UNIVERSIDAD REY JUAN CARLOS

INGENIERÍA INFORMÁTICA

Curso académico 2007-2008

Proyecto Fin de Carrera

Detección visual de velocidades de vehículos en la plataforma jdec

Tutor: José María Cañas Plaza

Autor: Víctor Manuel Hidalgo Blázquez

Están los que usan siempre la misma ropa.
Los que llevan amuletos.
Los que hacen promesas.
Los que imploran mirando al cielo.
Los que creen en superticiones....

Y están los que siguen corriendo cuando
le tiemblan las piernas.
Los que siguen jugando cuando se les
acaba el aire.
Los que siguen luchando cuando todo
parece perdido, como si cada vez fuera
la última vez. Convencidos que la vida misma
es un desafio.
Sufren, pero no se quejan porque saben que
el dolor pasa, el sudor se seca, el cansancio
termina, pero hay algo que nunca desaparecerá:
La satisfacción de haberlo logrado.

En sus cuerpos hay la misma cantidad de músculos.
En sus venas corre la misma sangre.
Lo que los hace diferentes es su espiritu.
La determinación de alcanzar la cima.
Una cima a la que no se llega superando a
los demás, sino superandose a uno mismo.

Agradecimientos

No tengo ninguna duda de quienes son los principales "culpables" de que a dia de hoy este escribiendo estas palabras. Mi *familia*, los que han estado día a día a mi lado, los que se han esforzado para permitirme estudiar, los que me ha enseñado a dedicarme a lo que me gusta y fundamentalente, los que me han aguantado todos estos años. A ellos gracias.

También quería agradecer a Estrella el esfuerzo por haberme ayudado y apoyado en todo lo que hago durante estos años. Y por esos maravillosos ratos que hemos pasado juntos.

De especial manera quiero dar las gracias a José María Cañas, ya no sólo por su esfuerzo, paciencia y los conocimientos que me ha aportado, si no también por haber contribuido enormemente a mi madurez como ingeniero y especialmente por haber despertado en mi el entusiasmo por investigar.

Tampoco puedo olvidar a mis amigos por todos esos buenos ratos juntos, que también han colaborado a mi desarrollo como persona.

Finalmente me gustaría desctacar el buen ambiente dentro de grupo de *robótica* de la *URJC*, algo esencial para poder trabaja en unas buenas condiciones. Dentro del grupo dar las gracias a todos mis compañeros y en especial a Julio Manuel Vega y Antonio Luís González por todos los buenos momentos que hemos pasado en estos últimos meses.

Resumen

En los últimos años, los grandes avances en la informática han permitido el desarrollo de nuevas disciplinas científicas como la visión artificial. Las cámaras son hoy en día sensores de bajo coste cuyas imágenes proporcionan gran cantidad de información del entorno donde se utilicen. Extraer dicha información e interpretarla posibilita la generación de aplicaciones que incorporen comportamientos inteligentes enormemente útiles en tareas complejas de nuestra vida cotidiana.

El objetivo de este proyecto fin de carrera es crear una aplicación capaz de realizar *estimaciones precisas* de la *velocidad* de los coches y además *contar* el número de vehículos que circulan por carretera, usando como única fuente de información las imágenes servidas por una cámara.

Para ello la aplicación debe ser capaz de detectar y seguir a los vehículos por la carretera. El objetivo de la detección es avisar al sistema de que un nuevo objeto ha aparecido en la imagen y debe ser seguido. Con el seguimiento nos referimos a la habilidad de estimar de manera contínua la posición de un vehículo. Dicha posición cambiará rápidamente por lo que debe actualizarse la estimación de manera contínua. Para poder realizar dicha funcionalidad se ha estudiado, diseñado y desarrollado un algoritmo evolutivo multimodal. El algoritmo que detecta por movimiento a los coches hipotetiza todas las posibles velocidades del vehículo (etapa de detección), y descarta a los que no son compatibles con el flujo de imágenes (etapa de seguimiento). Además para poder realizar el seguimiento de un objeto es necesario conocer su posición, para resolver esto se ha integrado en este sistema la rectificación de imágenes.

La aplicación ha sido desarrollada apoyándose en la plataforma software *jdec* e implementada en forma de hebras iterativas o esquemas. Este sistema es lo suficientemente vivaz como para seguir varios vehículos a gran velocidad en una carretera, y está dotado de un gran precisión a la hora de estimar la velocidad real de los vehículos. Se han realizado experimentos en el simulador *gazebo* y con imágenes reales de diferentes carreteras.

´

Índice general

Índice de figuras

Capítulo 1
Introducción

La vista es el sentido que más utilizamos para captar información de nuestro entorno. Mediante los ojos somos capaces de modelar nuestro entorno, conocer detalles de los objetos de alrededor. Gracias a estos detalles podemos reconocer dichos objetos por su color, por su forma, detectar movimiento en ellos o incluso estimar aproximadamente la distancia que nos separa.

La visión computacional es la rama de la inteligencia artificial que intenta emular la capacidad que tienen algunos seres vivos para extraer información útil desde las imágenes. Por este motivo la visión computacional sirve de herramienta para multitud de disciplinas como la robótica o la medicina.

Este proyecto fin de carrera consiste en una aplicación visual, que cuenta y estima la velocidad de vehículos en la carretera. Para ello, los vehículos son detectados y posteriormente seguidos en un pequeño tramo.

1.1. Visión artificial

Nuestros ojos están mucho más desarrollados que cualquier cámara de hoy en día,y más alla, el procesamiento de la señal que hace nuestro cerebro es mucho más complicado que cualquier programa de procesamiento de imágenes actual. De hecho, todavía hay ciertas habilidades que no se comprenden completamente y no somos capaces de imitarlas con las máquinas. Más alla de imitar al ser humano, en vez de entender toda la información contenida en una imagen, la visión por computador se centra en sacar cierta información de interés para alguna tarea especifica.

Se puede definir la visión artificial o visión computacional como el área de la inteligencia artificial que se dedica a extraer información de las imágenes con el fin de

comprender y asimilar lo que en ellas está sucediendo. La visión artificial tiene como objetivo por tanto interpretar las imágenes analizadas para obtener información relevante sobre elementos que en ellas aparecen.

La visión artificial surge en la década de los 60 con la idea básica de conectar una cámara de vídeo a una computadora. Esto implica no sólo la captura de imágenes a través de la cámara, sino también la comprensión de lo que estas imágenes representan. Un resultado muy importante de este trabajo, y que marcó el inicio de la visión artificial, fue un trabajo realizado por Larry Roberts, el creador de ARPAnet. En 1961 creó un programa, el mundo de micro-bloques, en el que un robot podía ver una estructura de bloques sobre una mesa, analizar su contenido y reproducirla desde otra perspectiva, demostrando así que esa información visual que había sido mandada al ordenador por una cámara, había sido procesada adecuadamente por él.

Sin embargo, hasta comienzos de la década de 1990 la visión computacional no dio un salto cualitativo importante. Este hecho se debe en gran medida a la aparición de ordenadores con velocidad de cómputo suficiente para procesar imágenes de forma ágil y con poca demora. Este hecho se le unió el abaratamiento de precios de las cámaras, siendo este tipo de sensores los mejores en la relación información/precio, actualmente. A partir de entonces la visión computacional comenzó a emplearse para múltiples tareas y su desarrollo fue concentrándose en problemas de tratamiento de las imágenes, ya que la principal dificultad de las cámaras no radica en tener los datos disponibles, sino en procesarlos. Para dicho procesamiento se siguen diversas técnicas como filtros, segmentación, detección de movimiento y visión 3d, entre otros. En los libros [Forsyth y Ponce, 2003] y [Hartley y Zisserman, 2004a], podemos obtener mucha información sobre técnicas de visión computacional.

Este tremendo desarrollo tecnológico ha sido decisivo para la consolidación de áreas de aplicación de la visión computacional muy lucrativas. Algunos de los productos más conocidos a día de hoy permiten alcanzar propósitos tales como:

- Interacción con humanos: como el *GoMonkey* [1] (figura videojuegos como el *eyetoy* [Santos, 2007], domótica.
- Organizar información: indexado en bases de datos de imágenes o vídeos.
- Modelar objetos o escenas: análisis de imágenes médicas, modelado topográfico

[1] http://www.gomonkey.at/

- Controlar procesos: control de calidad, robots industriales
- Reconocer objetos: seguridad, digitalización de textos (OCR).
- Seguimiento de objetos: contadores de objetos, reconocer características de objetos, vídeo vigilancia.

Actualmente la visión computacional es muy utilizada en el campo industrial para el control de calidad, ya que en general esta tarea es repetitiva y aburrida para los seres humanos y conlleva un elevado riesgo de error. Existen sistemas de control de calidad que se basan en capturar la imagen del producto y verificar si cumple ciertas características. En el ejemplo de la figura 1.1 podemos ver un sistema capaz de descartar piezas que no cumplen ciertas especificaciones de calidad, como el tamaño y la forma.

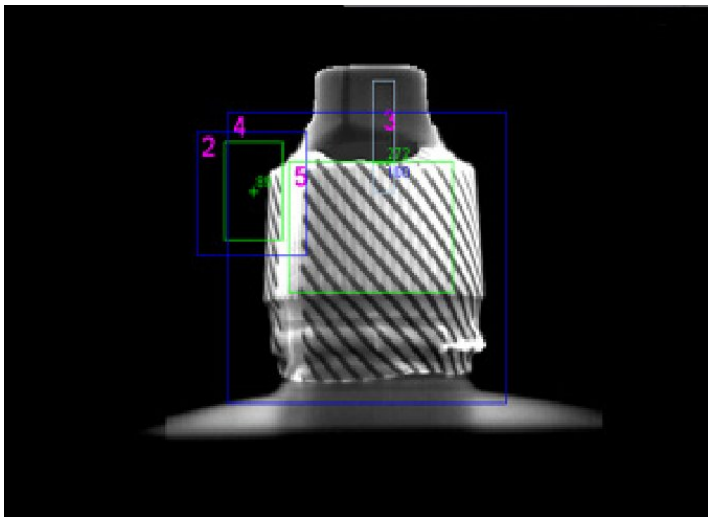
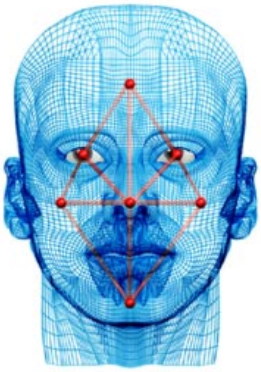


Figura 1.1: Sistema visual de control de calidad

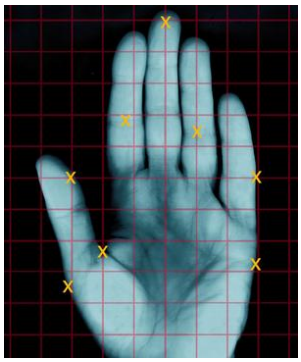
En materia de seguridad, el auge de la visión artificial está permitiendo un importante desarrollo de mejoras y nuevas posibilidades. Empresas y gobiernos invierten grandes cantidades de dinero en investigaciones en este campo, donde muchos horizontes aún están por descubrir. Por ejemplo, en el reconocimiento de personas mediante el estudio biométrico (figura 1.2), permite asi restringir el paso de personas a determinadas áreas.

En materia de seguridad, el auge de la visión artificial está permitiendo un importante desarrollo de mejoras y nuevas posibilidades. Empresas y gobiernos invierten grandes cantidades de dinero en investigaciones en este campo, donde muchos horizontes aún están por descubrir. Por ejemplo en el reconocimiento de personas mediante

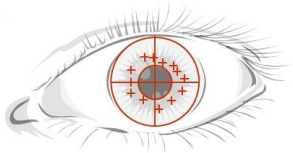
el estudio biométrico (figura 1.2), permitiendo asi restringir el paso de personas a determinadas áreas.



(a) Reconocimiento facial



(b) Reconocimiento dactilar



(c) Reconocimiento del iris

Figura 1.2: Reconocimiento de personas a través del estudio biométrico

Otro ejemplo de aplicación con visión artificial es el sistema EyeVision [2] (figura 1.3), que permite modelar en tiempo real una escena de manera foto realista con el fin de obtener nuevas imágenes virtualizadas de la misma. Este sistema ha sido empleado en la SuperBowl americana para la repetición de jugadas desde cualquier ángulo. A través de la colocación estratégica de un determinado número de cámaras alrededor de un estadio es posible obtener la realidad virtualizada de lo sucedido segundos antes. La escena tridimensional se obtiene de la composición de las imágenes 2D que consigue cada una de las cámaras.

Figura 1.3: Sistema de repetición tridimensional Eye-Vision

Entre las capacidades visuales que posee el ser humano se encuentra la de poder seguir objetos que tenemos delante. Aunque lo hagamos de forma inconsciente, mediante esta capacidad podemos realizar múltiples actividades cotidianas, como leer, caminar, conducir, etc. El seguimiento visual está estrechamente ligado a muchas de las tareas

[2] http://www.ri.cmu.edu/events/sb35/tksuperbowl.html

que realizamos. La introducción de estas capacidades en los sistemas visuales artificiales es una de las áreas de investigación en la visión por computador y la robótica de hoy en día.

1.2. Detección y seguimiento visual de objetos

En visión artificial, un algoritmo de seguimiento trata realizar una localización continua en el tiempo de uno o varios objetos que se encuentran en la imagen. El problema del seguimiento basado en la visión puede verse como la determinación, en una secuencia de imágenes, de la posición de un determinada región. Se trata de hacer corresponder dicha región en una imagen con la misma región en la siguiente imagen. Así pues, el objetivo principal del seguimiento de objetos es recuperar el estado (posición y orientación) de un objeto mientras se mueve en una secuencia de imágenes. Los componentes del estado del objeto calculados pueden incluir parámetros como la posición, velocidad u orientación .

Idealmente, un buen algoritmo de seguimiento puede devolver información completa y correcta sobre el estado del objeto para cada secuencia de imágenes, pero las condiciones ideales raramente se encuentran en el mundo real. Incluso el bien desarrollado sistema visual humano, no puede seguir objetos con precisión en condiciones adversas, por ejemplo cuando los objetos se mueven demasiado deprisa, cuando hay demasiadas distracciones en el fondo o se producen oclusiones. Las diferentes condiciones que se pueden dar en el mundo real hacen que el seguimiento visual sea un problema complejo.

Los seguidores pueden diferir mucho entre ellos según trabajen con uno o varios objetos, usen información 2D ó 3D, usen calibración de cámaras ([Kachach, 2008],[de Agapito, 1999]), geometría proyectiva, o se modelen por puntos, regiones, mallas deformables, etc. Existen algoritmos muy conocidos de seguimiento visual, un ejemplo de este tipo de algoritmos es el filtro de kalman. Los filtros de kalman tratan de estimar la posición de un punto o característica en movimiento en la siguiente imagen. Se trata de buscar la característica (punto, borde, esquina, región, etc.) en un área determinada de la siguiente imagen alrededor de la posición predicha, en la que estamos seguros de encontrar la característica dentro de un cierto grado de confianza. Por otro lado, existen otro tipo de algoritmos que facilitan el seguimiento de objetos. Un ejemplo de este tipo de algoritmos es el de flujo óptico. El flujo óptico de un píxel perteneciente a una imagen de una secuencia, es el vector velocidad de ese píxel en la secuencia, es

decir, hacia donde se mueve el píxel con el transcurso del tiempo. De esta forma, el problema de correspondencia entre dos imágenes de la secuencia consiste en identificar un píxel de la imagen en el instante t en la imagen en el instante t+1.

Hoy en día existen múltiples basadas para el seguimiento visual mediante técnicas de visión artificial. Muchas de ellas están inspiradas en lo que nosotros, como seres humanos, hacemos con nuestro sistema de seguimiento biológico. Sin embargo, a veces resulta difícil darse cuenta de que tareas aparentemente sin nada que ver con el seguimiento, son o están compuestas en alguna medida por seguimiento visual. El acto de leer es una de ellas, cuando nos fijamos en cada una de las palabras del texto, estamos a la vez siguiendo la línea en donde está incluida, saltando sucesivamente a las palabras situadas a la derecha de la actual.

- **Navegación visual en robots**: De la misma manera que las personas utilizan marcas visuales para caminar o evitar objetos, un robot puede hacer uso de ellas para obtener su posición y orientación relativa a ellas durante su trayecto. Con un sistema de seguimiento es posible no sólo usar una simple marca para realizar la tarea, sino varias marcas para obtener una mejora en la fiabilidad y hacer más sencilla la obtención de la posición y orientación

- **Realimentación visual** para brazos robotizados: Controlar brazos robotizados por medio de seguimiento visual es una técnica útil para interactuar con entornos reales.

- **Seguimiento de caras**: El reconocimiento de caras es un problema complicado. Existen numerosos trabajos sobre este tema, que tratan de identificar personas en entornos controlados y con vistas frontales de las caras. El problema se hace más difícil cuando la persona a reconocer no se encuentra inmóvil, sino que se mueve de forma natural delante de las cámaras. Un ejemplo de este tipo de aplicaciones es el proyecto *Detección y seguimiento de caras en la plataforma jdec* ([Martín, 2008]), que realiza detección y seguimiento de caras (figura 1.4).

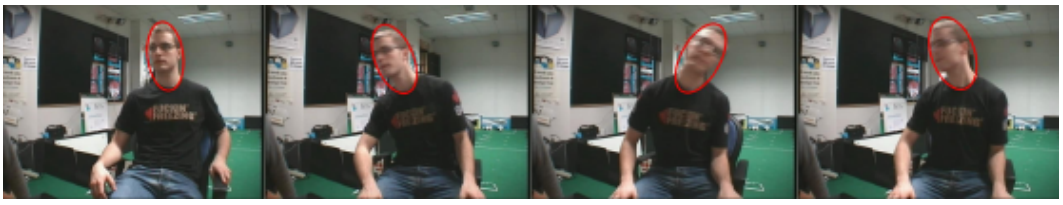


Figura 1.4: Seguimiento de caras

- **Control del tráfico**: La monitorización de vehículos en carreteras y autopistas es otra de las aplicaciones prácticas del seguimiento.

- **Vigilancia**: Ésta ha sido de las primeras aplicaciones útiles para algoritmos de seguimiento. Normalmente, los sistemas de vigilancia típicos constan de cámaras estáticas con módulos de detección que activan la grabación de secuencias. En la siguiente figura podemos ver un ejemplo de seguridad con cámaras.

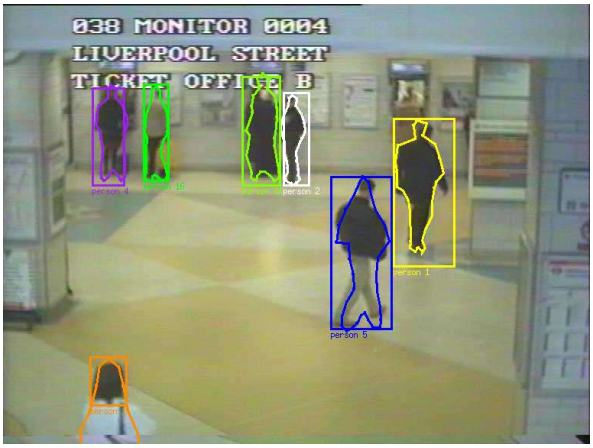


Figura 1.5: Sistema de vídeo vigilancia

- **Seguimiento de partes del cuerpo de personas**: En este tipo de aplicaciones no se sigue a la persona en sí, sino a una parte de su cuerpo. Esto tiene el atractivo de que los gestos del cuerpo y las manos pueden ser reconocidos. Por ejemplo, manos y dedos pueden ser usados como ratones o teclados para interactuar con un ordenador. En la siguiente figura 1.6 [Erik B. Sudderth, 2005] podemos ver un ejemplo del seguimiento de una mano.

Figura 1.6: Seguimiento del movimiento de una mano

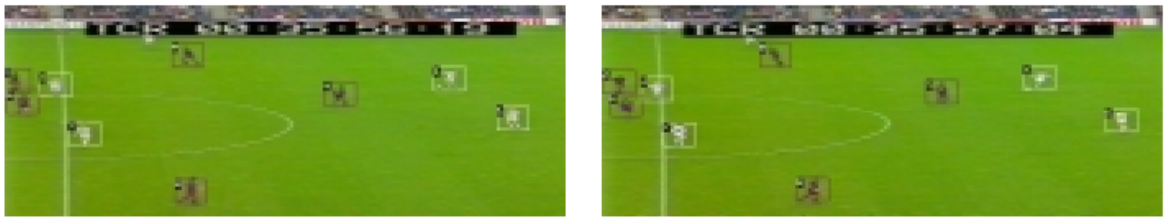
- **Deportes**: El seguimiento de objetos en eventos deportivos cuenta con numerosas aplicaciones. Pueden servir para la toma de decisiones arbitrales, como es el

ejemplo del tenis donde los tenistas pueden pedir al juez la repetición de alguna jugada polémica, gracias al sistema llamado ojo de halcón.(ver figura 1.7).

Figura 1.7: Seguimiento de la pelota de tenis durante un partido

También existen aplicaciones para el seguimiento de jugadores, en otros deportes como hockey y fútbol americano, para poder estudiar mejor las tácticas o el comportamiento de determinados jugadores. Otro ejemplo de un sistema visual aplicado a eventos deportivos es el de repetición virtual de jugadas de fútbol. Para ello se debe primero detectar el movimiento de cada jugador, calcular su posición 2D en el terreno de juego, y realizar el seguimiento visual (figura 1.8(a)). Así pues por cada fotograma se puede obtener un fotograma virtual (figura 1.8(b)).

En esta misma línea,muchas aplicaciones visuales siguen las mismas pautas que la repetición virtual de jugadas de fútbol, dividiendose la aplicación normalmente en tres fases. La primera fase consiste en la detección de movimiento en la imagen. En la segunda fase se pretende obtener la posición del objeto 2D ó 3D en el mundo. Y por último, una vez que se ha detectado el objeto y se sabe su posición, se procede al seguimiento del mismo.



(a) Detección,localización y seguimiento de jugadores de fútbol

(b) Reconstrucción virtual

Figura 1.8: Repetición virtual de jugadas de fútbol

1.3. Seguimiento visual de coches con algoritmo evolutivo

En el grupo de robótica de la URJC se han desarrollado diversos proyectos en el campo de la visión computacional, teniendo muchos de ellos en común un algoritmo para el seguimiento visual, denominado *algoritmo evolutivo multimodal*. En esta misma línea se encuentra este PFC, donde se ha realizado una adaptación de ese algoritmo para la detección de velocidades de coches. En este apartado comentaremos dichos proyectos, ya que en gran medida cada uno de ellos ha ido apartando un paso más, siendo el contexto inmediato para este proyecto fin de carrera, que parte de una base ya hecha.

1.3.1. Precedentes

El proyecto fin de carrera *Aplicación de seguridad basada en visión* ([Cabello, 2007]), consistió en una aplicación de seguridad para localizar dentro de una habitación a un

sujeto a partir del movimiento que genera en la imagen y el color predominante en dicha persona. En esta aplicación si el sujeto se adentra en una determinada zona restringida, el sistema avisa con una alarma sonora y visual. Para el desarrollo de esa aplicación se emplearon dos técnicas distintas: un algoritmo evolutivo monomodal y una técnica probabilística denominada filtro de partículas. La localización y el seguimiento 3D en dicho proyecto fueron aplicados a la vigilancia, a protección de objetos y al cuidado de ancianos. En este ultimo caso se utilizo la estimación en la coordenada Z para avisar de la caída de una persona mayor.

A raíz de esta aplicación de seguridad se llevó a cabo el proyecto fin de carrera *Seguimiento 3D visual de múltiples personas utilizando un algoritmo evolutivo multimodal* [Marugán, 2007] teniendo como objetivo principal lograr el seguimiento de múltiples objetos. Este proyecto fue un punto de partida muy importante para poder seguir varios coches en la imagen. En la figura 1.10 podemos ver la trayectoria seguida por cada individuo.

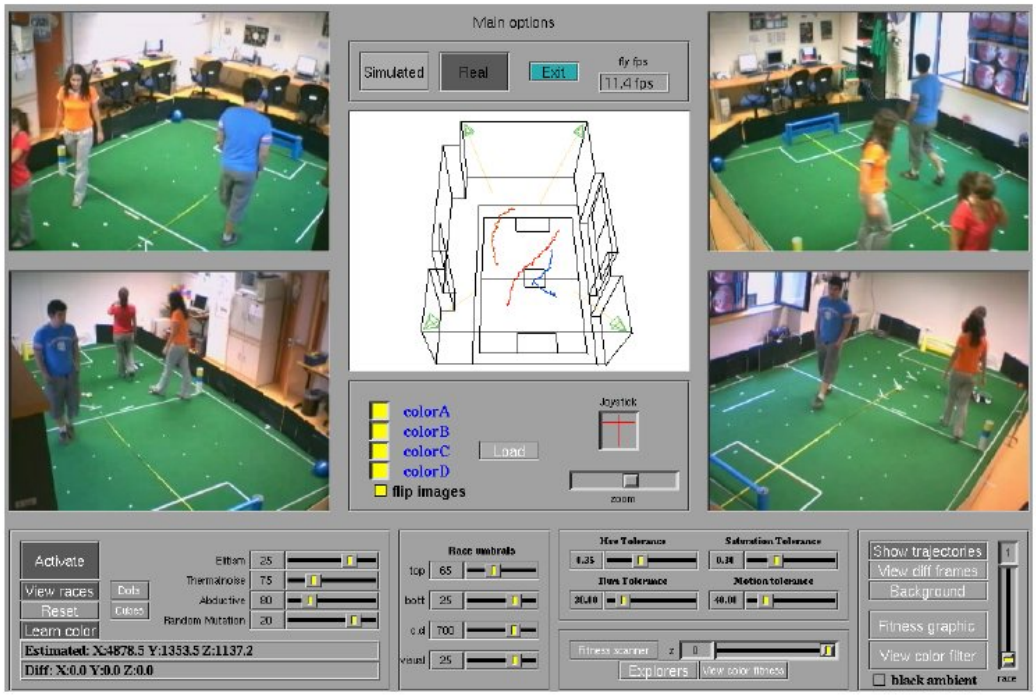


Figura 1.9: Seguimiento visual de múltiples personas

Otro antecedente dentro de este proyecto fin de carrera, es el de *Cálculo y aplicaciones de flujo óptico en tiempo real* [Santos, 2007]. Con él se desarrolló una aplicación capaz de seguir y contar coches. Para dicha aplicación se usó flujo óptico para estimar el movimiento a través de los cambios de intensidad de una secuencia de imágenes. En la siguiente figura podemos ver dicha aplicación.

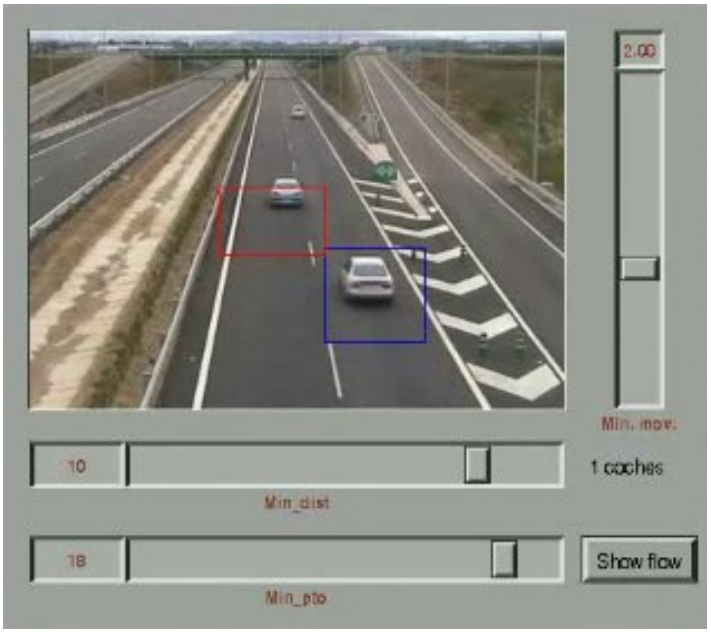


Figura 1.10: Seguimiento de coches con flujo óptico

Otros trabajos fines de carrera recientes llevados a cabo dentro del grupo de robótica con algoritmos evolutivos multimodales son los siguientes proyectos fin de carrera.

Localización y construcción de mapas en un robot de interiores [Ángel Cortés Maya, 2007], la parte de localización de este proyecto consistía saber en todo momento donde se encontraba el robot. Para ello utilizaba las medidas del sensor láser obtenidas y un algoritmo evolutivo multimodal que representaba varias posiciones similares en el mundo a las medidas reales.

Reconstrucción 3D visual con algoritmos evolutivos ([Martínez, 2007]), este proyecto consiste en una aplicación capaz de reconstruir objetos de nuestro entorno estimando la posición tridimensional de sus bordes a partir de un par estéreo de cámaras.

En línea del seguimiento multiobjeto cabe citar la tesis doctoral Aplicación de métodos secuenciales de Monte Carlo al seguimiento visual 3D de múltiples objetos [Gonzalez, 2005], que utiliza un filltro de partículas similar al utilizado en [Cabello, 2007], para seguir pelotas de un mismo color.

1.3.2. Estado del arte

En esta sección vamos a hacer una pequeña descripción más detallada sobre aplicaciones similares que abordan la medición de velocidades de coches.

Los radares o detectores de velocidad son unos sistemas electrónicos que permiten detectar objetos y determinar la distancia a que se encuentra o la velocidad a la que se desplazan. Actualmente los radares más usados son los denominados cinemómetros, pero también hay otro tipo de radares y siendo algunos de estos sistemas, como veremos aquí, los que mayor semejanza guardan con respecto la aplicación desarrollada en este proyecto fin de carrera.

Dentro de los cinemómetros distinguimos dos grandes tipos:

- **Cinemómetros de radar**, este tipo de radares dispone de una antena que proyecta sobre los vehículos ondas de radio. Estas ondas de radio son reflejadas por el vehículo y como consecuencia de este rebote la señal es de nuevo captada por la antena. La señal rebotada en el vehículo no es igual a la señal emitida, sino que se ha producido una distorsión en la señal, que permite al radar determinar a qué velocidad circulamos.
  En función de las diferencias en la frecuencia entre la señal emitida y de la señal que recibe rebotada por el vehículo el radar determina la velocidad a la que circula el vehículo y si ésta es superior a la velocidad permitida en dicho tramo, disparara una cámara fotográfica, en la que sobreimprimen sobre el vehículo la velocidad a la que circulaba, el nombre de la vía, la fecha, la hora, etc. Un ejemplo de este tipo de radares lo podemos ver en la figura 1.11.

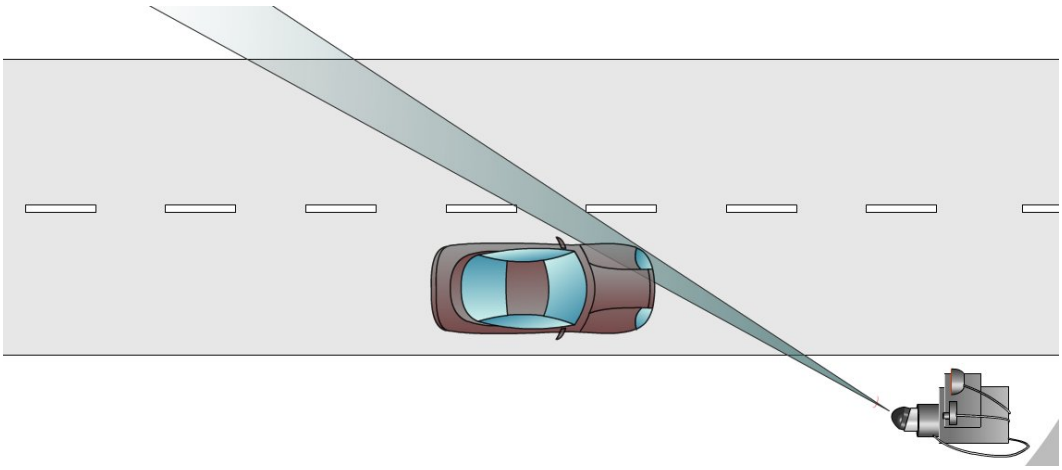


Figura 1.11: Cinemómetro de radar

- **Cinemómetros láser**, la principal diferencia de este tipo de radares con respecto a los anteriores es que éstos no proyectan ondas de radio, si no que que proyectan un haz láser.

Aparte de los cinemómetros, hay otro tipos de sistemas. Los radares por tramos miden la velocidad media en un tramo de carretera. Los radares por banda utilizan bandas de inducción debajo del asfalto, empleándolo para obtener la velocidad de los coches que pasan por encima. Los radares en helicóptero, a través de un haz láser pueden detectar la velocidad de los vehículos.

También existen otro tipo de prototipos basados en visión computacional, y donde sólo usan cámaras como sensores. Este tipo de radares no suelen funcionar en condiciones adversas de iluminación, como pueda ser por la noche, aunque hay sistemas actuales que resuelven este problema. Dichos radares han sido la referencia para crear un prototipo de radar visual en este proyecto fin de carrera.

En esta línea hay que destacar la aplicación *SpeedTrap* [3], que detecta la velocidad de vehículos y en la que también se puede analizar el tráfico. De la misma forma que en este proyecto, posee un plano imagen (imagen original capturada por la cámara) y otro plano donde se conocen las distancias de la carretera. Gracias a estas distancias si la posición de un vehículo es conocida en un fotograma y en el siguiente, se puede calcular su velocidad. Esta aplicación divide la carretera por carriles, y cada carril posee una línea en el centro. En la figura 1.12 podemos ver estas líneas dibujadas de verde. En cada píxel de dicha línea podemos tener dos valores 0 si no se ha detectado movimiento ó 1 si se ha detectado movimiento. Así pues, los píxeles de esa línea donde su valor sea 1, representan la existencia de un coche. Esta aplicación presenta un problema, que es el del seguimiento visual. El seguimiento de cada coche se realiza por la proximidad entre fotogramas. Por ejemplo si un coche A se encuentra en la posición 500, y en el siguiente fotograma se encuentra en la posición 600 por proximidad serían el mismo coche. Pero si en el fotograma donde el coche A se encuentra en la posición 600, un coche B que se encuentre en un carril adyadcente se cambia al mismo carril y justamente detrás del coche A, se producirá un error en el seguimiento visual.

[3] http://www.cs.princeton.edu/ cdecoro/traffic/

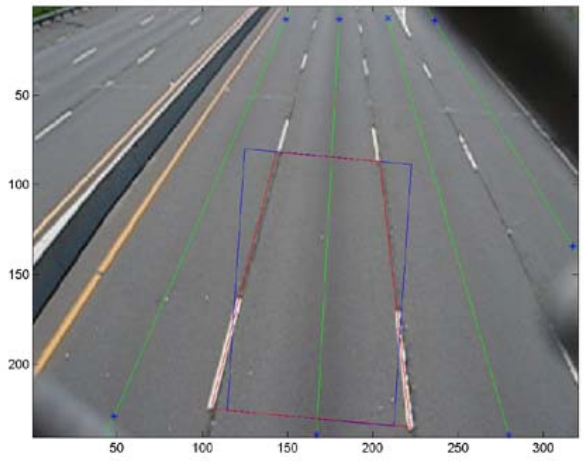


Figura 1.12: Líneas centrales en cada carril de la aplicación Speed Trap

Otras aplicaciones en esta línea son *detección y seguimiento de vehiculos para monitorización del tráfico* [Luca y Snidaro, 1998], donde usan filtros de Kalman y *VISATRAM* [Zhigang Zhu y Yang, 1998], aplicación que usa técnicas similares a las de *SpeedTrap*, donde además de detectar la velocidad, añaden otro tipo de información del vehículo como tamaño, marca etc.

Mas alla de la esta introducción la memoria de este proyecto fin de carrera está organizada en primer lugar con un capítulo dedicado a los objetivos y requisitos que ha de cumplir la aplicación . Posteriormente, en el capítulo 3 se describirá el entorno de desarrollo utilizado. Seguidamente encontraremos un capítulo dedicado a la descripción informática de este comportamiento, donde se explica la solución desarrollada para este proyecto. Una vez comentada la solución analizaremos los experimentos realizados para validarla. Finalmente se describirán las conclusiones y las líneas futuras de posibles trabajos sobre el mismo proyecto.

Capítulo 2
Objetivos

Tras haber presentado el contexto general y particular sobre el que se asienta la aplicación, en este capítulo se fijan los objetivos y requisitos concretos de este proyecto fin de carrera. Para ello se definirá de modo preciso el problema que se quiere resolver con este proyecto fijando el objetivo principal, los subobjetivos en los que lo hemos articulado, la funcionalidad requerida y las restricciones que han condicionado su desarrollo.

2.1. Descripción del problema

El objetivo de este proyecto consiste en diseñar, desarrollar y probar una aplicación software que cuenta coches y detectar la velocidad de vehículos con una sola cámara. Para ello se contará como único sensor con una cámara.

Dicho objetivo se ha articulado en tres subobjetivos más específicos que constituyen el camino hacia la solución final:

1. **Detección de vehículos**. Cuando aparezcan nuevos coches en la imagen, estos deben de ser detectados. Para poder detectar movimiento se han estudiado e implementado diversas técnicas propias de visión con la finalidad de obtener mejores resultados.

2. **Localización de vehículos en la carretera**. El sistema deberá conocer en todo momento la posición relativa en la calzada donde se encuentra el vehículo. De esta forma dado un píxel, el sistema tiene que conocer su posición en la carretera y viceversa.

3. **Seguimiento de vehículos**, para estimar su velocidad y poder llevar un contador de los vehículos que circulan por la calzada. La aplicación tiene que llevara

a cabo un seguimiento correcto, de los vehículos detectados. Para ello, se ha estudiado, diseñado y desarrollado un *algoritmo evolutivo*. Con este subobjetivo pretendemos construir un prototipo diferente, a las aplicaciones *SpeedTrap y VISATRAM* descritas en la sección 1.3.2.

2.2. Requisitos

En la siguiente sección se presentan los requisitos que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son esenciales, es decir, no sería aceptable un sistema que no satisfaga alguno de los requisitos aquí presentados.

2.2.1. Requisitos funcionales

La funcionalidad que se espera del sistema es la siguiente:

- **Requisito(01) *"Capturar imágenes"*:** El sistema deberá ser capaz de capturar imágenes mediante una cámara digital para poder disponer de información del entorno en el que se encuentra.
- **Requisito(02) *"Localización de vehículos"*:** El sistema deberá conocer en todo momento la posición de los vehículos sobre la carretera.
- **Requisito(03) *"Detección de vehículos"*:** El sistema debe detectar vehículos en una zona inicial de la calzada para después poder llevar a cabo su seguimiento.
- **Requisito(04) *"Seguimiento vivaz y preciso*:** Los diferentes vehículos detectados deberán ser evaluados constantemente, para poder llevar a cabo un seguimiento correcto. Además es de vital importancia que el seguimiento de los coches sea vivaz y preciso ya que estos circulan a velocidades elevadas.
- **Requisito(05) *"Estimación precisa de la velocidad y contador de vehículos"*:** El sistema deberá comunicar la velocidad, de forma muy precisa, a la que circula cada coche. Además la aplicación deberá llevar la cuenta del número de coches que pasan por la carretera.

2.2.2. Requisitos no funcionales

Una vez detallada la funcionalidad del sistema, se pasa a citar las restricciones del sistema final.

Requisitos del producto

- El sistema deberá ser lo más robusto posible, tolerante frente a cambios de iluminación en el entorno y a los cambios de carril que puedan realizar los vehículos.
- La captura de imágenes y su posterior tratamiento ha de ser vivaz y en tiempo real, por ello se deberá implementar algoritmos rápidos y eficientes. Al realizar una implementación vivaz, el sistema debe de funcionar en procesadores no muy potentes y por lo tanto más económicos.

Requisitos del proceso

- El lenguaje de programación para la implementación del sistema será *ansi- C.*
- El sistema operativo utilizado será GNU/Linux, puesto que es el usado en el laboratorio de robótica.
- El sistema será diseñado de acuerdo a la arquitectura *jde* y su implementación software actual jde.c. Se utilizará esta arquitectura porque, además de que es la propuesta por el grupo de robótica, soluciona muchos problemas como la captura de imágenes, la configuración de los *drivers*, etc.

2.3. Metodología

Para la realización de este proyecto se ha optado por seguir un modelo iterativo (Figura 2.1). Este tipo de modelo de desarrollo se intenta adaptar a los cambios, por lo que es idóneo para el desarrollo de proyectos de investigación, como es nuestro caso. Los modelos iterativos se basan en dividir el proyecto de desarrollo en varias etapas, llamadas iteraciones. La idea central es que en cada una de esas iteraciones se construye una parte del sistema. Para esa parte del sistema, se realiza todo el proceso: análisis, diseño, desarrollo y pruebas. Las principales ventajas que adopta este modelo son las siguientes:

- *Flexibilidad*, ya que los requerimientos no quedan totalmente fijados hasta el final del proyecto y se pueden realizar cambios de forma flexible. Por una parte, el conocimiento que se adquiere en una iteración sirve para plantear de forma más realista los requerimientos de la siguiente. Por otra parte, este conocimiento nos

puede hacer reformar partes del sistema construidas en iteraciones anteriores. En una palabra, todos los documentos del sistema (requerimientos, diseño y código) no son rígidos sino que pueden cambiarse durante todo el proceso de desarrollo

- *Mitigación de riesgos*, como las pruebas se hacen desde el principio del proyecto, puede determinarse la viabilidad o eficiencia de las decisiones de diseño.

- *Retroalimentación*, los desarrolladores tienen una rápida retroalimentación de lo que funciona y lo que no, ya que las pruebas se realizan desde el comienzo mismo del proyecto y no se debe esperar al final para hacer las modificaciones necesarias.

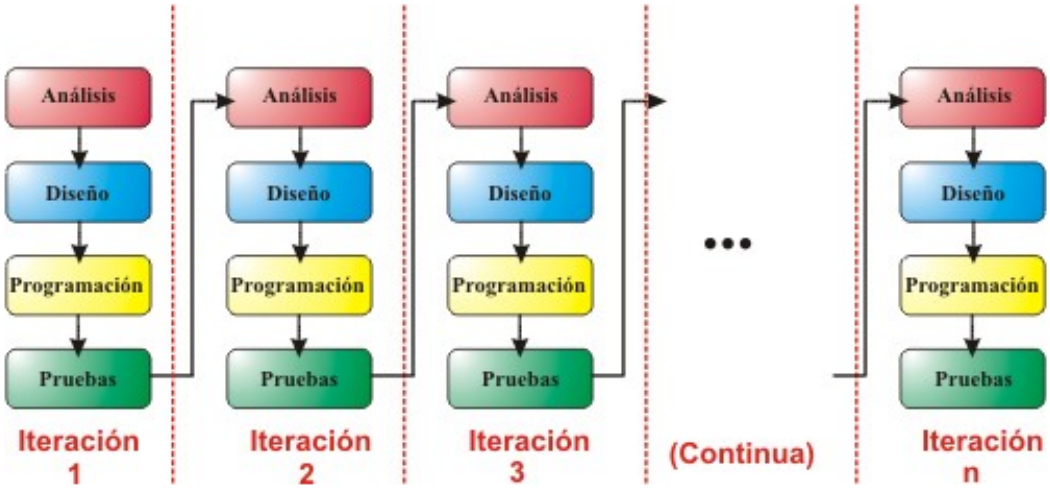


Figura 2.1: Modelo de desarrollo iterativo

Durante todo el desarrollo del proyecto se han mantenido reuniones semanales con el tutor para establecer y afinar los puntos llevados a cabo en cada etapa, y para comentar los resultados de etapas anteriores.

Para la consecución de este proyecto se han completado las siguientes iteraciones:

1. *Formación y familiarización con la infraestructura básica* (duración 1 mes)

   a) *Familiarización con la plataforma software jde.c* y estudio a fondo de la estructura de la misma. La primera iteración consistirá en la creación de varios esquemas para comprender el funcionamiento general de jde.c.

   b) *Estudio de técnicas de visión artificial y otros conocimientos relativos* tales como filtros de color, calibración de cámaras, seguimiento visual y visión

3D. Diseño y creación de nuevos esquemas en los que se apliquen dichas técnicas.

   c) *Estudio de otros esquemas afines*, como el rectificador [Kachach, 2008] o el de aplicaciones con flujo óptico [Santos, 2007], con el fin de poder integrarlos y adaptarlos a las necesidades de este proyecto.

   d) *Estudio del algoritmo evolutivo multimodal*, implementado en diferentes proyectos fin de carrera del grupo de robótica, como el de seguimiento 3D de personas [Marugán, 2007] , reconstrucción 3d visual con algoritmos evolutivos [Martínez, 2007] y localización de mapas en un robot de interiores [Ángel Cortés Maya, 2007].

2. *Desarrollo de infraestructura especifica: Simulador Gazebo* (duración 3 semanas)

   a) *Instalación* del simulador.

   b) *Cambios en el simulador*, para adoptar una representación del mundo similar a una situación real, donde la cámara se encuentra encima de una carretera.

   c) *Driver para jdec*, desarrollo del driver *GazeboCars* para poder interactuar desde la plataforma *jdec* con el simulador

3. *Relación entre la imagen y el espacio métrico*

   a) *Calibración de cámaras*, para poder manejar un espacio 3d en nuestra carretera (duración 1 mes)

   b) *Estudio e integración* de algoritmos de rectificación de imágenes [Kachach, 2008]. Con esta rectificación en nuestras imágenes conseguiremos manejar un espacio 2d en la carretera, sin tener la cámara calibrada. (duración 2 semanas)

4. *Técnicas de detección de movimiento en imagenes* (duración 2 semanas)

   a) *Flujo óptico*, estudio e integración del proyecto fin de carrera aplicaciones con flujo óptico [Santos, 2007]

   b) *Diferencia de imagenes*, estudio y desarrollo de detección de movimiento mediante diferencia de imágenes.

5. *Estudio, Diseño e implementación del algoritmo evolutivo multimodal*, para realizar el seguimiento correcto de los vehículos, realizar una estimación de velocidad de cada uno de ellos en cada iteración y llevar la cuenta de coches que pasan por la carretera. (duración 4 meses)

6. *Experimentos globales con imágenes reales* La última iteración realizada fue probar el sistema con imágenes reales, realimentando la implementación y persiguiendo la mejora. (duración 1 mes)

Capítulo 3
Plataforma de desarrollo

En este capítulo vamos a presentar la plataforma software sobre la cual se ha desarrollado este proyecto. Además de apoyarse en la arquitectura *jdec*, nuestro software hace uso de otras bibliotecas auxiliares y de un simulador que le facilitan algunas tareas concretas. Para la elección de estas librerías se ha tenido en cuenta los requisitos y restricciones impuestos en las secciones anteriores.

3.1. Plataforma software

Este proyecto fin de carrera se ha desarrollado sobre el sistema operativo GNU/linux, la arquitectura *jdec* y ha sido implementado en el lenguaje de programación C.

GNU/Linux es un sistema operativo similar a Unix, de libre distribución, multitarea y multiusuario. Es un sistema de 32 bits, robusto, ágil, muy completo y portado a la mayoría de los procesadores del mercado. La distribución seleccionada dentro de la amplia gama ha sido Ubuntu [1] por ser completamente libre y fácil de usar. Está accesible gratuitamente a través de internet y su actualización es muy sencilla.

Una de las desventajas de este sistema operativo es que no es un sistema de tiempo real, ya que no permite acotar plazos porque su sistema de planificación de procesos no implementa esas garantías. Sin embargo, GNU/Linux resulta suficientemente ágil para los requisitos necesarios en este proyecto.

En cuanto al lenguaje de programación, como ya se ha mencionado, se ha empleado C, ya que la plataforma *jdec* usada para la elaboración de este proyecto tiene soporte para él. Además, al existir una comunidad muy amplia de grupos de investigación que realizan sus desarrollos en este lenguaje, la reutilización o integración de software se

[1] http://www.ubuntu.com

vuelve más factible.

Este proyecto fin de carrera se apoya en la plataforma Jdec [2] (Jerarquía Dinámica de Esquemas), que ha sido desarrollada íntegramente en la URJC para facilitar la programación de robots y de aplicaciones relacionadas con la visión artificial. Su origen se remonta al año 1997 como fruto de una tesis doctoral [Plaza, 2003] desde entonces ha ido creciendo de un año para otro con nuevas funcionalidades. Este proyecto se ha desarrollado sobre la versión 4.2 de jdec.

En *jdec* las aplicaciones se conciben y organizan como un conjunto de componentes concurrentes. Los componentes de la aplicación de un usuario se conocen como *esquemas* y los que facilitan el manejo de sensores y actuadores, *drivers*. Cada uno de ellos tiene un hilo de ejecución propio y realiza una tarea concreta de forma iterativa. Todos ellos pueden compartir entre sí determinadas variables con distintos propósitos. Además, en *jdec* existen componentes ya listos para usarse, un ejemplo de estos componentes son los *schemas* rectificador [Kachach, 2008] y *opflow* [Santos, 2007], los cuales han sido reutilizados en este proyecto fin de carrera.

Esta posibilidad de conectar los esquemas y *drivers* da lugar al desarrollo de aplicaciones de diferentes niveles de complejidad. Para aplicaciones reactivas sencillas, con un único esquema que tenga muchas iteraciones por segundo resulta suficiente. Para aplicaciones algo más complejas *jdec* propone dividir la funcionalidad en esquemas perceptivos y esquemas de actuación concurrentes, todos al mismo nivel. Los esquemas perceptivos tratan los datos sensoriales y los esquemas de actuación recogen la información sensorial o de otros esquemas perceptivos y toman decisiones de actuación. Para aplicaciones aún más complicadas *jdec* propone organizar la colección de esquemas en una jerarquía de varios niveles, donde unos activan a otros.

Este proyecto fin de carrera usa básicamente un esquema perceptivo, por lo que sólo hay un nivel y no utiliza jerarquía. Además en dicho esquema se ha integrado la funcionalidad aportada por el esquema *rectificador*, y puede usar los datos proporcionados por el esquema de *cálculo de flujo óptico*. Para poder desarrollar la aplicación dichos esquemas se han debido conectar a un *driver* para que nos facilite imágenes. En *jdec* existen varios drivers que nos faciliten imágenes, en concreto son 3 los que hemos usado en este proyecto : *gazeboCar* que nos ofrece las imágenes obtenidas del

[2] https://trac.robotica-urjc.es/jde/

simulador gazebo, *firewire* las imágenes son adquiridas desde cámaras *ieee 1934*, y por último *mplayer* que proporciona imágenes a partir de archivos de vídeos o cámaras IP.

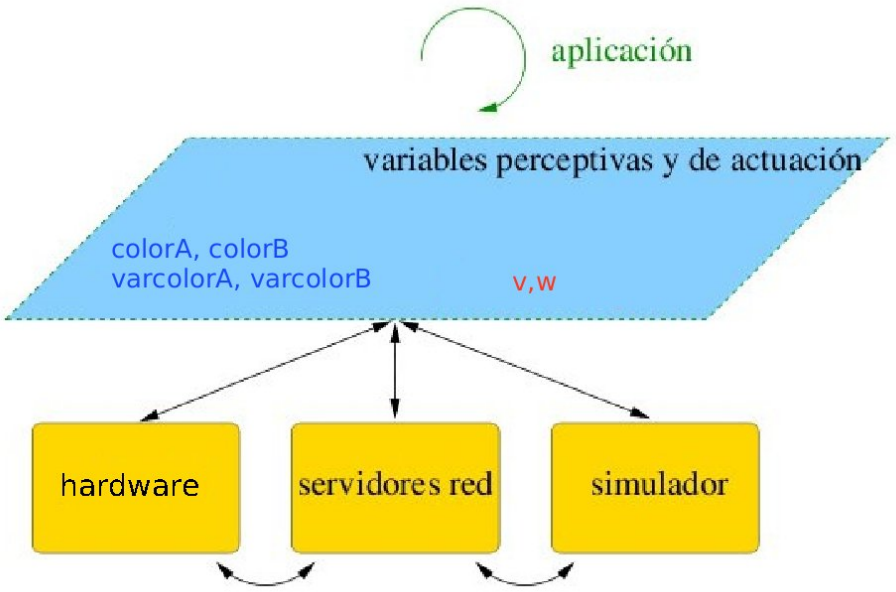


Figura 3.1: Jdec básico

Jdec que ofrece una interfaz de variables de percepción y de actuación. La aplicación obtiene las últimas observaciones sensoriales leyendo las variables perceptivas, que la plataforma mantiene permanentemente actualizadas. Por otra parte ordena comandos a los motores escribiendo en las variables de actuación, que la plataforma se encarga de materializar y hacer llegar hasta los actuadores correspondientes. Si vemos la figura 3.1, las variables de color azul se corresponden con las variables perceptivas, en concreto estas se encargan de ofrecer las imágenes .

Como podemos ver en la figura 3.1 existen tres posibles configuraciones. Ya que los sensores y actuadores pueden estar conectados físicamente al ordenador, pueden ser simulados o bien *jdec* se puede conectar a simuladores servidores de red, donde están conectados o simulando los actuadores y sensores.

Si bien la plataforma está inicialmente orientada a la programación de comportamientos en robots, en los últimos años se ha trabajado en incluir facilidades para el desarrollo de aplicaciones que usen visión artificial. En esta línea, se ha desarrollado la biblioteca *progeo* que posibilita conseguir información 3D en las imágenes, teniendo las cámaras calibradas. En este proyecto fin de carrera se ha desarrollado el driver *gazeboCars* que proporciona acceso de imágenes y envía diversos comandos de actuación, al

simulador gazebo.

3.2. Simulador *Gazebo*

Gazebo es un simulador multirrobot para entornos abiertos, que es capaz de simular un conjunto de robots, sensores y objetos en un entorno tridimensional. Además incorpora simulación física realista (figura 3.2), de modo que los objetos tienen masa, se pueden empujar unos a otros, etc. Entre los sensores que simula se incluyen cámaras como la sonyVID30, la canonVCC4, el robot *Pioneer* o la aspiradora robótica *Roomba*.

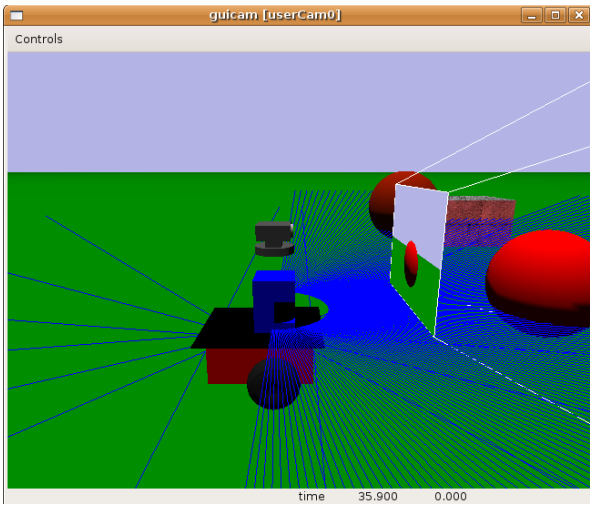


Figura 3.2: Simulador Gazebo

Dentro del contexto de este proyecto fin de carrera, el objetivo por el cual usamos este simulador, versión *0.7*, es el de tener un entorno sencillo y fiable donde probar nuestro algoritmo con imágenes sintéticas y con colores sencillos antes de probarlo en un entorno real.

El simulador Gazebo es parte del entorno software abierto Player/Stage/Gazebo [3], en cuya página web del proyecto se puede descargar el código fuente y ver la documentación actualizada.

[3] http://playerstage.sourceforge.net

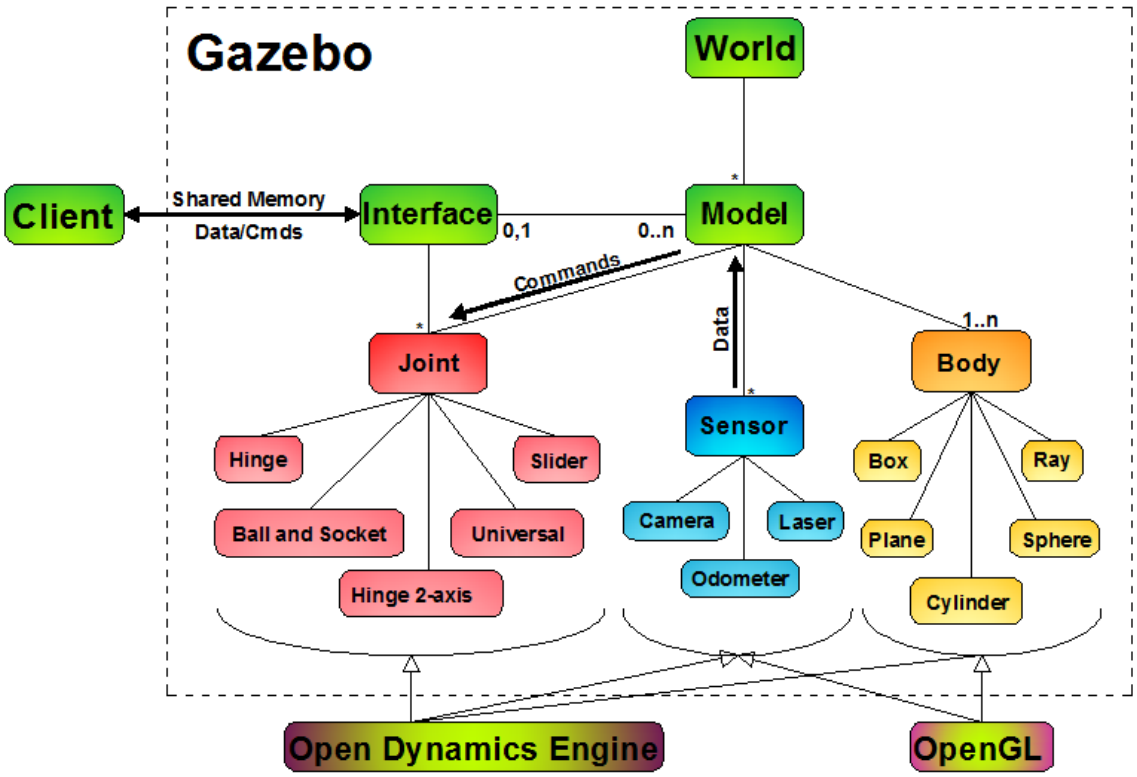


Figura 3.3: Arquitectura software del simulador gazebo

Este simulador se apoya en librerías de software libre como: *ODE* [4] (Open dinamic Engine), *Gdal* [5] (Geospatial Data Abstraction Library) y *opengl* [6].*Gazebo* ha sido desarrollado en C++, aunque ofrece una librería en C (libgazebo), que facilita a otros programas leer datos sensoriales y enviar comandos a los distintos actuadores. En la figura 3.3 podemos ver la estructura de dicho simulador. Este simulador lo podemos ver como una arquitectura *cliente - servidor*, donde el cliente son nuestros programas y el servidor el simulador. En la parte del simulador (servidor), en el *wordfile* hay una descripción del mundo que será simulado por *Gazebo*. En este mundo se describen robots, sensores, cámaras, coches, fuentes de luz o la gravedad. Cada objeto descrito en el *wordfile* tiene un modelo, dicho modelo está compuesto de 1 a infinitos cuerpos, que pueden ser cajas o cilindros entre otros, puede llevar o no sensores (de donde obtenemos los datos) y *joints* (uniones entre dos cuerpos, y donde se envían los comandos a los actuadores). Por último tenemos la interfaz, que es la que nos ofrece la librería *libgazebo* que usa interprocesos de comunicación para intercambiar datos entre el cliente y el servidor, mientras el simulador se está ejecutando.

[4] http://www.ode.org/
[5] http://www.gdal.org/
[6] http://www.opengl.org/

Como podemos ver en la figura 3.4, para este proyecto fin de carrera hemos simulado una carretera, con un determinado número de coches. Para poder usar dicho simulador con la arquitectura *Jdec*, se ha desarrollado un *driver*, *gazeboCars* que satisface nuestras necesidades. Los principales características de este *driver* son las siguientes:

- Se obtienen datos y envían comandos para poder controlar a los diferentes coches definidos en el mundo.
- Proporcionar las imágenes obtenidas por las cámaras simuladas a la arquitectura *jdec*
- Finalmente, para no estar creando dinámicamente coches en el simulador, desde el cliente, que pasarán por el campo visual de la cámara y no saturarán el procesador del PC, se optó por cambiar de posición a los coches. De esta forma, cada vez que un coche pase el campo visual de la cámara se le volverá a colocar en una posición anterior a la del campo visual definido por la cámara.

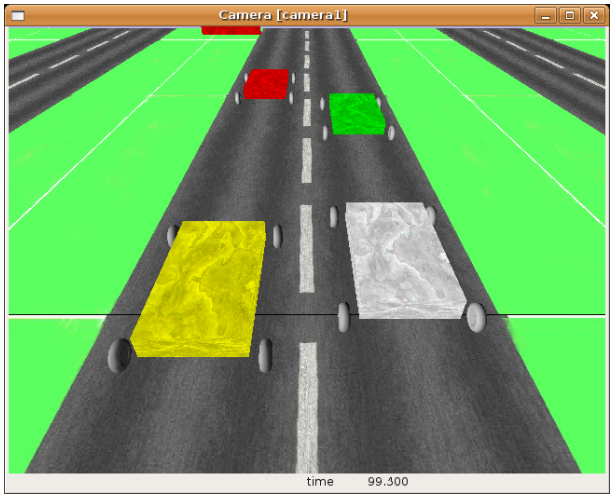


Figura 3.4: Simulación de coches sobre una carretera

3.3. Bibliotecas auxiliares

3.3.1. GSL

GSL [7] (GNU Scientific Library) es una librería de uso libre que ofrece un conjunto muy amplio de operaciones matemáticas de todo tipo. La biblioteca ha sido escrita de forma entera en el lenguaje C y ofrece un API muy fácil de usar facilitando al programador el uso de las distintas funcionalidades que ofrece. Entre las distintas rutinas que

[7] http://www.gnu.org/software/gsl/

esta biblioteca ofrece podemos encontrar: Álgebra lineal, números complejos, polinomios, permutaciones, vectores y matrices entre otras.

GSL ha facilitado los cálculos matriciales que son tan frecuentes en la geometría proyectiva. Gracias a estos cálculos, *GSL* nos ha ayudado a realizar la rectificación de la carretera [Kachach, 2008], en el siguiente capítulo veremos este hecho con detenimiento.

3.3.2. IPP

Esta biblioteca está diseñada por Intel y dirigida a realizar cálculos pesados y costosos de una manera más ágil y rápida, utilizando para ello instrucciones MMX del procesador. Es una biblioteca de bajo nivel, pero ofrece un API en C que es el que se ha utilizado en los programas de este proyecto para calcular el flujo óptico y realizar operaciones con imágenes.

IPP (*Integrated Performance Primitives*) [Ipp, 2002] ofrece una amplia gama de funcionalidad como tratamiento de señales de audio, trabajo con imágenes y vídeos, criptografía. En este proyecto fin de carrera la usaremos para detectar movimiento en las secuencias de tráfico. Todas estas funcionalidades tienen un rendimiento muy superior al habitual conseguido con las mismas funciones codificadas en c, ya que se utiliza toda la potencia del procesador.

Se dispone de varias versiones de esta biblioteca, optimizadas para la familia del procesador que se está utilizando. La que se ha utilizado está optimizada para la arquitectura Intel, que es capaz de funcionar sobre cualquier procesador de 32 bits, incrementando el rendimiento de forma diferente en unos procesadores que en otros, se ha probado también sobre procesadores AMD, en donde el rendimiento también es superior al normal.

3.3.3. Xforms

Uno de los elementos más importantes de nuestra aplicación es la interfaz gráfica, ya que nos permite visualizar los resultados obtenidos y depurar el funcionamiento.

Xforms es una biblioteca de libre uso escrita en C y está soportada por la plataforma *jdec*. Su misión principal es facilitar la creación y uso de interfaces gráficas sobre el sistema X-Window de Linux, ocultando la complejidad de ésta al programador. Para ello

ofrece un extenso repertorio de elementos gráficos sencillos (botones, sliders, canvas..) que juntos permiten crear interfaces complejas. La biblioteca ofrece una herramienta *fdesign* (ver figura 3.5) de uso visual que nos permite crear y personalizar la interfaz de nuestro programa de manera muy sencilla.

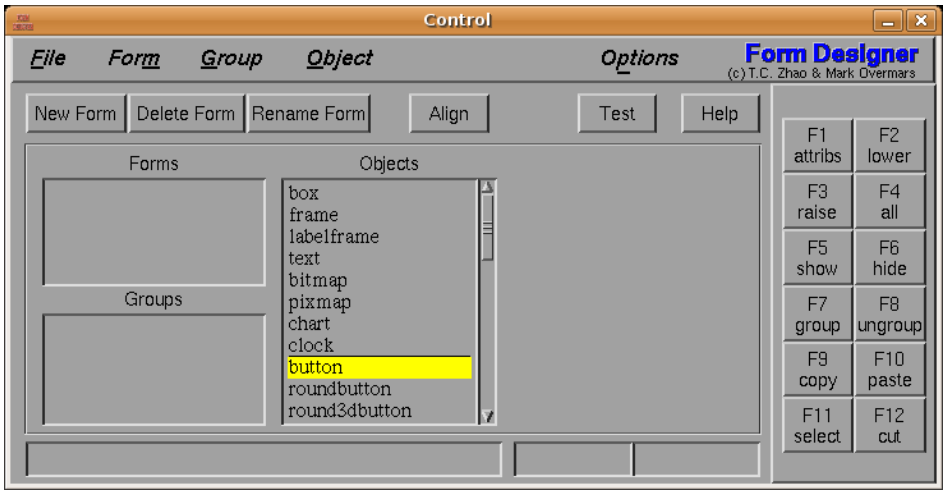


Figura 3.5: Fdesign

En este proyecto *Xforms* ha sido utilizada para crear la interfaz gráfica del esquema *carSpeed*.

Capítulo 4
Descripción Informática

Tras haber explicado en capítulos anteriores los requisitos y las herramientas necesarias para la elaboración del proyecto, en este capítulo se detalla la solución desarrollada. A continuación veremos el diseño global de la solución donde describiremos cada uno de los componentes de la aplicación, para posteriormente tratar el diseño detallado y la implementación que se ha llevado a cabo en cada una de las partes.

4.1. Diseño global

Como se dijo en el capítulo 2, el objetivo de este proyecto es conseguir contar y detectar la velocidad de vehículos con una sóla cámara. La aplicación desarrollada es capaz de detectar los vehículos que entran en el plano imagen de la cámara, y una vez detectados, seguirlos. El seguimiento visual de objetos no es más que una localización continuada del objeto en la imagen, y gracias a ello el sistema es capaz de conocer la posición de cada vehículo en la carretera en todo momento. Además, el sistema, una vez detectado el vehículo informará en cada iteración de la velocidad que éste lleva y mantiene la cuenta de los vehículos que han pasado por delante de la cámara.

El objetivo de este proyecto se ha resuelto a partir de la plataforma software *jdec* con la creación de dos esquemas, *carSim* y *carSpeed*, el desarrollo del driver *gazeboCars* y la reutilización del esquema *opflow* y el driver *mplayer*. El diseño global de la aplicación se puede observar en la figura 4.1. En ella los colores de los esquemas perceptivos están en color azul, ellos son los encargados de recoger y procesar la información de las imágenes (variable *colorA*). Los nombres de los esquemas y variables actuación están en color rojo, en nuestro caso *vCars*, *wCars*, *positionCars* se encargan de dar velocidad y cambiar la posición de los coches del simulador. Las variables que sirven de intermediarias para la comunicación entre los esquemas creados en nuestro proyecto se han puesto de color negro. Los *drivers* son los componentes de color amarillo. Por último,

los componentes que nos sirven las imágenes aparecen en color verde.

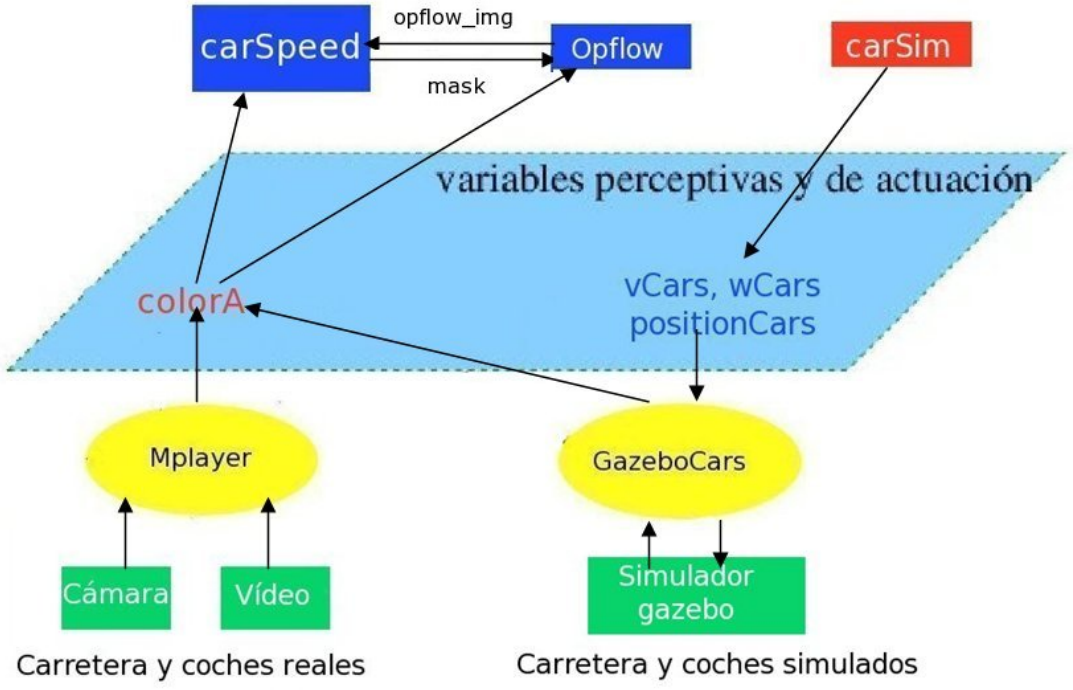


Figura 4.1: Diseño global con esquemas

CarSpeed es el esquema fundamental de la aplicación, que recibe como entradas las imágenes proporcionadas por la variable *colorA* de *jdec* y a veces las variable *opflow_img*.

Como podemos ver en la figura 4.1, el sistema sólo hace uso de imágenes como datos de entrada. Estas imágenes pueden ser simuladas (obtenidas desde *gazebo*), o reales (obtenidas a partir de vídeos ó cámaras reales). Si las imágenes son simuladas, es el *driver gazeboCars* el encargado de actualizar la variable correspondiente en *jdec*. Por el contrario, si las imágenes son reales es el driver *mplayer* el encargado de hacerlo. En este proyecto fin de carrera, la variable sensorial actualizada por dichos drivers es *colorA*. De esta manera conseguimos que la aplicación sea completamente independiente de la fuente de imágenes.

Al igual que obtenemos información de la plataforma *jdec* a través de las imágenes, la aplicación puede escribir sobre las variables de actuación, en caso de estar usando el simulador *gazebo*. El esquema *carSim* da velocidad a los seis coches simulados por *gazebo*, además de poder cambiarles de posición. El hecho de poder cambiar a los coches

de posición es para no tener que estar creando en tiempo de ejecución nuevos coches simulados, si fuese así, en ejecuciones prolongadas *gazebo* simularía demasiados coches sobrecargando de esta forma demasiado la CPU. Por este motivo, cuando los coches están muy lejos de la cámara son trasladados a una posición detrás a la del plano imagen. En la figura 4.2 vemos un ejemplo de un coche que ya ha rebasado el plano imagen de la cámara, por lo que es trasladado a una posición anterior a la cámara. *CarSim* actualiza las variables *carsPosition*, *carsW* y *carsV*, y el driver *gazeboCars* se encarga de enviar los comandos oportunos al simulador.

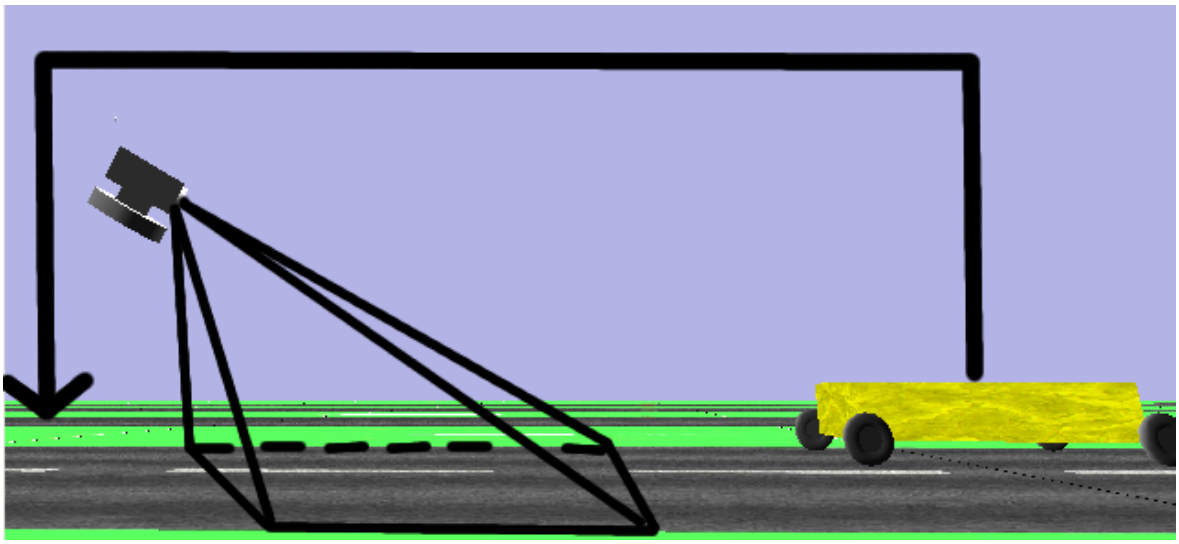


Figura 4.2: Cambio de posición de un vehículo en gazebo

Por otro lado, el esquema *opflow* se encarga de detectar movimiento en la zona inicial de la carretera. Para ello trata de encontrar la correspondencia entre los píxeles de dos imágenes consecutivas, es decir consiste en identificar un píxel de la imagen en el instante t en la imagen en el instante t+1. Este esquema ofrece al esquema *carSpeed* la variable *opflow_img*, que es una estructura que informa si en un píxel ha habido movimiento. *CarSpeed* ofrece la variable *mask* que permite al esquema *opflow* seleccionar las zonas donde se calcula el flujo óptico. Como veremos más adelante, *opflow* no es la única manera por la que se detecta movimiento en la imagen, por lo que este esquema puede estar desactivado.

4.2. Algoritmo de detección y seguimiento

El algoritmo en el que se basa esta aplicación está inspirado en un algoritmo evolutivo multimodal, el cuál ha sido implementado en varios proyectos del grupo de robótica

de la URJC. Para comprender mejor la solución adoptada explicaremos primero los fundamentos teóricos de los algoritmos evolutivos en los cuales se inspira el algoritmo de esta aplicación.

Los llamados algoritmos genéticos o evolutivos son algoritmos de búsqueda iterativos que tratan de hallar una solución óptima al problema planteado. Estos algoritmos combinan las propiedades de las hipótesis en el espacio de soluciones para obtener nuevas generaciones de hipótesis más adecuadas. Son algoritmos iterativos que guardan cierta similitud con la evolución natural y se utilizan a menudo en tareas de optimización, diseño de controladores borrosos o sistemas de aprendizaje automático.

Un determinado individuo puede ser en si mismo una posible solución (enfoque Pittsburgh) o bien sólo una contribuyente a la misma (enfoque Michigan). Estos algoritmos cuentan con un conjunto o población de N individuos, cada individuo n_i tiene asociado un determinado peso o salud h_i, que indica cómo de buena es esa solución. La población de individuos evoluciona con el tiempo y en cada iteración se actualiza la salud correspondiente a cada individuo. U

Los algoritmos genéticos se dividen principalmente en dos fases que se ejecutan iterativamente:

1. *Generación de la próxima población.* Se aplican *operadores genéticos* para obtener la nueva población de individuos. La nueva población se genera gracias a los operadores genéticos, utilizando en el proceso a los individuos que reúnen las mejores propiedades de la población anterior. Estas propiedades dependerán del propósito del algoritmo, y los operadores genéticos pueden ser diseñados expresamente para captar estos rasgos de manera más eficiente. Existen varios operadores genéticos clásicos que se suelen utilizar siendo adaptados al contexto específico de cada problema. Algunos de estos operadores son: Mutación aleatoria, Ruido térmico, Cruce o Repulsión.

2. *Computación de la salud.* Se calcula la salud para cada uno de los individuos en función de las entradas del problema en cuestión, en nuestro caso las imágenes. El cálculo de salud consiste en evaluar a cada uno de ellos a partir de unos criterios muy dependientes del problema concreto. Cada uno de estos criterios se corresponde con características que determinan la calidad del individuo. Cuantos más criterios satisface un individuo mayor será su valor de salud h_i.

En este proyecto hemos diseñado una solución a nuestro problema de seguimiento inspirada en los algoritmos evolutivos implementados en los proyectos fin de carrera de Sara Marugán [Marugán, 2007] y Ángel Cortés [Ángel Cortés Maya, 2007]. En este caso debemos detectar y seguir objetos móviles (vehículos) en una carretera para poder estimar la velocidad que llevan. Para determinar la velocidad de cualquier objeto es necesario conocer el espacio que recorre dicho objeto en un determinado tiempo. Por ese motivo es indispensable establecer una correspondencia entre la imagen y el espacio métrico de la carretera. Así la carretera queda definida en un espacio bidimensional *x,y* donde podemos saber la distancia entre dos puntos cualesquiera que se encuentren sobre ella. La primitiva de información utilizada es un conjunto de individuos no puntuales. El espacio de búsqueda para estos individuos es el de las posiciones y velocidades de los vehículos en la carretera. Los individuos están definidos por una determinada posición en la carretera , una determinada velocidad y su correspondiente salud. La población de individuos tiene como objetivo el seguimiento de los vehículos en la carretera, para ello inicialmente se hipotetizan todas las velocidades posibles de cada vehículo y según avanzan los vehículos se van descartando velocidades (individuos). De esta manera, el seguimiento de cada vehículo queda definido sobre un espacio tridimensional (*x,y, velocidad*).

Siguiendo la idea del proyecto fin de carrera de Sara Marugán [Marugán, 2007] distinguimos dos grupos diferentes de individuos que trabaja en zonas diferentes de la imagen: exploradores y explotadores.

- *Población de Exploradores.* Está compuesta por M individuos destinados a explorar el espacio de la zona inicial de la carretera. Permite descubrir los vehículos nuevos que aparecen en la imagen, por lo que podemos hablar de una zona de detección en la parte inicial de la carretera (ver figura 4.3). La salud de esta población se asigna tomando en cuenta la información de movimiento extraída de las imágenes. Un individuo será prometedor si su salud supera un umbral dado, es decir, si ha detectado un movimiento considerable. Un individuo es aquel que detecta un vehículo en la carretera. Este individuo prometedor genera una seríe de individuos explotadores que se encuentran en la misma posición pero todos ellos con una velocidad diferente. Al conjunto de todos estos individuos se les llama *raza*. El principal objetivo en esta etapa del algoritmo es que un coche nuevo de lugar a una única raza de explotadores, que incluye como hipótesis todas las posibles velocidades.

- *Población de Explotadores.* Esta población se compone de un número de razas indeterminado, inicialmente será cero. Cada raza cuenta con N individuos cuya finalidad es seguir a un vehículo, que ha sido detectado por la población de exploradores, en el espacio *(x,y,velocidad)*, de esta manera podemos hablar de una zona de seguimiento en la carretera (ver figura 4.3). Cada individuo de la raza, tiene una velocidad diferente a todos los de su raza. Así pues, si una raza esta formada por N individuos cada uno de estos tendrá un valor único en el espacio *0 - N km/h*. De esta forma no hay una estimación inicial de la velocidad de cada individuo, si no que dichas velocidades se hipotetizan y en la zona de seguimiento se van filtrando todos los individuos cuya velocidad sea incompatible con la secuencia de imágenes. La salud de cada individuo se define a través de la información de movimiento de las imágenes. Cada raza tiene un individuo representante (líder), que es aquel con mayor salud de toda la población de la raza. Las razas explotadoras se crean durante la ejecución del algoritmo, nacen a partir de individuos prometedores de la población exploradora y se extinguen si su salud es baja. Generalmente surgirá una sóla raza por vehículo y a través de ésta se realizará su seguimiento. Durante la zona de seguimiento se visualiza por pantalla la velocidad de cada vehículo correspondiente con el líder de cada raza. La estimación de la velocidad lo más cercana a la realidad se consigue cuando el coche se encuentra la parte final del seguimiento, pues esa estimación ha sido ya contrastada con todas las imágenes de ese coche.

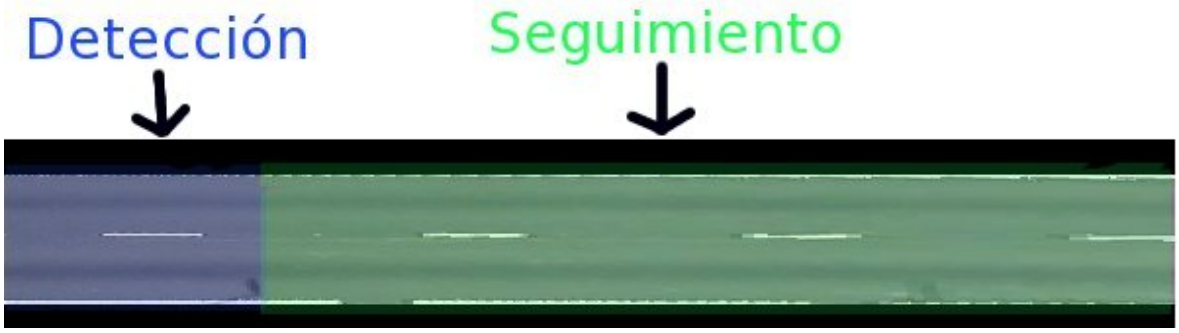


Figura 4.3: Zona de detección y seguimiento en la carretera

Como acabamos de comentar la población de exploradores se corresponde con la zona de detección de vehículos y la población de explotadores con la de seguimiento. En la figura 4.3, podemos ver gráficamente las diferentes zonas de la carretera, la zona azul se corresponde con la etapa de detección y la verde con la de seguimiento. Hay

que destacar que una vez que el vehículo ha sido detectado, comienza la etapa de seguimiento, por lo que la etapa de seguimiento de un vehículo puede solaparse con la de zona de detección.

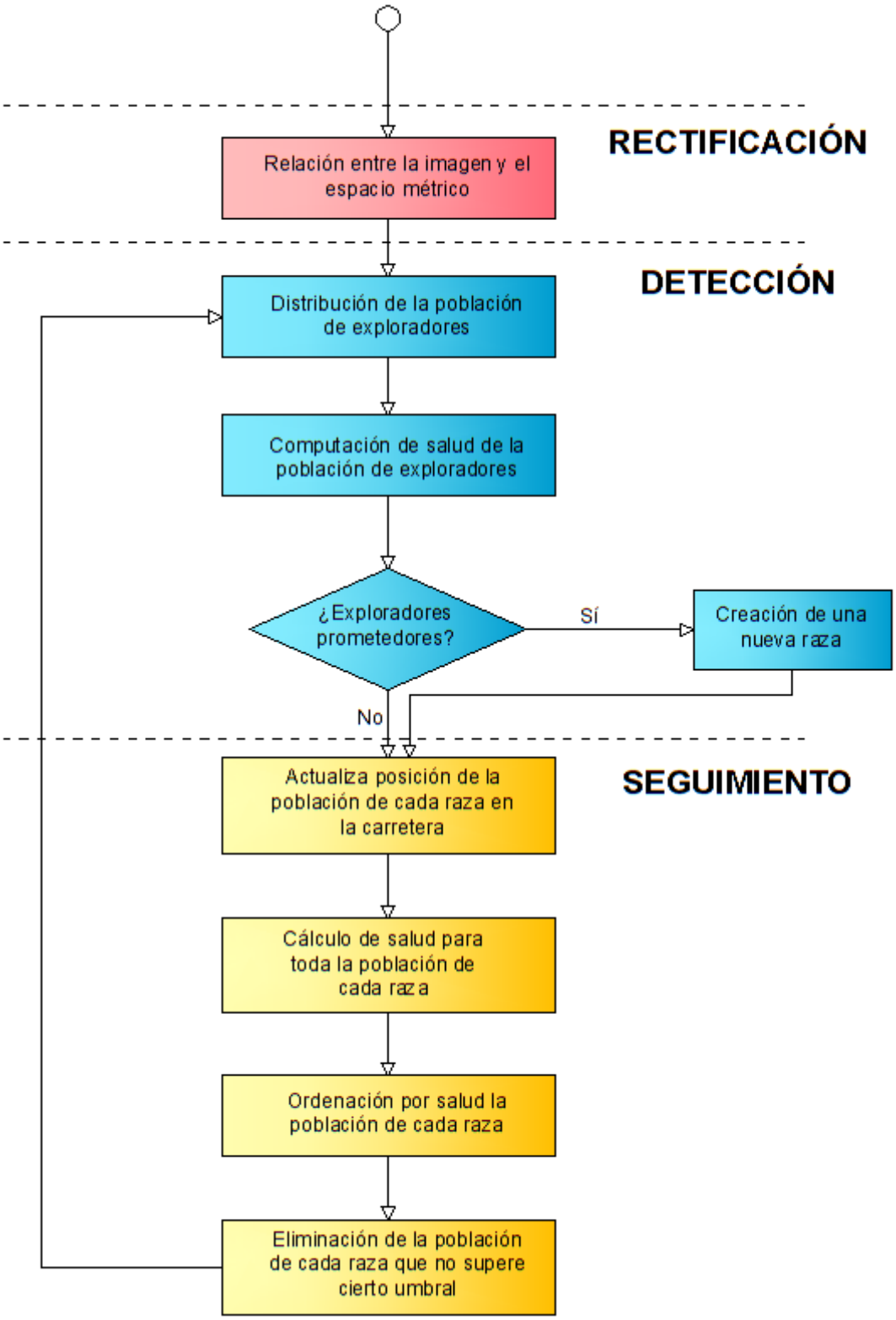


Figura 4.4: Esquema general de la solución adoptada

En la figura 4.4 mostramos una visión general de la solución propuesta. En primer lugar llevamos a cabo la correspondencia entre la imagen y el espacio métrico de la carretera. De este modo dado un píxel podremos obtener su posición en la carretera y viceversa. A continuación entramos en la zona de detección, donde se descubren por movimiento los nuevos coches que entran en la zona inicial de la carretera. Una vez detectados los coches entramos en la etapa de seguimiento, donde se estima la velocidad de cada vehículo en cada iteración y se eliminan todas las velocidades de aquellos individuos que posean una baja salud.

Como vemos en la figuran 4.4, existen tres partes bien diferenciadas: rectificación, detección y seguimiento. Una vez planteado el algoritmo completo en la sección 4.3 detallamos la rectificación, en la sección 4.4 explicamos la función salud y en la sección 4.5 y 4.6 explicaremos más detalladamente las etapas de detección y seguimiento.

4.3. Relación entre la imagen y el espacio métrico

Para seguir (localización continuada) y estimar la velocidad de un vehículo, es necesario saber la posición del coche en la carretera. Para ello es útil conocer la distancia entre dos puntos cualesquiera de la carretera, es decir, realizar mediciones de distancias. Hemos realizado un rectificación de la imágen, estableciendo una homografía entre el plano imagen y el plano de la carretera, para poder estimar distancias. Además hay que destacar, que hemos asumido que la carretera es plana en el tramo observado por la cámara.

El problema de la recticación consiste en deshacer el efecto que produjo la cámara en la imagen capturada. Estos efectos pueden ser básicamente de dos tipos: Distorsión por perspectiva (fruto de capturar la imagen con cierto ángulo) o distorsión radial (fruto de la distorsión de la lente). En este proyecto al colocar la cámara encima de un puente la imagen de la carretera sufre una distorsión de perspectiva por la posición de la cámara. Para recuperar una vista desde arriba de la carretera y poder manejar distancias se ha utilizado la rectificación de imágenes.

La rectificación se basa en geometría proyectiva [Hartley y Zisserman, 2004b]. Una proyección transforma una figura en otra equivalente manteniendo todas sus propiedades de proyección invariantes. Aprovechando el modelo de cámara Pinhole, intentaremos deshacer las transformaciones resultantes de la proyección.

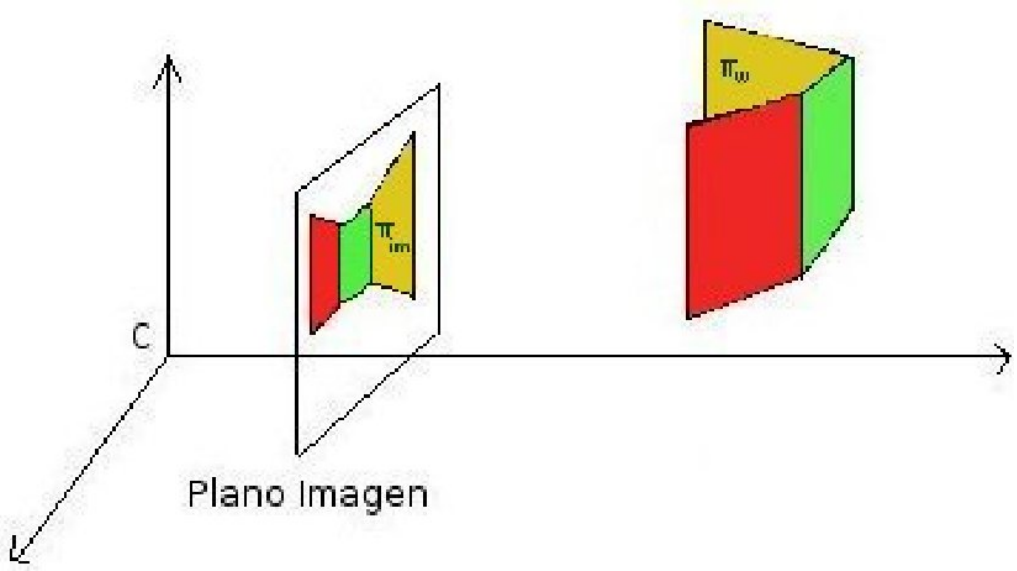


Figura 4.5: Mapeo entre dos planos

En la imagen 4.6 podemos ver cómo la cámara transforma el plano π_w del mundo en el plano π_{im} de la imagen. La rectificación intenta darle la vuelta a este mapeo y obtener el plano π_w a partir del plano de la imagen π_{im}.

El modelo de cámara Pinhole se basa en la proyección perspectiva que a su vez es una transformación lineal. En la figura 4.6 podemos ver cómo se pasa de una imagen que sufrió una transformación de proyección perspectiva a una imagen frontal sintetizada donde se reconstruye el plano de la pared tras haber elegido las cuatro esquinas de la ventana para formar las correspondencias.

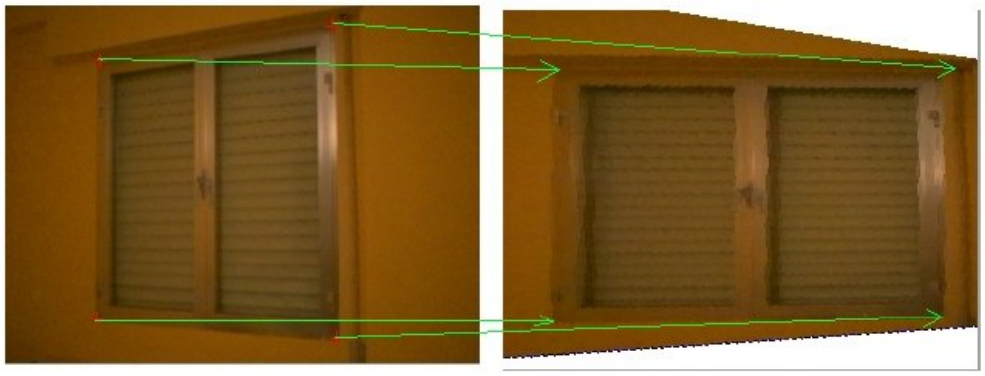


Figura 4.6: Reconstrucción del plano pared

Según el modelo Pinhole la ecuación general del modelo de cámara que relaciona

un punto cualquiera del espacio P_w con un píxel P_{im} de la imagen:

$$P_{im} = HP_w \tag{4.1}$$

$$P_w \in \pi_w$$

$$P_{im} \in \pi_{im}$$

Donde π_w es el plano que queremos reconstruir y H_{3x3} una matriz homogénea de ocho grados de libertad (el noveno es para la escala). Para deshacer la transformación basta con calcular la matriz H y aplicar la transformación inversa:

$$P_w = H^{-1}P_{im} \tag{4.2}$$

La relación 4.1 de proyección se puede representar con una matriz homogénea H de nueve elementos, donde $x1$,$x2$, $x3$ es un punto en el plano π_{im} en coordenadas homogéneas y $x1^{'}$,$x2^{'}$, $x3^{'}$ es un punto en el plano π_w en coordenadas homogéneas.

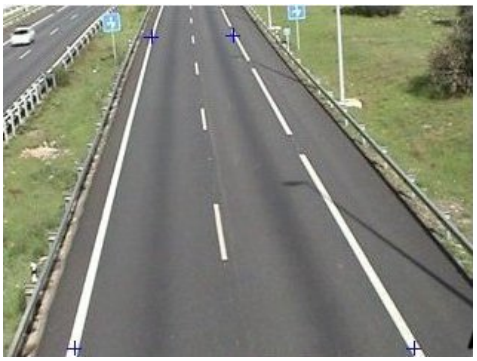
$$\begin{pmatrix} x1^{'} \\ x2^{'} \\ x3^{'} \end{pmatrix} = \begin{pmatrix} h11 & h12 & h13 \\ h21 & h22 & h23 \\ h31 & h32 & h33 \end{pmatrix} . \begin{pmatrix} x1 \\ x2 \\ x3 \end{pmatrix}$$

Para poder calcular la matriz H es necesario conocer cuatro puntos en la imagen de entrada y otros cuatro puntos que se correspondan con los ya elegidos en la imagen de salida. Para ver cómo calcular la matriz H y ver con más detalle el análisis con geometría proyectiva para rectificar imágenes, consultar el proyecto fin de carrera de Redouane Kachach [Kachach, 2008].

Una vez calculada la matriz H ya podemos construir la imagen rectificada. El proceso de construcción consiste en recorrer la imagen de salida y por cada píxel $(x0, y0)$ calculamos su correspondiente (x, y) en la imagen de entrada y lo pintamos con el mismo color. Se puede dar el caso de que el mismo píxel de la imagen de entrada se mapee a varios píxeles de la imagen de salida. El resultado es una imagen sintetizada reconstruida a partir de la imagen de entrada y la matriz H.

En la figura 4.7 podemos ver la rectificación de la carretera. La figura 4.7(a) es la carretera original, la figura 4.7(b) es la carretera rectificada donde realizamos el seguimiento, en ella podemos observar que las esquinas superior derecha e inferior

derecha de la imagen de salida son de color blanco. Esto se debe a que es imposible reconstruirlas ya que la información necesaria para hacerlo no ha sido capturada en la imagen de entrada. La figura 4.7(c), muestra la carretera rectificada tal y como se muestra en la interfaz gráfica de nuestra aplicación.



(a) Carretera original

(b) Carretera rectificada

(c) Carretera rectificada vista en la aplicación

Figura 4.7: Rectificación de la carretera

4.4. Función Salud

En esta sección vamos a comentar cómo calculamos la salud perteneciente a cada individuo, ya sean exploradores o explotadores. La función salud de cada individuo la definimos como un valor entre 0 y 1. En cada iteración se obtendrá un valor correspondiente a cada individuo. Si ese valor no supera un determinado umbral el individuo es eliminado, en el caso de ser explotador. De esta manera, esta salud permite valorar si la velocidad de una raza es la real a la que circula el vehículo, descartando los individuos (es decir, las velocidades) que no son compatibles con las imágenes.En el caso de ser explorador, si tiene salud baja, no será un individuo prometedor por lo que no podrá dar lugar a una raza.

Como hemos comentado anteriormente, un individuo queda definido por su velocidad v y posición x,y en la carretera rectificada. La posición no es una primitiva de información puntual, es decir, un individuo no es un punto, sino que abarca una superficie en la carretera. La superficie que abarca un individuo queda establecida por

los conceptos de *cuerpo y falda* que nos sirven para calcular la salud de dicho individuo. En la figura 4.8(b) vemos un ejemplo de un individuo con una salud ideal, ya que tiene la parte del cuerpo (cuadrado rojo) completamente dentro del vehículo y la falda (cuadrado verde) completamente fuera del vehículo. El *cuerpo* se corresponde con la media de píxeles, correspondientes al rectángulo rojo de la figura 4.8(b), donde se ha detectado movimiento, y la *falda* con la medía de píxeles, correspondientes con el rectángulo de la figura 4.8(b) verde, donde no se ha detectado movimiento. Es en la carretera rectificada donde esta definida el cuerpo y la falda de cada individuo. La solución desarrollada para el cálculo de la salud asociada a cada individuo viene dada por la ecuación 4.3. Utilizando la relación de correspondencia entre la carretera y la imagen proyectamos el *cuerpo* y la *falda* en la imagen, que es donde se calcula la salud asociada al individuo. Además, gracias a la relación de correspondencia entre imagen y carretera rectificada, la falda y el cuerpo, al igual que los vehículos, disminuyen su tamaño en la imagen cuando se alejan de la cámara. En la figura 4.8(a) mostramos como el cuerpo y la falda del coche más alejado ocupa un menor tamaño.

(a) Individuos proyectados en la imagen

(b) Cuerpo y falda de un individuo

Figura 4.8: Cuerpo y falda de la función salud

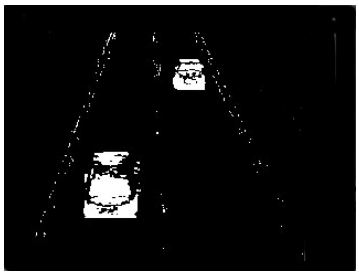
$$salud = 0{,}5 * salud_{cuerpo} + 0{,}5 * salud_{falda} \tag{4.3}$$

En el párrafo anterior hemos comentado que para calcular la salud, nos basamos en la detección de movimiento en el cuerpo, y justamente todo lo contrario en la falda. Para detectar movimiento en el cálculo de la función de salud usamos una diferencia de imágenes entre la imagen actual y una imagen de fondo. En la figura 4.9 mostramos un ejemplo de cómo se calcula salud para la etapa de seguimiento. En la figura 4.9(a),

tenemos la imagen de fondo, mientras que en la figura 4.9(b) vemos la imagen actual donde se va a calcular la salud. El resultado de restar ambas imágenes lo mostramos en la figura 4.9(c), donde los píxeles de color blanco son aquellos donde se ha detectado movimiento. En la figura 4.8(a) mostramos la proyección del individuo líder de las dos razas existentes en ese momento para cada vehículo en la imagen.

(a) Imagen de fondo

(b) Imagen con los coches sobre la carretera



(c) Detección de movimiento en la imagen

Figura 4.9: Detección de movimiento para el cálculo de la función salud

La salud asociada a cada individuo se va acumulando en cada iteración. De tal forma que la salud de un individuo es la media de la salud total acumulada. El motivo de utilizar un salud media acumulada, es que la salud instantánea es inconsistente debido a ciertas holguras en los cálculos de tiempo entre iteraciones del algoritmo.

Seguidamente, una vez comentada la solución adoptada, haremos un estudio espacial de la función salud y para terminar comentaremos otras implementaciones de la función salud realizadas antes de conseguir la solución final.

4.4.1. Estudio espacial de la función salud

Como comentamos anteriormente, nuestra función salud está compuesta por cuerpo y falda. La funcionalidad de la falda es que la altura de los vehículos no afecte negativamente a la estimación de la velocidad y hace que la función salud sea más discriminante. De esta forma para coches entrantes (los que se acercan a la cámara) la falda se encuentra en la parte delantera del coche, y en los coches salientes (los que se alejan de la cámara) la falda se encuentra en la parte trasera del vehículo.

En la figura 4.10 podemos ver cómo se distribuye la salud en un vehículo. El eje x e y se corresponden con la posición a lo largo y ancho de un individuo en la carretera.

Mientras que el eje z muestra la salud de un individuo en una determinada posición de la carretera. En la figura 4.10 podemos ver cómo la salud del vehículo es mucho mayor en la zona trasera. Según avanzamos a la zona delantera dicha salud disminuye, debido a que en la zona de la falda se va detectando movimiento. En la parte derecha de la imagen (figura 4.10), vemos el cuerpo y la falda del individuo de mayor salud del vehículo. El cuerpo y la falda de dicho individuo también esta representado en el plano 2d (x,y) de la gráfica. De esta forma podemos comprobar que dicho individuo se corresponde con el pico de mayor salud de la distribución espacial del vehículo.

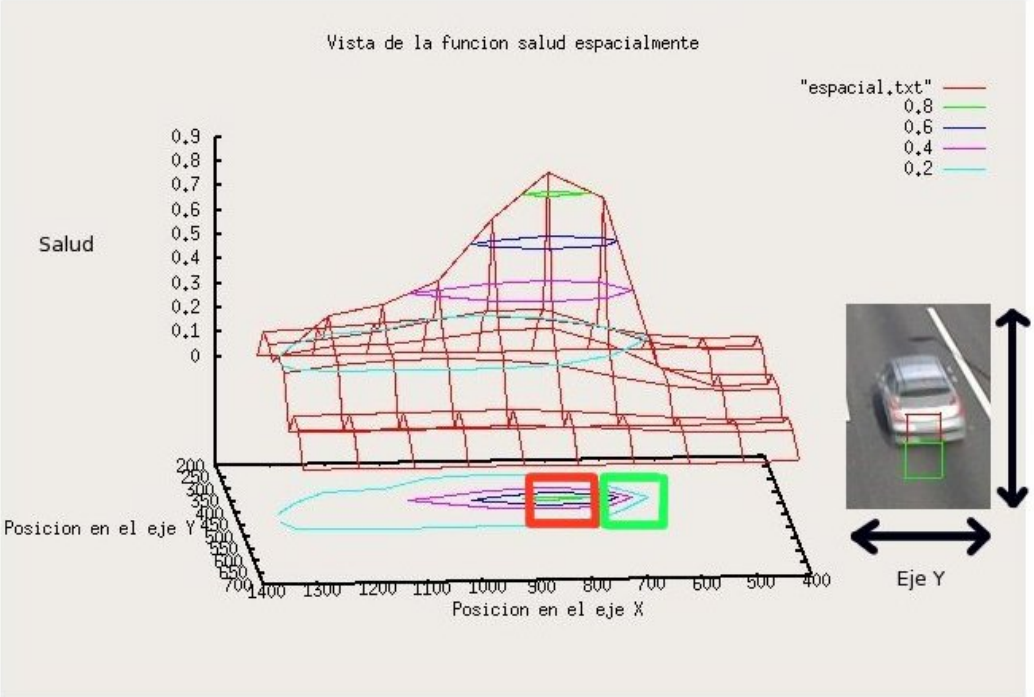


Figura 4.10: Distribución de la salud en un vehículo

En la figura 4.11 mostramos un análisis más exahustivo de la función salud. Donde no sólo vemos la salud asociada a la posición de cada individuo, si no que podemos ver en qué píxeles se detecta movimiento, la posición del *cuerpo* y la *falda* en determinados individuos de la raza junto con su probabilidad. Al igual que en la figura 4.10 el eje x e y se corresponden con la posición del vehículo en la carretera, y en el eje z podemos ver la salud asociada a un individuo. En la esquina inferior izquierda mostramos los píxeles donde se ha detectado movimiento al pasar el vehículo. En las cuatro fotografías de los coches, podemos ver las diferentes probabilidades de diferentes individuos de la raza. Si observamos la gráfica podemos ver que la mayor probabilidad del coche está localizada en aquellos individuos de la raza que se encuentran en la parte trasera del vehículo.

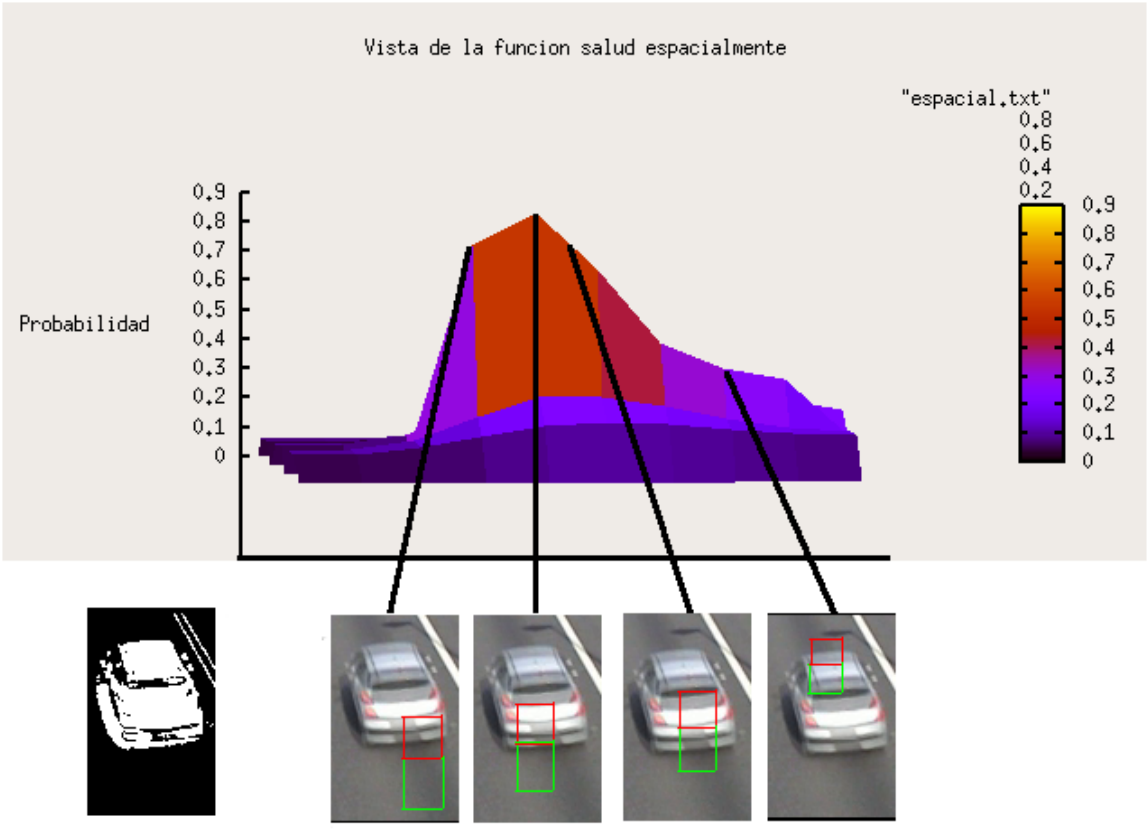


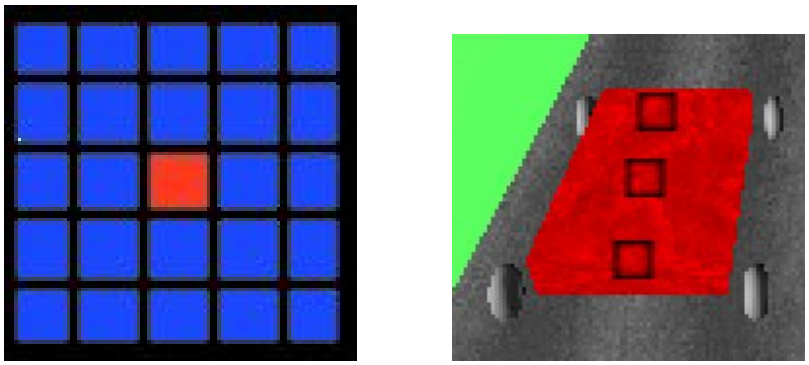
Figura 4.11: Análisis de la función salud

4.4.2. Implementaciones alternativas de la función salud

Antes de llegar a la solución final desarrollamos dos implementaciones alternativas de la función salud. A continuación detallamos cada una de ellas.

En la primera implementación los individuos estaban representado por su posición (x,y) en la carretera rectificada y su velocidad. A partir de esta posición se proyectaba en la imagen el individuo donde se calculaba su salud. Para calcular la salud se tenía en cuenta los píxeles vecinos (vecindad 5x5), en la imagen, además del píxel del cuál se quiere conocer su salud. Así calcula el porcentaje de píxeles donde se ha detectado movimiento. En la figura 4.12(a), podemos ver que el píxel de color rojo es del que se quiere conocer su salud, y los píxeles de color azul son sus píxeles vecinos. Pero esta función salud presenta dos problemas. El primero de ellos, era que no se podía saber con exactitud la velocidad del vehículo, sobre todo en zonas de la carretera donde la resolución para los vehículos es mayor, ya que existen varios individuos de la raza que tienen la salud máxima, como podemos ver en la figura 4.12(b). Por otro lado, en zonas donde la resolución es menor (zonas alejadas de la cámara), el vehículo ocupa muy pocos

píxeles y el individuo sigue manteniendo el mismo número de píxeles (vecindad 5x5), por lo se detecta movimiento en un menor número de píxeles y se eliminan todos los individuos de la raza.



(a) Vecindad 5x5

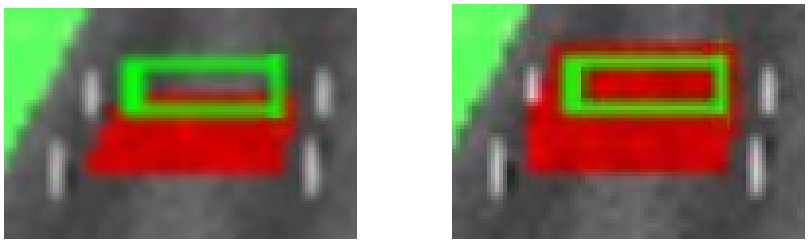
(b) Salud poco discriminante

Figura 4.12: Cálculo de la función salud mediante píxeles vecinos

En la segunda implementación la posición en la carretera rectificada de los individuos no era puntual. Es decir, un individuo no estaba ligada únicamente a dicha posición, si no que quedaba definido a partir de su posición por el concepto de *cuerpo* (rectángulo en la carretera rectificada). El cuerpo queda definido en la carretera rectificada y se proyecta sobre la imagen donde se calcula la salud del individuo. En la imagen 4.13 mostramos el *cuerpo* proyectado sobre la imagen, la salud se corresponde a la medía de pixeles correspondientes con el rectángulo verde 4.13 donde se ha detectado movimiento. De esta manera, resolvíamos el problema de las resoluciones, ya que al estar manejando distancias en el rectángulo y no píxeles, conseguimos que cuanto menor es la resolución menos píxeles ocupa el rectángulo y cuanto mayor es la resolución más pixeles ocupa éste. Esto lo conseguimos, con el uso de la rectificación de la imagen, que dado un punto en la carretera podemos conocer su correspondiente píxel en la imágen. Sin embargo, seguíamos teniendo el problema de que no se podía saber con exactitud la velocidad del vehículo, ya que había varios individuos de la raza con la máxima salud.

Además surgió otro problema, relacionado con la altura de los coches: supongamos vehículos salientes cuando inicialmente pasan por zonas cercanas a la cámara, se puede ver toda la longitud del vehículo. Es decir, la altura del vehículo apenas influye. Según avanza el vehículo, vamos viendo el coche más pequeño en lo que al espacio 2D se refiere. Sin embargo, como en el mundo real el coche tiene volumen, cuanto más se aleja el coche de la cámara, podemos observar que su altura hace la longitud visible

del coche mayor y siendo éste más largo en nuestra carretera rectificada. Para entender mejor dicho problema vamos a fijarnos en la figura 4.13. En ambos casos la velocidad hipotetizada del individuo (representado por el rectángulo verde) es mayor a la real. En la figura 4.13(a), el coche representado es casi plano. Al calcular su salud obtendríamos una salud baja, ya que en parte del rectángulo verde no se detecta movimiento, siendo el individuo correctamente eliminado. Sin embargo, en un coche que no es plano (figura 4.13(b)), la salud obtenida es alta y por lo tanto el individuo no es eliminado. Esta implementación de la función salud era poco discriminante para lugares en la imagen con poca resolución.



(a) (b)

Figura 4.13: Problema de la altura en la segunda implementación de la función salud

4.5. Etapa de Detección

El objetivo principal en esta etapa es detectar a los vehículos nuevos que aparecen en la imagen, quedando cada vehículo definido por una única raza, que contiene todas las velocidades posibles del vehículo. En la etapa de detección se distribuye la población de individuos de exploradores por la zona inicial de la carretera en las zonas donde se ha detectado movimiento.

Así en cada iteración a cada individuo se le asigna un peso o salud h_i, correspondiente al movimiento detectado, aquel cuya salud supere un umbral determinado se considera un individuo prometedor y éste provoca la creación de una nueva *raza*, siempre y cuando no exista una raza cercana a esa posición (para que no se cree más de una raza por vehículo). Una raza está formada por N individuos. Estos individuos inicialmente se sitúan en la misma posición en la carretera rectificada, y por lo tanto en el mismo píxel (al usar la funcionalidad proporcionada por la rectificación de imágenes). La diferencia entre estos individuos es que cada uno posee una velocidad diferente dentro de la raza.

La solución adoptada estudia si existe movimiento en unos determinados píxeles de la zona inicial de la carretera basándose en las técnicas de *rejilla* o *muestreo*. Para determinar si se ha detectado movimiento en estos píxeles usamos una técnica que se basa en la diferencia entre la imagen actual y una imagen de fondo aprendida. Si la diferencia supera cierto umbral se ha detectado entonces movimiento . Si se detecta movimiento ese píxel pasa a convertirse en un individuo explorador. El individuo explorador se encarga entonces de calcular la salud asociada para su posición x,y en la carretera rectificada. Si este individuo obtiene una buena salud, es un individuo prometedor que crea una raza siempre y cuando no exista otra en una posición cercana.

La funcionalidad para detectar movimiento mediante una diferencia de imágenes está implementada en el esquema *carSpeed*.

Como acabamos de comentar la técnica usada en la solución final *diferencia entre el fotograma actual y una imagen de fondo aprendida*. Para el aprendizaje del fondo de la carretera se toma una imagen como muestra inicial y sobre ella se van sumando fotogramas cada ciertas iteraciones de manera ponderada, así de forma acumulativa se obtiene una imagen de fondo. La diferencia entre la imagen actual y la de fondo aprendido se genera restando píxel a píxel. En la figura 4.15 mostramos el resultado de hacer la resta entre fotogramas, donde vemos que la zona de detección se corresponde con la zona negra al inicio de la carretera y dentro de ella los píxeles en blanco es donde se ha detectado movimiento.

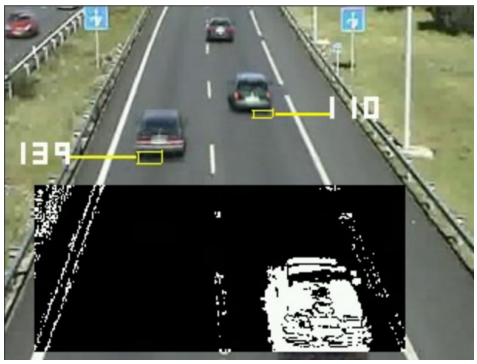


Figura 4.14: Diferencia entre el fotograma actual y una imagen de fondo aprendida

Una vez explicada la técnica de detección de movimiento vamos a comentar cómo se distribuye la población de exploradores en la zona inicial de la carretera. Debido a que esta técnica localiza una gran cantidad de píxeles donde se ha detectado movimiento, el cálculo de salud en todos estos píxeles supone un alto coste computacional.

Por este motivo, la diferencia entre imágenes con esta técnica se hace en unos píxeles determinados. Las dos formas para distribuir a población de exploradores son las siguientes.

- *Rejilla.* Se establece una rejilla en la zona inicial de la calzada donde se realizará la diferencia de imágenes, para comprobar la existencia de movimiento. Esta rejilla calcular la salud para un píxel, cada N columnas y filas. Siendo N una constante, que puede ser modificada. Generalmente las pruebas realizas N tomaba el valor de cinco. En la siguiente imagen podemos ver dicha rejilla.

Figura 4.15: Distribución de la población de exploradores en rejilla

- *Muestreo.* En esta ocasión es en cada N píxeles en la zona inicial donde se calcula su salud asociada. Siendo N una constante, que en las pruebas realizadas toma el valor de 25, es decir cada 25 píxeles del espacio de la zona inicial se realiza una diferencia entre píxeles.

4.5.1. Implementaciones alternativas para la detección de movimiento

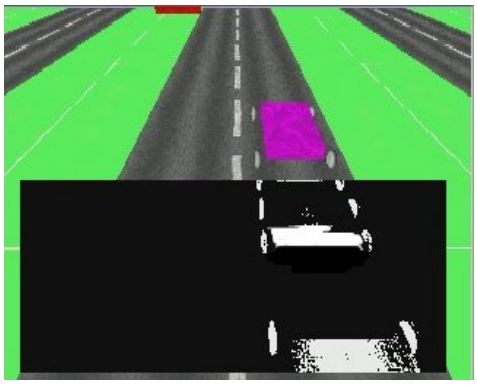
Además de la diferencia de imágenes entre la imagen actual y una imagen de fondo aprendida, se han desarrollado otras implementaciones para la detección de movimiento. En concreto las técnicas implementadas, se basan en la diferencia de dos imágenes consecutivas y flujo óptico.

Filtrado de movimiento utilizando dos fotogramas consecutivos

La técnica para detectar movimiento consiste en restar a cada píxel de la imagen anterior su valor anterior de RGB de la imagen anterior. Para ello es necesario almace-

nar la imagen anterior de forma que se pueda contrastar cada píxel en el instante t con el mismo píxel en el instante *t-1*. Se establece una tolerancia de movimiento y aquellos píxeles cuya diferencia supere esa tolerancia se considera que han sufrido variación debido a un movimiento.

En este caso, tenemos el problema que al restar dos fotogramas consecutivos, sólo se obtiene movimiento en la parte final e inicial del vehículo, como podemos ver en la figura 4.16(a). Los píxeles de la parte trasera tendrán el *cuerpo* fuera del vehículo y los que están en la parte delantera tendrán la falda dentro del vehículo. Por lo que la función salud de cada individuo no obtiene un buen valor y no existe ningún individuo prometedor para crear una raza.



(a) Filtrado de movimiento utilizando dos fotogramas consecutivos

(b) Píxeles donde se calcula la salud para resta entre fotogramas consecutivos

Figura 4.16: Filtrado de movimiento utilizando fotogramas consecutivos

Los píxeles donde se aplica la técnica de detección de movimiento son los correspondientes a la distribución de la rejilla o muestreo. Es en estos píxeles donde se realiza la diferencia entre la imagen actual y anterior. Si la diferencia supera un umbral determinado, a partir de ese píxel distribuiremos la población de exploradores para calcular la salud en cada un de ellos. Observando la figura 4.16(b) vemos como se distribuye la población de exploradores a partir de un píxel donde se detecto movimiento. La cruz roja es donde se ha detectado movimiento y los píxeles en azul la población de exploradores. Los individuos azules que se encuentran, en la carretera rectificada, en una posición más lejana a la de la cámara, nos permiten crear una raza si la cruz roja se encuentra en la parte trasera del vehículo. Por otro lado, si esta cruz roja se encuentra en la parte delantera del vehículo son los píxeles azueles con una posición más cercana

a la de la cámara, los que nos permiten crear una raza.

Flujo óptico

La funcionalidad de flujo óptico para poder detectar movimiento en la imagen está implementada en el esquema *opflow*. Este esquema ha sido reutilizado del proyecto fin de carrera de José Antonio Santos [Santos, 2007].

El esquema *opflow* busca en la imagen de entrada puntos relevantes sobre los que calcular el flujo óptico. Estos puntos serán los correspondientes con bordes y esquinas. Como el flujo óptico es una operación muy costosa, es necesario ganar eficiencia. Para ello usa la multirresolución, que es una técnica que se basa en el cálculo recursivo del flujo óptico en imágenes con menor resolución pudiendo así utilizar ventanas de búsqueda más pequeñas y ahorrar cómputo. El resultado de calcular el flujo óptico se traslada a la siguiente capa, en la que se procederá a al cálculo del flujo óptico centrando la ventana de búsqueda en el resultado de la capa inmediatamente superior (imagen 4.17). Para poder usar la multirresolución en los programas, se utilizan estructuras llamadas pirámides, éstas guardan la imagen en diferentes resoluciones y permiten así que el cálculo del flujo óptico se realice de forma escalonada en todas las capas (resoluciones). Una vez obtenida las pirámides se implementa un algoritmo iterativo, que busca la correspondencia entre dos imágenes consecutivas.

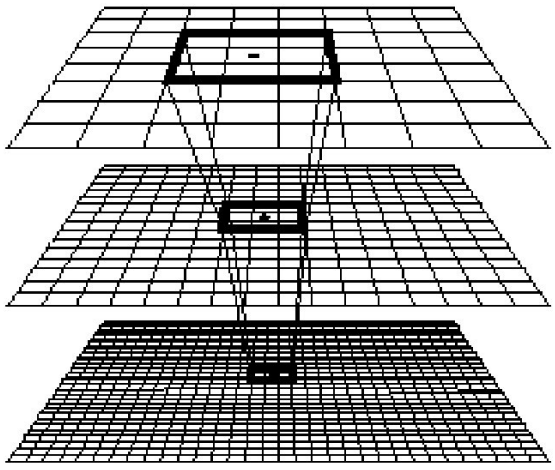
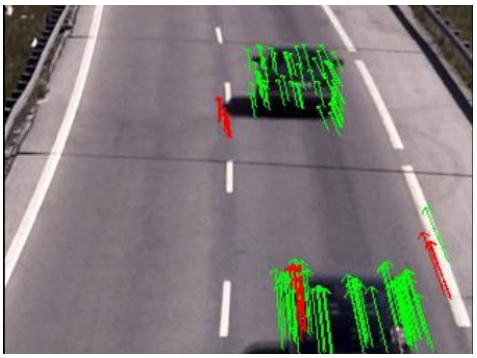


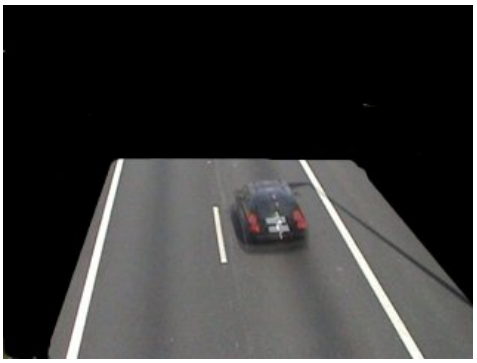
Figura 4.17: Cálculo del flujo óptico con multirresolución

Opflow ofrece al esquema *carSpeed*, la variable *image_opf* y *carSpeed* hace lo mismo con la variable *mask* al esquema *opflow*. La variable *image_opf* nos proporciona vectores con sentido hacia donde se ha producido el movimiento (figura 4.18(a)). La

variable *mask* sirve para reducir el cálculo de flujo óptico a determinadas zonas de la imagen (figura 4.18(b)). Estas zonas son introducidas por el usuario a través de una máscara, que no es más que una imagen de tres bytes puestos al valor máximo, si se quiere calcular el flujo óptico en ese píxel, o al valor mínimo, si no se quiere calcular en él flujo óptico. Con el uso de esta variable, se agiliza el cómputo de CPU, ya que el flujo óptico no es calculado en toda la imagen. En la figura 4.18(b) no se calculó el flujo óptico en las zonas negras de la imagen.



(a) Detección de movimiento con flujo óptico



(b) Mascara para calcular flujo óptico

Figura 4.18: Calculo de flujo óptico

Los datos obtenidos por el esquema *opflow* no son suficientes, ya que los puntos de interés que nos da el flujo óptico son los bordes y las esquinas. Esto se debe a que la implementación de la función de salud necesita que el cuerpo se encuentre sobre el vehículo y la falda sobre la carretera. Así pues a partir de los píxeles donde el flujo óptico ha detectado movimiento, se han desarrollado dos técnicas diferentes para distribuir la población de exploradores. En la población de exploradores se calculara la salud y si estamos ante un individuo prometedor se crea una raza. En esta raza se hipotetizan todas las posibles velocidades para el vehículo, estando la posición de cada individuo definida en la carretera rectificada. A continuación comentamos, las dos técnicas para distribuir la población de exploradores.

- **Roseta de flujo óptico**. Para poder explicar bien esta técnica hay que fijarse en la figura 4.19. La cruz de color rojo, es el pixel donde *opflow* ha detectado movimiento, a partir de este píxel se generan distintas hipótesis. Estas hipótesis se corresponden con los seis píxeles de color azul en la figura, que tienen su correspondiente posición en la carretera rectificada. Estos seís pixeles son las esquinas

y los puntos superior e inferior de un rectángulo imaginario. El rectángulo tiene un tamaño definido fijo, siendo éste pequeño para poder detectar vehículos como motos.

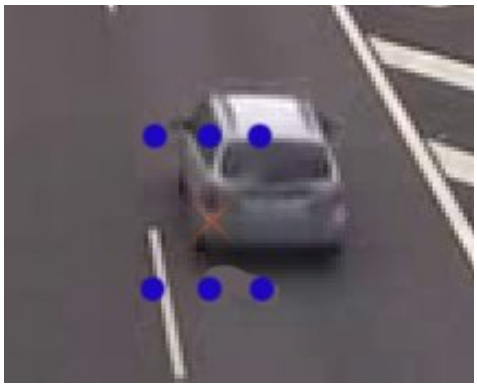


Figura 4.19: Roseta de flujo óptico

- **Ruido térmico a partir de flujo óptico**. En este caso a partir de la hipótesis inicial donde se ha detectado movimiento, se aplica el operador genético de ruido térmico en la carretera rectificada, que genera un explorador cercano a este y proyectando este individuo sobre la imagen se calcula su salud.

4.6. Etapa de seguimiento

En esta etapa se pretende que los N individuos de cada raza evolucionen en el tiempo para seguir al vehículo, a la vez que se visualiza la velocidad del individuo con mejor salud en cada iteración y la velocidad final de la raza, que es la velocidad real estimada por el sistema para cada vehículo. Esta etapa comienza para cada vehículo cuando es detectado en la zona inicial de la carretera, y todos los individuos de la raza se encuentran en la misma posición pero con diferentes hipótesis de velocidad. Es a partir de este momento cuando comienza el seguimiento. Poco a poco se van descartando individuos, quedando al final de la etapa de seguimiento muy pocos, de esta manera podemos realizar una estimación más exacta de la velocidad a la que circula el vehículo. Para ello como se muestra en la figura 4.4 se actualiza la posición de cada individuo de la raza en cada iteración, calcular la salud de cada individuo en su nueva posición, ordenar la posición por salud y evaluar si debe o no ser eliminado de la raza. A continuación veremos cada una de estas partes más detalladamente:

- *Actualización de la posición.* Desde el momento en que el vehículo fue detectado se va actualizando la posición de todos los individuos de la raza en cada iteración. Como el seguimiento de los vehículos se realiza en un corto espacio, entre unos

40 y 50 metros, asumimos que la velocidad de los vehículos es constante. Al ser un tramo de carretera tan corto, no da tiempo a que la velocidad de los coches cambie drásticamente. Así, al conocer la posición de todos los individuos de la población, su velocidad y el tiempo transcurrido por iteración (medida con la función *gettimeofday*), es fácil calcular el desplazamiento a lo largo de la carretera de los vehículos. La nueva posición viene dada por la ecuación 4.4, siendo x_0 la posición inicial en el eje x (largo de la carretera). Con esta ecuación los individuos de la raza se van dispersando espacialmente según sea su velocidad.

$$x(t) = x_{t-1} + V * \Delta t \tag{4.4}$$

- *Cálculo de salud y ordenación de los individuos de una raza por salud.* Una vez que se ha actualizado la posición de toda la población existente de una raza, calculamos de nuevo la salud asociada a cada individuo. Esta salud mide cuan compatible es cada velocidad hipotetizada con las imágenes. Después de esto ordenamos los individuos existentes por salud.

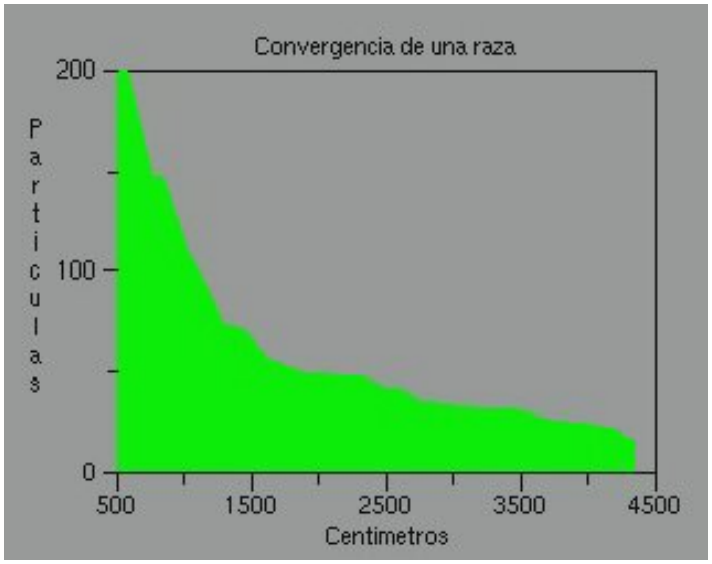


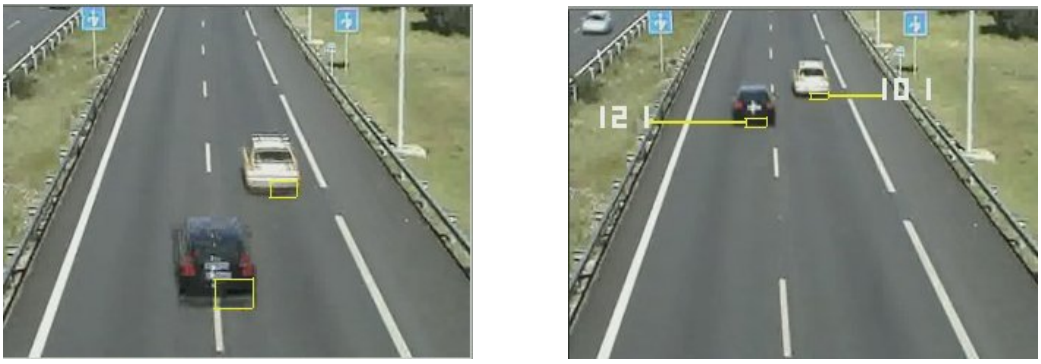
Figura 4.20: Convergencia de la población de una raza

- *Evaluación.* En este punto se eliminan todos aquellos individuos pertenecientes a una raza que no superen un cierto umbral de salud. De esta forma conseguimos que la población de una raza vaya disminuyendo, sobreviviendo sólo los individuos cuya velocidad esté más próxima a la real. En la figura 4.20, podemos ver una gráfica de cómo converge la población de una raza. Vemos cómo rápidamente el algoritmo elimina en los 10 primeros metros, las velocidades que no están cerca de

la velocidad que lleva el coche. A partir del metro 15 y según avanza el vehículo en la carretera se van eliminando individuos de una forma más regular. Así al final de la etapa de seguimiento sobreviven muy pocos individuos, todos ellos con velocidades similares, siendo la velocidad del individuo que tenga mayor salud media la velocidad estimada por el sistema para ese vehículo.

4.6.1. Desplazamientos laterales

Como acabamos de comentar, la evolución de los individuos en el tiempo se basa en la ecuación (4.4). Esta fórmula nos limita el seguimiento de los vehículos únicamente en una sóla dimensión en concreto a lo largo de la carretera (eje x). Para realizar también el seguimiento en el ancho de la carretera y seguir correctamente vehículos que cambian de carril hemos añadido una funcionalidad en la fase de la actualización de la posición. En cada iteración comprobamos que el ancho del cuerpo del individuo líder de cada raza se encuentre sobre el vehículo en la imagen. Es decir, comprobamos que el individuo líder no pierda salud por los laterales del cuerpo. Si es así, y alguno de los laterales del cuerpo no se encuentra sobre el vehículo, se modifica la posición de todos los individuos de la raza a lo largo del eje y, hasta que el ancho del cuerpo se sitúe sobre vehículo en la imagen. De esta manera podemos reubicar horizontalmente a los individuos de cada raza, cuando un vehículo realice un cambio de carril. En las figuras 4.21 podemos ver cómo un vehículo es seguido, y por lo tanto el algoritmo actualiza bien su posición al cambiarse de carril.



(a) (b)

Figura 4.21: Actualización de posición en cambio de carril

4.7. Interfaz gráfica de la aplicación

La interfaz gráfica es un elemento importante para la aplicación ya que ha permitido la depuración de errores del algoritmo y se utiliza para la visualización de los resultados del seguimiento, estimación de la velocidad y la cuenta de los coches. La interfaz del esquema *carSpeed* se presenta en la figura 4.22, distinguiendo ocho bloques gráficos que se explican a continuación.

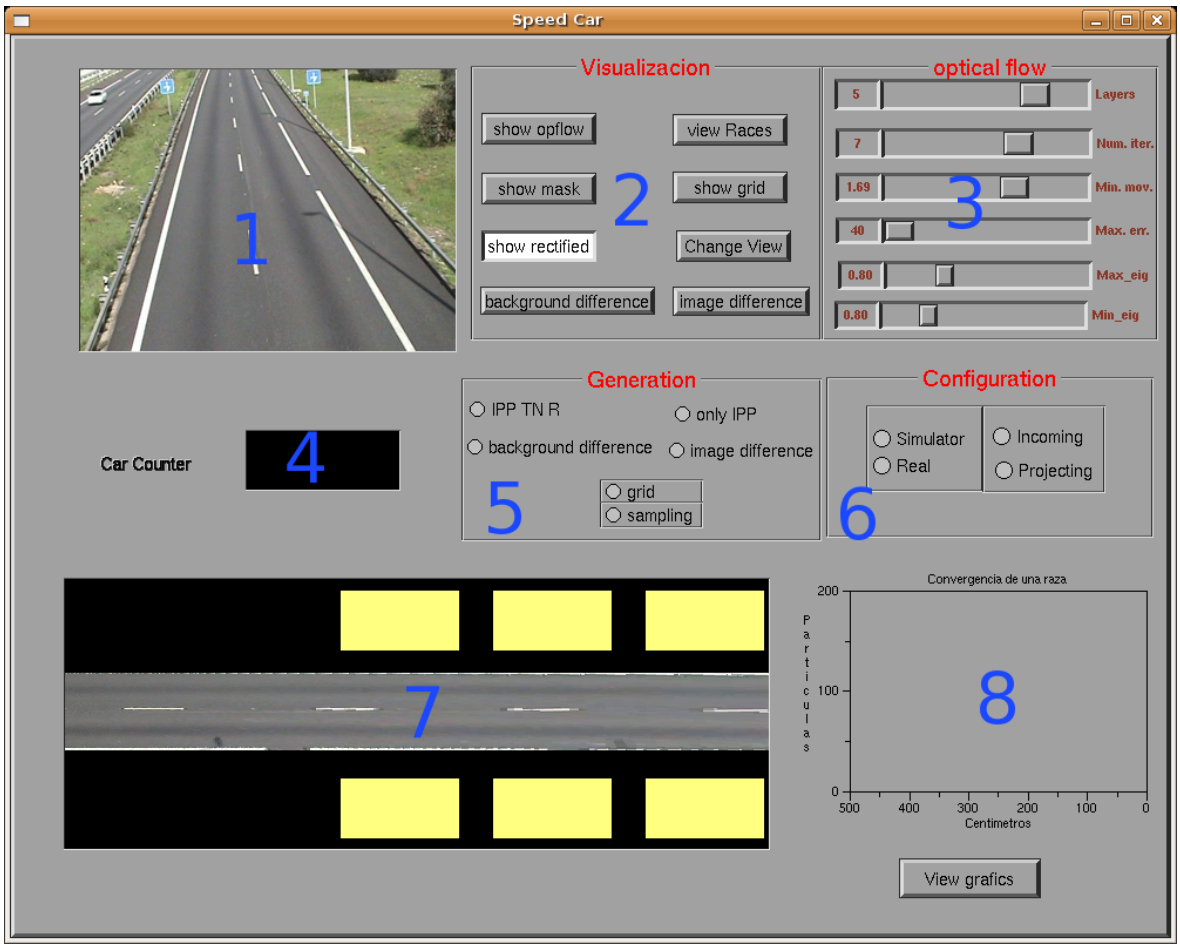


Figura 4.22: Interfaz gráfica del esquema carSpeed

1. *Imágenes de entrada.* En este bloque se visualizan las imágenes de entrada, ya sean simuladas o reales, la visualización de la velocidad del individuo con mayor salud en cada iteración de cada uno de los vehículos o los píxeles en la zona de detección donde se produce movimiento.

2. *Visualización* En este bloque se eligen las opciónes de visualización. Los botones *show oflow y show mask*, siempre y cuando el esquema *opflow* esté activo, muestran respectivamente el flujo óptico calculado en la imagen y las zonas donde éste es calculado. El botón *show rectified* hace que se muestre en el bloque 7 la

rectificación de la imagen. El botón *view Races* hace que se muestre la evaluación de los coches en la carretera y la velocidad estimada por la aplicación, por defecto muestra estos datos en la imagen rectificada (bloque 7). *Show grid* hace que se muestre la rejilla de la zona de detección en la imagen de entrada. *Change view* cambia la visualización del seguimiento y la estimación de la velocidad a la imagen real (bloque 1). Los botones *background difference* e *image difference* muestran la resta de la imagen real con la imagen de fondo aprendida y la diferencia de la imagen actual con la imagen anterior.

3. *Configuración del flujo óptico. Layers* modifica el número de capas de las pirámides que se generan para calcular el flujo óptico. *Num. iter.* permite modificar el número de iteraciones que se emplearán en cada capa para encontrar la similitud. *Min. mov*, cambia el valor del módulo a partir del cual se considera un movimiento suficiente para detectar movimiento. *Max. err.* modifica el valor máximo de error permitido.*Max. eig. y min. eig.*, indican los valores máximos y mínimos de los autovalores para buscar los puntos de interés.

4. *Car counter*, visualiza la cuenta que se lleva de los vehículos que pasan por la carretera.

5. El bloque *Generation* se corresponde con la etapa de generación, y nos da la opcion de elegir la técnica con la que realizar la detección, y en el caso de las técnicas que llevan consigo diferencia de imágenes, permite elegir la distribución de la población de exploradores, rejilla o muestreo.

6. En el bloque *Configuración*, decimos a la aplicación si las imágenes son reales o simuladas, y si los vehículos llevan sentido entrante o saliente.

7. *Carretera rectificada*, en este bloque podemos visualizar los coches seguidos por la carretera con su velocidad estimada. En cualquier caso, ya se visualicen el seguimiento en la imagen real o en la carretera rectificada, siempre que un coche finaliza su etapa de seguimiento, se muestra la velocidad estimada de cada coche en los marcadores amarillos.

8. *Convergencia de una raza*, aquí se va mostrando cómo converge la población de una determinada raza.

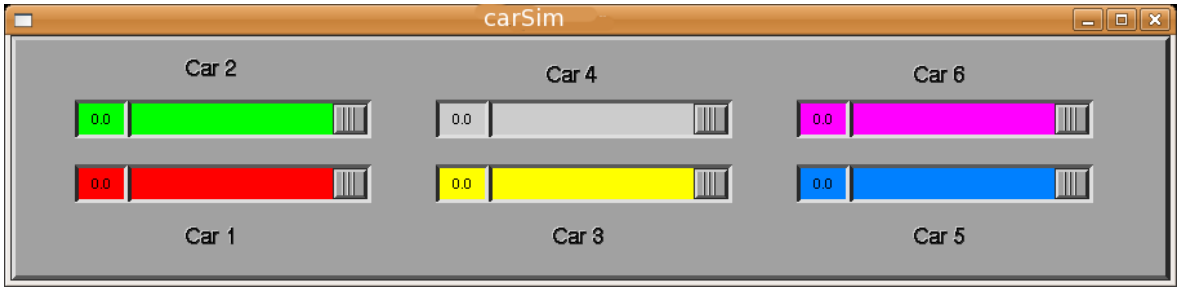


Figura 4.23: Interfaz gráfica del esquema carSim

En segundo lugar, podemos ver la interfaz del esquema *carSim* en la figura 4.23, a través de la cual podemos gobernar el movimiento de hasta 6 coches simulados. Como podemos ver la interfaz del esquema *carSim* está compuesta por seis deslizadores. Cada deslizador sirve para dar velocidad a cada coche simulado por gazebo. Además cada slider tiene un color idéntico al coche correspondiente simulado por gazebo.

Capítulo 5
Experimentos

En el capítulo anterior hemos descrito la solución final que hemos desarrollado. En este capítulo explicaremos qué pruebas, experimentos y observaciones hemos realizado para validar y mejorar la implementación en el camino hacia esta solución, según distintos prototipos desarrollados iterativamente, como vimos en el capítulo 2.

Mediante los experimentos se intenta analizar el comportamiento del algoritmo y obtener los parámetros óptimos para realizar un seguimiento vivaz y una estimación de la velocidad lo más exacta a la real. Comenzaremos mostrando el entorno de pruebas de los experimentos. A contiunación mostramos la ejecución típica de la solución final, para después hacer un análisis de errores del sistema. Seguidamente haremos un estudio sobre la discriminación de velocidades de la aplicación. Por último veremos todas las alternativas probadas antes de llegar a la solución final.

5.1. Entorno de pruebas

Las pruebas realizadas para probar y ajustar el algoritmo se han desarrollado en dos entornos. Por un lado, un entorno ha sido simulado usando el simulador *gazebo*. Por otro lado, se han realizado pruebas con vídeos reales para probar el funcionamiento del algoritmo en un entorno real.

El material *hardware* usado, consiste en una cámara de vídeo y un ordenador. Si bien es cierto que el algoritmo ha sido probado en dos ordenadores con características diferentes. En ambos el algoritmo ha funcionado correctamente. Las características del primer ordenador son las siguientes: 4 núcleos a 2.4 Ghz, 3 GB de memoria RAM y sistema operativo GNU/Linux. Las características del 2 ordenador son las siguientes: 1 núcleo a 3.2 Ghz, 1GB de memoria RAM y sistema operativo GNU/Linux. La cámara de vídeo la hemos usado, para grabar vídeos reales que luego usamos como entrada en la aplicación, ya que es mucho más cómodo que montar un dispositivo que nos sirviera

las imágenes directamente desde la cámara de vídeo colocada en un puente sobre una carretera al ordenador.

5.2. Ejecución típica

A continuacion mostramos los experimentos realizados en un entrono real, una vez que hemos depurado y probado el algoritmo en un entorno simulado,

Para llevar a cabo las pruebas en un entorno real, nuestro sistema necesita una correcta instalación para un buen funcionamiento. Dicha instalación es simple. El sistema necesita saber cuatro puntos de entrada (píxeles) en la imagen original y cuatro puntos de salida en la imagen rectificada para poder calcular la matriz de homografía que nos permite conocer la relación entre la imagen y el espacio métrico de la carretera. En la figura 5.1 mostramos los cuatro puntos de entrada del sistema. Los puntos de salida se corresponden con la longitud a lo largo y ancho de la carretera en ese tramo. Una vez que el sistema conoce estos puntos ya esta preparado para llevar a cabo su funcionalidad.

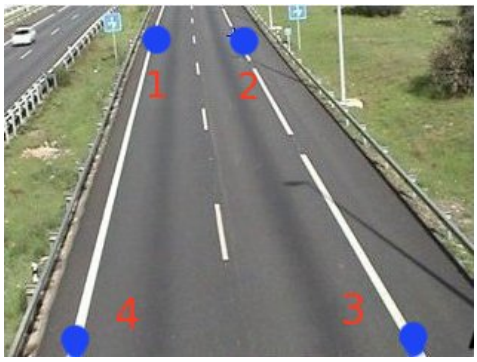
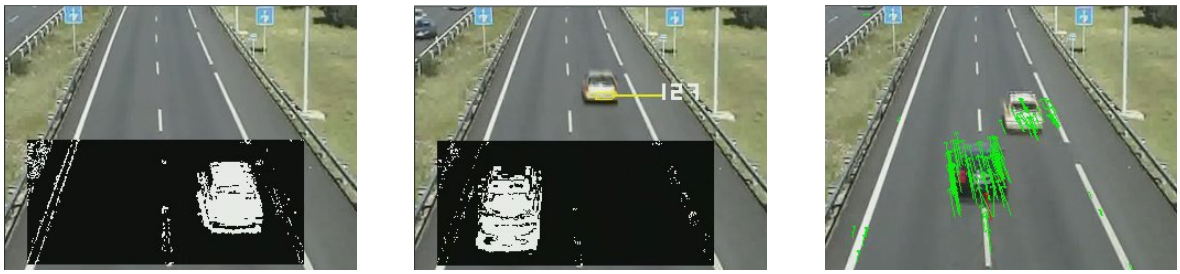


Figura 5.1: Puntos de entrada necesarios para el sistema

Inicialmente la aplicación necesita detectar los vehículos en la zona inicial de la calzada. Si para ello usamos las técnicas basadas en resta de imágenes sólo hace falta tener activado el esquema principal de la aplicación *carSpeed*, si por el contrario nos basamos en flujo óptico, tendrá que estar activo el esquema *opflow*. En la figura 5.2, podemos ver las diferentes técnicas de detección de movimiento aplicadas para coches salientes. En concreto en la figura 5.2(a) se realiza una diferencia de imágenes entre la imagen actual y una de fondo aprendida, en esta figura podemos ver cómo se detecta movimiento en las líneas que delimitan la carretera y en las líneas centrales, esto se debe al ruido producido por los cambios de luminosidad. En la figura 5.2(b) se realiza la

diferencia de imágenes entre la imagen actual y la anterior. En la figura 5.2(c), tenemos un ejemplo de detección de movimiento con flujo óptico. De estas tres técnicas la de flujo óptico lleva con sigo un gran coste computacional, a pesar de utilizar la máscara para agilizar el cálculo de flujo óptico, por lo que depende de la potencia del computador donde se ejecute para realizar un seguimiento y estimación de la velocidad correctos. Con la técnica de diferencia entre imágenes consecutivas, como vimos en el capítulo 4, por cada píxel donde se detecta movimiento, se distribuyen cuatro exploradores. Sin embargo, la técnica entre la imagen actual y una de fondo aprendida surge un sólo explorador por cada píxel que se detecta movimiento por lo que tiene un menor coste computacional. Las tres técnicas cumplen adecuadamente el objetivo de esta fase, que es generar 1 raza por cada vehículo.



(a) (b) (c)

Figura 5.2: Detección de vehículos en coches salientes

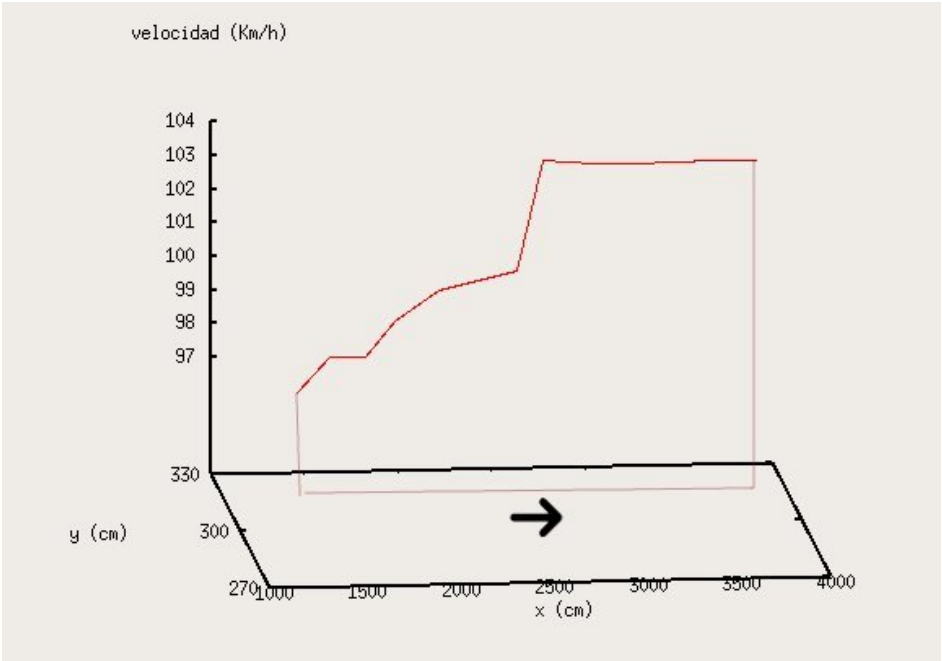
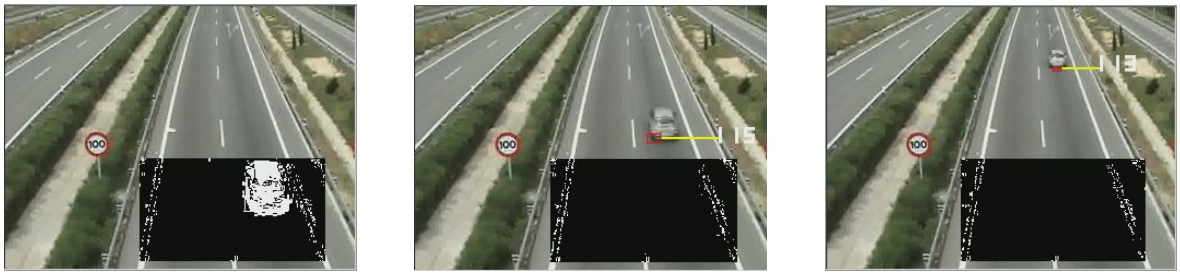


Figura 5.3: Seguimiento de un vehículo en el espacio *x,y,velocidad*

Una vez detectado los vehículos en la zona inicial, pasamos a la etapa de seguimiento donde se llevará acabo la estimación de la velocidad en cada iteración. En la figura 5.3 podemos ver el seguimiento de un vehículo en nuestro espacio 3d. Las tres dimensiones vienen dadas, por la posición del vehículo en la carretera viéndose esta como un espacio bidimensional (x, y) y la velocidad de dicho vehículo. En la gráfica la posición del vehículo (x, y) esta representada por los ejes x,y , la velocidad se corresponde con el eje z. Aproximadamente en el metro 12, es cuando el vehículo es detectado y comienza la etapa de seguimiento. Desde el momento en que comienza la etapa de seguimiento hasta el metro 25, la velocidad estimada para el vehículo no es constante. En este tramo los vehículos con menor y mayor velocidad a la velocidad real van perdiendo salud, y algunos llegan a ser eliminados, ya que dichos individuos se van quedando detrás y delante del vehículo original. A partir del metro 25 , la velocidad se estabiliza siendo esta la velocidad final estimada por el algoritmo. Observando la gráfica, podemos decir que el algoritmo tarde entre unos 10 y 15 metros en estabilizar la velocidad del vehículo.

En las figuras 5.4 y 5.6 podemos ver un ejemplo de ambas etapas en un vehículo saliente. En la primera imagen vemos la detección del coche y en las otras el seguimiento.



(a) Detección (b) Seguimiento inicial (c) Estabilización de la velocidad

Figura 5.4: Seguimiento de vehículo saliente

Hasta ahora hemos comentado la solución aplicada para coches salientes. Además dadas las características del algoritmo y su fácil instalación se ha desarrollado una implementación para coches entrantes. Los únicos cambios con respecto a los coches entrantes, es cambiar la etapa de detección, seguimiento y los puntos de salida en la instalación que definen el largo y el ancho de la carretera. La etapa de detección es la zona más alejada de la cámara y la etapa de seguimiento es la más cercana a la cámara. El cambio de los puntos de salida en la instalación, se realiza para que el coche avance en el eje *x positivo* (largo de la carretera). Además existe otro cambio, pero no

con respecto a la implementación si no con respecto a la hora de entender la función salud. En este caso, el objetivo de la función salud para que no afecte la altura de los vehículos, es que los individuos con mejor salud se encuentren en la zona delantera del vehículo.

En la figura 5.5 podemos ver las mismas técnicas para detectar movimiento usadas para coches salientes, pero esta vez aplicadas para coches entrante. En este caso podemos ver como la zona de detección ha cambiado, encontrándonos ahora en la zona inicial de la carretera. Además esta zona de detección es menor que para coches salientes ya que la zona de detección esta definida en la carretera rectificada y al hacer la proyección sobre la imagen existen menos píxeles por que es una zona de menor resolución. Esto implica que el coste computacional en la zona de detección sea menor, por que se detecta movimiento en menos píxeles. Si nos fijamos en la figura 5.5(c) vemos que apenas se ha detectado movimiento con flujo óptico, como resultado de esto el flujo óptico no funciona bien para coches entrantes ya que hay ocasiones donde no se detecta un nuevo vehículo. Las técnicas que utilizan diferencia de imágenes (figura 5.5(a) y 5.5(b)), tienen un funcionamiento correcto siendo algo menor el coste computacional para la diferencia de imagenes entre la actual y la de fondo aprendida, por el motivo que comentamos para coches salientes. El seguimiento para coches entrantes cambia la zona de seguimiento y el sentido de los vehículos, motivo por el que se cambiaron los cuatro puntos de salida en la instalación del sistema.En la figura 5.6 mostramos un casa particular de un coche entrante desde su detección hasta el fin de la etapa de seguimiento. En la figura 5.17(a) vemos la detección del vehículo y en las figuras 5.17(b) y 5.6(c) vemos el seguimiento de dicho vehículo.

(a) (b) (c)

Figura 5.5: Detección de vehículos en coches entrantes

(a) (b) (c)

Figura 5.6: Seguimiento de vehículo entrante

Otro de los objetivos de este proyecto fin de carrera es el de llevar la cuenta de los vehículos de los que el sistema estima su velocidad. En la figura **??** vemos el contador de la aplicación, marcando el número total de coches de los que ha estimado su velocidad, junto con una imagen real donde se estiman las velocidades de los coches seguidos.

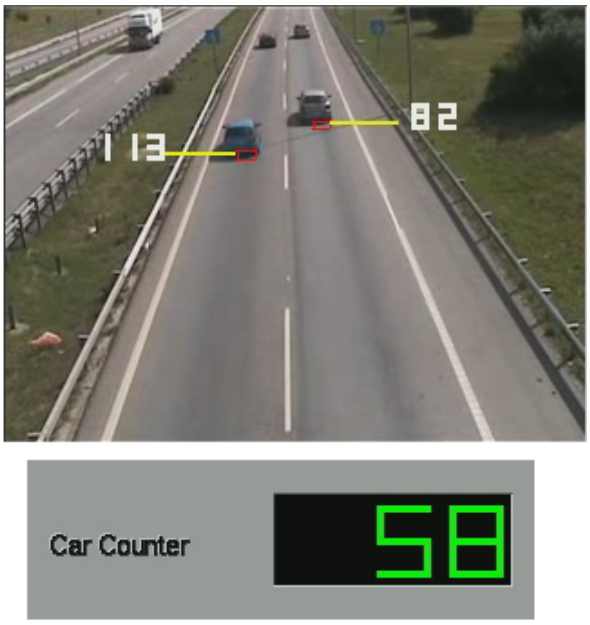


Figura 5.7: Contador de vehículos

5.3. Discriminación entre velocidades

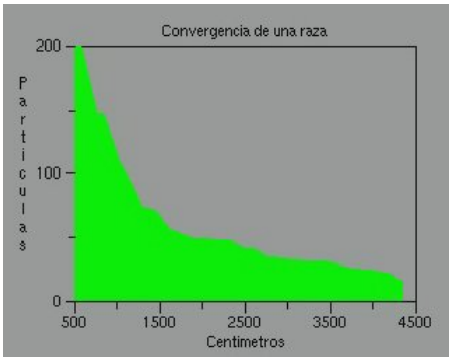
En esta sección vamos a realizar un estudio sobre los individuos de una raza a lo largo de la etapa de seguimiento. El objetivo es ver cómo evolucionan las razas viendo la convergencia de cada raza desde el momento que el vehículo es detectado y durante toda la etapa de seguimiento.

Las gráficas 5.8 de la interfaz nos dan una importante intuición de cómo converge la

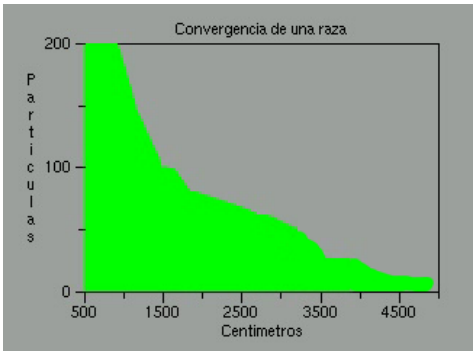
población de una raza. En estas gráficas en el eje x se muestra la posición del vehículo y en el eje y el número de individuos de la raza. En la figura 5.8(a) para coches salientes, podemos ver cómo en los primeros 10 metros (antes del metro 15) se eliminan entorno a 150 individuos. A partir de este momento se eliminan individuos cada vez más lentamente, ya que los individuos existentes tienen velocidades muy parecidas.

En la grafica 5.8(b) vemos la misma gráfica pero esta vez para coches entrantes. En ésta se eliminan unos 100 individuos en los primeros 10 metros. A partir de hay se eliminan individuos a una menor velocidad pero de una manera más o menos constante, quedando al final de la etapa de seguimiento muy pocos individuos que son los que compiten por la estimación de velocidad final del sistema para ese vehículo.

Si comparamos las dos gráficas en la figura 5.8(a) podemos ver cómo en los primeros 10 metros la población converge más deprisa que en la figura 5.8(b). Esto se debe a que para coches salientes la resolución es mayor que para coches entrantes en la zona inicial de seguimiento y como consecuencia de esto se discrimina un mayor número de coches. Una vez rebasada la zona inicial del seguimiento por el vehículo, los vehículos salientes van perdiendo resolución en la imagen según se alejan de la cámara por lo que es más difícil discriminar individuos. Sin embargo para los coches entrantes, cada vez se van acercando más a la cámara por que van ganando en resolución y es más fácil discriminar individuos. Si nos fijamos en la parte final de la etapa de seguimiento vemos como la población para coches entrantes es menor que para coches salientes. De esta manera el sistema da una estimación más precisa para coches entrantes que salientes.



(a) Convergencia para vehículos salientes



(b) Convergencia para vehículos entrantes

Figura 5.8: Convergencia de la población par vehículos entrantes y salientes

Una vez que hemos visto como convergen los individuos de una raza, vamos a ver unas comparativas sobre los individuos que son eliminados al principio, donde la raza converge a mayor velocidad, y cuales son eliminados en la parte final de la etapa de seguimiento. Primero veremos como se comportan los individuos que tienen una velocidad muy distinta, que coinciden con los individuos que son eliminados primero, y después los que se eliminan en la etapa final del seguimiento.

En la figura 5.9 mostramos una gráfica para estudiar el comportamiento de velocidades que difieren entre si 20 *km/h*. En la gráfica el eje x se corresponde con la posición del individuo a lo largo de la carretera, en el eje y vemos la salud asociada a cada individuo para su correspondiente posición y velocidad. Inicialmente todos los individuos se encuentran en la misma posición y por tanto con la misma salud. Antes del metro 15 los individuos con velocidades más bajas y más altas son eliminados. Los individuos con velocidades más bajas se quedan atrás enseguida y los individuos con velocidades más altas van por delante del vehículo. A partir del metro 15 sólo observamos tres individuos cuyas velocidades están en un rango de 10 km/h. El individuo con velocidad de 100 km/h es eliminado aproximadamente en el metro 27, ya que su velocidad es un poco más lenta que la real, quedando hasta el final únicamente dos individuos con velocidades de 104 y 110 km/h.

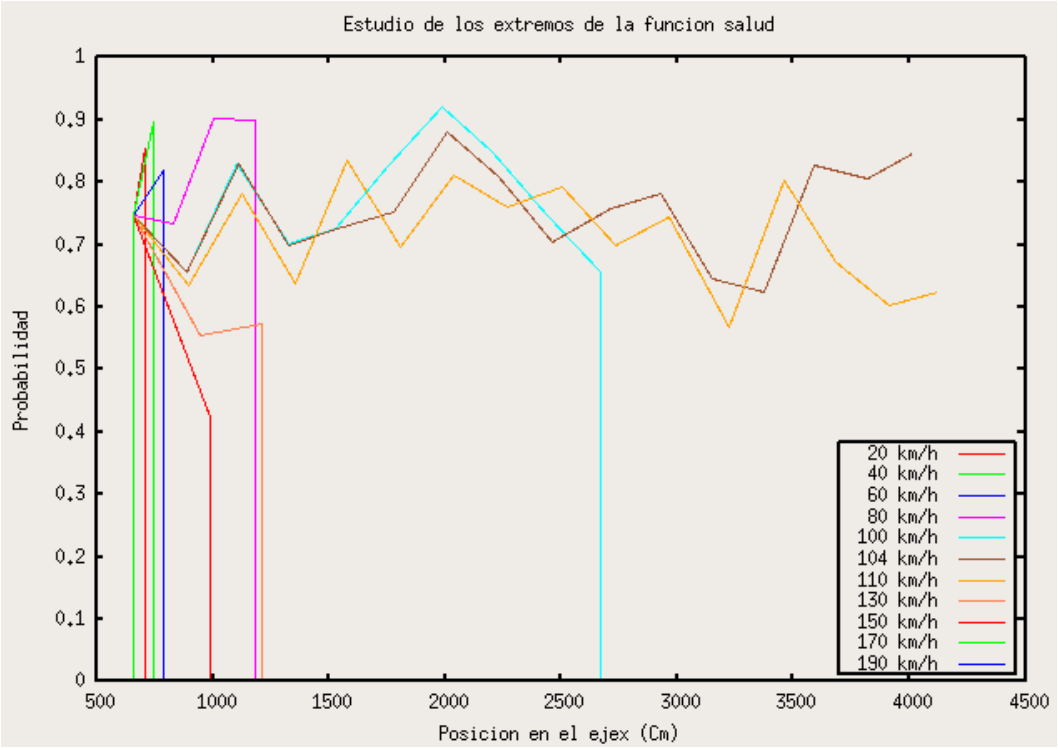


Figura 5.9: Individuos con velocidades lejanas

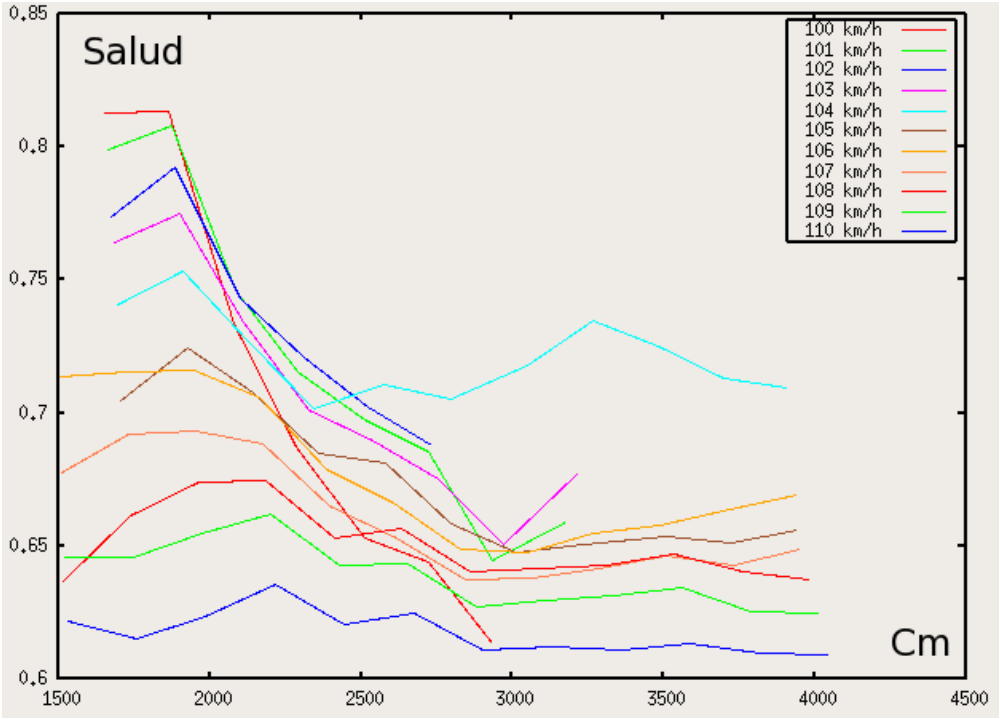


Figura 5.10: Salud media de individuos con velocidades cercanas

En la figura 5.10, vemos un ejemplo de la evolución de individuos de una raza con velocidades cercanas. En esta gráfica se muestra a los individuos a partir del metro 15, no desde el comienzo del seguimiento. En la gráfica 5.10 vemos como antes del metro 25, no hay una raza que destaque sobre las demás en salud. Es a partir del metro 25 donde donde la raza que tiene velocidad de 104 *km/h* destaca sobre las demás. Otra importante apreciación es que ninguna de las velocidades de la gráfica es eliminada, ya que al ser velocidades parecidas, están muy próximas espacialmente. El objetivo de estas dos gráficas ha sido mostrar como individuos con velocidades muy diferentes son eliminados rápidamente y como individuos con velocidades muy próximas a la real llegan al final de la etapa de seguimiento.

Como acabamos de comentar, a partir del metro 25 es el individuo con velocidad de 104 *km/h* el que tiene la máxima salud hasta el final, siendo esta consistente hasta el final. En este sentido otra importante comparación es entre la salud media acumulada con la salud instantánea. En la figura 5.10 vemos como la salud media acumulada es bastante estable. En la siguiente figura 5.11 vemos la salud instantánea para individuos con velocidades cercanas. Inicialmente todos los individuos parten del mismo punto, la gráfica muestra la evolución de los individuos a partir del metro 15, en este momento los individuos que tienen velocidades muy bajas o altas con respecto al coche ya han sido eliminados. En la gráfica podemos ver cómo hasta el metro 25 no hay ningún in-

dividuo que destaque sobre las demás. A partir de ahí es el individuo con velocidad de 104 *km/h* el de máxima salud hasta el metro 35, donde es el individuo de 106 *km/h*, el que tiene la máxima salud. Pero no sólo hay inconsistencias en el individuo que tiene la máxima salud. Si nos fijamos, veremos que todos los individuos tiene diferentes picos de salud, cambiando de posición constantemente en lo que al ranking de máxima salud se refiere. El principal motivo de que la salud instantánea sea inconsistente es el ΔT, usada en la fórmula 4.3 para actualizar la posición del individuo en cada interacción.

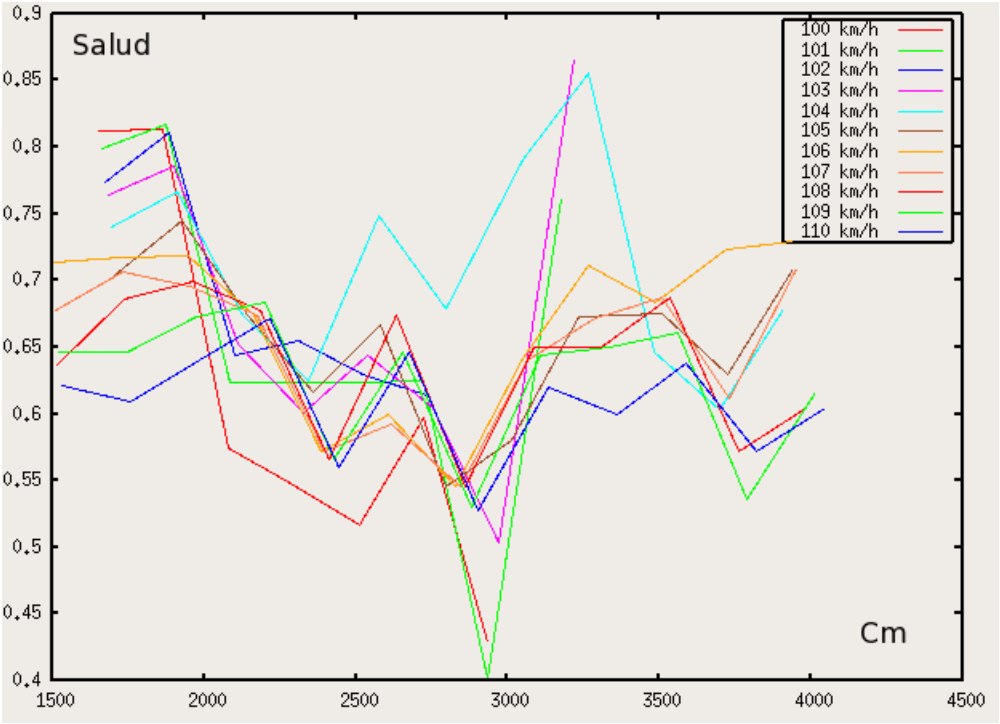


Figura 5.11: Salud instantánea de individuos con velocidades cercanas

5.4. Análisis de errores

En esta sección analizaremos el error sobre las pruebas realizadas en el simulador, ya que nos permite ver la diferencia entre la velocidad de los coches simulados y la velocidad estimada. El motivo de realizar los experimentos para analizar el error sobre el simulador es por que no disponemos de medios precisos para saber la velocidad real a la que circulan los vehículos en la carretera.

En la figura 5.12 podemos ver un ejemplo de cómo podemos comparar la velocidad estimada con la velocidad real del coche. La velocidad estimada la podemos ver al lado de los coches en la imagen real y la real está indicada en los deslizadores con el

mismo color que los coches. Además en la misma figura 5.12 podemos ver un ejemplo del seguimiento realizado sobre varios vehículos en gazebo.

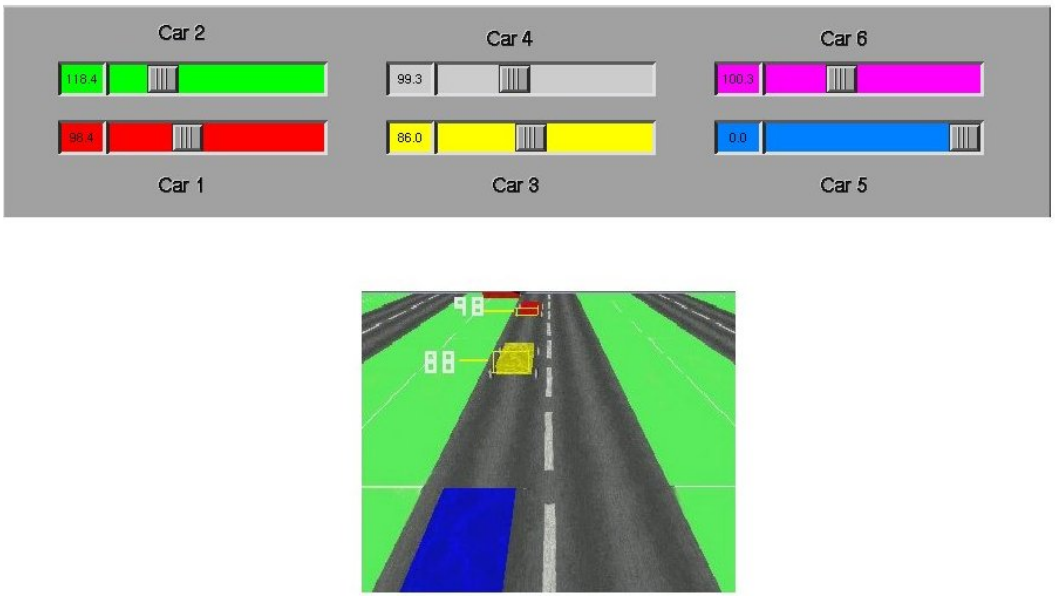


Figura 5.12: Comparación entre la velocidad estimada y la real

Inicialmente veremos las pruebas realizas sobre coches salientes y con una resolución de imagen de 320x240 píxeles, que es la configuración habitual del sistema. Después compararemos estos resultados con los obtenidos para coches entrantes y misma resolución. Finalmente compararemos los resultados iniciales con coches salientes y con una resolución de 640x480 píxeles. Hay que destacar que todas las pruebas se han realizado con la estimación de la velocidad para unos mil vehículos simulados.

5.4.1. Análisis con vehículos salientes

A continuación realizamos un análisis de los errores obtenidos en el sistema, con su configuración inicial: imágenes de 320x240 y coches salientes.

En la siguiente gráfica podemos ver los resultados obtenidos para coches salientes con una resolución de imagen de 320x240. En el eje x se encuentra el error entre la velocidad estimada y la real, y en el eje y el número de coches que tiene este error. En dicha gráfica podemos ver cómo el error varía entre *-6* y *2 km/h*. El error medio cometido es de -2.18 *km/h* y la desviación de típica es de *1.95*. Así el error de la población queda principalmente concentrada entre -4 y 0 *km/h*. Esto implica que las velocidades estimadas son por lo general un poco más lentas que las reales. El motivo

de que las velocidades estimadas sean algo más lentas puede ser por las holguras a la hora de medir tiempo que transcurre entre cada iteración.

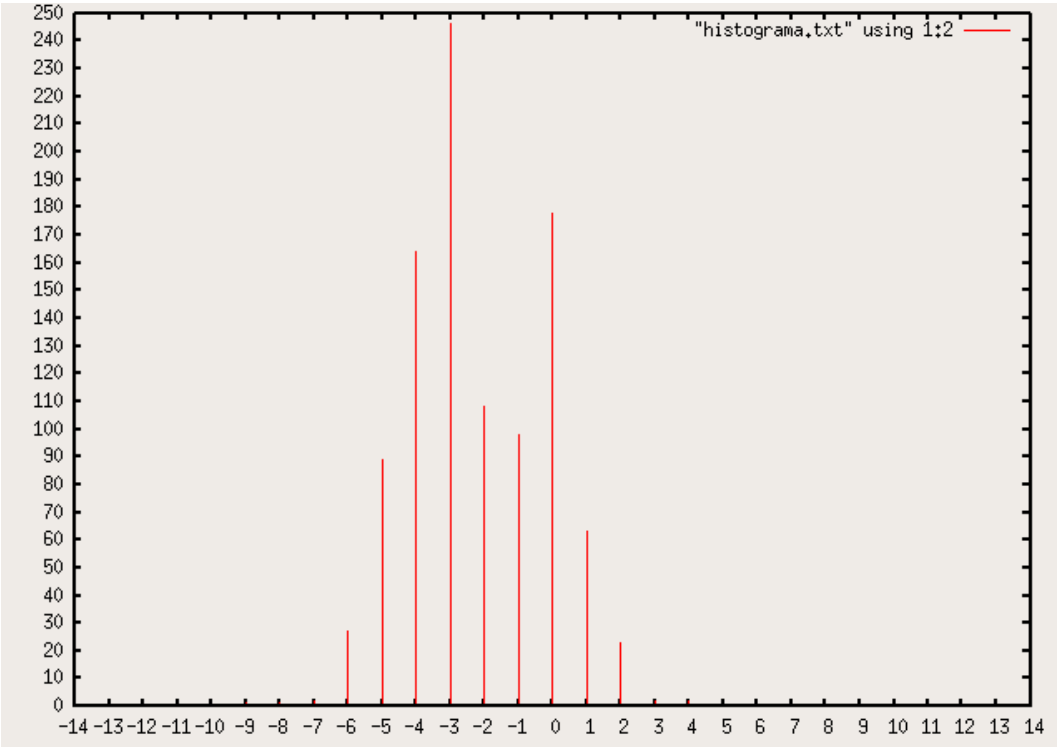


Figura 5.13: Error para vehículos saliente y resolución 320x240

5.4.2. Análisis con vehículos entrantes

Una vez que hicimos las pruebas para coches salientes pensamos en como mejorar el error. El principal problema que veíamos para esta configuración es que en zonas alejadas de la cámara existe menos resolución. Que haya menos resolución implica que es más difícil descartar individuos y por tanto tener una estimación de velocidad menos exacta. El problema de la resolución al final de la carretera se debe a que individuos con velocidades parecidas y por tanto espacialmente muy próximos, les corresponden los mismos píxeles para cuerpo y falda al hacer la correspondencia entre la posición del individuo y la imagen. Por lo que es más difícil que el algoritmo en estas zonas discrimine velocidades.

Otro de los experimentos realizados fue la de usar el algoritmo para vehículos entrantes. De esta manera perderíamos resolución en la zona de detección pero lo ganaríamos según el vehículo se va acercando a la cámara, siendo la velocidad final más exacta.

En la figura 5.14 podemos ver la gráfica obtenida. En el eje x se encuentra el error entre la velocidad estimada y la real, y en el eje y el número de coches que tiene

este error. El error se encuentra entre $\pm 5 km/h$. También vemos cómo la gráfica es más simétrica y cómo se concentran más los valores de error entorno a cero. Estos datos visuales los podemos contrastar con el error medio y la desviación típica. En este caso el error medio es de -0.707 *km/h* y la desviación típica de 1.85. Por lo tanto la mayor parte del error de la población se concentra aproximadamente entre -2.5 y 1.5 *km/h*. En este caso las velocidades estimadas siguen siendo un poco menores a la real. Podemos concluir que el sistema obtiene mejores resultados con un configuración para vehículos entrantes. Esto es debido a la mayor resolución de la imagen en la zona final de seguimiento. Al haber mayor resolución, los individuos que se encuentran próximos espacialmente es más complicado que coincidan en los mismos píxeles de imagen para cuerpo y falda.

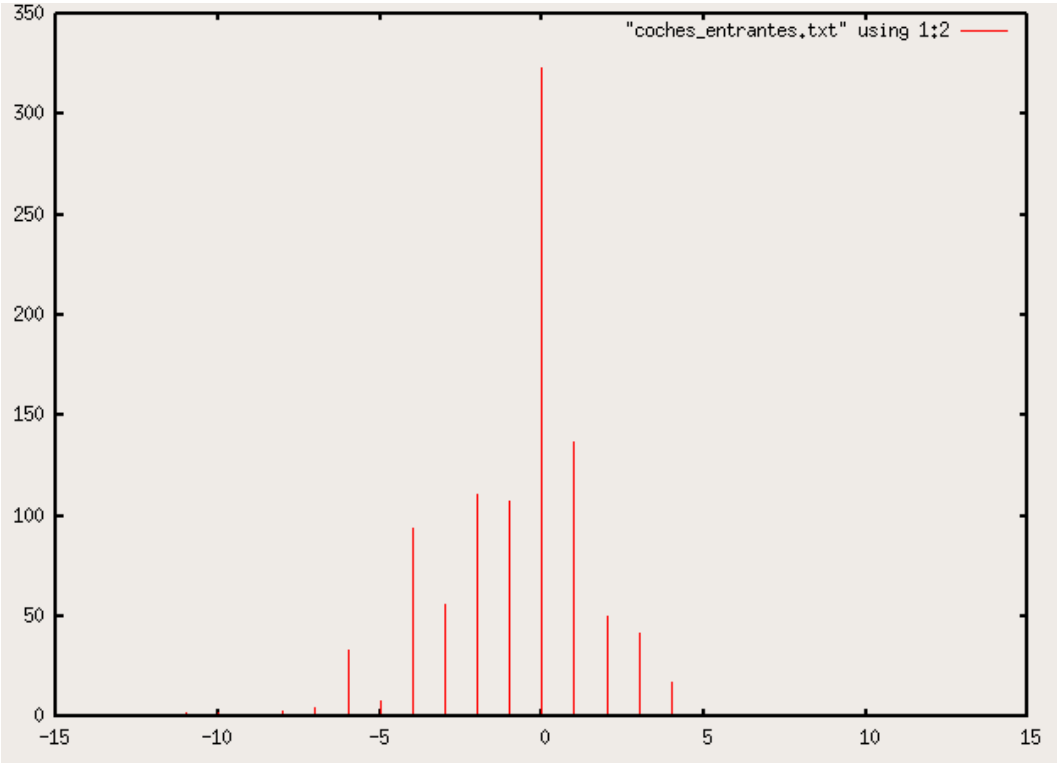


Figura 5.14: Error para vehículos saliente y resolucion 320x240

5.4.3. Análisis con imágenes de mayor resolución

Otra de las implementaciones realizadas para mejorar los resultados de los coches entrantes fue tratar imagenes de mayor resolución, en concreto de 640x480 píxeles para coches salientes. De esta manera ganamos resolución en la parte final del seguimiento (más alejada de la cámara) y las estimaciones de velocidad serán más exactas.

En la figura 5.15 podemos ver la gráfica obtenida. Si la comparamos con la figura

inicial 5.13 vemos que los resultados no mejoran a los iniciales, hay un mayor rango de error, en concreto entre -8 y 8 *km/h*. El error medio es de -0.4243 y la desviación típica de 3.9. Por lo que la mayor parte del error de la población queda concentrada entre -4.5 y 3.5. Idealmente los resultados deberían de ser mejores, ya que al haber una mayor resolución de la imagen es más difícil que individuos de diferentes velocidades y próximos espacialmente les correspondan los mismos píxeles, en cuerpo y falda, en la imagen a la hora de calcular la salud. Sin embargo, los resultados experimentales obtenidos no verifican lo que pensamos en un principio. Uno de los motivos de este resultado es que aumenta la carga computacional al tratar una imagen mayor. Esto implica que disminuyan las imágenes por segundo, afectando a la vivacidad del seguimiento y a la estimación de la velocidad.

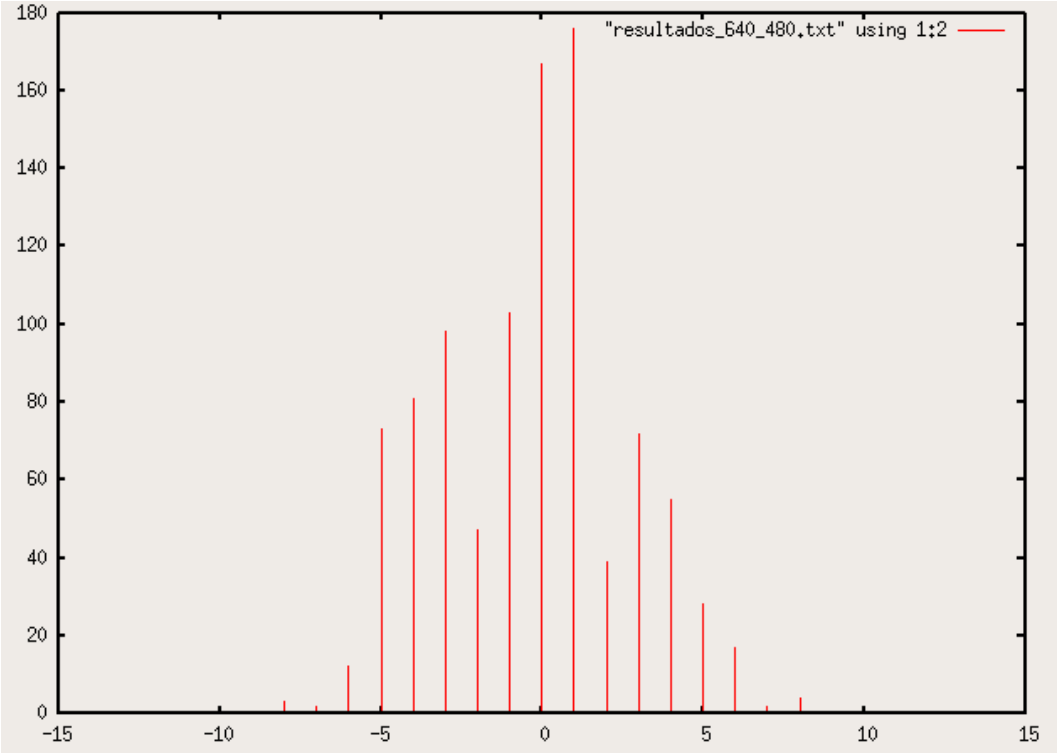


Figura 5.15: Error para vehículos saliente y resolucion 320x240

5.5. Robustez del algoritmo

En esta sección vamos a analizar la robustez del algoritmo, en la instalación , detección y seguimiento.

Un punto muy importante de esta aplicación, es que ésta sea capaz de detectar y seguir a varios vehículos a la vez. En la figura 5.16(a) podemos ver cómo se detectan dos vehículos simultáneamente y en la figura 5.16(b) vemos como la aplicación es capaz de seguir varios vehículos al mismo tiempo. Por lo tanto vemos que el algoritmo es robusto

a la detección y seguimiento de varios vehículos. Si bien es cierto, que el detectar y seguir a varios vehículos a la vez supone un mayor coste computacional. Este hecho hace que en ordenadores con menor capacidad de cómputo a los que se ha probado el algoritmo disminuya el número de imágenes por segundo (*ips*), por lo que la estimación de velocidad puede ser menos exacta e incluso se puede perder al individuo en la zona de seguimiento.

(a) Detección múltiple (b) Seguimiento múltiple

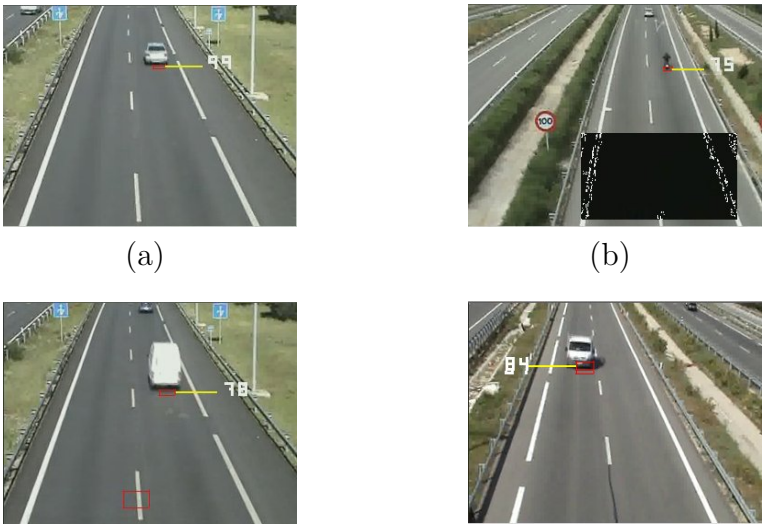
Figura 5.16: Detección y seguimiento de múltiples vehículos

Para no sobreadaotar el algoritmo a una carretera y situación concreta se han llevado a cabo pruebas reales realizadas en diferentes carreteras: A-5 , M-506 y M-511. Además de probar en diferentes carreteras se han probado diferentes orientaciones de cámaras como podemos ver en la figura 5.17. También hay que destacar que dicho sistema es robusto frente a cambios de iluminación. Si nos fijamos en las imágenes de la figura 5.17 vemos que la hay diferentes condiciones de iluminación en cada una de ellas. Si bien es cierto, que no se ha probado el sistema para condiciones extremas como pueda ser con lluvia y que con imágenes nocturnas el sistema no funciona correctamente ya que muchos vehículos no son detectados y otros son seguidos por más de una raza a la vez.

(a) M-506 (b) M-5011 (c) A-5

Figura 5.17: Diferentes orientaciones de la cámara en las pruebas realizadas sobre diferentes carreteras

La aplicación es también robusta a diferentes tamaño de vehículos. En la figura 5.18 podemos ver desde motos, coches, furgonetas hasta camiones pequeños. Sin embargo, para vehículos muy largos, como autobuses, al ocupar a lo largo más que la zona de detección, generan en su seguimiento muchas razas.



(a) (b)

(c) (d)

Figura 5.18: Pruebas sobre diferentes tamaños de vehículos

En esta sección hemos probado el buen funcionamiento de la aplicación, que es capaz de detectar, seguir y estimar correctamente la velocidad de vehículos. Además el sistema es vivaz (entre unas 10 y 15 imágenes por segundo) ya que permite realizar la detección y el seguimiento de varios coches a la vez. También hemos visto que es muy robusto frente a cambios de carretera y diferentes orientaciones de la cámara, haciendo que la aplicación tenga una fácil instalación. Además se han realizado varias pruebas sobre los mismos vídeos, obteniéndose los mismos resultados en la estimación de la velocidad para los mismos vehículos. De esta manera aunque no podamos comparar en un entorno real la velocidad estimada con la velocidad real, ya que no disponemos de medios, podemos decir que los resultados son consistentes.

5.6. Alternativas probadas

En esta sección describiremos varias alternativas implementadas antes de haber llegado a la solución final. Dichas implementaciones las fuimos descartando por uno u otro motivo hasta finalmente llegar a la solución propuesta en el capítulo 4 que es la que mejor nos ha funcionado. A continuación detallamos cada una de ellas.

5.6.1. Calibración de cámaras

Como hemos comentado en a lo largo de la memoria, para que la aplicación funcione correctamente necesitamos establecer una correspondencia entre la imagen y su posición en la carretera. Esto es necesario para que los individuos de cada raza puedan evolucionar a lo largo del tiempo.

Inicialmente para conseguir este objetivo pensamos en desarrollar el sistema basándonos en la calibración de la cámara que proporciona las imágenes. De esta manera dado un píxel seríamos capaces de conocer su correspondiente punto 3D en la carretera y viceversa. De esta manera los individuos de la raza estarían definidos por la posición 3D (x,y,z)en la carretera y su correspondiente *velocidad*.

Para las pruebas realizadas sobre el simulador, necesitamos saber la relación entre las cámaras simuladas en *gazebo*, implementadas con OpenGL, y *Progeo*, biblioteca de geometría proyectiva que usamos en el grupo de robótica de la URJC. Ambas bibliotecas utilizan el modelo de cámara *pinhole*, por lo que únicamente difieren en la forma de especificar sus parámetros. Estos parámetros pueden ser extrínsecos o intrínsecos. Los extrínsecos son expresados por ambas bibliotecas como la posición de la cámara en el mundo, el foco de atención y la orientación o roll. Sin embargo los parámetros intrínsecos se caracterizan de forma diferente. Progeo necesita la distancia focal y la posición del centro del plano imagen. Por el contrario, OpenGl utiliza el ángulo de apertura y la relación de aspecto del plano imagen. La equivalencia existente entre ambas se determina fácilmente ya que el tamaño de las imágenes es conocido (320*240 relación de aspecto), el centro óptico de la cámara OpenGl es 160,120 (al tratarse de una cámara ideal) y el ángulo de apertura equivale a $\alpha = 2 * \arctan \frac{alto}{2*f}$. Tenemos determinada la equivalencia entre las diferentes formas de expresar los parámetros intrínsecos que tiene cada biblioteca.

Una vez calibrada la cámara correctamente, con la biblioteca *Progeo* podemos relacionar puntos en la imagen con su posición 3D. *Progeo* nos ofrece la función *project* que nos permite proyectar un píxel 3D en una linea visual, es decir, nos dice que la posición del píxel en la realidad se encuentra en un punto de esa línea. Más en concreto, para saber la posición exacta nos hace falta una tercera coordenada (coordenada z ó altura). En nuestro caso como queríamos la posición sobre la carretera, asumimos que $z=0$ y obtendríamos la posición 3D. En la siguiente figura 5.19 podemos ver el rayo visual

obtenido en la realidad al proyectar un píxel de la imagen.

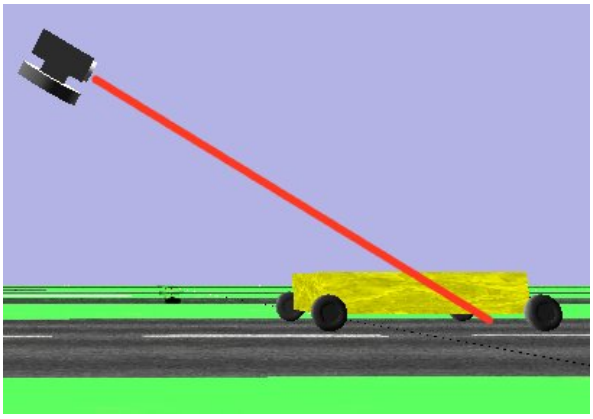


Figura 5.19: Rayo visual proyectado por progeo

Dentro de este pfc se implemento un módulo en el cual se calibro la cámara simulada. En la siguiente figura 5.20, podemos ver cómo se ha conseguido una correcta relación entre la cámara simulada por el *simulador gazebo* y la biblioteca *progeo*. Así conociendo la posición 3D de las diferentes esquinas de los rectángulos al proyectar la posición de cada punto sobre la imagen (función inversa a *project*), los píxeles proyectados coinciden con las esquinas de los rectángulos.

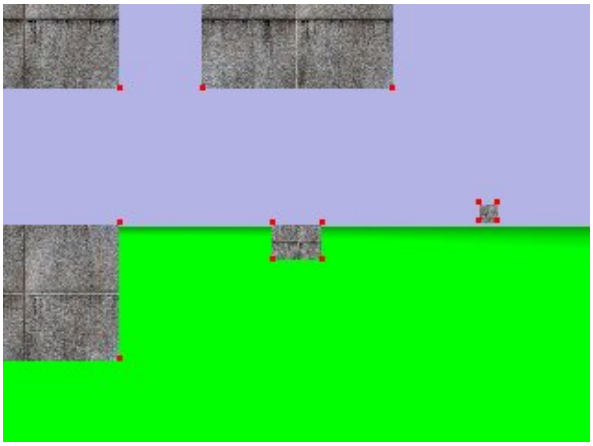


Figura 5.20: Correspondencia entre cámaras simuladas con OpenGl y Progeo

Sin embargo, emplear la calibración de cámaras para nuestro proyecto presentaba un gran problema, la dificultad de saber exactamente la posición de la cámara en la realidad. Esto llevaría a una difícil instalación del sistema. Por lo tanto decidimos emplear otra técnica para manejar distancias en la carretera.

5.6.2. Algoritmo evolutivo multimodal

Como comentamos en el capítulo 4 el algoritmo desarrollado en está aplicación esta inspirado en un algoritmo multimodal evolutivo. Inicialmente se realizó una primera implementación del algoritmo con operadores genéticos para obtener la nueva población de cada raza. Pero el hecho de utilizar operadores genéticos daba lugar a grandes problemas para estimar la velocidad, ya que individuos con velocidades con mala salud no eran eliminados y podían ser la estimación final del sistema.

En esta implementación, al igual que en la aplicación final, teníamos dos etapas bien diferenciadas. Por un lado estaba la etapa de detección de vehículos en el tramo inicial de la calzada. Por otro lado, teníamos la etapa de seguimiento. Según fuese la zona donde nos encontráramos aplicábamos diferentes operadores genéticos. Los operadores genéticos usados en la etapa de detección (individuos exploradores) son los siguientes:

- *Operación de mutación aleatoria* Este operador posiciona aleatoriamente a un individuo en la zona de detección de la carretera (ver figura 4.3). Este operador permite explorar uniformemente la zona inicial de la calzada. Sus probabilidades de éxito son bajas pero permite evitar los máximos locales en la búsqueda de la solución.

- *Operador genético de abducción* Este operador generara un individuo explorador en aquellas zonas de la etapa de detección donde se detecte movimiento. De esta manera si detectamos movimiento gracias al flujo óptico o a la diferencia de imágenes, surge un nuevo explorador.

A continuación comentamos los operadores genéticos usados en la etapa de seguimiento.

- *Operador genético de elitismo.* Este operador necesita que los individuos estén ordenados por salud y elige los explotadores con mayor salud de la población permitiendo que permanezcan en la nueva población, manteniendo su posición y velocidad. Este operador es poco eficiente ya que se necesitan generar muchos para detectar los vehículos.

- *Operador genético de ruido térmico.* Este operador se encarga de operar con los individuos que poseen menor salud. En este caso las nuevas posiciones son cercanas a las que ocupaba un individuo de la población de la raza seleccionado mediante el algoritmo de la ruleta. La velocidad del nuevo individuo en la raza es cercana a la anterior, en concreto $\pm$ 5 *kmh*.

- *Operador genético de cruce* Mediante el operador de cruce se realiza la fusión de tres individuos de la población anterior para generar el nuevo individuo. Los tres individuos se seleccionan mediante el algoritmo de la ruleta y cada uno de ellos aportara la posición x,y y la velocidad respectivamente.

Acabamos de comentar que para seleccionar ciertos individuos usamos el algoritmo de la ruleta. Este algoritmo asigna un sector a cada individuo proporcional a la salud de este. Así cuanta más salud tenga un individuo, más grande será su sector y más probabilidad tendrá ese individuo de ser seleccionado. Se girará la ruleta tantas veces como individuos se quiera seleccionar. Con este algoritmo los individuos con más probabilidad tienen más opciones de salir y por tanto la nueva población estará formada a partir de los individuos más probables.

Una vez explicados los fundamentos del algoritmo evolutivo multimodal, pasamos a comentar los problemas que presenta esta implementación. Para explicar mejor estos problemas nos apoyamos en la figura 5.21. Con la primera de estas figuras 5.21(a), vamos a comentar los problemas del operador genético de ruido térmico. Supongamos que el punto azul es el individuo seleccionado para aplicar ruido térmico. Este punto por las características de nuestra función salud debería ser eliminado, ya que su velocidad con respecto el coche es menor. Ahora bien al aplicar el operador surge otro individuo, punto amarillo, relativamente cerca y con velocidad muy similar. Esto hace que los individuos con velocidades menores y mayores a la real sean muy difíciles de eliminar. De esta manera surgen líderes en la raza que no han sido contrastados durante muchos fotogramas.

(a) Ruido térmico (b) Cruce

Figura 5.21: Problemas de los operadores genéticos

Por otro lado, en la figura 5.21(b) tenemos un ejemplo de los problemas que presenta el operador genético de cruce. Supongamos que los puntos rojos, son los seleccionados por el algoritmo de la ruleta para realizar el cruce, y el punto amarillo el resultante. Como podemos ver el individuo que debía ser eliminado (punto rojo con una velocidad de 80 km/h) proporciona su velocidad al individuo resultante. Por lo tanto como en el anterior caso, es muy difícil eliminar individuos con menor y mayor velocidad.

Capítulo 6
Conclusiones y trabajos futuros

Descrita la solución propuesta para esta aplicación y comentados algunos de los experimentos más relevantes, terminamos esta memoria exponiendo las conclusiones principales sacadas de este proyecto y las posibles líneas futuras en las que se puede seguir investigando.

6.1. Conclusiones

Haciendo un repaso de los objetivos marcados en el capítulo 2, vamos a describir en qué medida se han cumplido éstos satisfaciendo los requisitos también expuestos en ese capítulo. El objetivo principal de desarrollar un algoritmo para estimar la velocidad y contar los vehículos que pasan por la carretera se ha cumplido, consiguiendo un correcto funcionamiento del sistema, como se ha validado en el capítulo 5.

El objetivo global estaba articulado en 3 iteraciones principales

1. Localización de vehículos en la carretera, para conocer en todo momento la posición relativa de los vehículos en la calzada.

2. Detectar los vehículos nuevos que aparecen en la imagen.

3. Realizar un seguimiento correcto de los coches que nos permita contar y estimar la velocidad de los vehículos

El primer subobjetvio era el localizar los vehículos en la carretera, para concer en todo momento su posición la calzada. Para solucionar este subobjetivo, hemos integrado el rectificador de imágenes existente en la plataforma *jdec*. Este rectificador establece una homografía entre la imagen y la carretera, de tal forma que podemos saber que posición (x,y) de la carretera le corresponde a un determinado píxel. Gracias a que hemos conseguido localizar los vehícuns en la carretera, nuestro espacio de trabajo ha sido la carretera rectificada. Este subobjetivo es muy importante para poder realizar

una estimación de la velocidad. Ya que para poder estimar la velocidad nos hace falta saber el espacio que recorre un individuo en un determinado espacio de tiempo. Este subobjetivo fue inicialmente probado en el simulador y una vez que se obtuvieron buenos resultados pasamos a probarlo con imágenes reales.

El segundo subobjetivo era el de detectar los nuevos vehículos que aparezcan en la zona inicial de la carretera. Para ello se han llevado a cabo diferentes técnicas que detectan movimiento en la zona inicial de la carretera. Tal y como se vio en el capítulo 4 de la memoria, una de ellas se basa en la diferencia de imágenes y la otra en el cálculo de flujo óptico. Una vez que se ha detectado el movimiento en la imagen se generan una seríe individuos exploradores de manera inteligente, con el objetivo principal de que surja una raza por cada vehículo. Esta raza parte con N individuos todos ellos situados en la misma posición pero con diferentes hipótesis de velocidad. Estos individuos además tienen asociada una salud que nos dice cuan buena es la posición del individuo. Inicialmente probamos la funcionalidad en el simulador *gazebo*, y una vez habiendo comprobado que esta funcionaba correctamente pasamos a probarla en la realidad. Los experimentos descritos en el capítulo 5, muestran que el cálculo de flujo óptico no funciona tan bien para coches entrantes como para coches salientes. Además el cálculo de flujo óptico lleva mucha carga computacional. Por lo tanto podemos concluir para este subobjetivo es mejor utilizar diferencia de imágenes para detectar movimiento.

El subobjetivo 3 era el de realizar un seguimiento correcto de los diferentes vehículos que nos permita contar y estimar la velocidad de los vehículos. Para ello nos inspiramos en los proyectos fin de carrera de Sara Marugán [Marugán, 2007] y Ángel Cortés [**?**]. Como comentamos en el subobjetivo 2, una vez que un vehículo es detectado surge una raza. Es en este momento donde comienza el seguimiento. En el seguimiento cada individuo va evolucionando en el tiempo, descartándose aquellas cuya salud asociada no supera un cierto umbral. El objetivo principal en esta etapa es que la población de la raza converga de N individuos hasta tener muy pocos individuos al final de la carretera para poder contabilizar al vehículo y dar una estimación de velocidad muy precisa. Este subobjetivo fue probados inicialmente en el simulador. El simulador nos ofrecia la ventaja de poder comparar la velocidad estimada y la real. Una vez que obtuvimos nuevos resultados pasamos a probar nuestro algoritmo a la realidad.

Una vez verificado experimentalmente el comportamiento, podemos concluir que ha sido generado correctamente, y se ha desarrollado íntegramente en la plataforma *jdec*,

en la que como requisitos importantes debía implementarse el proyecto mediante esquemas, siendo programados en el lenguaje C, conseguir un algoritmo capaz de localizar, detectar, realizar un seguimiento vivaz de vehículos a la vez que obtiene una estimación de la velocidad muy precisa y lleva la cuenta de los vehiculos que circulan por la calzada.

6.2. Líneas futuras

Algunas posibles mejoras que se podrían realizarse sobre este proyecto y que sirven como nuevas líneas de investigación para otros proyectos fin de carrera son los siguientes.

- Comprobar que las estimaciones de velocidad de nuestro sistema son precisas comparandolas con las velocidades de los vehículos en la realidad.
- Una mejora considerable sería hacer funcionar el sistema en condiciones extremas con lluvia, niebla e por la noche, ya que dadas las difíciles condiciones de luminosidad es muy difícil detectar movimiento. Una buena ídea sería el intentar seguir a los coches a partir de sus luces
- Un interesante proyecto sería el de integrar este proyecto con otro que fuera capaz de detectar las matrículas de los coches y almacenara tanto la matrículá como el video de prueba de aquellos vehículos que sobrepasen el límite de velocidad de la carretera correspondiente

□

Bibliografía

[Cabello, 2007] Antonio Pineda Cabello. Aplicación de seguridad basada en visión. *Proyecto fin de carrera, URJC*, 2007.

[de Agapito, 1999] Lourdes de Agapito. Linear calibration of a rotating and zooming camera. *Proceedings on Computer Vision and Pattern Recognition*, 1999.

[Erik B. Sudderth, 2005] Michael I. Mandel Erik B. Sudderth. Visual hand tracking using occlusion compensated message passing. 2005.

[Forsyth y Ponce, 2003] David A. Forsyth y Jean Ponce. Computer vision: A modern approach. 2003.

[Gonzalez, 2005] Pablo Barrera Gonzalez. *Aplicación de los métodos secuenciales de Monte Carlo al seguimiento visual 3D de múltiples objetos*. PhD thesis, Universidad Rey Juan Carlos, 2005.

[Hartley y Zisserman, 2004a] Hartley y Zisserman. Multiple view geometry in computer vision. *Cambridge University Press, ISBN: 0521540518, second edition*, 2004.

[Hartley y Zisserman, 2004b] Hartley y Zisserman. Multiple view geometry in computer vision. *Cambridge University Press*, 2004.

[Ipp, 2002] Ipp. Intel integrated performance primitives v5.3 for linux on ia-32. *Intel Architecture, Release Notes*, 2002.

[Kachach, 2008] Redouane Kachach. Calibración automática de cámaras en la plataforma jdec. *Proyecto Fin de carrera, Universidad Rey Juan Carlos*, 2008.

[Luca y Snidaro, 1998] Gian Luca y Lauro Snidaro. Vehicle detection and tracking for traffic monitoring. *University of Udine*, 1998.

[Lucas y Kanade, 1981] B. Lucas y T. Kanade. Proceedings of darpa image understanding workshop, chapter an iterative registration technique with an apllication to stero vision, pages 121-130. 1981.

[Martín, 2008] Javier Martín. Detección y seguimiento de caras en la plataforma jdec. *Proyecto fin de carrera, URJC*, 2008.

[Martínez, 2005] Marta Martínez. Sistema de atención visual en escena. *Proyecto fin de carrera, URJC*, 2005.

[Martínez, 2007] Iván García Martínez. Reconstrucción 3d visual con algoritmos evolutivos. *Proyecto Fin de Carrera, URJC*, 2007.

[Marugán, 2007] Sara Marugán. Seguimiento 3d visual de múltiples personas utilizando un algoritmo evolutivo multimodal. *Proyecto fin de carrera, URJC*, 2007.

[Plaza, 2003] José María Cañas Plaza. Jerarquía dinámica de esquemas para la generación de comportamiento autónomo. *Tesis doctoral, Universidad Politécnica de Madrid*, 2003.

[Santos, 2007] José Antonio Santos. Cálculo y aplicaciones de flujo óptico en tiempo real. *Proyecto Fin de carrera, Universidad Rey Juan Carlos*, 2007.

[Zhigang Zhu y Yang, 1998] Guangyou Zhigang Zhu y XuBo Yang. a real-time vision system for automatic traffic. *University of Tsinghua*, 1998.

[Ángel Cortés Maya, 2007] Ángel Cortés Maya. Localización y construcción de mapas en un robot de interiores. *Proyecto fin de carrera, URJC*, 2007.