación al problema de la falta de muestras. Esta técnica consiste en remuestrear, con reemplazamiento, los datos originales y obtener con ello una nueva variable aleatoria. El estadístico elegido ha sido la media, que denotaremos como media Bootstrap.

En las gráficas obtenidas, el eje vertical representa los mensajes recibidos por segundo y el eje horizontal es el lugar del recorrido. La línea verde que representa el semáforo virtual, se encuentra en el punto 1350m aproximadamente. Por otro lado, la línea discontinua vertical representa la posición de la antena del semáforo. La línea discontinua horizontal muestra el máximo de mensajes enviados por el semáforo, que son 100 mensajes por segundo.

Los puntos de la línea roja representa la media de las medias Bootstrap para los diferentes intervalos de 15m. La sombra que se encuentra alrededor de la línea, de color rosado oscuro, indica la desviación estándar de las medias Bootstrap. La sombra de color rosado claro hace referencia al valor máximo y mínimo de las medias.

A modo general, se puede observar que la zona de máxima cobertura de es de unos 150 metros. En esta área se ve que se recibe casi la totalidad de paquetes (100 por segundo).

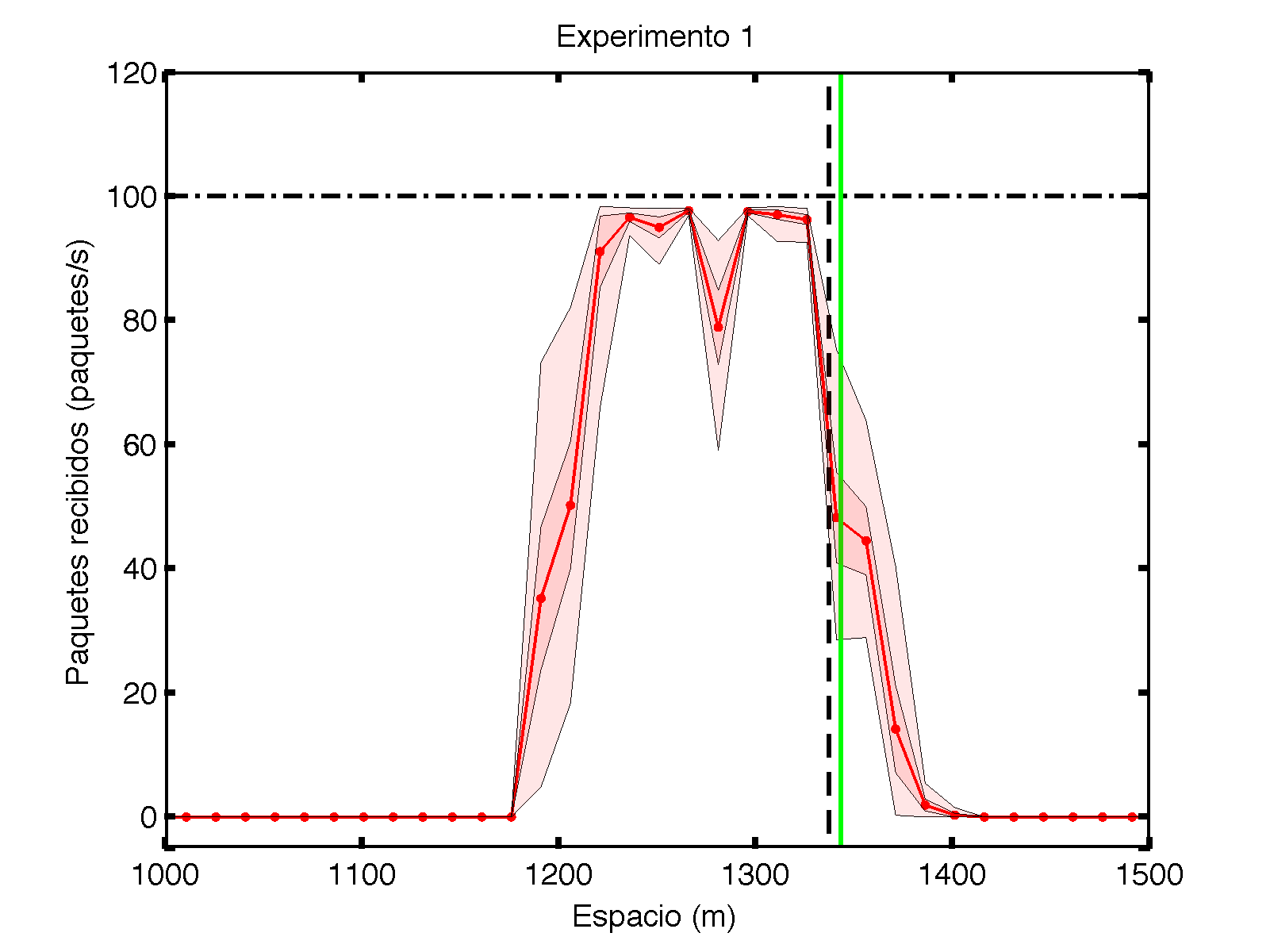


Figura . Paquetes recibidos/s del experimento 1. V = 40km/h.

En esta gráfica podemos observar que la recepción en casi toda el área de cobertura es superior al 90%, rozando incluso en algunos puntos el 100% de los paquetes recibidos. A partir del paso por el semáforo observamos que los paquetes empiezan a perderse en gran cantidad, puesto que la antena es muy directiva y la comunicación que se produce es con los lóbulos traseros del diagrama de radiación.

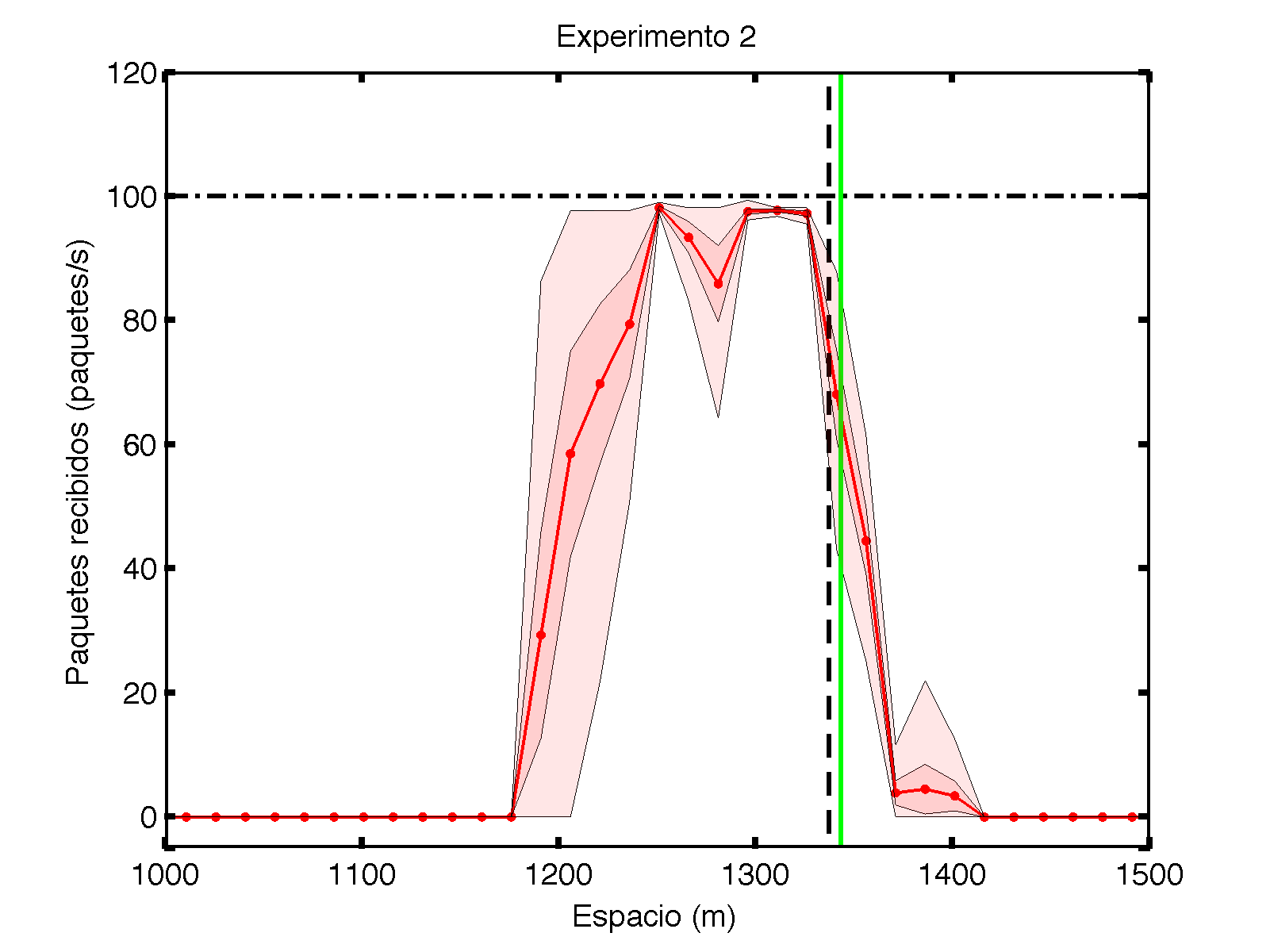


Figura . Paquetes recibidos/s del experimento 2. V= 40km/h.

El 2º experimento observamos que la recepción de paquetes ha sido peor que en el caso anterior. Lo podemos notar en la entrada a la zona de cobertura, los valores medios son más bajos que los anteriores.

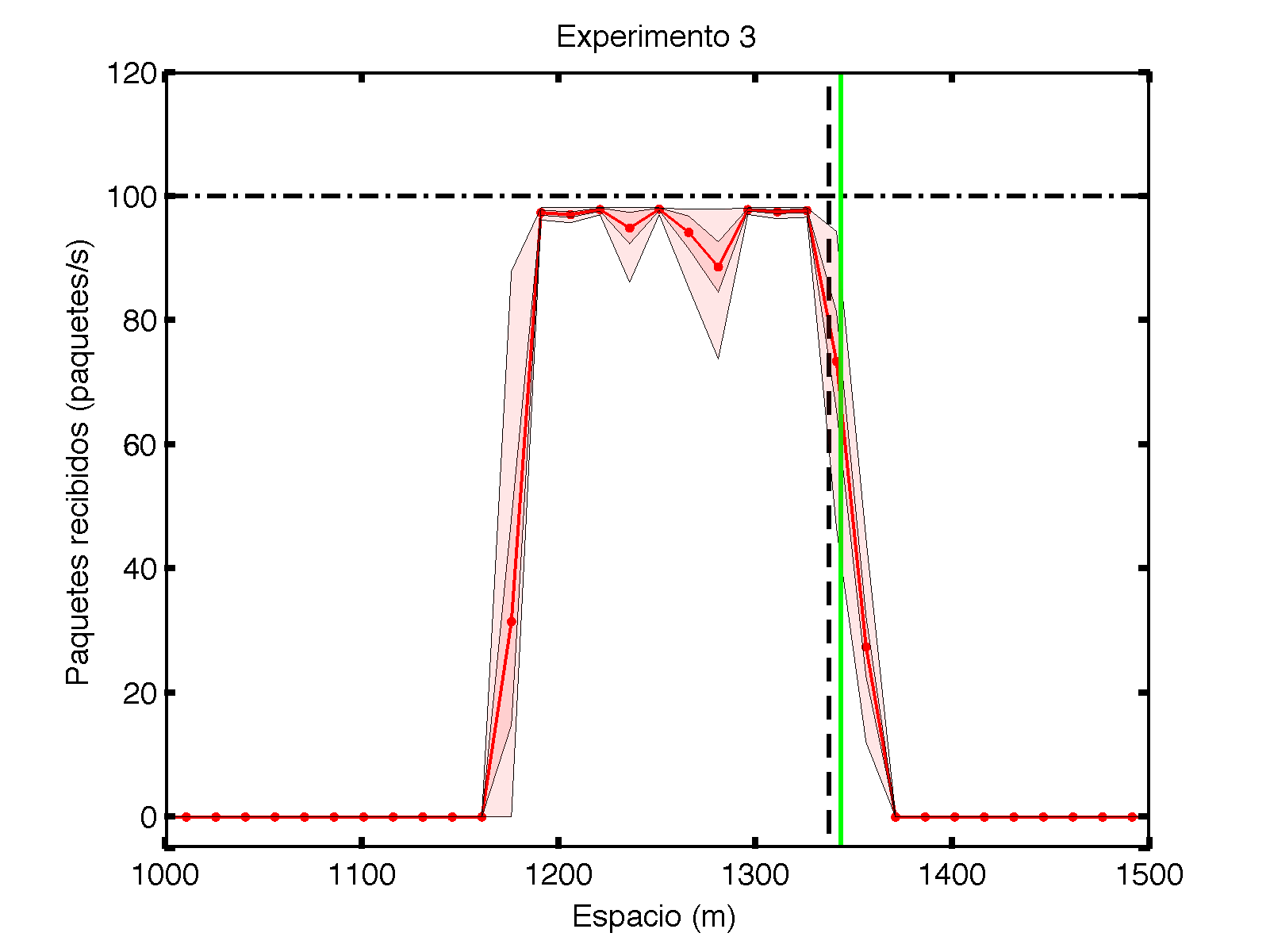


Figura . Paquetes recibidos/s del experimento 3. V=20km/h.

Sin embargo, este tercer experimento ha dado unos resultados muy satisfactorios en cuanto a la recepción. Prácticamente en todo momento se ha recibido más del 95% de lo enviado. La media se ajusta más al total de paquetes enviados, como vemos en la figura.

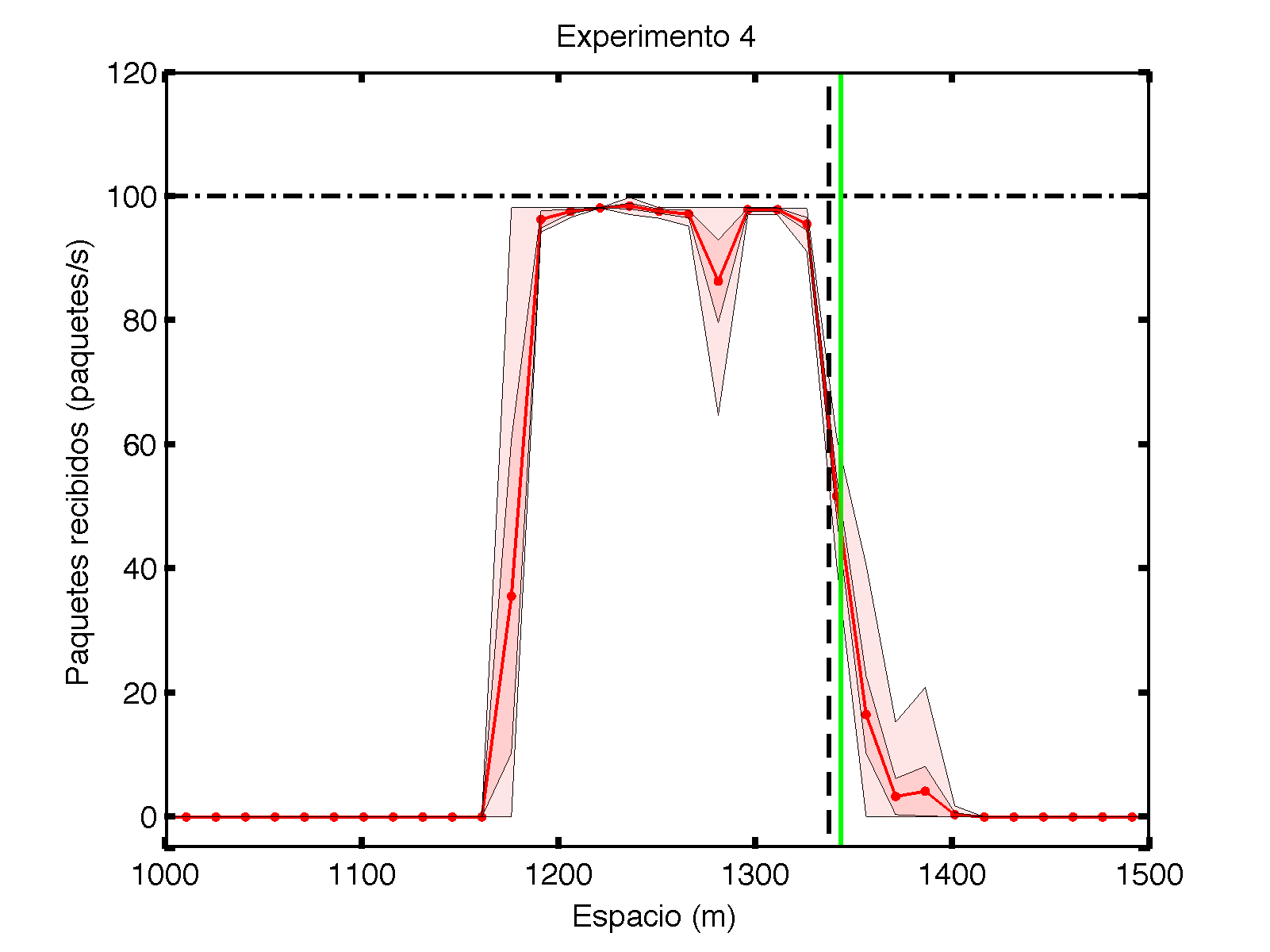


Figura . Paquetes recibidos/s del experimento 4. V=20km/h.

En este último experimento se han obtenido unos valores muy parecidos a los del experimento 3. Muy buena recepción durante casi toda la zona de cobertura.

A falta de realizar más experimentos, pensamos que las mejores prestaciones en los casos 3 y 4 se deben a que fueron realizados a una velocidad inferior.

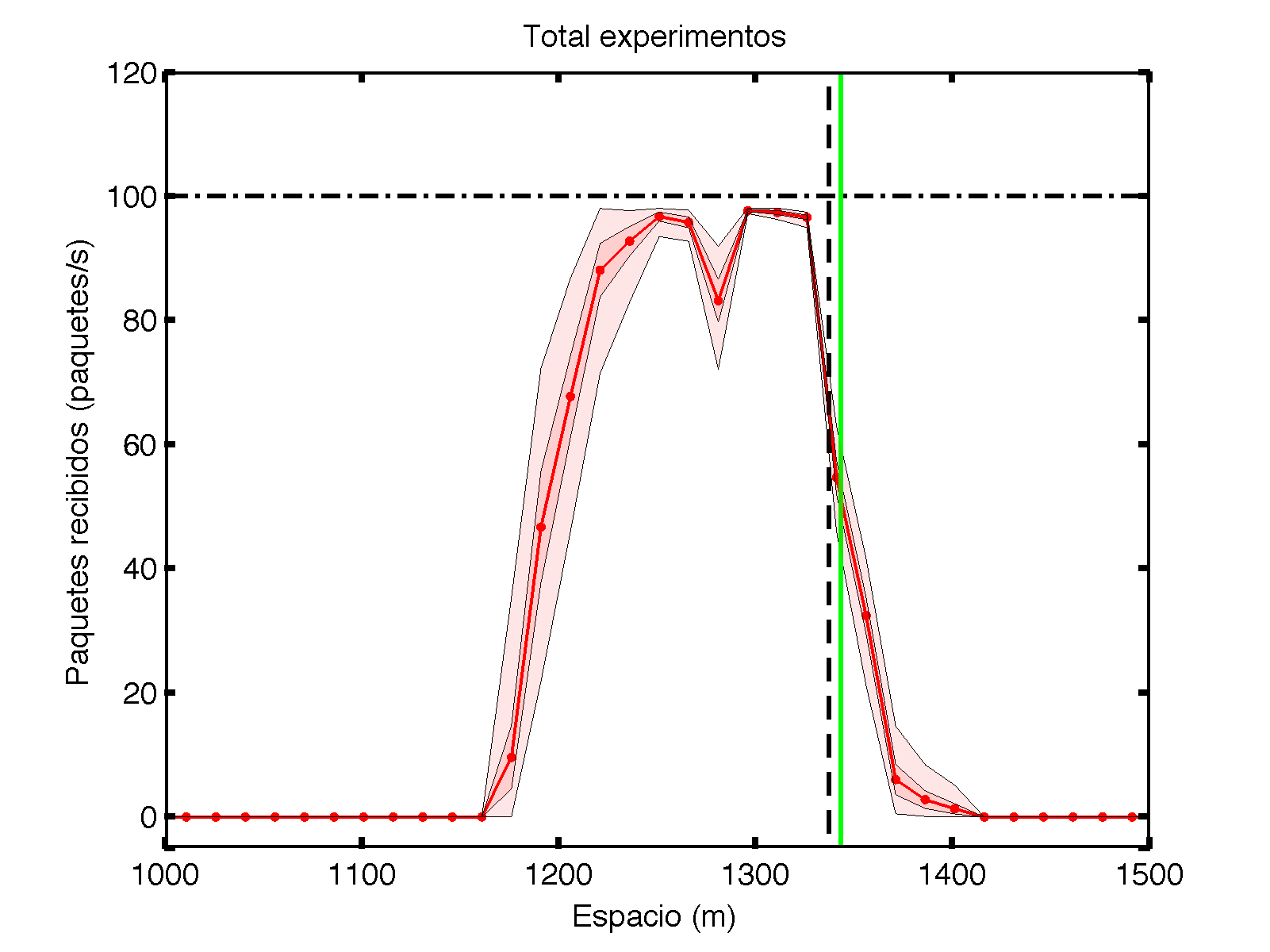


Figura . Datos de los 4 experimentos

Esta última gráfica representa los datos obtenidos en los 4 experimentos. Se utiliza el mismo método que en los casos anteriores, pero concatenando las muestras de los 4 experimentos (12 vueltas al campus en total).

Vemos como en la entrada en el área de cobertura la recepción es peor. Esto es debido a los dos primeros experimentos realizados y sus resultados obtenidos. A medida que nos acercamos al semáforo, los resultados se ven mejorados, fruto de la buena comunicación existente V2I.

Como dato a analizar, en todas las gráficas se observa un pico en el medio que representa una bajada en la recepción. No se ha podido saber la causa exacta aunque se propone una hipótesis: Al no moverse el semáforo y tener una antena muy directiva el vehículo podría pasar del lóbulo primario a uno un lóbulo secundario.

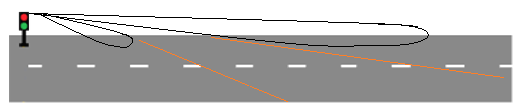


Figura . Hipótesis de pérdida de paquetes.

En la figura 40 se puede observar una representación de la hipótesis que se baraja. Al pasar el coche por la zona delimitada entre las dos líneas naranjas se produciría la pérdida de paquetes. Es lógico pensar esto puesto que estos lóbulos no se mueven al igual que la antena que los produce.

Los resultados obtenidos muestran que se ha conseguido desarrollar una aplicación de inyección de tráfico con funciones de geo-localización.

Los siguientes resultados muestran la parte del funcionamiento del modelo de sistema propuesto para la solución de los problemas del tráfico.

## Demostración de sistema

En las siguientes figuras podemos observar gráficas Espacio – Tiempo – Velocidad, en las que se aprecia cómo influye el sistema en la conducción.

El eje horizontal representa el tiempo desde el comienzo del experimento, tomando como referencia la hora del sistema (sincronizada inicialmente con Internet para que en ambos subsistemas se tenga la misma referencia de tiempo). El eje Y muestra el espacio del recorrido cercano al semáforo, que es el área de representación de interés. La línea horizontal representa el semáforo que, como se puede apreciar, permanece en el mismo sitio conforme avanza el tiempo. Los colores verde y rojo muestran el estado en cada momento.

Los puntos en diagonal es la trayectoria del vehículo. La flecha negra corresponde con la trayectoria que seguiría el vehículo manteniendo la velocidad actual y si no hubiera semáforo.

En estos casos, vemos como se modifica el comportamiento del conductor a medida que se acerca al semáforo. Esto es debido a que el semáforo se ha comunicado con el vehículo y éste ha producido sus mensajes de recomendación para que el conductor los ejecute.

En la demostración de la siguiente figura, podemos ver como se ha modificado la trayectoria inicial al recibir datos de estado en rojo. Se ha reducido la velocidad para poder cruzar el semáforo en color verde sin haberse detenido. También se aprecia cómo, si no hiciera caso y mantuviera su velocidad, se cruzaría en rojo.

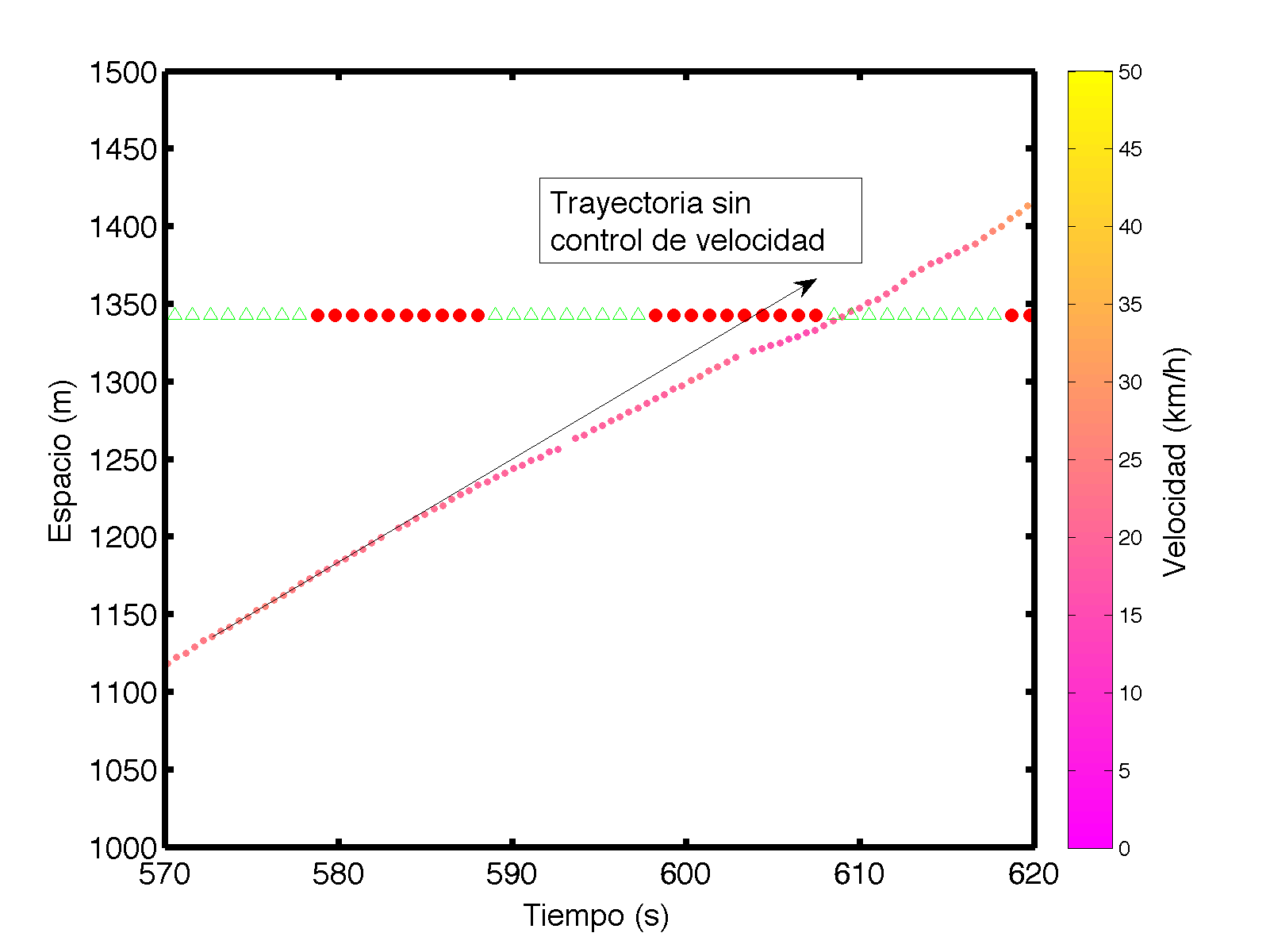


Figura . Control de velocidad, reducción.

La siguiente figura muestra otra vez la misma comunicación pero en otra vuelta diferente al circuito. Esta vez se ve representada la zona de control o zona uno. Aquí se empieza a hacer caso a los mensajes recibidos. Se ha decidido dejarlo en 50 metros porque es una distancia en la que se puede reaccionar sin realizar deceleraciones bruscas.

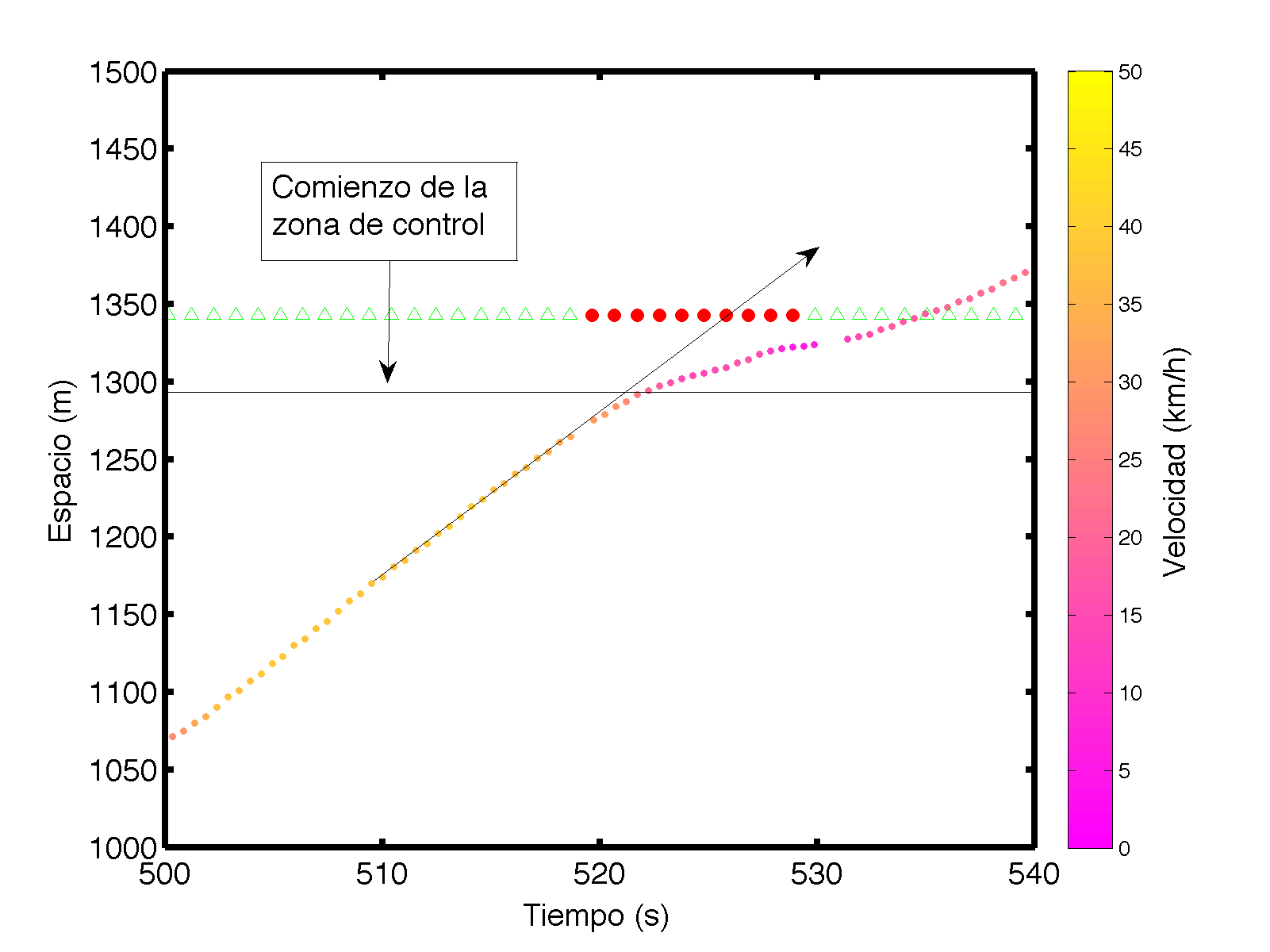


Figura . Control de velocidad, zona de control.

Los siguientes datos representados coinciden con una de las veces que el semáforo se encontraba verde y se conducía a una velocidad que permitía pasarlo sin problemas.

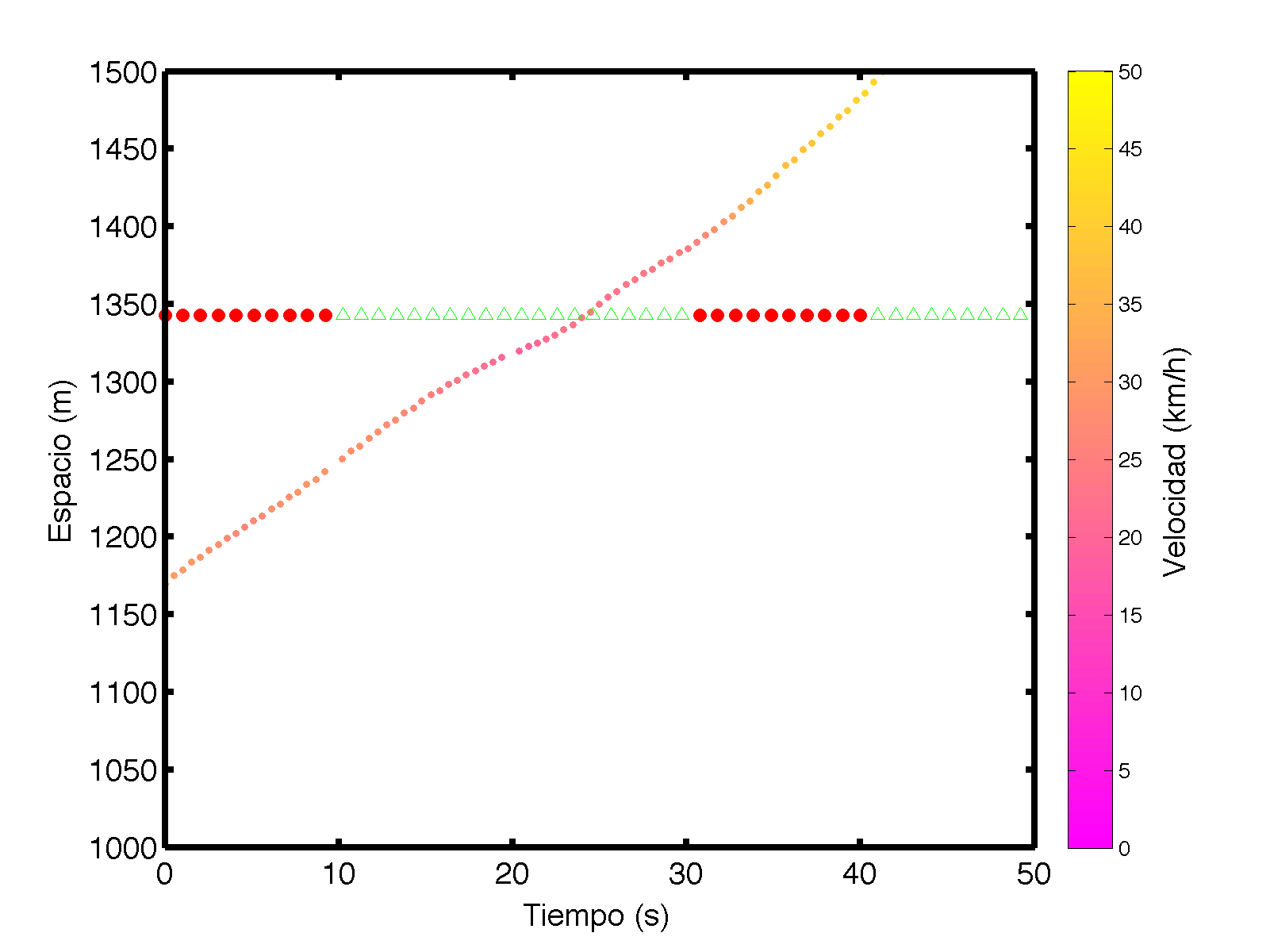


Figura . Paso en verde, mínimo de velocidad.

Esta última gráfica muestra como el vehículo se tuvo que detener ante el semáforo. Esto es debido a que el estado actual era rojo y la velocidad calculada para intentar pasar el semáforo, era muy baja como para intentar no pararse hasta que se ponga el semáforo en verde de nuevo, así que la recomendación del sistema al conductor fue la de detenerse con suavidad.

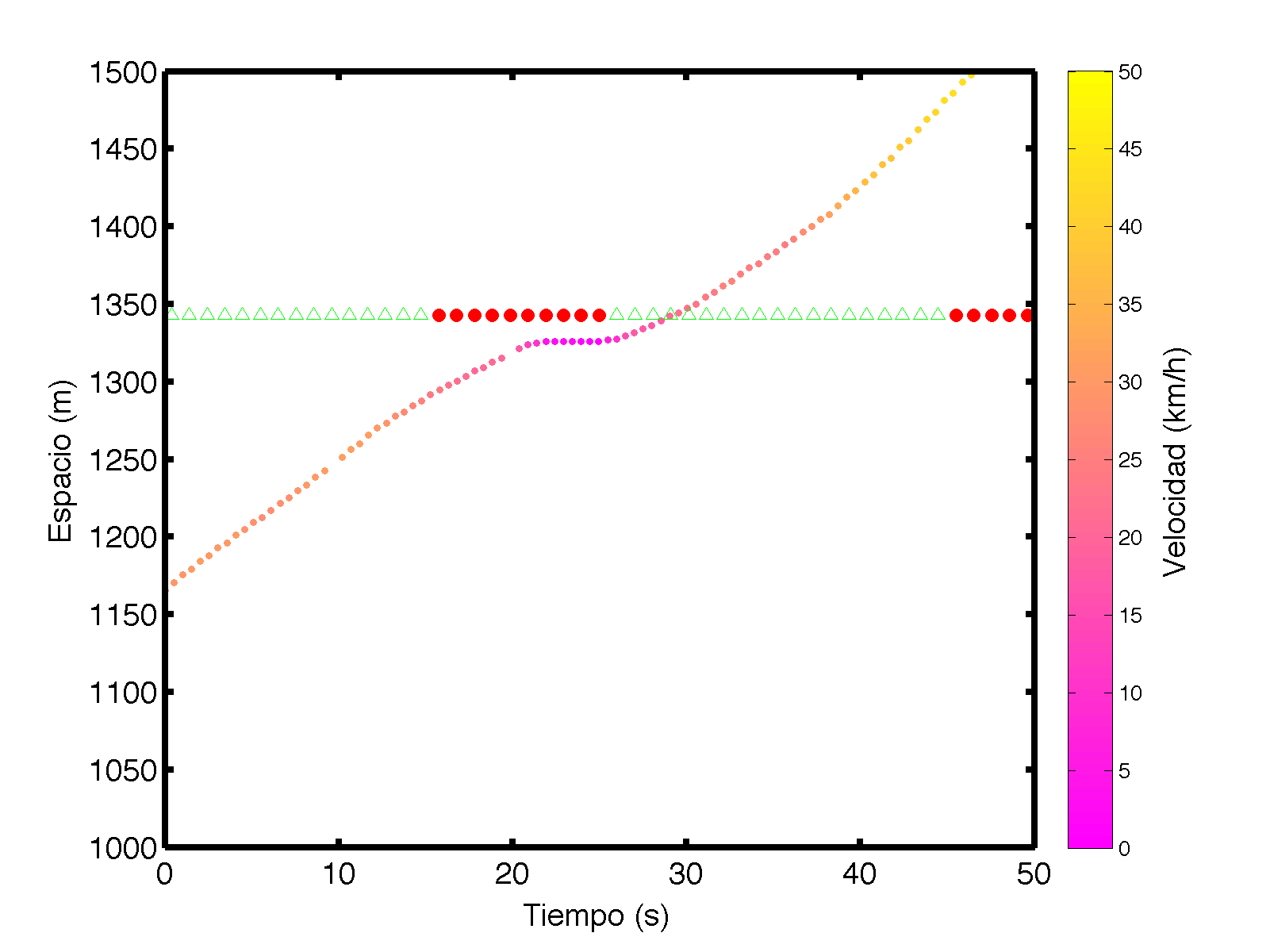
****

Figura . Parada y arranque.

La zona en la que el vehículo empieza a escuchar activamente al semáforo, se encuentra bastante dentro de la cobertura existente. Esto es así para asegurarnos que en todo momento recibimos cuando se empieza a realizar los cálculos para proporcionar una velocidad recomendable al conductor.

##### CAPÍTULO V

# Conclusiones y líneas futuras

1. resultados pricipales
2. lo que he aprendido
3. lineas futuras
4. highlights de proyecto (resultados mas importantes)
5. - MEJORAS
6. - Logstash
7. - Otros mÃ©todos de clasificaciÃ³n
8. - MÃ¡s mÃ©todos y mejores generadores de trÃ¡fico para asimilarse mejor a la predicciÃ³n por DTW y flows
9. - Usar coordenadas
10. - Mas usuarios

## Conclusiones finales

Una vez que llegamos a este punto, toca analizar y evaluar todo lo realizado.

En primer lugar, se puede confirmar que el objetivo perseguido, conseguir la comunicación V2I, se ha alcanzado y con buenos resultados.

Por otro lado, se ha podido realizar la unión inyector de tráfico con las coordenadas LLA, herramienta que programas como Iperf o DITG no tienen.

En cuanto a los objetivos secundarios, se ha podido comprobar que con la velocidad recomendable aumentamos la probabilidad de cruzar el semáforo en verde sin tener que parar. Esto nos proporciona un ahorro de tiempo y una conducción más confortable, todo lo contrario a tener que parar y arrancar de nuevo al cambio de color. El objetivo del ahorro de combustible no es fácilmente medible con la instrumentación que se poseía pero, al reducir el número de paradas se puede decir que se ha consumido menos.

Referido a los primeros resultados obtenidos se puede decir que, pese al pico en el que se recibía una menor tasa, la comunicación existente entre ambos puntos del enlace ha sido muy satisfactoria debido al número de paquetes recibidos. Por lo tanto, se pueden incluir mejoras en el resto de la aplicación, que por esta parte se asegura un buen enlace. Además, podemos estar más seguro de esto gracias a que los resultados se han obtenido mediante el preprocesado Bootstrap y los promedios estadísticos calculados se ajustan más a la realidad.

La evaluación de funcionamiento de sistema también ha sido positiva. Las gráficas de resultados nos han mostrado que se ha conseguido llevar a la práctica el modelo de conducción y de sistema que se había diseñado en un principio. La comunicación ha sido posible y gracias a ella el conductor puede modificar su comportamiento mediante órdenes sencillas de entender y ejecutar.

También la cobertura obtenida nos ha proporcionado una comunicación segura y un buen funcionamiento del sistema.

Pero no todo son ventajas. Como se ha ido explicando durante el presente trabajo, esta aplicación presenta limitaciones como la existencia de un único vehículo y un único semáforo entre otras. Por ello requiere de un siguiente paso que consistirá en optimizar el funcionamiento y conseguir que sirva para cualquier tipo de escenario y ante cualquier tipo de situación que se pueda presentar.

## Base y aprendizaje

Para el desarrollo del presente trabajo he utilizado conocimientos adquiridos durante la realización del Grado en Ingeniería en Sistemas de Telecomunicación.

Durante el desarrollo del proyecto he tenido que combinar los conocimientos adquiridos del tutor con los adquiridos de forma autodidacta.

### Conocimientos previos

El primero de los conocimientos de los que partía de base es el lenguaje de programación utilizado para el desarrollo de la aplicación; Python. Debido que su aprendizaje vino únicamente de una asignatura en toda la carrera, he tenido que investigar por mi cuenta para incrementar mis conocimientos sobre este lenguaje.

Por otro lado, durante los dos primeros años de carrera, adquirí interés sobre las clases impartidas acerca de enrutamiento y configuración de redes. De ahí la elección de un proyecto en el que se incluyera el WiFi y el uso de Routers.

No se puede dejar sin mencionar la posesión del permiso de conducción, básico para la realización de pruebas con vehículos.

Los conocimientos sobre Bootstrap adquiridos en la carrera se han complementado con los impartidos por el tutor del proyecto.

Por último cabe mencionar los conocimientos adquiridos, durante el estudio de asignaturas de comunicaciones, sobre modulaciones, tasas y enlaces inalámbricos.

### Conocimientos adquiridos

El primer tema sobre el que tuve que investigar fue el protocolo WiFi usado y sus posibles aplicaciones en la vida real. El descubrimiento de la existencia del mismo y los nombres de los equipos utilizados en aplicaciones con esta tecnología para vehículos, viene de una de las clases de último curso.

Para dichas clases se tuvo que realizar un trabajo por grupos sobre ello. Aquí comencé mi investigación en solitario ya que me tocó realizar pruebas con los Routers como trabajo individual. En estos experimentos en laboratorio, con dos ordenadores y dos Routers con sus antenas WiFi de 15 cm, apliqué los conocimientos previos sobre redes para conectarme a la interfaz del router mediante el comando “ssh” y la IP de la máquina.

Una vez dentro, probé a cambiar parámetros de la capa física con el comando wlan11p, sobre el que tuve que aprender su funcionamiento. Para la obtención de resultados acerca de estos cambios, utilicé un inyector de tráfico llamado IPERF. Es otro ejemplo de conocimientos adquiridos durante el desarrollo del proyecto.

Metidos más en la aplicación desarrollada, ha sido necesario el conocimiento básico sobre el sistema de coordenadas UTM, así como la investigación acerca del demonio de comunicación entre GPS y ordenador llamado GPSD y el protocolo de comunicación marítimo entre GPS y máquina, entendido y gestionado por este programa.

Por último he tenido que profundizar en el método “multi-thread” de programación y la instanciación de clases en el lenguaje Python, necesario para mantener las comunicaciones y el GPS en continuo funcionamiento e independiente del resto del programa.

## Líneas futuras

Para el desarrollo de esta aplicación se han asumido unas condiciones de entorno específicas, como la existencia de un vehículo, un semáforo, estado verde y rojo, circuito cerrado y cíclico… Con estas condiciones se pretendía dar un comienzo a una aplicación muy potente si se guía por buen camino.

La primera mejora que se debería introducir en el programa es el estado naranja. Lo que se busca con estas mejoras es la aproximación a la realidad y en ella los semáforos poseen tres colores; verde, rojo y naranja. Este nuevo estado supondría una nueva división de posibilidades en el controlador. Para la zona uno, de control, actuaría como un estado rojo. Se calcularían las velocidades de igual forma que en este color, pero con la discrepancia de la suma del tiempo de duración del naranja al rojo. Esto perjudicaría la velocidad recomendada, ya que estamos incrementando el tiempo que queda para volver al estado verde, y en muchos casos nos haga frenar porque no podemos estar en movimiento, a una velocidad reducida, hasta que el semáforo cambie a verde. Para zona dos en movimiento actuaría como para zona uno. Si estamos parados, actuaría igualmente que un estado rojo. En zona tres se obviarían los mensajes recibidos, como se hace actualmente con los otros colores.

La siguiente mejora a incorporar sería la de un modelo de conducción que tenga en cuenta la velocidad actual y las aceleraciones. Ahora mismo, el modelo implantado calcula velocidades con la simple pero eficaz fórmula de velocidad es igual a espacio partido de tiempo. En ciertas ocasiones, la velocidad recomendada que ha calculado el vehículo, es un valor muy dispar de la actual. Esto supone una aceleración o deceleración muy brusca para poder obtener el número deseado para cruzar el semáforo en verde. En una gráfica, en la que el eje X es el tiempo y el eje Y el espacio, pasaríamos de un modelo de comportamiento de velocidad de rectas, a un modelo de curvas. Para ello, se debe estudiar cuál es el valor óptimo de aceleración confortable, tanto para aumentar la velocidad como para disminuirla. A partir de aquí, teniendo en cuenta la velocidad actual, recomendada y como valor de referencia la aceleración confortable, se puede influir en la toma de decisiones en función de si la velocidad objetivo requiere de una aceleración confortable. Se deberían cambiar las fórmulas de cálculo de velocidad. Si se quiere “rizar el rizo” y entrando en tecnología mecánica, se podría tener en cuenta las características motoras del vehículo para ver si esos valores teóricos son admisibles por las características del vehículo.

Como se ha mencionado con anterioridad, nuestro modelo sólo admite un semáforo. Al tener el trayecto parametrizado y con las coordenadas vemos entre que dos puntos de referencia de la carretera estamos, y el sentido que llevamos, se podría aumentar el número de semáforos. Para ello habría que identificar cada carretera, sentido de recorrido del trayecto y semáforo con unos identificadores únicos. De esta forma, el vehículo detectará en que carretera se encuentra y hará caso a los mensajes recibidos por el semáforo que tiene su mismo identificador.

Como conclusión final, los resultados finales han sido satisfactorios para este primer prototipo de sistema aunque aún queda trabajo que hacer para poder llegar a ser un sistema generalizado.

#### BIBLIOGRAFÍA

**[1] “Analyzing Android Encrypted Network Traffic to Identify User Actions” Mario Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, Nino Vincenzo Verde.**

IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, VOL. 11, NO. 1, JANUARY 2016

<http://ieeexplore.ieee.org/document/7265055/>

**[2] “Encrypted Traffic Analytics” Cisco, 2017.**

<https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf>

**[3] Ubuntu (versión 16.04)**

<https://www.ubuntu.com/>

**[4] Python (versiones 2.7 y 3.5)**

https://www.python.org/

**[5] Shell Bash**

<https://es.wikipedia.org/wiki/Bash>

**[6] Docker**

<https://www.docker.com/>

**[7] Apache Kafka**

<https://kafka.apache.org/>

**[8] ElasticSearch**

<https://www.elastic.co/>

**[9] Kibana**

<https://www.elastic.co/products/kibana>

**[10] Spark**

<https://spark.apache.org/>

**[11] Spark Streaming**

<https://spark.apache.org/streaming/>

**[11] Spark MLlib**

<https://spark.apache.org/mllib/>

**[12] Spark ML**

<https://spark.apache.org/docs/latest/ml-guide.html>

**[13] Random Forest**

<https://es.wikipedia.org/wiki/Random_forest>

**[14] Anaconda**

<https://docs.continuum.io/anaconda/>

**[15] Jupyter notebook**

<http://jupyter.org/>

**[16] TCPDUMP**

<http://www.tcpdump.org/tcpdump_man.html>

**[17] Definición Big Data**

<https://es.wikipedia.org/wiki/Big_data>

**[18] Definición Ciencia de Datos (Data Science)**

<https://es.wikipedia.org/wiki/Ciencia_de_datos>

**[19] Machine Learning**

<https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico>

**[20] Streaming**

<https://es.wikipedia.org/wiki/Streaming>

**APÉNDICE A**

#### CÓDIGO DEL PROGRAMA

URL: https://github.com/IABcb/TFM.git

1. RJ-45: Interfaz física de comunicación en redes cableadas. [↑](#footnote-ref-1)
2. HDMI: High Definition Multimedia Interface. Cable de envío de imágenes y sonido entre dos dispositivos. [↑](#footnote-ref-2)
3. Jack: conector de audio de 3.5mm de diámetro. [↑](#footnote-ref-3)
4. SD: Secure Digital. Tarjeta de almacenamiento de datos. [↑](#footnote-ref-4)
5. www.raspbian.org [↑](#footnote-ref-5)
6. www.linux.org [↑](#footnote-ref-6)
7. UTC: Universal Time Coordinated: Sistema de referencia universal del tiempo. [↑](#footnote-ref-7)
8. UTM: Universal Transverse Mercator: Sistema de coordenadas basada en la proyección cartográfica de Mercator. [↑](#footnote-ref-8)
9. WiFi: Wireless Fidelity: procotolo de comunicaciónes inalámbricas. [↑](#footnote-ref-9)
10. TCP: Transmission Control Protocol [↑](#footnote-ref-10)