INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMAS

Escuela Superior de Ciencias Experimentales y Tecnología

Curso académico 2006-2007

Proyecto Fin de Carrera

Cálculo y aplicaciones del flujo óptico en tiempo real.

Tutor: José María Cañas Plaza

Autor: José Antonio Santos Cadenas

"Si seguimos haciendo lo que siempre hemos hecho, continuaremos obteniendo los mismos resultados que hasta ahora. Para obtener resultados diferentes, hay que hacer cosas diferentes."

Albert Einstein

Agradecimientos.

En primer lugar quiero agradecer a mi novia Tere por estar siempre ahí y por todo el esfuerzo que ha hecho para corregir la memoria.

En segundo lugar a mi tutor José María todo el esfuerzo que ha hecho para que este proyecto estuviera terminado a tiempo.

También quiero agradecer a mi familia por apoyarme en todo lo que emprendo, gracias por confiar en mi.

Para terminar quiero agradecer a mis compañeros de proyecto Javier Martín y Sara Marugán la buena convivencia que ha habido en estos meses de trabajo y las ayudas prestadas.

Gracias a todos, no hubiera sido posible sin vosotros.

Resumen.

En la actualidad, la visión por computador es una de las disciplinas de la informática que más posibilidades ofrece. Puesto que, las imágenes contienen gran cantidad de información y las cámaras tienen un precio reducido. Por estos motivos, se hace interesante poder utilizarlas para automatizar ciertas tareas. La visión por computador utiliza diferentes formas de extraer datos de las imágenes, una de ellas es el flujo óptico, que es una señal rica en información y que permite conocer el movimiento en la imagen de forma precisa, indicando origen y destino del mismo en ciertos puntos interesantes para su cálculo.

El objetivo de este proyecto ha sido construir un módulo integrado en la plataforma *jdec* que calcule la señal del flujo óptico, así como construir dos aplicaciones que ilustran posibles usos de ésta señal. El cálculo del flujo óptico se apoya en la funcionalidad de la biblioteca *Ipp* de *intel*, que permite mejorar el rendimiento de los algoritmos, dado que está implementada en bajo nivel y utiliza instrucciones más específicas de los procesadores.

La primera de las aplicaciones de ejemplo *eyeoperator* permite operar el robot realizando movimientos delante de la cámara. Para ello, captura el flujo óptico que generan los movimientos del usuario y lo traduce a movimientos del robot. La segunda es la aplicación *cuenta-coches*, que utiliza el flujo óptico para detectar y contar los coches que pasan por una carretera. A raíz del flujo óptico, realiza una segmentación en cada fotograma y establece correspondencias entre dos segmentaciones consecutivas para seguir los coches mientras avanzan por la carretera.

Para comprobar que las aplicaciones realizadas cumplían los objetivos, se han hecho pruebas experimentales de funcionamiento. Incluso se ha tenido que desarrollar un nuevo módulo para la plataforma *jdec*, que permite tomar las imágenes de entrada desde un vídeo en lugar de una cámara, para facilitar los experimentos de la aplicación *cuenta-coches*.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1
Introducción

Una de las principales aportaciones de la informática a la vida moderna es la automatización de tareas más o menos complejas, de tal forma que se realicen correctamente sin la supervisión directa de un humano. El potencial que demuestran las máquinas en la realización de ciertas tareas frente a los seres humanos es muy grande en campos en los que los problemas son sencillos pero requieren una velocidad de cálculo o de procesamiento de datos muy alta. Otras tareas para las que son útiles las máquinas son aquellas en las que la fuerza física necesaria es demasiado grande para el humano, el riesgo que entraña la actividad es muy alto o, simplemente, son aburridas.

Figura 1.1: Registro de vehículos automatizado en la entrada de un parking

Un factor importante a la hora automatizar las tareas es que la máquina pueda tomar buenas decisiones de forma autónoma, para ello, debe disponer de datos ricos en información. Una de las mayores fuentes de información para el ser humano son los ojos, gracias a ellos podemos percibir el entorno que nos rodea e interactuar con él. De la misma forma, una fuente muy importante de información para las máquinas son las cámaras.

Las cámaras son sensores muy rentables, ya que tienen un precio reducido y la cantidad de información que pueden aportar es muy grande. Pero tienen un problema, extraer la información de las imágenes es costoso, tanto por el coste computacional que entraña como por la dificultad de diseñar algoritmos que consigan extraer información. La parte de la informática que se encarga de trabajar con las imágenes y de sacar información de ellas es la visión artificial o visión por computador.

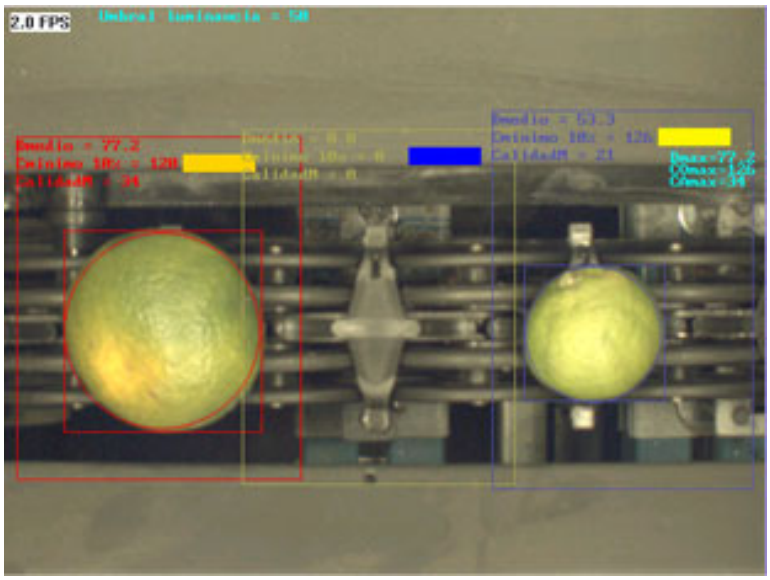


Figura 1.2: Control de calidad de la fruta utilizando visión computacional

En la actualidad la visión artificial es una rama muy importante de la informática, ya que, el abaratamiento y perfeccionamiento de las cámaras y de los ordenadores, así como la mejora de los algoritmos, que ahora permiten extraer mucha más información, han hecho que en numerosos lugares se automaticen tareas utilizando estas técnicas. Por ejemplo, en la entrada de algunos parkings hay cámaras que se encargan de grabar las matrículas y registrar las entradas y salidas, reconociendo los caracteres que hay en las mismas (ver imagen 1.1). También se utiliza la visión artificial en ciertos procesos de calidad, para comprobar que los productos cumplen las características establecidas (Ver figura 1.2).

La visión computacional también se puede encargar de procesos como el reconocimiento de textos (OCR) en una imagen. Permitiendo así convertir a formato digital casi cualquier texto de forma automática.

En el campo de la localización 3D, cabe destacar un par de aplicaciones. La primera

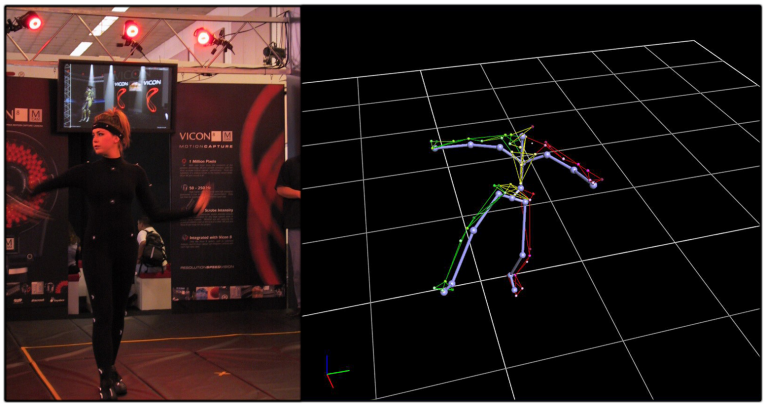


Figura 1.3: Captura de movimientos mediante el sistema *Vicon*

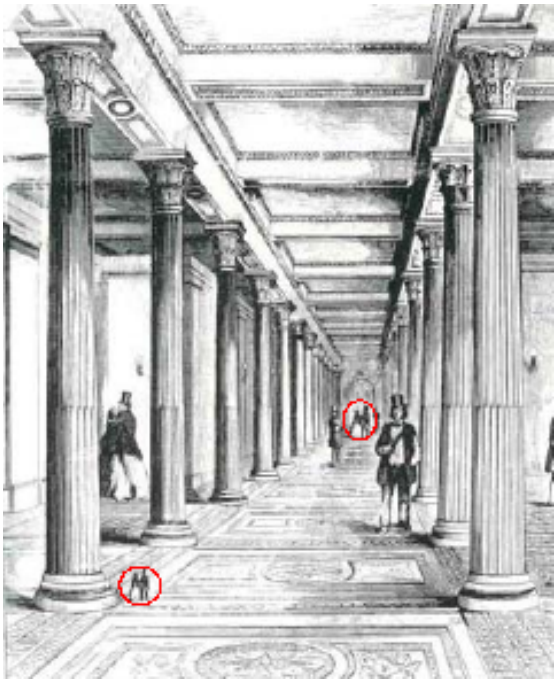
la aplicación *Vicon*[1], que permite capturar movimientos a partir de la realidad. Se una mucho para la animación digital en la industria cinematográfica. El sistema detecta, mediante el uso de varias cámaras, los marcadores que se colocan en ciertos puntos del cuerpo de la persona y realiza una estimación de su posición en 3D, generando un esqueleto que al que se le pueden aplicar luego diferentes pieles dependiendo del personaje.

Otra aplicación relacionada con las estimaciones 3D es *EyeVision*. *EyeVision* es un sistema que permite la reconstrucción de escenas 3D de tal manera que no se disferencia entre la reconstrucción y las fotos originales. Este sistema se utiliza en las repeticiones de las jugadas en la liga *Superbowl*, permitiendo ver la jugada desde cualquier ángulo. Para ello, se graba el partido desde numerosas cámaras situadas estratégicamente y se raliza una reconstrucción a partir de todas ellas. El resultado son imágenes de la misma calidad que las originales, pero desde nuevos ángulos, calculadas en pocos segundos (en la figura 1.4 se puede obsevar algunas de las imágenes generadas).

Figura 1.4: Sistema de repetición de jugadas en 3D *Eyevision*

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

En definitiva, la visión por computador no trata más que de realizar composiciones de la realidad a partir de las percepciones que recibe y, al igual que le pasa al ser humano (ver figura 1.5), las máquinas pueden encontrarse con entornos que les resulten confusos y por lo tanto, las conclusiones extraídas de la imagen no sean completamente correctas.



(a) Dificultad al determinar el tamaño de un objeto, los dos detalles marcados son del mismo tamaño

(b) ¿Mujer o saxofonista?

Figura 1.5: Imágenes que presentan interpretaciones dudosas para el humano

En visión computacional, los métodos más comunes para extraer información de las imágenes son [Gonzalez y Woods, 1993]: los filtros de color, de bordes o esquinas, búsqueda de patrones en la imagen, entre otros. Con estos métodos es fácil realizar segmentaciones e incluso alguna estimación de distancia monocular utilizando hipótesis adicionales, como por ejemplo en [Díaz Peña, 2005] se utiliza un filtro de bordes y la hipótesis del suelo [Horswill, 1993] para calcular las distancias.

Los métodos citados anteriormente son muy buenos para realizar ciertas operaciones más o menos sencillas. Pero cuando se quiere conocer el movimiento en una sucesión de imágenes, no de manera aproximada sino precisa, hay que emplear otras técnicas. Para estimar movimientos se pueden realizar diferencias entre fotogramas consecutivos, pero este método no aporta datos más allá de qué puntos son diferentes en las dos

imágenes (ver imagen 1.6). Se desconoce el sentido del movimiento y su naturaleza. Por su parte, el flujo óptico es un método de extracción de información que sí cumple esos requisitos, indica la dirección, el sentido y la intensidad del movimiento, describiendo adecuadamente los desplazamientos que se producen en la imagen.

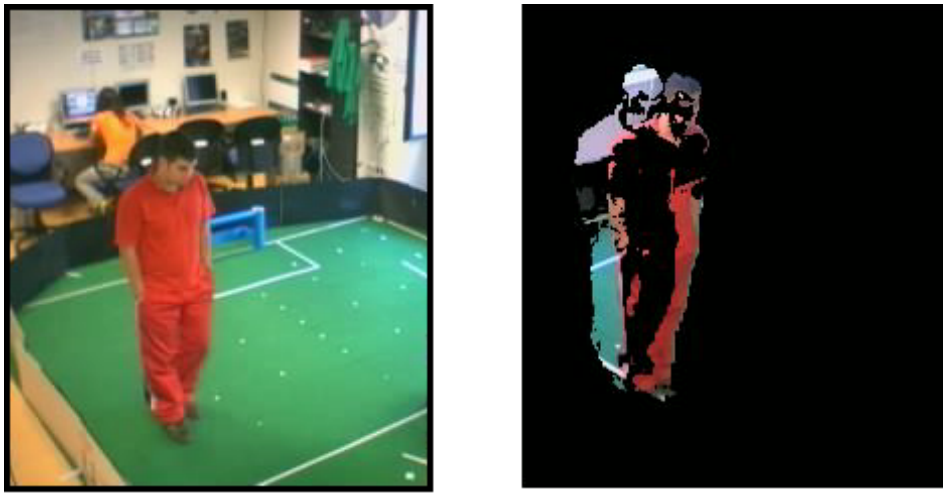


Figura 1.6: Búsqueda del movimiento mediante el método de las diferencias

Conocer cómo se mueven los objetos de la imagen es un modo de extraer información muy rico. De hecho, los primates tenemos un subsistema visual dedicado a la percepción del movimiento. A partir del flujo óptico se puden calcular otros datos, por ejemplo, se pueden reconocer objetos, porque si se conocen los puntos que se están moviendo en cada dirección y con diferentes intensidades, se podrá estimar qué partes de la imagen pertenecen al mismo objeto (por tener características similares). Otro posible cálculo que se puede realizar a partir del flujo óptico es medir las distancias de la cámara a los objetos, ya que el movimiento de los objetos lejanos genera flujo óptico de menor intensidad que el movimiento de los objetos cercanos. De esta forma se puede estimar si un objeto está más alejado que otro, he incluso, calibrando la cámara, se puede conocer la distancia exacta a los objetos utilizando simplemente una cámara.

El flujo óptico también es útil en aplicaciones que simulen atención visual, es un buen disparador para esta tarea, ya que, además de indicar que hay movimiento, permite conocer la naturaleza del mismo. Pudiendo descartarlo, en caso de no ser suficiente, o utilizar los datos que aporta para realizar un seguimiento, aprovechando que se puede hacer una previsión del futuro emplazamiento del objeto en la imagen siguiente, suponiendo que su movimiento sea constante.

(a) Erik (b) Visión en primera persona del robot Erik recorriendo un pasillo

Figura 1.7: Robot Erik del MIT, capaz de navegar utilizando flujo óptico

Otra aplicación muy interesante que se puede apoyar en la señal de flujo óptico es la navegación visual. Se puede lograr que un robot deambule por un entorno semi-estructurado utilizando tan solo una cámara y el flujo óptico para esquivar los obstáculos, puesto que los objetos generan flujo óptico cuando el robot se acerca a ellos. De esta forma, intentando mantener equilibrado el flujo óptico a ambos la dos de la imagen, el robot avazará por el centro del pasillo. Si aparece algún obstáculo, girará hacia el lado contrario para equilibrar el flujo óptico (que en ese lado de la imagen habrá aumentado) y así esquivará el bache. En el año 2001, un grupo de trabajo del *Massachusetts Institute of Technology* (MIT) desarrolló una aplicación de estas características[2] (En la figura 1.7 se puede ver el robot y una visión en primera persona del robot navegando).

En definitiva, la visión por computador y en concreto el flujo óptico, son uno de esos avances que pueden conseguir aliviar de ciertos trabajos pesados a las personas. Este proyecto de fin de carrera trata de calcular, conocer y utilizar esta señal para abrir el camino de su uso en nuevas aplicaciones.

En los capítulos siguientes se presentan, en primer lugar, los objetivos que se pretenden conseguir con el proyecto (capítulo 2). Posteriormente, se introduce el contexto en el que se realiza el desarrollo (capítulo 3) para continuar explicando en los capítulos 4 y 5 el trabajo realizado. Por último, en el capítulo 6, se exponen las conclusiones del trabajo realizado y se proponen futuros trabajos a raíz del desarrollado.

[2]http://people.csail.mit.edu/lpk/mars/temizer_2001/Optical_Flow/index.html

Capítulo 2
Objetivos

Una vez presentado el contexto del proyecto en el capítulo 1, en este capítulo se explican las líneas de trabajo seguidas durante el desarrollo del proyecto. Se describe el problema que se quiere abordar, los requisitos que deben cumplir las soluciones que se adopten y qué método de trabajo se ha seguido.

Como se describe en el capítulo anterior, el flujo óptico es una señal muy rica en información que se puede aplicar en gran cantidad de aplicaciones. Siendo el flujo óptico una forma de analizar la señal de imagen tan rica, sería interesante utilizar el cálculo de esta señal en algunas aplicaciones robóticas que utilizan como sensor principal la cámara. Así como preparar al futuro usuario de esta señal aportándole algún ejemplo de cómo manejarla.

2.1. Objetivos

Por los motivos descritos en el párrafo anterior, se ha decidido que *jdec* cuente con un módulo capaz de calcular el flujo óptico de la imagen. En esta línea, los objetivos a cumplir con este proyecto son:

1. Desarrollar un componente apoyado en la plataforma *jdec* que sea capaz de calcular el flujo óptico a partir de las imágenes de entrada de la plataforma (ya sea capturada por la cámara o por otros medios que la plataforma permita). Esta aplicación deberá tomar la señal de vídeo que aporta la plataforma *jdec* y exportar como datos de salida los valores del flujo óptico para cada punto en el que proceda ser calculado.

2. Crear varias aplicaciones que utilicen la señal de flujo óptico como señal de entrada para que ilustren algunas de las ventajas de esta señal, así como mostrar la forma de tomar los datos de la aplicación que calcula el flujo óptico a posibles usuarios de la misma.

Como aplicaciones se proponen dos:

- Una que sea capaz de manejar el robot a partir de los datos que reciba por la cámara. Algo parecido a los juegos de la serie *eyetoy*[1] de *PlayStation®2*[2]. La aplicación deberá utilizar la señal de flujo óptico para generar comportamientos en el robot, es decir moverlo hacia delante, hacia atrás o girarlo.
- La otra permitirá contar los coches que pasan por un tramo de carretera utilizando el flujo óptico.

2.2. Requisitos

Ahora que se han definido los objetivos que se pretenden alcanzar al terminar el proyecto, hay que describir exactamente qué requisitos mínimos tienen que cumplir las aplicaciones que realizadas.

- El componente que calcule el flujo óptico tendrá que funcionar en tiempo real, esto es, que calcule el flujo óptico de la secuencia a la vez que la está recibiendo, además de consumir el menor tiempo de cómputo posible, para que otros programas puedan procesar la señal que genere.
- Los datos que calcule tienen que ser fiables, es decir que el programa presente datos con un índice de error muy bajo.
- Por último, los cálculos que realice deben ser robustos, es decir, independientes de las condiciones externas, el error no debe aumentar con los cambios de iluminación, con objetos demasiado pequeños o con movimientos bruscos.
- El requisito fundamental de las aplicaciones de ejemplo es que funcionen utilizando como señal de entrada el flujo óptico. Ya que el fin es ilustrar el trabajo con esta señal.
- La forma de manejar el robot tiene que ser intuitiva, para facilitar en lo posible la adaptación del usuario a la aplicación.
- Por último, la aplicación que cuenta los coches realizará un seguimiento de los mismos, mostrando un recuadro en torno a cada vehículo que detecte.

[1] http://www.eyetoy.com
[2] http://es.playstation.com/ps2/

2.3. Metodología

Para el desarrollo del proyecto, se ha seguido un plan de trabajo basado en el modelo en espiral. Se ha elegido este sistema porque es flexible, permite planear las actividades a corto plazo basadas en los resultados que se van obteniendo, permitiendo cambiar ciertos objetivos cercanos en cada reunión semanal con el tutor. De esta forma, se pueden suplir carencias de conocimiento o realizar pruebas parciales del software de una forma más natural que planificando el proyecto completo desde el inicio.

Al tener que crear varias aplicaciones diferentes, cada una de ellas se ha desarrollado de forma independiente, debiendo ser la primera la aplicación que calcule el flujo óptico, ya que las otras dependen de ella.

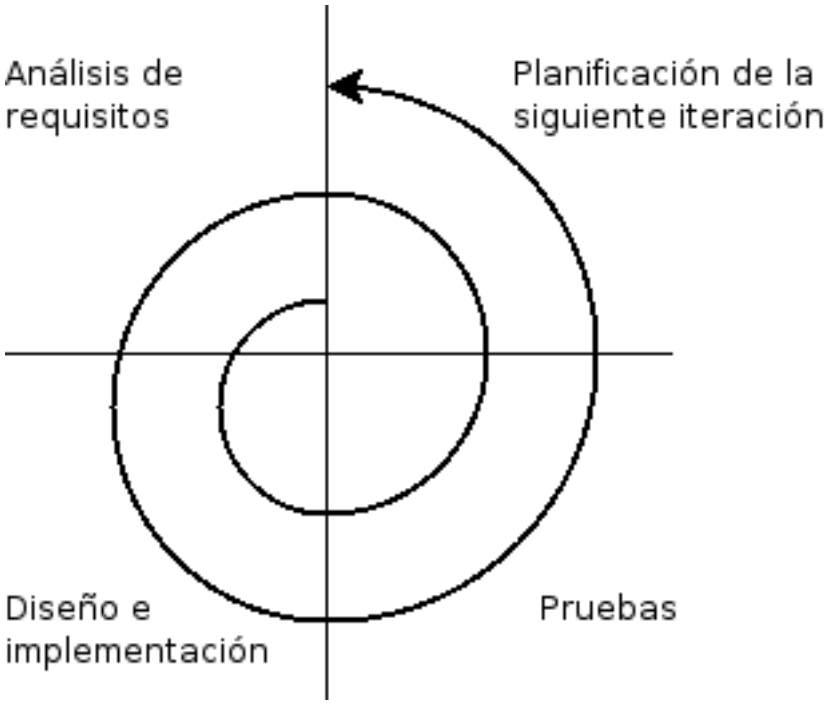


Figura 2.1: Modelo de desarrollo en espiral

El modelo de desarrollo utilizado (modelo de desarrollo en espiral) consiste en realizar iteraciones sucesivas de cuatro etapas (ver figura 2.1): *análisis de requisitos*, *diseño e implementación*, *pruebas* y *planificación de la próxima iteración*. Con cada iteración se van cumpliendo nuevos objetivos o sub-objetivos y además se va comprobando que el camino que se está siguiendo es el correcto, porque la etapa de *planificación de la próxima iteración* permite comprobar si el avance generado en la iteración actual ha sido productivo o si por el contrario se ha avanzado en una dirección errónea, en cuyo

caso habrá planear una nueva etapa que lleve el desarrollo al camino correcto.

Las líneas generales de actuación han sido:

1. Durante las dos primeras semanas, un proceso de adaptación a la plataforma *jdec* que será la que se utilice como base para la realización del proyecto. Habrá que aprender todo lo necesario para poder realizar aplicaciones basadas en esa plataforma, así como conocer su funcionamiento a fondo para sacar el máximo rendimiento.

2. Recopilar información sobre flujo óptico, así como posibles métodos para el cálculo y el trabajo con él.

3. Realizar la implementación de un componente que calcule el flujo óptico. Conforme a las herramientas y métodos que se han estudiado en la fase anterior. Esta etapa, la más costosa y larga, ha habido que dividirla en varios hitos diferentes, de manera que no se ha abordado la aplicación al completo, sino que se han desarrollado módulos de forma independiente en iteraciones sucesivas.

4. Abordar la aplicación que cuenta coches, buscando las formas de segmentar y de realizar el seguimiento que mejor se adapten de las que se hayan estudiado en el punto 2.

5. Implementar la aplicación que teleopera el robot, también basando el trabajo en lo métodos encontrados en el punto 2.

6. Por último, ha habido que documentar todo el proceso que se ha realizado escribiendo una memoria en la que se detalle el resultado final del producto, así como las decisiones más importantes que se han tomado en toda la etapa de diseño.

Capítulo 3
Entorno de desarrollo

En este capítulo se explica todo el software y hardware sobre los que se apoya el proyecto, tanto la plataforma *Jdec* como la biblioteca *Ipp* de *intel*, el hardware utilizado y otras bibliotecas auxiliares.

3.1. *Jdec*

Jdec es una plataforma que facilita la programación de robots desarrollada por el grupo de robótica de la universidad rey Juan Carlos. Nació en 1997 como fruto del trabajo de una tesis doctoral [Cañas, 2003] y actualmente es la plataforma principal que utiliza el grupo de robótica de esta universidad. Dado que se trata de la principal plataforma de programación de robots que utiliza el grupo de robótica, es la que se ha utilizado también para realizar este proyecto. De esta forma, se facilita parte del trabajo (ya que la plataforma aporta ciertas ventajas que veremos a continuación). La versión que se ha utilizado es la 4.2, las instrucciones para su descarga se pueden encontrar en la página web de la plataforma (`https://trac.robotica-urjc.es/jde/`). Todas las aplicaciones y componentes desarrollados en este proyecto funcionan dentro de esta plataforma.

La principal ventaja que ofrece *Jdec*, es que proporciona una API al programador de las aplicaciones muy sencillo, pudiendo acceder a los *sensores y actuadores* utilizando simplemente *variables compartidas*. Además ofrece un encapsulado para las aplicaciones llamado *esquema*, una aplicación puede constar de uno o varios esquemas funcionando concurrentemente. La plataforma provee un esqueleto, formado por una serie de funciones, que el programador deberá rellenar con la funcionalidad que desee implementar en cada esquema.

Un esquema, según se explica en el manual de *jdec* [Cañas *et al.*, 2007], es la unidad

mínima de programación que maneja *Jdec*, es una hebra de ejecución encapsulada para estructurar el comportamiento y generalizar la forma de compartir variables y funciones. Además, nos ofrece una funcionalidad mínima, como puede ser lanzarlo, detenerlo o dejarlo en espera hasta que se den las condiciones necesarias para que se ejecute. Los esquemas se pueden organizar de forma jerárquica (padres e hijos) haciendo que el funcionamiento de un esquema esté basado en el funcionamiento de otros. Por ejemplo, un esquema padre puede resolver varias situaciones, gracias a que sus hijos las resuelven, simplemente tendrá que lanzarlos o pararlos según sea necesario.

La naturaleza del esquema es iterativa, es decir, el código escrito por el programador se ejecutará repetidamente. Para que el consumo de CPU no se dispare, la plataforma proporciona un mecanismo para controlar el número de iteraciones por segundo que se desean ejecutar.

Hay dos tipos de esquemas: los esquemas creados por usuarios, cuyo objetivo es resolver ciertas situaciones con las que se encuentre el robot o facilitar a otros esquemas datos derivados de otros más sencillos, y los esquemas básicos. Éstos pueden ser tanto perceptivos como motores y ambos son generados por los *drivers*. El objetivo de los esquemas básicos es poner a disposición de otros esquemas las variables que representan los sensores o actuadores. Los otros esquemas leerán los datos de los sensores o escribirán las acciones a realizar en esas variables.

La función de los *drivers* es sumamente importante, ya que son los encargados de facilitar el acceso a sensores y actuadores uniformando su apariencia. Para ello, se comunica con el hardware y genera esquemas que comparten variables que representan estos sensores. *Jdec* dispone de un conjunto de drivers ya creados, pero también permite la creación de nuevos drivers que se pueden utilizar, ya que el mecanismo de carga de esquemas y drivers es dinámico (en tiempo de ejecución). Para que el usuario decida qué drivers cargar, se utiliza un fichero de configuración, en el que se indican los drivers que se utilizarán, junto con las opciones pertinentes para cada driver (ver tabla 3.1) y los esquemas que se cargarán en una ejecución concreta de la plataforma.Para más detalles sobre el fichero de configuración de *jdec*, se puede consultar su manual [Cañas *et al.*, 2007].

Los drivers que se han utilizado para la realización de este proyecto son:

- Driver *player*: Permite conectar con *Player*, que es un servidor para el control de

robots (ver apartado 3.2). Éste servidor permite uniformar en cierta medida el acceso a los robots, ocultando, incluso, si se trata de un robot real o de un robot simulado. El *driver* adapta la forma de trabajo del servidor *Player* a la plataforma *jdec*, ofreciendo variables que representan los sensores del robot, como el láser o los sónar, además de los actuadores (mover los motores del robot). Para poder mover el robot, este driver permite utilizar dos variables, v y w. La variable v permite controlar la velocidad de avance o retroceso del robot, de tal forma que el robot se moverá a la velocidad indicada en dicha variable en m/s y durante el tiempo que tenga asignado ese valor. El mismo caso se da para la variable w, que controla la velocidad de giro del robot en grados/segundo.

En la configuración de éste driver (tabla 3.1), hay que indicar qué sensores y actuadores se quieren obtener, la posición inicial del robot en el mundo, y la dirección y el puerto en el que escucha el servidor *Player*.

- Driver *firewire*: proporciona acceso a las imágenes que capturan las cámaras por medio del bus *firewire* (IEEE1394). El acceso a las imágenes para el programador es una variable compartida (*colorA*) en la que las imágenes están representadas en formato BGR mediante un *array* que contiene 3 bytes por píxel. La posición de los píxeles en el array es consecutiva, comenzando por el píxel superior izquierdo y terminando por el inferior derecho hasta completar el total de los 320X240 píxeles.

  En la tabla 3.1 se puede observar cómo es la configuración de este *driver*, se indica el nombre de la variable compartida (*colorA*), el nodo en el que está conectada la cámara (*0*) y si se quiere *autofocus*.

- *Driver mplayer*: para poder realizar pruebas con flujos de imágenes diferentes a los que se podían capturar con la cámara, así como para poder tener la misma secuencia de vídeo para comprobar las mejoras en los algoritmos, se realizó dentro este proyecto un driver capaz de tomar un archivo de vídeo como entrada y proporcionar como salida una variable compartida que represente la imagen al estilo de la plataforma. El ritmo al que se muestran los fotogramas es al que se indique en el vídeo, además se realiza un escalado para ajustar el tamaño del vídeo al de la plataforma.

  El *driver* está apoyado en la funcionalidad del reproductor de películas para Linux *mplayer* y el codificador *mencoder*. Éste se encarga de lanzar *mplayer* para decodificar la película al ritmo al que debe ser mostrada, y escribir la decodifica-

```
### DRIVER LOAD FOR SENSOR SOURCES AND MOTOR DEVICES  -------------------
driver firewire
provides colorA 0 autofocus_off
#provides colorB 1 autofocus_on
end_driver

#driver mplayer
#provides colorA videos/coches.mpg repeat_on
#provides colorA <filename> repeat_on
#end_driver

driver player
hostname localhost port 6665
#initial position X(mm) Y(mm) Theta(deg)
initial_position -6500 -6500 45
#provides laser
#provides sonars
provides encoders
provides motors
end_driver

### SCHEMA LOAD --------------------
load opflow
#load cuentacoches
load eyeoperator
```

Cuadro 3.1: Posible fichero de configuración de *jdec*

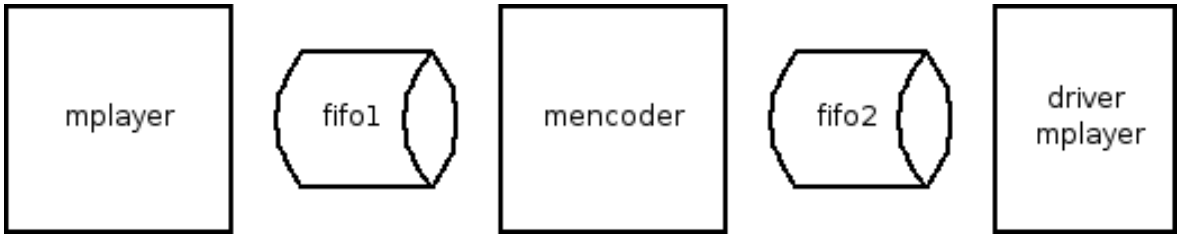


Figura 3.1: Esquema de funcionamiento del *driver mplayer*

ción en un *fifo*[1]. De este *fifo* leerá una instancia de *mencoder* lanzada también por el *driver* y convertirá los datos al formato raw BGR24, que es el mismo que utiliza *jdec*, escribiendo los resultados en otro *fifo*. Por último, el *driver* leerá de ese *fifo* e irá copiando los datos de cada *frame* a la variable compartida que representa la imagen En la figura 3.1 se puede ver un esquema del funcionamiento del driver. Son necesarios dos intermediarios porque *mplayer* no convierte directamente a BGR24 que es el formato necesario para la plataforma *jdec*.

Como los *fifos* tienen una capacidad limitada, cuando se intenta realizar una operación de escritura en uno que está lleno, el programa que lo intenta queda bloqueado. Esta característica permite que cuando el usuario detenga la captura de imágenes de la plataforma, *mplayer* se detenga, ya que no se consumen los datos que está escribiendo en el *fifo*, y al volver a arrancar la captura continúe en la misma posición del vídeo.

La ventaja que representa utilizar una aplicación externa para poder decodificar los vídeos es que no es necesario empotrar los *codecs* en el programa ni hay que restringir los vídeos a algunos *codecs*. El *driver* aceptará todos los vídeos que *mplayer* pueda reproducir y esto es una gran ventaja dada la gran variedad de tipos de vídeo que soporta. Además, como *mplayer* es un proyecto vivo, existe un grupo de desarrolladores que se encarga de su mantenimiento y la incorporación de nuevos *codecs* es más rápida.

La forma de configurar el *driver* es mediante el fichero de configuración de *jdec* (ver tabla 3.1). En el que se indica el nombre de la variable en la que va a compartir la imagen mediante la cláusula"*provides*", el nombre del fichero de vídeo (y posiblemente la ruta de acceso) y por último, se indicará si se desea que el vídeo se repita indefinidamente, al término del mismo, o si bien que se muestre sólo una vez mediante "*repeat_on*" o "*repetat_off*".

Finalmente, *jdec* permite dos formas de interactuar con él como usuario. La primera mediante un entorno gráfico que permite lanzar y detener esquemas, así como visualizar qué esquemas se han cargado y cuales están ejecutando en cada momento, además permite arrancar el entorno gráfico de cada uno de los esquemas por separado (Ver figura 3.2). La otra forma de comunicarse con la plataforma es utilizar la *shell* de texto que proporciona. Esta *shell* ofrece la misma funcionalidad que el entorno gráfico, pero

[1]Herramienta del sistema operativo Linux, que permite compartir memoria entre dos programas diferentes a los cuales se les presenta esta región de memoria como un fichero

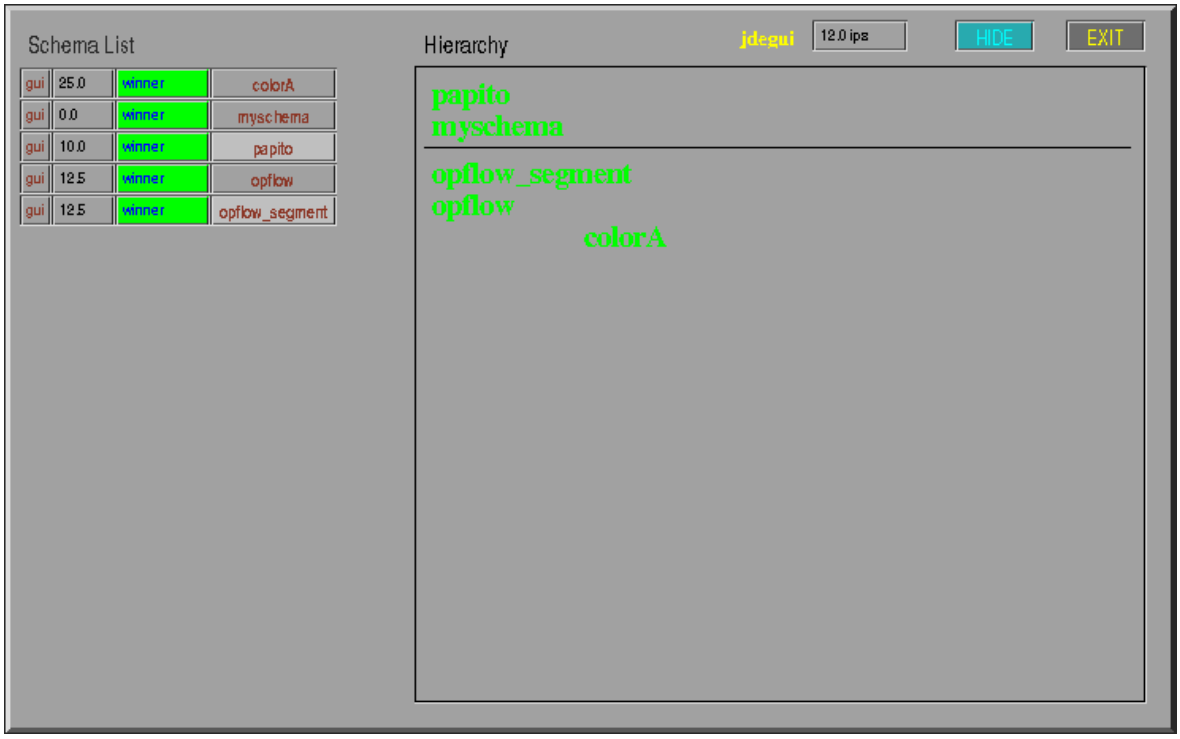


Figura 3.2: Entorno gráfico de la plataforma *jdec*

con la ventaja de no tener que consumir tiempo de ejecución en mostrarlo.

3.2. Simulador *Stage* y servidor *Player*

Stage es un simulador de entornos robóticos en dos dimensiones capaz de simular varios robots en un mismo entorno. Pertenece al mismo proyecto que *Player* y generalmente se utilizan conjuntamente. *Player* es un servidor que permite conectarse al simulador *Stage*, así como a robots reales, proporcionando acceso a sus sensores y actuadores a través de una conexión de red. Se han utilizado las versiones 2.0.3 de *Stage* y la 2.0.4 de *Player* para realizar las pruebas de funcionamiento de la aplicación *eyeoperator*. Ambos programas se puede descargar de la página web del proyecto *Player* `http://playerstage.sourceforge.net/`.

En la figura 3.3 se puede ver el simulador *Stage* en con el que se han generado varios robots diferentes en un entorno de oficinas.

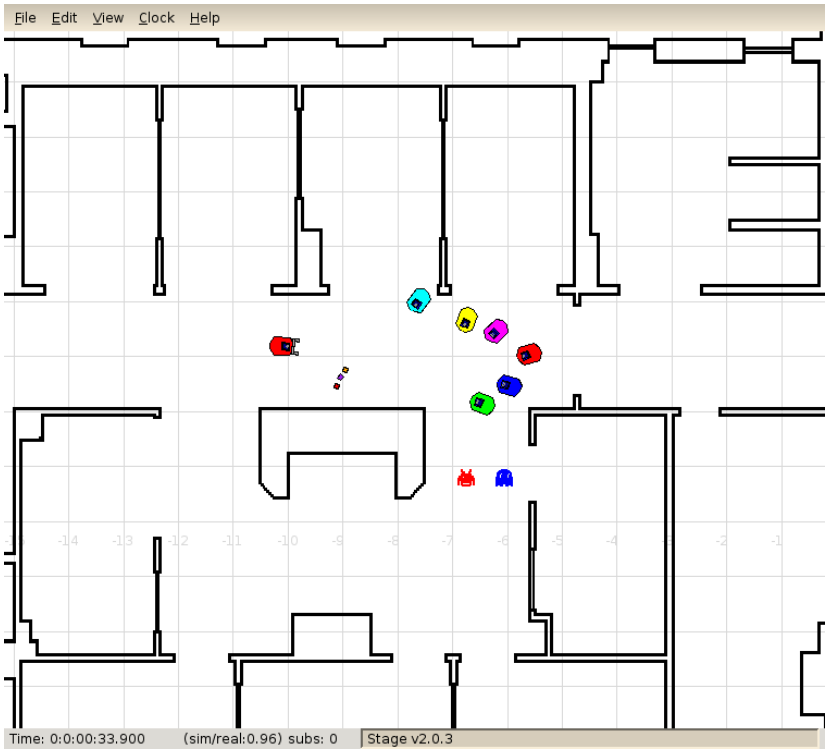


Figura 3.3: Simulador *Stage*

3.3. Biblioteca *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*) ofrece una amplia gama de funcionalidad: tratamiento de señales de audio, trabajo con imágenes y vídeos, criptografía y operaciones matriciales, vectoriales y con cadenas de textos [Ipp, 2002]. Todas ellas con 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 esté utilizando. Estas versiones ofrecen un rendimiento muy alto ya que cada versión está preparada para cada tipo de procesador. La que se ha utilizado está optimizada para la arquitectura IA-32, que es capaz de funcionar sobre cualquier procesador de 32 bits, incrementando el rendimiento de forma diferente en unos pro-

cesadores que en otros, se ha probado también sobre procesadores AMD, en los que el rendimiento también es superior al normal, aunque muy por debajo del rendimiento habitual de IPP en procesadores de intel.

La decisión final de utilizar esta biblioteca se tomó porque el rendimiento que ofrece es muy alto frente a otras bibliotecas de visión computacional similares (permite tiempo real), como puede ser *OpenCV*. Es por esto que pese a los inconvenientes que tiene Ipp (licencia, escasez de información y complejidad en el manejo de las funciones) finalmente se decidió que la implementación del software se apoyase en ella.

Se utilizó para este proyecto la versión 5.1 para Linux. Se puede descargar de la página web de intel[2] (Actualmente se oferta la versión 5.2 de la biblioteca).

3.4. *XForms*

Es la biblioteca gráfica que utiliza actualmente *Jdec* (ver figura 3.2) y la que se utilizó para generar las interfaces gráficas de los esquemas creados. *XForms* trabaja directamente sobre el servidor X de la máquina, utilizando la biblioteca *Xlib*; lo que hace que el sistema de ventanas generado sea altamente compatible y ligero.

XForms permite la creación y visualización de ventanas, formularios, botones, barras de desplazamiento, espacio para visualizar imágenes, etc. Para facilitar el diseño de las ventanas y los formularios, *XForms* proporciona una herramienta gráfica llamada *fdesign* que permite diseñar visualmente el interfaz gráfico simplemente arrastrando los componentes que nos interesan a la zona deseada.

Figura 3.4: Cámara web Apple iSight

[2]http://www.intel.com

3.5. Cámaras *iSight*

Las cámaras de Apple iSight 3.4 son cámaras para vídeo conferencia, pero con una resolución y calidad de imagen suficientemente buena para los objetivos de las aplicaciones. Además, dado que se conecta al ordenador por el bus *firewire*, proporciona una buena cadencia de imágenes (30 imágenes por segundo a la resolución a la que trabajan los programas). Se ha utilizado como fuente de las imágenes sobre las que se calcula el flujo óptico.

Capítulo 4
Flujo óptico

En este capítulo se establece el pilar fundamental del proyecto, sobre el que se apoyan el resto de aplicaciones. Se explican algunos principios teóricos sobre el cálculo de flujo óptico, así como las implementaciones realizadas.

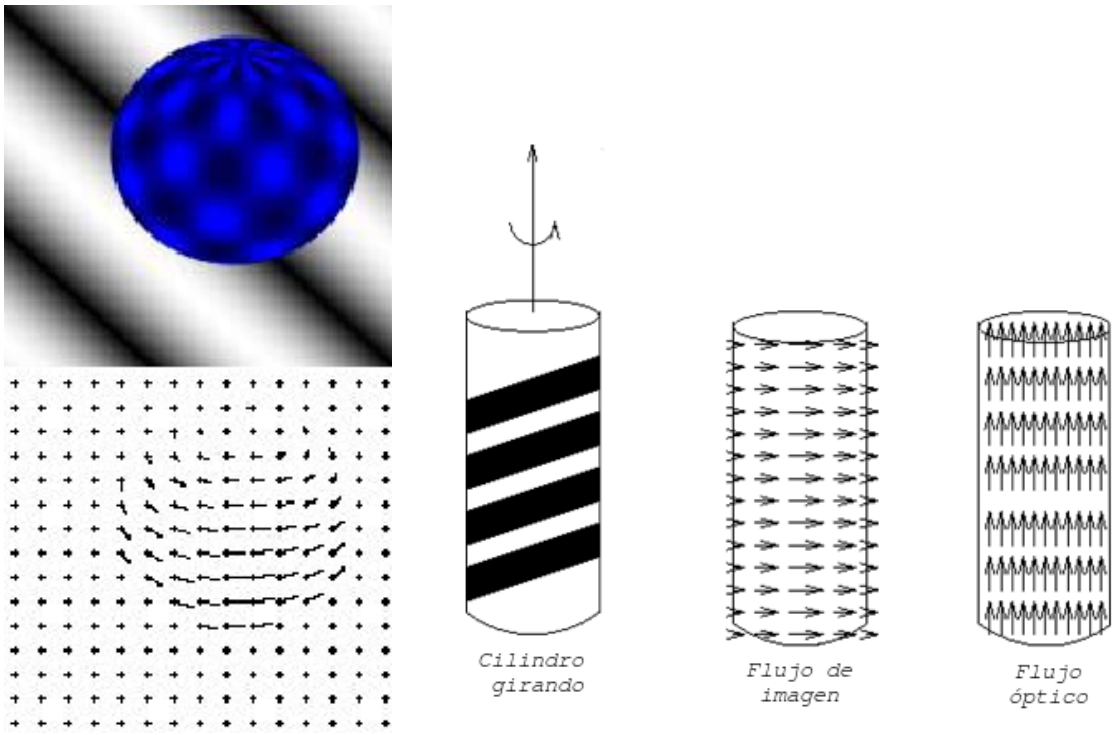
4.1. Fundamentos teóricos

El flujo óptico es una de las formas de extraer información de las imágenes. Para entender qué es el flujo óptico y qué propiedades tienen, hay que entender antes varios conceptos [Sánchez *et al.*, 2006]:

- Campo de movimiento: es el campo de velocidades tridimensionales de cada objeto en todos los puntos del espacio.
- Flujo de imagen: es la proyección bidimensional del campo de movimiento sobre el plano de la imagen.
- Flujo óptico: campo de velocidades en la imagen asociadas a los cambios de intensidad. Se puede interpretar en muchos casos como el movimiento de los objetos en la imagen, pero en algunos casos el flujo óptico se produce por el movimiento de la fuente de luz ya que es el movimiento de los puntos de intensidad luminosa lo que define el flujo óptico.

  Un ejemplo típico de este problema es una esfera sin textura girando. Evidentemente existe movimiento, pero no se registra flujo óptico ya que la intensidad luminosa de la esfera no varía. El caso opuesto se puede ejemplificar con una esfera estática (también si textura), pero cuya iluminación es móvil. En este caso se percibe flujo óptico sobre la esfera, a pesar de estar estática, porque ha habido un cambio en la luminosidad de la misma a causa del movimiento de la fuente de iluminación. Otro ejemplo que muestra las diferencias entre flujo óptico y flujo

de imagen se percibe en la figura 4.1(b)



(a) Flujo óptico de una esfera con textura dando vueltas

(b) Diferencia entre flujo óptico y movimiento real

Figura 4.1: Flujo óptico

Existen dos grandes familias de algoritmos que permiten calcular el flujo óptico, a continuación se presentan los fundamentos teóricos de ambas.

4.1.1. Método basado en gradiente

El método basado en gradiente parte de la suposición de que un punto de la imagen en movimiento mantiene alguna propiedad constante en el tiempo, en el caso que nos interesa se considera que mantiene su luminosidad constante. Es decir:

$$\frac{dI}{dt} = 0 \tag{4.1}$$

Teniendo en cuenta esta suposición, y tal y como se dice en [Horn y Schunck, 1981], se puede hacer una estimación del flujo óptico, representado como el vector $V(v_x, v_y)$, basándose en las derivadas parciales:

$$\frac{\delta I}{\delta x}v_x + \frac{\delta I}{\delta y}v_y + \frac{\delta I}{\delta t} = 0 \tag{4.2}$$

La ecuación 4.2 explica matemáticamente que si un punto cambia su luminosidad, es porque ha habido movimiento. Siempre que un punto varíe su luminosidad, la derivada espacial de la luminosidad en ese punto será distinta de cero.

Con esta ecuación, como se ha comentado anteriormente, solo se percibirá el movimiento cuando éste conlleve un cambio de luminosidad. Por esto es importante elegir bien los puntos, si se elige para calcular el flujo óptico un punto sin textura, aunque se produzca un movimiento real, no se podrá calcular el flujo óptico porque no se registrará un cambio en la luminosidad del punto. Habrá que determinar qué puntos son relevantes para el cálculo del flujo óptico y qué puntos no lo son. Generalmente, los puntos esquina son buenos puntos de interés porque sí se percibe el cambio de intensidad al haber movimiento en ellos. Hay que tener especial cuidado con los puntos borde, ya que resulta imposible determinar la dirección del movimiento. En estos casos hay que contar con algún dato más para suponer la dirección del movimiento. Este es el *problema de la apertura* un problema muy común al intentar calcular el flujo óptico en puntos borde en los que no se tiene una perspectiva suficientemente grande como para saber la dirección del movimiento (en la figura 4.2 se ilustra este problema. Pese a que el movimiento se produce hacia arriba y hacia la derecha, sólo viendo el interior del círculo parece que el movimiento sólo ha sido hacia arriba).

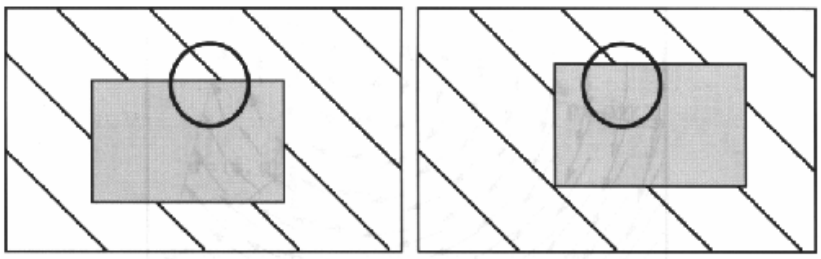


Figura 4.2: El problema de la apertura

En el caso del método basado en gradiente, se supone que la el vector del flujo óptico en los puntos borde tiene la dirección del gradiente, esto es, perpendicular al borde. Es sencillo realizar esta suposición dado que la ecuación 4.2 ya contiene las derivadas parciales que definen el gradiente.

4.1.2. Método basado en correspondencias

En [Lucas y Kanade, 1981] se propone, para calcular el flujo óptico, suponer que el entorno del punto en el que se está realizando el cálculo tiene el mismo movimiento; es decir, que el entorno del punto se mantiene constante. El flujo se calcula, en este caso, mediante correspondencias. Se crea para ello una ventana alrededor del píxel que define su entorno y se intentará encontrar su ventana correspondiente en la imagen siguiente. La ecuación 4.3 realiza una minimización de las diferencias de cada píxel con su supuesto correspondiente. La ventana de correspondencia que minimice la función será la que defina el destino del flujo óptico hacia donde se ha movido el punto original en la imagen siguiente.

$$\Sigma(W(x,t)(\nabla I(x,t)v) + I_t(x,t))^2 \tag{4.3}$$

Otra forma de explicar la ecuación 4.3 es decir que la ecuación calcula la diferencia de la ventana original del píxel, con todas las posibles ventanas de la imagen siguiente en un entorno acotado del punto. La correspondencia será con la ventana con la que haya habido una diferencia menor.

Una vez más, aparece el problema de la apertura, ya que los puntos de borde pueden tener varias correspondencias. En este caso el flujo óptico apuntará hacia el píxel que esté centrado en la ventana que minimice las diferencias, pero, para las ventanas que sólo incluyan bordes, las diferencias serán muy pequeñas y no se podrá asegurar que la dirección del movimiento sea la que se está indicando. Por este motivo conviene calcular el flujo óptico sólo en puntos de interés (es decir, esquinas).

4.2. Esquema *opflow*

En esta sección se explica la implementación del módulo que se ha realizado para calcular el flujo óptico. El esquema *opflow*, apoyado en la funcionalidad de la biblioteca IPP (ver apartado 3.3), implementa un algoritmo que calcula el flujo óptico utilizando el método basado en correspondencias descrito en el apartado anterior. El cálculo se realizará de forma continua en el tiempo, utilizando para para ello pares de imágenes (imagen nueva e imagen anterior) que se van renovando en cada iteración (la nueva pasa a ser vieja y la imagen recién capturada pasa a ser la nueva).

Este esquema permitirá que otras aplicaciones obtengan la señal de flujo óptico sin necesidad de calcularla, simplemente accediendo a la variable compartida que ofrece el esquema. De esta forma se podrá incluir de forma sencilla en otras aplicaciones.

4.2.1. Entradas y salidas

Dentro de las posibles funciones de un esquema de *Jdec*, *opflow* se encarga de generar señales perceptivas procesadas a raíz de la información recibida por los sensores.

Como señal de entrada, recibe el flujo de imágenes de la cámara. La información de las imágenes viene dada por la plataforma *jdec* de acuerdo con el formato establecido (Ver en el apartado 3.1 los drivers que exportan imágenes).

Además, el usuario puede introducir una máscara para que el esquema sólo calcule el flujo óptico en ciertas regiones deseadas. La máscara es una imagen de tres bytes por píxel puestos al valor máximo, si se quiere calcular el flujo óptico en ese píxel; o al valor mínimo, si no se quiere calcular. Se puede acceder a ella a través de la variable compartida *mask* que ofrece el esquema.

```
typedef struct{
   int calc;
   unsigned char status;
   float error;
   floatPoint dest;
   float hyp;
   float angle;
}t_opflow;
```

Cuadro 4.1: Tipo de datos de la variable de salida de *opflow*

Opflow procesa esta señal de imagen y proporciona como resultado una imagen de flujo(t), a raíz de las imágenes (t) y (t-1), dónde cada píxel se describe con la estructura *t_opflow* representada en la tabla 4.1. Cada campo de esta estructura indica:

- ***calc*:** marca que indica si se ha calculado el flujo óptico en ese píxel. En caso de estar calculado, se marca ese campo con un *1* y el resto adquieren significado; en otro caso, deberán ser ignorados siendo su valor indeterminado.

- ***status*:** este campo señala la capa en la que se ha encontrado la última correspondencia [1] (Para entender mejor este parámetro ver el apartado 4.2.4).

- ***error*:** muestra el índice de error que tiene el cálculo realizado para ese píxel, indica las diferencias de la ventana original del píxel, con la ventana de correspondencia. A mayor error, menor fiabilidad.

- ***dest*:** es una estructura de datos que se utiliza para representar puntos. Tiene dos campos, dos números reales que indican: la coordenada x y la coordenada y del punto. En este caso, el valor de *dest* será la posición del punto que tiene correspondencia en la imagen siguiente con el punto que se está estudiando.

- ***hyp*:** para facilitar las operaciones se realiza una conversión a coordenadas polares del punto de destino. *hyp* contiene el valor al cuadrado del módulo de esas coordenadas polares.

- ***angle*:** muestra el valor del ángulo que corresponde con las coordenadas polares del punto de destino del flujo óptico.

4.2.2. Captura de señales de entrada

El esquema, en primer lugar, captura la imagen que *Jdec* ofrece como un array. Se copia este array en un *buffer* propio del esquema y además se guarda la señal de luminancia en otro *buffer* local (con esto se consigue que las imágenes sobre las que se calcula el flujo óptico no cambien durante el cálculo, además de guardar una copia para tener "imagen anterior" en la próxima iteración).

Cada píxel de la señal de luminancia se multiplica por su correspondiente en la máscara, esto hace que los puntos en los que no se desee calcular el flujo óptico queden negros de forma uniforme; y consecuentemente no se generen puntos relevantes en esas regiones. Será esta señal, la de *luminancia con la máscara aplicada*, la que se utilice *para calcular el flujo óptico*. Además, para reducir tiempos de copia se utiliza doble *buffer*, consistente en un almacenamiento estático manejado mediante punteros que cambian en cada iteración para apuntar al dato concreto. De esta forma, simplemente con cambiar la dirección de destino de los punteros, se consigue el mismo efecto que copiando los datos de una zona de memoria a otra. Así, es sencillo convertir la imagen

[1] Solo se deberán tener en cuenta los valores que han sido calculados en la capa inferior (capa 0), ya que esta se corresponde con la imagen original. El resto no son más que indicios calculados en las capas superiores que no tenían una correspondencia en la imagen real.

nueva en imagen vieja para la siguiente iteración (ver figura 4.3).

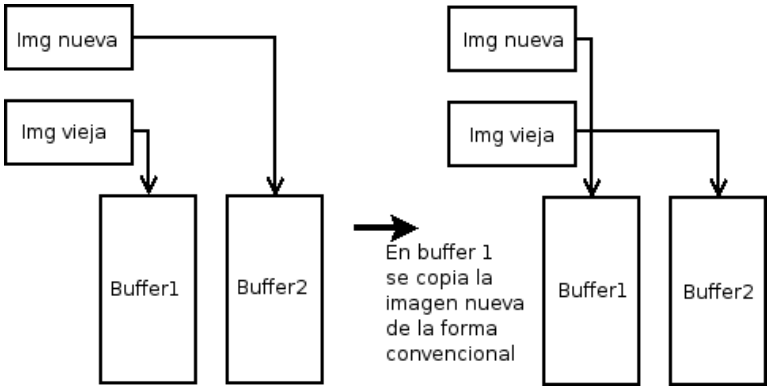


Figura 4.3: Almacenamiento con doble *buffer*

A partir de la segunda iteración, se puede comenzar el cálculo, ya que se dispone de dos señales de luminancia consecutivas: una correspondiente a la imagen actual y otra a la inmediatamente anterior. Partiendo de estas dos señales, se pueden establecer correspondencias para determinar el flujo óptico. Será entonces cuando se empiecen a realizar los cálculos.

4.2.3. Búsqueda de puntos de interés

El segundo paso consiste en encontrar en la imagen puntos relevantes sobre los que calcular el flujo óptico. Se realiza esta operación por dos motivos: el primero, porque como se ha comentado en los apartados 4.1.1 y 4.1.2 el cálculo de flujo óptico es más fiable para ciertos puntos que para otros y el segundo, porque al ser una operación pesada, calculando sólo el flujo óptico en ciertos puntos, se consigue reducir el coste computacional.

Para ello, se realiza un cálculo apoyado en una función de la biblioteca Ipp que calcula para cada punto el autovalor menor asociado a la matriz 4.4, dónde D_x es la derivada parcial con respecto a x de la luminancia, D_y la derivada parcial con respecto a y y el sumatorio se realiza para un entorno del punto a tratar (ventana de 3 o 5 píxeles de lado). El valor resultante de la función 4.4 mide el valor del gradiente espacial en ese píxel, es decir los puntos con una alta derivada espacial (bordes y esquinas) tendrán un valor más alto y en concreto los que tengan los valores mayores (esquinas) serán los que se utilicen para calcular el flujo óptico.

$$\begin{pmatrix} \Sigma D_x^2 & \Sigma D_x D_y \\ \Sigma D_x D_y & \Sigma D_y^2 \end{pmatrix} \tag{4.4}$$

El resultado de este cálculo, según se explica en [Ipp, 2006] y en el párrafo anterior, se puede utilizar para determinar los puntos esquina. En este caso, se realiza una selección de los puntos cuyo autovalor quede entre dos umbrales calculados experimentalmente (*min_eig* y *max_eig*), de ellos se seleccionan los n mayores y se procede al cálculo del flujo óptico sobre esos n puntos (en la figura 4.8(b) se pueden ver los puntos de interés que calcula el esquema).

4.2.4. Pirámides y multirresolución

Ya que el cálculo del flujo óptico es una operación muy costosa computacionalmente, es necesario ganar eficiencia. La multirresolución es una técnica que permite ganar la eficiencia buscada, ayudando a ampliar la ventana de búsqueda sin aumentar en exceso la complejidad del algoritmo.

La técnica se basa en el cálculo recursivo del flujo óptico en imágenes con menor resolución, pudiendo así utilizar ventanas de búsqueda más pequeñas y ahorrando cómputo.

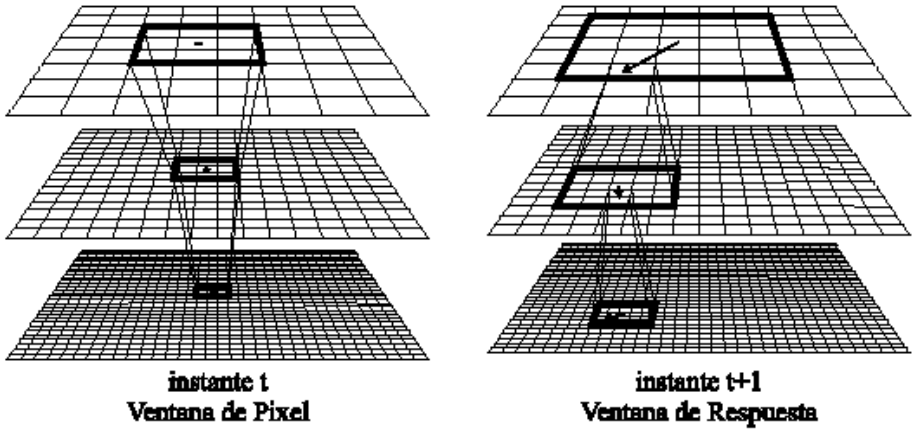


Figura 4.4: Cálculo del flujo óptico con multirresolución

El cálculo comienza en la imagen con menor resolución (con menos píxeles). El resultado de calcular el flujo óptico se traslada a la siguiente capa, en la que se proce-

derá al cálculo del flujo óptico centrando la ventana de búsqueda en el resultado de la capa inmediatamente superior (figura 4.4).

Al realizarse los primeros cálculos en una imagen con menos resolución, la ventana cubre más imagen, pero el cálculo del flujo óptico resulta menos preciso. A medida que se va descendiendo a capas con mayor resolución, el cálculo se vuelve más preciso. El ahorro de cómputo se produce al reducir la ventana de búsqueda, porque en las capas con menor resolución se dan los indicios de movimientos grandes, y luego se van afinando en las capas con mayor resolución.

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). La biblioteca *IPP* define una implementación de estas estructuras y aporta funciones para reducir el tamaño de imágenes y así rellenar las capas. En cada iteración, hay que reservar memoria dinámicamente según las capas que se vayan a utilizar e ir calculando las imágenes con menor resolución a partir de las imágenes originales.

4.2.5. Algoritmo de Lukas y Kanade

Una vez obtenidas las pirámides, se procede a llamar a la función que calcula el flujo óptico. Esta función [Ipp, 2006] implementa el algoritmo iterativo de Lukas y Kanade, que busca correspondencias entre las imágenes aplicando para ello el método de mínimos cuadrados y apoyándose en la multirresolución (ver apartados 4.1.2 y 4.2.4). Recibe como argumentos: las dos pirámides, un listado con los puntos sobre los que calcular el flujo (los puntos de interés calculados anteriormente) y una serie de parámetros que afinan el funcionamiento del algoritmo, como por ejemplo el número de iteraciones por cada punto (*num_iter*), cuantas capas de las pirámides se utilizarán (*layers*) o el umbral mínimo para saber si se ha encontrado correspondencia.

Al finalizar la ejecución de esta función se obtiene: un listado con las coordenadas de los puntos correspondientes a los puntos de interés en la imagen actual (el punto que corresponde con el punto de interés en la imagen nueva), otro que indica el grado de diferencia entre la ventana del píxel en la imagen anterior y su correspondiente en la imagen nueva (una medida del error de cálculo) y un tercero que indica en qué nivel de la pirámide se encontró la correspondencia descrita (ver campo *status* de los píxeles

de la imagen de flujo en el apartado 4.2.1).

Después de calcular el flujo óptico, se libera toda la memoria que se ha reservado para guardar la pirámides.

4.2.6. Resultados

Posteriormente, se procede a la inserción de los resultados obtenidos en el tipo de datos devuelto por el esquema. Para ello, en primer lugar, se inicializa la estructura que guarda los datos, indicando para cada punto que no se ha calculado el flujo óptico. A continuación, se revisan los datos para ver en qué puntos se ha calculado el flujo óptico y para cada uno de ellos se comprueba si el movimiento supera un umbral mínimo (*min_mov*) y si el error queda por debajo de otro umbral (*max_err*). En caso de cumplirse las condiciones descritas, se rellena la estructura de datos para el píxel origen del movimiento con los datos descritos en la tabla 4.1, para ello se copian los datos y se calculan las coordenadas polares. De esta forma queda indicado para los demás esquemas este movimiento.

4.2.7. Interfaz gráfica

Para que el usuario pueda ver los resultados y realizar pruebas de configuración de forma rápida y cómoda, se ha puesto a su disposición una interfaz gráfica. En la figura 4.5 se puede observar esta interfaz. La plataforma permite que se pueda activar y desactivar la interfaz gráfica de todos los esquemas a voluntad, de esta forma se ahorra cómputo.

El elemento más característico de la interfaz es el display que muestra la imagen de entrada. Con los botones que hay debajo de ella, el usuario puede hacer que se muestre el flujo óptico[2] (representado con flechas[3]), los puntos de interés calculados para esa imagen (representados como puntos azules) y la máscara (representada como regiones negras en la imagen).

[2]En realidad las flechas que se muestran son el doble de largas que el flujo óptico al que representan para que se vean mejor en la representación.

[3]Las flechas verdes indican que el error está por debajo de la mitad del error máximo permitido, el resto serán rojas.

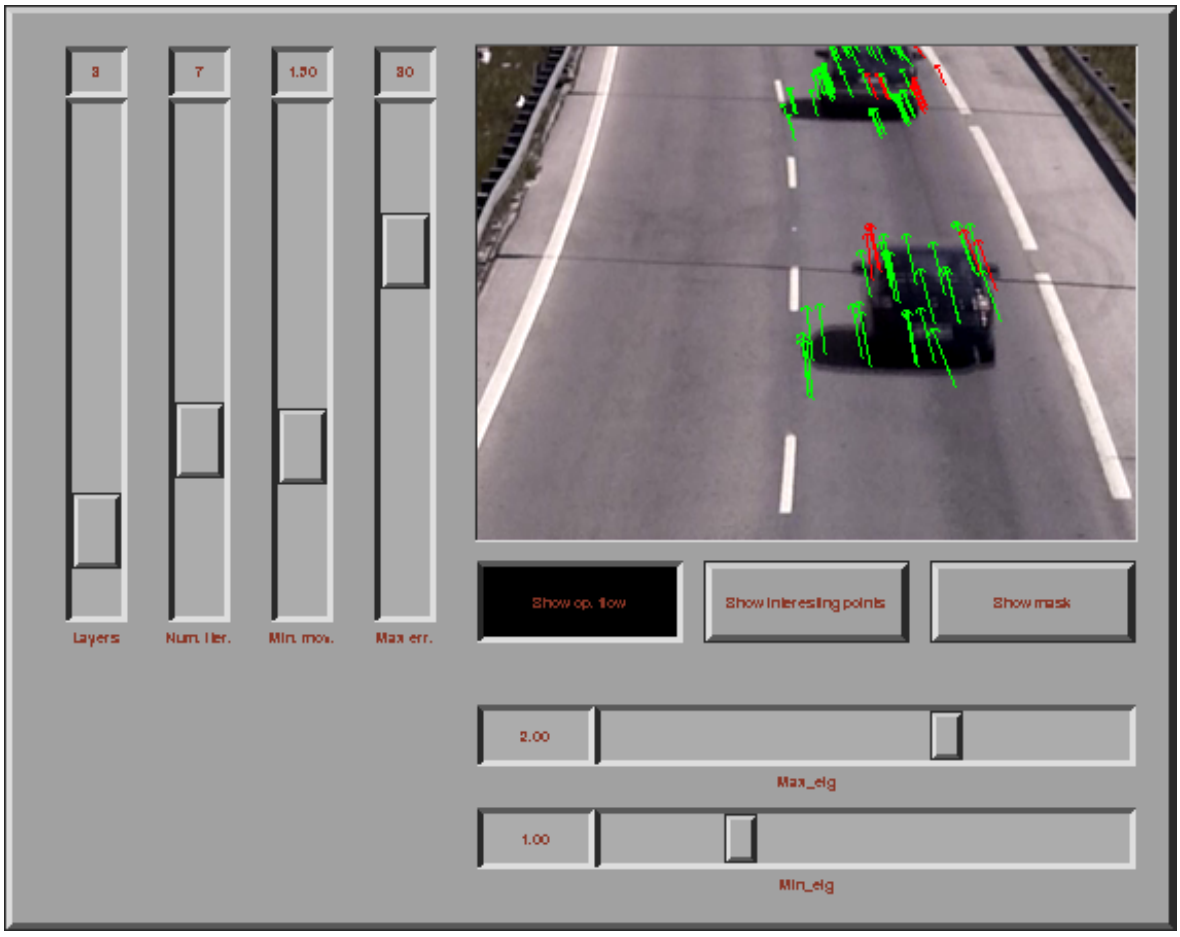


Figura 4.5: Interfaz del esquema opflow

Además, la interfaz permite al usuario manipular ciertos parámetros que varían el funcionamiento de los algoritmos, estos son:

- **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 pasarlo a los demás esquemas.
- **Max err.**: modifica el valor máximo de error permitido para notificar a los demás esquemas y mostrar en el display.
- **Max eig y min eig**: indican los valores máximos y mínimos de los autovalores para buscar los puntos de interés (Para más detalles ver 4.2.3).

4.3. Experimentos

Una vez descrito el funcionamiento del esquema *opflow* hay que comprobar que cumple los requisitos deseados, es decir, comprobar que el esquema calcula el flujo óptico de forma robusta independientemente de las condiciones externas (cambios de iluminación, movimientos bruscos, movimientos de cámara, etc.). También hay que comprobar que funcione en tiempo real, que consuma poco tiempo de cómputo y que la fiabilidad sea alta (poco error en los resultados).

Para todos los experimentos se tomará como medida de velocidad las iteraciones por segundo y el porcentaje de carga de CPU en la máquina de pruebas (Procesador AMD Athlon XP 2800+ con 1Gb de memoria RAM DDR Dual Channel).

4.3.1. Ajuste de parámetros

La primera prueba que se llevó a cabo (antes de finalizar completamente el esquema) fue la de cambiar los parámetros que definen como realizar el cálculo del flujo óptico, probando diferentes valores para cada unos de ellos. Con estas pruebas se consiguió ajustar el algoritmo para que el funcionamiento fuese mucho más rápido y preciso.

El parámetro principal que hay que ajustar es el de la frecuencia de cálculo del propio esquema, ya que una frecuencia muy alta puede provocar que las diferencias

entre las imágenes sean demasiado pequeñas y que el flujo real no se pueda distinguir del producido por el ruido de la imagen, además conlleva un coste computacional muy alto (y se dejaría de cumplir uno de los requisitos). Por el contrario, una frecuencia excesivamente baja causa que las diferencias sean demasiado grandes, perdiendo movimientos y posibles correspondencias porque el objeto desaparezca de la imagen (en este caso no se podría afirmar que funciona en tiempo real ni que los resultados son fiables). Tras varios experimentos se pudo comprobar que un buen equilibrio entre resultados y coste computacional es el que está en el intervalo que va de las 10 a las 15 iteraciones por segundo[4]. Con menos de 10 los resultados empeoraban sensiblemente, y con más de 15 el coste computacional aumentaba demasiado, no percibiéndose, además, mejores resultados. El valor que se dejó por defecto para todas las demás prueba es de 13 ips.

También son muy importantes otros parámetros propios de este algoritmo de cálculo del flujo óptico:

- el tamaño de la ventana
- el número de iteraciones máximo del algoritmo en cada capa
- el umbral de diferencia mínima para el que se considera encontrada la correspondencia
- el número de capas de las pirámides
- el número de puntos sobre los que calcular el flujo óptico

De todos ellos, el tamaño de la ventana de correspondencia y el número de capas (*layers*) son parámetros muy dependientes y hay que ajustarlos uno en función del otro. Por ejemplo, si el tamaño de la ventana es muy grande, tener muchas capas no será útil, porque ya se está comprobando la correspondencia en puntos suficientemente alejados de la imagen, además es posible que la ventana no quepa dentro de las capas superiores. En concreto, se observó que un valor de ventana grande (mayor de 15 de lado) hace que el coste computacional del algoritmo sea muy elevado, ya que se tiene que comprobar la correspondencia en muchos más puntos, en consecuencia el coste computacional aumenta del 50 % al 80 %. Después de varias pruebas, se comprueba que la mejor opción es tener un valor de ventana en torno a 9 píxeles de lado y, que para

[4]Es importante reseñar en este punto que no se realizaron estos experimentos de manera única, sino que se llevaron a cabo a medida que se ajustaban el resto de parámetros, los resultados que se muestran aquí son aquellos con los que se obtuvieron mejores resultados con todos los parámetros bien configurados

que se puedan percibir movimientos más grandes, se utilice la multirresolución (ver apartado 4.2.4) con pirámides de 3 a 5 capas[5], más capas producen que no se encuentre flujo óptico ya que la ventana se vuelve demasiado grande para las capas superiores.

En cuanto al resto de parámetros, cabe destacar que elevar demasiado el número de iteraciones máximas por capa (*max_iter*), a más de 20, así como un valor del umbral muy bajo hacen que aumente el tiempo de ejecución, la aplicación vuelve a no cumplir las características, ya que no deja tiempo de cómputo a otras aplicaciones, consumiendo en torno al 80 % del tiempo de cómputo. Valores apropiados para esos parámetros son 7 para el número de iteraciones, y para el umbra un valor de 0,3.

Por último, es importante comentar la decisión sobre el número máximo de puntos en los que calcular el flujo óptico. De la cantidad de puntos sobre los que se calcule el flujo óptico dependerá la exactitud de los datos, pero si este número es demasiado alto nuevamente existen problemas de rendimiento temporal, un valor que encuentra el equilibrio es 300.

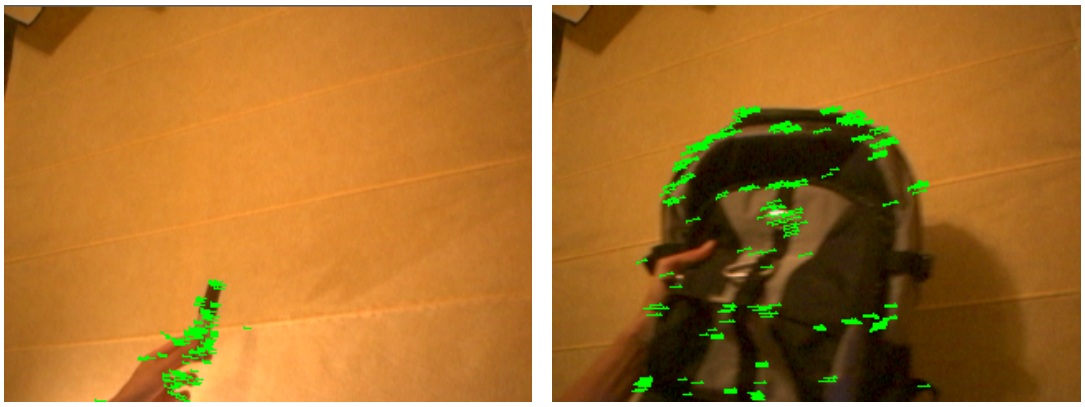
Para los parámetros que configuran los puntos de interés, *eig_min* y *eig_max*, se comprueba que si el valor máximo es demasiado alto, los puntos de interés se concentrarán en regiones con bordes muy marcados, quedando otras regiones de la imagen olvidadas. En el caso de que no existan este tipo de regiones en la imagen, el máximo puede no estar definido. En cuanto al valor mínimo, no tiene que ser excesivamente bajo porque podrían incluirse como puntos interesantes algunos que no son realmente bordes o esquinas y esto produciría que el flujo óptico calculado para esos puntos no sea correcto (evitar el problema de la apertura, ver 4.1.1), aunque un valor elevado de este parámetro produce que se descarten muchos puntos. Valores de 1,0 y 2,0 respectivamente funcionan bien en casi todos los entornos.

4.3.2. Pruebas de funcionamiento

Ahora que se conoce el efecto de cada parámetro en los resultados obtenidos y se han configurado para que los resultados sean óptimos, habrá que realizar pruebas de funcionamiento.

[5]Este parámetro se puede modificar mediante la interfaz de usuario para que el programa pueda adaptarse a entornos con movimientos grandes y pequeños, aunque, con tres capas y una ventana del tamaño indicado, ya es suficiente para el formato de imagen que se utiliza

En un principio se realizaron pruebas de funcionamiento teniendo como entrada la cámara iSight (Ver 3.5). Se probó la cámara produciendo movimiento en la imagen tanto de objetos pequeños como de objetos grandes, alejados de la cámara y cerca. Para todos estos experimentos se obtuvieron resultados satisfactorios. Si bien cuando los objetos son muy pequeños o están muy alejados el esquema detecta peor el flujo óptico, ya que dispone de menos puntos de interés por objeto y además realizar correspondencias es más complicado. Pero aún así, como se observa en la figura 4.7, los resultados son satisfactorios.



(a) Flujo óptico de un elemento pequeño

(b) Flujo óptico de un elemento grande

Figura 4.6: Experimentos con el esquema opflow (movimiento de objetos grandes y pequeños)

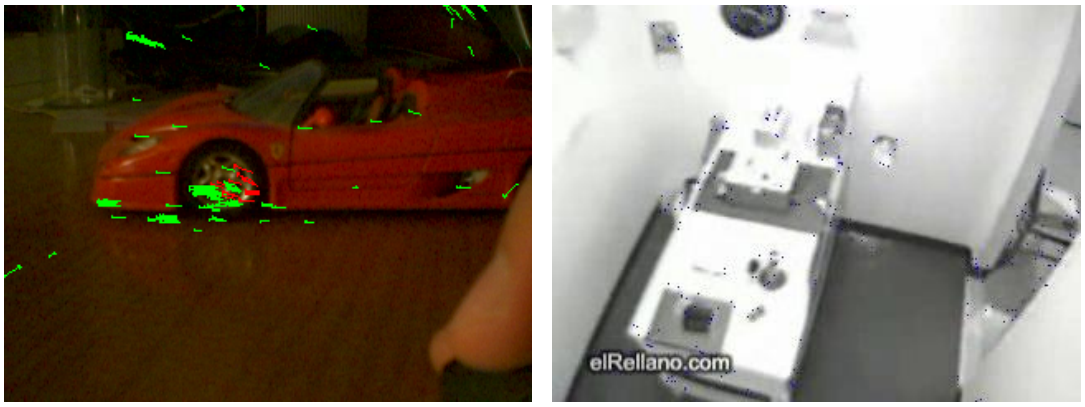
Otra prueba interesante consiste en mover la cámara, tanto en entornos estáticos como dinámicos. Mover la cámara en entornos estáticos permite observar qué objetos están más alejados y cuáles más cerca, ya que los objetos cercanos al moverse la cámara aparentemente se desplazan más que los lejanos. Teniendo en cuenta este fenómeno, se pueden hacer estimaciones de profundidad con una sola cámara.

También se comprobó el funcionamiento correcto en entornos con diferentes iluminaciones, observando que los resultados eran mejores cuanto mejor era la iluminación. El problema fundamental es que al bajar la iluminación se suele reducir el contraste y es precisamente esa bajada de contraste en la señal de luminancia la que perjudica al cálculo del flujo óptico porque, al tener toda la imagen un todo similar, es difícil encontrar correspondencias al igual que asegurar que la correspondencia está en ese punto concreto y no en otro. En la figura 4.8(a) se puede observar que detrás del coche

(a) Flujo óptico de una persona alejada (b) Movimiento de la cámara en un entorno estático

Figura 4.7: Experimentos con el esquema opflow (movimiento de objetos grandes y pequeños)

(único elemento en movimiento), en la zona oscura se genera flujo falso debido a esa falta de contraste.



(a) Flujo óptico con poca iluminación (b) Puntos de interés

Figura 4.8: Experimentos con el esquema opflow (poca iluminación y puntos de interés)

Para ahorrar cómputo se intentó evitar el cálculo de los puntos de interés en cada iteración. Para ello se utilizaban los puntos de destino del flujo óptico en una iteración como nuevos puntos relevantes en la siguiente, ya que estos puntos son correspondientes a los puntos de interés en la imagen nueva. Si bien este razonamiento es completamente correcto, surge un problema. No todos los puntos de interés se mueven o tienen una correspondencia correcta. Siguiendo este razonamiento, va descendiendo el número de puntos que realmente son de interés incluidos en ese grupo, quedando en pocas iteraciones casi anulado. Para solventar este problema, al menos en parte, se buscaban de nuevo los puntos relevantes cada 5 o 6 iteraciones (antes de que se anulasen completamente).

Pero, a pesar de todo, la fiabilidad de la aplicación descendía muchísimo, llegando a aportar resultados escasos y llenos de falsos positivos, mientras que el rendimiento no mejoraba visiblemente. Por este motivo se decidió abandonar este procedimiento y recalcular los puntos de interés en cada iteración.

4.3.3. Implementaciones alternativas

Antes de decidir utilizar la biblioteca IPP se realizaron aproximaciones utilizando otros métodos. Para la primera aproximación, basada en el gradiente, se utilizó como base unos experimentos anteriores de otro compañero. El programa realizaba derivadas parciales, tanto espaciales como temporales, y combinando todas ellas realizaba una estimación del flujo óptico similar a la que se describe en 4.1.1. En este caso, la velocidad aparente se calculaba mediante las fórmulas 4.5 y 4.6:

$$v_x = \frac{-\frac{\delta I}{\delta t}\frac{\delta I}{\delta x}}{|\nabla|^2} \tag{4.5}$$

$$v_y = \frac{-\frac{\delta I}{\delta t}\frac{\delta I}{\delta y}}{|\nabla|^2} \tag{4.6}$$

Este método, pese a ajustarse a la teoría, aportaba demasiados datos falsos; además de ser especialmente delicado ante el problema de la apertura. A pesar de que se realizaban los cálculos en puntos esquina, para evitar en lo posible este problema, los malos resultados seguían persistiendo. Por eso se decidió abandonar el método basado en gradiente y realizar una implementación fundamentada en el método basado en las correspondencias locales (ver 4.1.2), que es el que finalmente se ha utilizado en el algoritmo.

Otra implementación apoyada en la biblioteca OpenCV también llegó a ser funcional y a mostrar resultados satisfactorios. Pero, dado que el rendimiento era más bajo (no permitía tiempo real dejando tiempo de cómputo a otras aplicaciones, ya que el consumo de CPU estaba en torno al 85 % cuando se ejecutaba el algoritmo a 8 iteraciones por segundo). Además había que desperdiciar mucha memoria para hacer la adaptación del modo de trabajo con imágenes de OpenCV al de *Jdec* (además de los motivos indicados en 3.3) se decidió utilizar Ipp.

Capítulo 5
Aplicaciones

En este capítulo se describen las aplicaciones de ejemplo que se han desarrollado y que sirven para ilustrar alguna de las formas de utilizar la señal de flujo óptico. Al final del capítulo 1 se mencionan algunas posibles aplicaciones que se pueden apoyar en el flujo óptico, entre ellas: navegación de un robot basada en el flujo óptico, atención visual (mirar hacia dónde se percibe movimiento), etc. En este caso, se realizan dos aplicaciones concretas: una aplicación capaz de teleoperar el robot según los movimientos capturados por la cámara (*eyeoperator*) y la otra capaz de contar los coches que pasan por una carretera (*cuenta-coches*).

5.1. Aplicación *eyeoperator*

Bajo el nombre *eyeoperator* se agrupan dos aplicaciones que permiten teleoperar el robot usando como única comunicación entre la persona y la máquina las imágenes que se capturan por la cámara. Los movimientos de la persona son interpretados por la aplicación y transformados en órdenes para el robot. La diferencia entre las dos es la forma en que interpretan los movimientos capturados.

Ambas aplicaciones se basan en el esquema *opflow*, utilizando la señal de flujo óptico para conseguir su funcionalidad. Sirviendo de nuevo como ejemplo de un posible uso de la señal de flujo óptico.

5.1.1. Diseño general

Desde el punto de vista del usuario los programas son sencillos. Éste simplemente tiene que situarse frente a la cámara y observará su propia imagen en la pantalla, sobre la que se habrán superpuesto los límites de cierto número de regiones, dependiendo de la aplicación serán:

- Una región: en este caso el usuario utilizará la aplicación del mismo modo que utilizaría un ratón de ordenador portátil (*touchpad*) realizando movimientos dentro de la región marcada.

- Cuatro regiones: utilizando esta aplicación la forma de manejar el robot es más similar a manejarlo con un *joystick*. Cada una de las cuatro regiones representa un posible movimiento: avance (en la región superior), retroceso (en la región inferior), rotar a la derecha (en la región de la derecha), o hacia la izquierda (en la región situada a la izquierda de la pantalla). Realizar movimientos en cada una de las regiones, equivale a mover el *joystick* hacia ese lado. Además el efecto causado será acorde al movimiento medio detectado en la región, a mayor desplazamiento, mayor efecto.

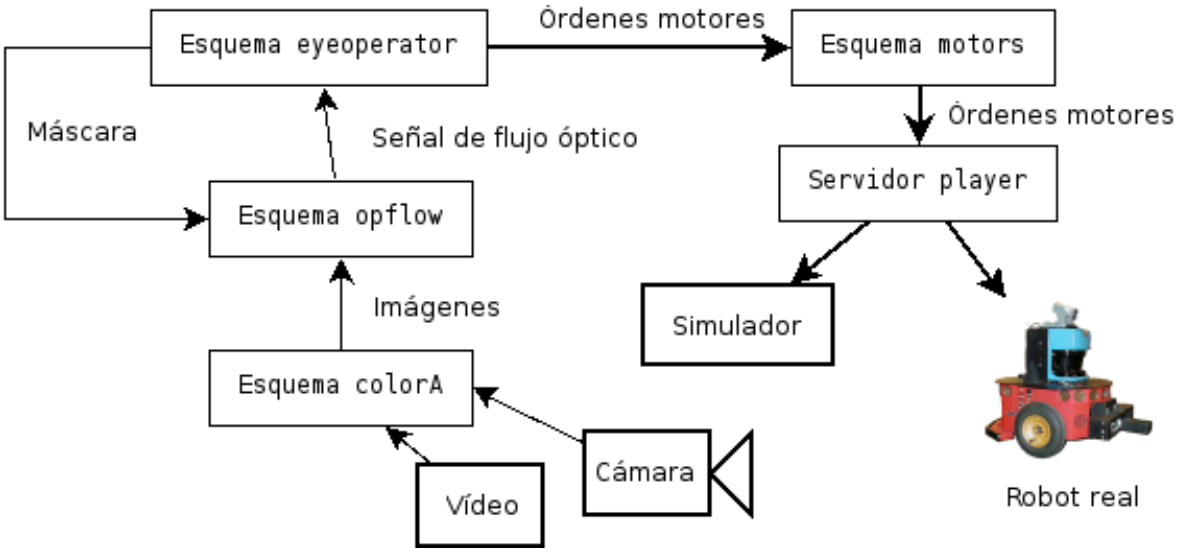


Figura 5.1: Esquema de bloques de la aplicación *eyeoperator*

Algo muy importante a tener en cuenta, y que esta aplicación tendrá que hacer, es la comunicación con un robot (real o simulado). Para ello, la aplicación utilizará las facilidades que ofrece la plataforma *jdec*. La aplicación está formada por varios esquemas que realizan tareas independientes y que encapsulan su funcionamiento (ver figura 5.1). En este caso el esquema, *colorA* proporciona imágenes (sin importar la fuente: cámara *firewire*, cámara USB o vídeo) por medio de una variable compartida. Estos datos de imágenes son tomados por el esquema *opflow* que calcula el flujo óptico de la secuencia de imágenes que recibe de *colorA*. Con los datos del flujo óptico, el esquema *eyeoperator* calcula los movimientos que tiene que realizar el robot y, mediante las

variables compartidas que le proporciona el *motors*, ordena su realización. *Motors* se relaciona por medio de una conexión de red con la aplicación *player* que se encarga de manejar el robot o el simulador y de esta forma llevar a cabo los movimientos que ordenó *eyeoperator*.

5.1.2. *Eyeoperator* con una región

Restricción de zonas

Antes de comenzar ningún cálculo se debe especificar en qué regiones se quiere calcular el flujo óptico. Para ello se utiliza la máscara que se describe en el apartado 4.2.1. Sobre la imagen que representa la máscara, se define al vuelo una máscara dejando visibles solo los puntos que están dentro de la región de interés.

Generar órdenes para el robot

El esquema *eyeoperator* para esta aplicación, realiza un cálculo para hallar el flujo medio de toda la región. Sumando las componentes x e y de todos los puntos con flujo óptico y dividiendo entre el número de puntos. Se calculan también, a partir de éstos dos valores, las coordenadas polares para dibujar en el display el flujo medio. Y por último se convierten estos datos en desplazamientos.

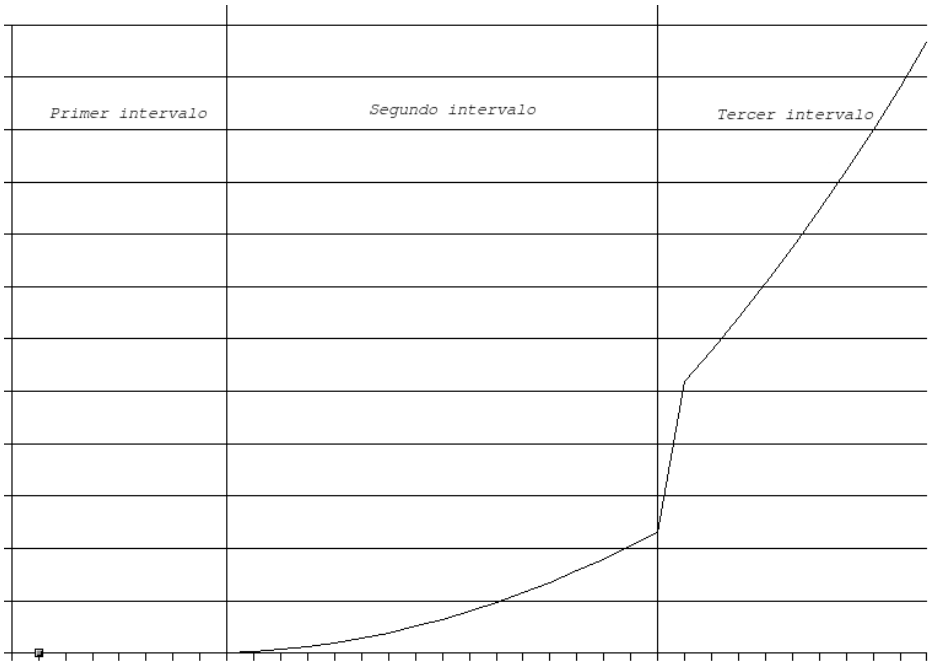


Figura 5.2: Relación entre el flujo (eje x)y la velocidad del robot (eje y)

Para convertir los datos en desplazamientos se utiliza el flujo horizontal como velocidad angular del robot y el flujo vertical como velocidad de avance o retroceso del mismo. Pero no basta con esto, para que el manejo del robot sea más natural se han distinguido tres intervalos en las que la velocidad de giro y de avance se calculan de forma diferente según la región (ver figura 5.2):

- Si está en la *primer intervalo* (la que alberga los valores más bajos), no se realizará desplazamiento.
- En la *segundo intervalo* (que contiene los valores intermedios), la velocidad será el cuadrado del flujo óptico en esa dirección (con el mismo signo que tenía el desplazamiento), para la velocidad de giro, y el cubo del flujo óptico, para la velocidad lineal.
- Por último en la *tercer intervalo* (que contiene todos los valores a partir de un segundo umbral), la conversión será la misma que en la segunda región, pero multiplicando el resultado por un parámetro que haga que el efecto sea mayor.

Utilizar varias regiones de conversión, así como utilizar funciones cuadráticas o cúbicas para realizar los cálculos, hace que los movimientos pequeños sean más precisos, ya que no se producen aceleraciones bruscas por un cambio pequeño en el flujo óptico, y que los movimientos más grandes sirvan para realizar desplazamientos más bruscos, así un aumento pequeño del flujo óptico produce una aceleración grande.

Interfaz gráfica

La interfaz gráfica que ofrece esta aplicación, pese a ser imprescindible es muy sencilla. En este caso no se permite realizar cambios en los parámetros, sino que simplemente muestra la captura de la cámara invertida (como un espejo) añadiendo el cuadrado que delimita el *touchpad*. El fin de poner la imagen invertida es que sea más sencillo para el usuario moverse enfrente de la cámara, ya que de esta forma los movimientos se realizan de forma muy parecida a como se realizan delante de un espejo.

En la figura 5.3 se puede observar la interfaz de esta aplicación.

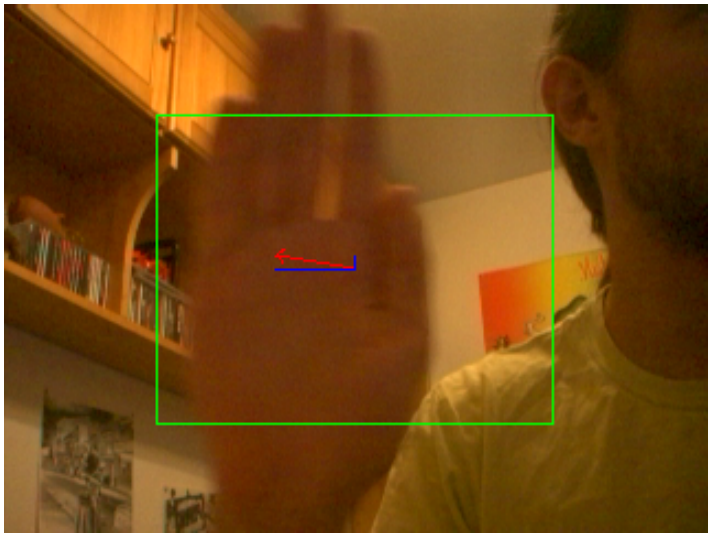


Figura 5.3: Interfaz del esquema *eyeoperator* en la versión con una región

5.1.3. *Eyeoperator* con cuatro regiones

Restricción de zonas

Del mismo modo que en la otra aplicación *eyeoperator*, hay que utilizar la máscara que proporciona *opflow* para dejar visibles sólo las cuatro zonas en las que se desea capturar el movimiento y sólo en ellas calcular el flujo óptico.

Generar órdenes para el robot

En este caso, el esquema *eyeoperator*, calcula el módulo medio del flujo óptico en cada zona, así como una estimación burda del ángulo medio que se utilizará para mostrar en el display una aproximación del flujo medio en cada región. Los cálculos no son más que una media aritmética del valor del módulo de los puntos con flujo dentro de la región.

En este caso, para convertir los datos de flujo medio de cada región en órdenes de velocidad de giro y avance, hay que:

- Sumar el valor de la región superior e inferior para calcular la *velocidad lineal*, comprobando siempre que el módulo medio del flujo óptico de cada región supere un umbral mínimo (bajo el cual se considera el movimiento como ruido). Hay que tener en cuenta que una de las regiones representa la velocidad negativa, así que habrá que cambiarla de signo para sumar.

- Proceder de la misma forma con las regiones laterales para calcular la *velocidad angular*.

Al igual que en el modelo que sólo contiene una región, se han delimitado tres zonas para que el movimiento sea más natural. Se procede al cálculo exactamente igual que en el modelo anterior, tomando como datos de entrada las sumas descritas en el párrafo anterior.

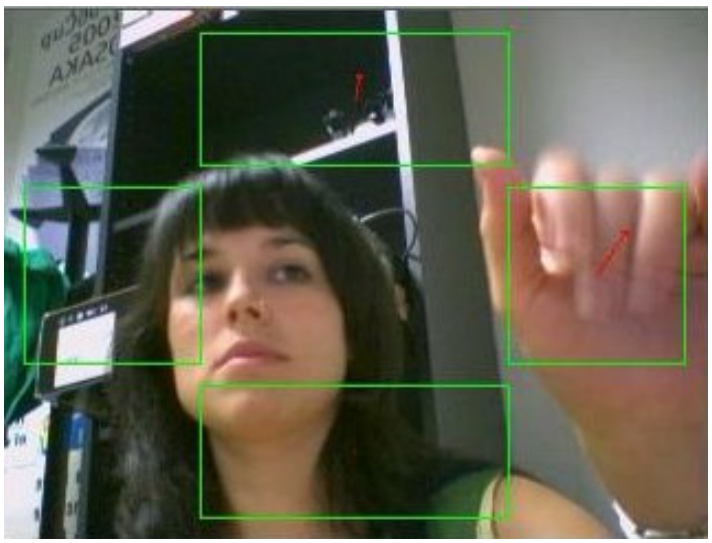


Figura 5.4: Interfaz del esquema *eyeoperator* en la versión con cuatro regiones

Interfaz gráfica

La interfaz gráfica de la aplicación *eyeoperator* con cuatro regiones es también simple, muy similar a la de *eyeoperator* para una región. La imagen capturada por la cámara mostrada en forma de espejo en la que se han pintado las cuatro regiones necesarias para manejar el robot (avance, retroceso y giros).

En la figura 5.4 se puede observar la interfaz del esquema para la versión que posee cuatro zonas.

5.1.4. Experimentos

Para comprobar que todo funciona correctamente, lo primero que hay que conseguir es un robot. En primer momento lo que se utilizó fue un simulador. En concreto se utilizó *player/stage* que permite simular un entorno (*world*) en el que se pueden

emplazar uno o varios robots (ver apartado 3.2). Una vez simulado el entorno y los robots, *player* permite que se conecten los usuarios a cada robot mediante una conexión cliente-servidor.

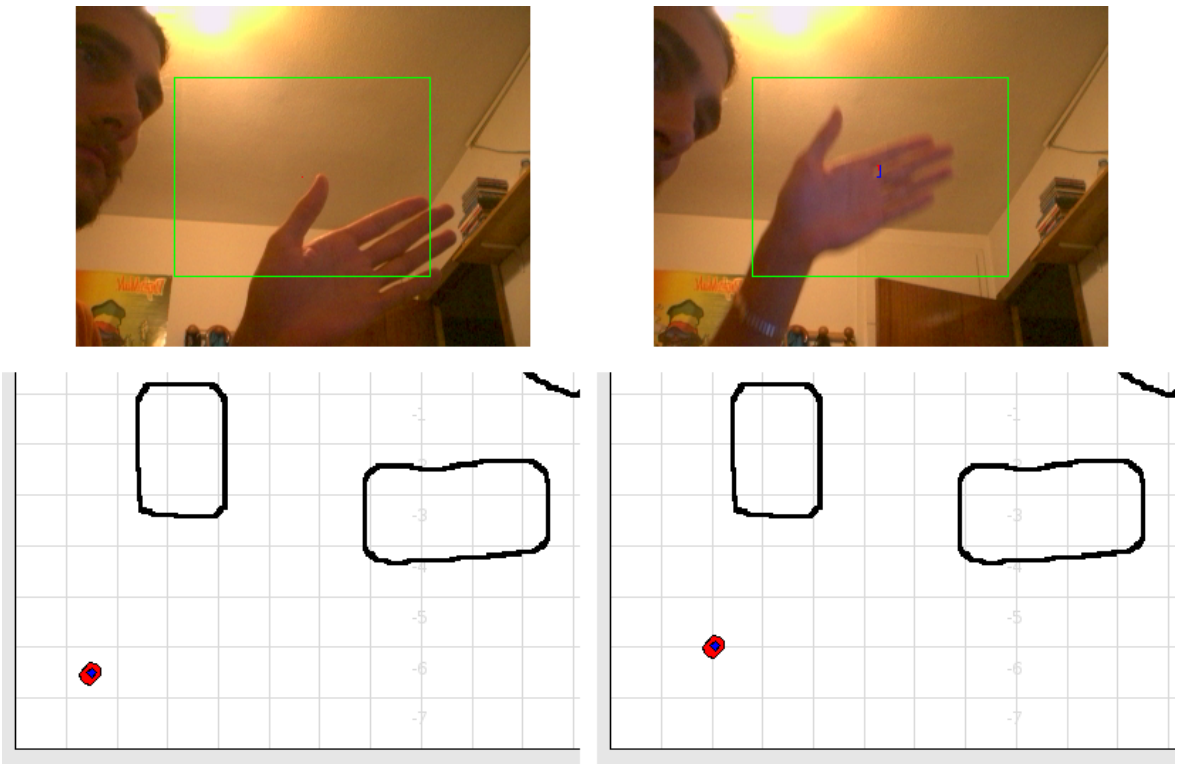


Figura 5.5: Ejemplo de ejecución de la aplicación *eyeoperator* y vista del simulador *stage*

Las pruebas que se han realizado utilizando el simulador han sido satisfactorias, se consigue manejar el robot correctamente, respondiendo siempre a las señales que se pretenden enviar mediante los movimientos. La forma de manejo es intuitiva, ya que es similar a otras interfaces conocidas (como un ratón o un *joystick*), facilitando así el aprendizaje (En la figura 5.5 se puede observar como con los movimientos del usuario se producen desplazamientos del robot en el simulador).

Para terminar el ciclo de experimentos, se decidió utilizar un robot real y realizar los movimientos con él (ver figura 5.6). Para ello, se hizo que los datos del robot real fueran servidos por *player*, que esta vez no aportaba los datos de un simulador (*stage*) sino que se conectaba al robot real por un puerto serie (USB), de esta forma, la aplicación no percibía ningún cambio en su funcionamiento, simplemente player debería adaptarse a interactuar con el robot y no con el simulador. La experiencia de

manejar un robot real con este método fue muy positiva, el robot respondía perfectamente a las indicaciones, incluso de una forma más natural que el robot en el simulador.

Figura 5.6: La aplicación *eyeoperator* manejando el robot real

Una vez más hay que destacar que esta aplicación se apoya en la señal de flujo óptico, lo que conlleva que el funcionamiento correcto de esta aplicación dependerá en gran medida del funcionamiento correcto del esquema *opflow*, por eso la iluminación tiene que ser adecuada para que el esquema pueda aportar flujo óptico acorde a los movimientos del usuario.

Hay que destacar un par de problemas que surgieron al probar la aplicación con una sola región. El primero es que el manejo a veces se vuelve un poco difícil si, entre el fondo y el cuerpo, no se genera suficientes puntos de interés, en ese caso será difícil generar flujo óptico, y en consecuencia se dificultará el manejo del robot. El segundo problema es que los bordes de la máscara provocaban que el cálculo del flujo fuera erróneo, debido al problema de la apertura, ya que estos bordes generan, entre la imagen y la máscara, puntos relevantes y, a veces, flujo paralelo a ellos. Esto produce que el movimiento detectado no sea el que se pretendía. Para resolver este problema se amplió la máscara 10 píxeles en torno a la zona de interés, de esta forma esos falsos positivos quedaban fuera.

En cuanto a la aplicación que dispone de cuatro regiones, cabe destacar que en ocasiones es fácil que se produzcan movimientos en una zona que no se desee, por ejemplo meter el codo en una región lateral al intentar poner la mano en la superior, pero esto no es muy problemático, dado que el umbral de ruido suele amortiguarlos.

5.2. Aplicación *cuenta-coches*

Esta aplicación cuenta los coches que pasan por una carretera, además de realizar un seguimiento de los mismos.

5.2.1. Diseño general

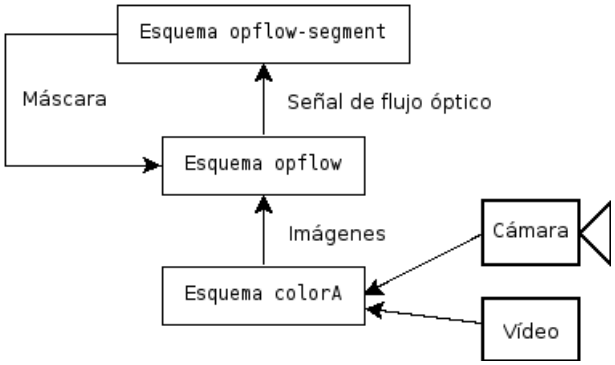


Figura 5.7: Estructura de la aplicación *cuenta-coches*

La aplicación está formada por varios esquemas, que conjuntamente ofrecen la funcionalidad esperada. En la figura 5.7 se puede ver un diagrama con los esquemas que utiliza la aplicación así como los datos que intercambian. Cada uno de ellos cumple una función específica, encapsula y oculta a los demás su funcionamiento, facilitando el trabajo del resto. De esta forma, *colorA* se encarga de conseguir las imágenes, ya sea de un vídeo o directamente de la cámara; *opflow* recibe las imágenes y la máscara para generar la señal de flujo óptico, y por último el esquema *opflow-segment* que toma como señal de entrada la imagen de flujo óptico aportada por el esquema *opflow* y, realizando una segmentación y un seguimiento, indica el número de coches que han pasado hasta el momento por la carretera, además de indicar la posición de cada uno de ellos permanentemente: en la imagen y mediante una estructura como la de la tabla 5.1.

```
typedef struct{
    int xmax;
    int xmin;
    int ymax;
    int ymin;
    int nPto;
    int min_angle; /*deg*/
    int max_angle; /*deg*/
    float mid_angle; /*rad*/
    float min_mov;
    float max_mov;
    float mid_mov;
    int vida;
    int encontrado;
    FL_COLOR color;
}t_opflow_segment;
```

Cuadro 5.1: Tipo de datos que guarda la segmentación en *cuenta-coches*

En la estructura se indican los límites superior, inferior y laterales mediante los campos: *ymin, ymax, xmin* y *xmax*. *nPto* es un contador del número de puntos con flujo óptico que dan soporte a ese segmento. Después, mediante: *min_angle*, *max_angle*, *mid_angle*, *min_mov*, *max_mov* y *mid_mov* se define el movimiento del segmento según el flujo óptico que lo soporta. Por último, *vida* muestra las veces consecutivas que el segmento puede no tener soporte para seguir considerando que existe, *encontrado* se utiliza como contador de las veces que el segmento ha sido confirmado y *color* indica el color con el que será pintado en el display, para que el usuario pueda comprobar que se trata del mismo segmento.

Enmascarar zonas no interesantes

Para que el cálculo de flujo óptico se centre sólo en las zonas interesantes, se utiliza la máscara que proporciona el esquema *opflow* y se rellena con una imagen proporcionada por el usuario. Éste debe proporcionar la máscara adecuada para cada entorno, la forma de hacerlo será dejar un fichero llamado "filtro.ppm", que contenga una imagen al estilo de la plataforma *jdec* con los datos de la máscara (ver apartado 4.2.1), en el directorio en el que se ejecute la plataforma.

Sólo se establecerá la máscara en la primera iteración del esquema, ya que se supone

que la cámara permanecerá estática.

5.2.2. Búsqueda de orientaciones relevantes

En primer lugar y antes de comenzar con la segmentación, se debe realizar una búsqueda de las características relevantes. En este caso, la característica relevante será la existencia de flujo óptico significativo y su dirección, ya que el flujo óptico que generan los puntos del coche tiene aproximadamente la misma dirección.

La forma de determinar qué conjunto de ángulos forman un grupo que constituye una característica consiste en realizar un histograma en el que se contabilicen el número de puntos con flujo óptico que tiene cada dirección, en grados y con una precisión de un grado. Para ello se recorre la matriz que describe el flujo óptico aportada por *opflow* y para todos los puntos con un flujo mínimo (*min_mov*) se suma uno al grupo de ángulos correspondiente.

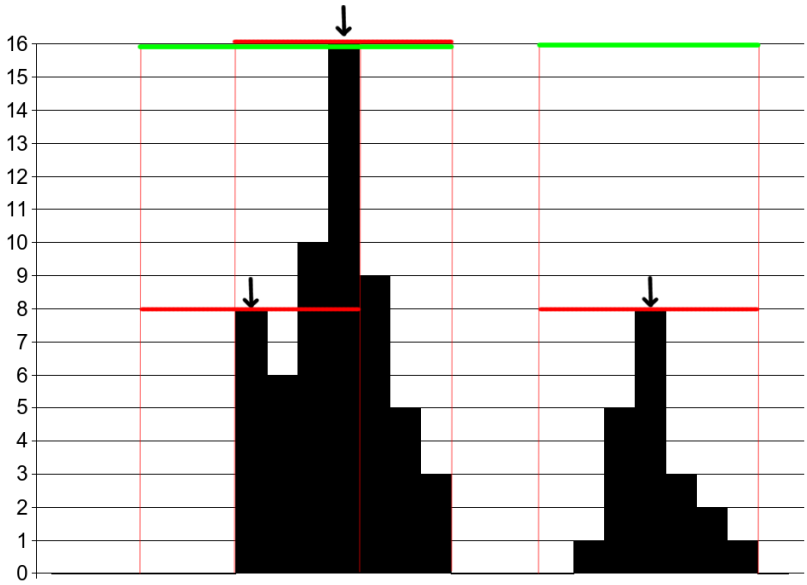


Figura 5.8: Búsqueda de ángulos relevantes

Con el histograma realizado hay que buscar las acumulaciones de ángulos. El método elegido se fundamenta en que cada grupo tiene un máximo (o varios cercanos) que lo hace diferenciarse del resto. Para implementar este método se recorre el histograma y se comprueba si ese valor del ángulo es un máximo local, comprobando que el valor del histograma es mayor que sus vecinos, tal y como se describe en la fórmula

5.1. En el caso de serlo, se genera un nuevo grupo en torno a ese máximo, que abarca un intervalo definido por un parámetro de error (*Angle error*). Si ese nuevo grupo no se solapa con el anterior, es decir que el mínimo del grupo nuevo no es menor que el máximo del grupo creado justo antes, se genera un nuevo grupo de ángulos que definen una nueva característica relevante para realizar la segmentación. Cuando se termina, hay que comprobar que el primer y el último grupo no se solapan. Estas comprobaciones se realizan porque se puede dar el caso de que el grupo de ángulos tenga más de un máximo local, en ese caso se deberán unir (en la figura 5.8 se puede ver gráficamente la selección de grupos hecha en un histograma ejemplo, en él se reflejan en el eje x los ángulos, y en el eje y la cantidad de puntos con flujo óptico en esa dirección).

$$\begin{aligned} h(n) &> h(n-1) \\ h(n) &< h(n+1) \end{aligned} \tag{5.1}$$

Para realizar los grupos de ángulos se podrían utilizar otros métodos aparentemente más precisos, por ejemplo, se podría tomar como grupo desde el máximo local hasta un umbral o hasta encontrar un mínimo local. Sin embargo, los resultados realizando este método no mejoran y oscurecen el código, por este motivo se tomó utilizó este método.

5.2.3. Segmentación espacial

Una vez extraídas las orientaciones relevantes, hay que comprobar si existen píxeles próximos con características similares, de esta forma se localizan los coches, de esta forma se sientan las bases para poder contarlos.

La segmentación es un proceso que consiste en agrupar en una imagen digital en regiones homogéneas con respecto a una o más características. En este caso habrá que realizar una segmentación para cada orientación relevante encontrada en el paso anterior.

Para implementar la segmentación se ha decidido utilizar un método basado en histogramas y utilizando doble umbral, similar al que se describe en [Martínez de la Casa Puebla, 2003]. Este método consiste en realizar histogramas, como los de la figura 5.9, tanto verticales como horizontales, contabilizando los píxeles que pasan el filtro que define al objeto que se busca (en este caso, este filtro lo constituye la existencia de flujo relevante y la orientación del mismo). Posteriormente, busca la localización de los

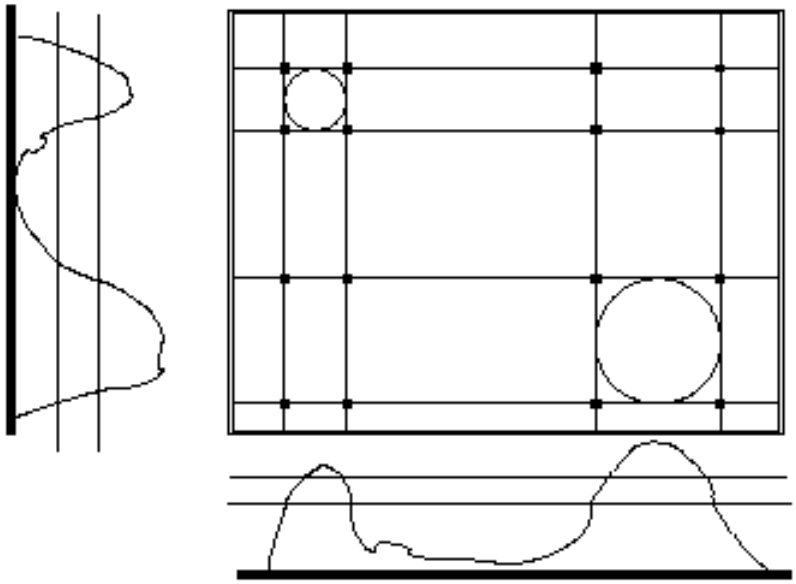


Figura 5.9: Ventanas con doble umbral

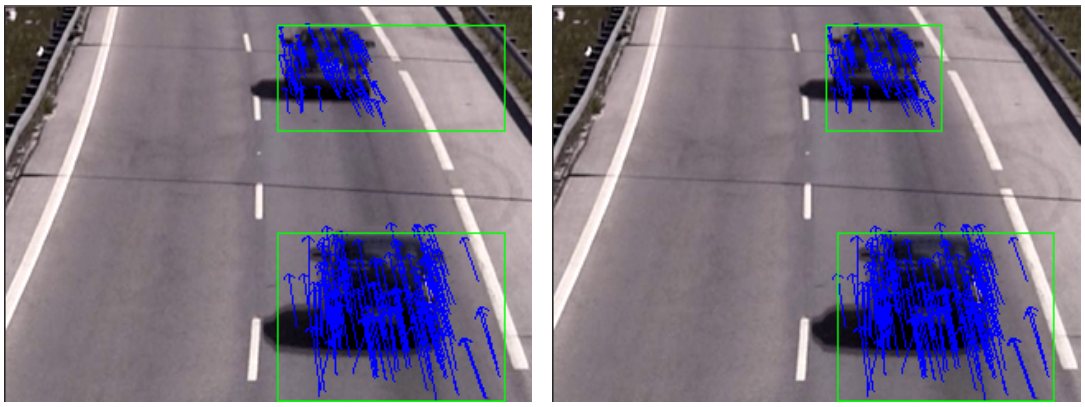
posibles objetos comprobando qué partes del histograma superan un cierto umbral. Si se reúne la información de ambos ejes, se obtienen ventanas en las que posiblemente haya objetos. Como se percibe en la figura 5.9, es posible que se creen ventanas falsas en las cuales no exista un objeto. Para eliminar estas ventanas falsas hay que comprobar que en su interior existan un número mínimo de píxeles que pasan el filtro (*Min_pto*).

Para terminar de describir completamente el algoritmo de segmentación, cabe destacar que es un algoritmo recursivo, al que se le ha aplicado una poda definida por el usuario. La función que se encarga de la segmentación (Tabla 5.2) admite como parámetros la región en la que buscar segmentos (*boundaries*, definido como un array de cuatro posiciones ordenadas de la forma siguiente: xmin, xmax, ymin, ymax) y el número de veces que se quiere llamar recursivamente a la función (*times*). Así, cuando el algoritmo encuentra los segmentos, antes de guardarlos en la variable de salida (*out*), comprueba si se debe ejecutar de nuevo el algoritmo, en caso afirmativo se vuelve a ejecutar pero centrado en la región que abarca el segmento que se acaba de localizar. Esta recursividad consigue que los segmentos sean mucho más ajustados al objeto real que hay en la imagen. Además, el hecho de poder controlar el número de llamadas recursivas, deja a elección del usuario (programador) cuánto se quiere afinar en la segmentación o cuánto tiempo de cómputo se quiere consumir. En la figura 5.10 se puede comprobar la diferencia de utilizar recursividad a no utilizarla.

```
int segmentation(int *boundaries, t_opflow *im, t_opflow_segment *out,
                 int angle_min, int angle_max, int offset, int times);
```

Cuadro 5.2: Función que realiza la segmentación

La función *segmentation* permite también guardar los resultados en el array de salida (*out*, del tipo de datos visto en al tabla 5.1) a partir de una posición indicada por el parámetro *offset*. De esta forma, se consigue que se pueda llamar a la función varias veces para diferentes características determinantes y agrupar todos los resultados en una única estructura. Por último, el parámetro *im* es la imagen de flujo óptico que proporciona el esquema *opflow*.



(a) Segmentación utilizando sólo una pasada del algoritmo

(b) Segmentación utilizando dos pasadas

Figura 5.10: Segmentación basada en flujo óptico

5.2.4. Correspondencias entre segmentos

Para conseguir realizar un seguimiento y posteriormente contar el número de coches que pasan por la carretera, es necesario que se identifiquen unos segmentos con otros, es decir que se establezca una relación entre dos poblaciones de segmentos consecutivas. Para ello, habrá que estimar en qué posición han de encontrarse los segmentos de la iteración anterior (utilizando el flujo óptico) y comprobar si en esa posición, o cerca, hay algún segmento que corresponda.

La forma de abordar la búsqueda de correspondencias a la ahora de programar es realizando una estimación de la posición actual de los segmentos encontrados en la

iteración anterior. Para ello, se utilizan las coordenadas polares del flujo óptico con las que se calculan los nuevos bordes del segmento. De esta forma se prevé la posición del segmento en la imagen nueva.

Después, se comienza a comprobar con cuál de todos los actuales tiene más área en común. Con el que tenga mayor área en común será con el que corresponda, siempre que los perímetros sean similares, es decir que la relación entre ellos esté dentro de ciertos límites.

Para el caso de los segmentos que no encuentren una correspondencia, se actualizan en la población nueva conservando todas sus propiedades excepto la vida (número de segmentaciones sin correspondencia consecutivas) y los límites, que se incluyen los límites estimados a raíz de los antiguos y el flujo óptico. Este caso se debe a pérdidas momentáneas del segmento (por falta de flujo óptico en una iteración) o bien a que el coche ha desaparecido de la imagen.

También puede ser que exista un segmento en la segmentación nueva que no corresponda con ninguno, en este caso se trata de un coche que ha aparecido en la imagen.

Cuenta de segmentos

A la vez que se establece la correspondencia, se va comprobando qué elementos deben dejar de ser seguidos, porque haya habido demasiadas segmentaciones sin que vuelva a haber una correspondencia, es decir, el coche se ha salido del plano. Cuando se deja de seguir un segmento, hay que comprobar cuántas veces ha sido confirmado por una nueva segmentación (cuántas veces ha encontrado un correspondiente) y si supera un umbral calculado experimentalmente, habrá que contar el segmento como un coche que ha pasado por la carretera.

5.2.5. Interfaz gráfica

La interfaz del esquema *cuenta-coches*, como se puede ver en la figura 5.11, permite al usuario de la aplicación controlar ciertos parámetros importantes a la hora de segmentar, así como comprobar los resultados del esquema gráficamente en el display que posee.

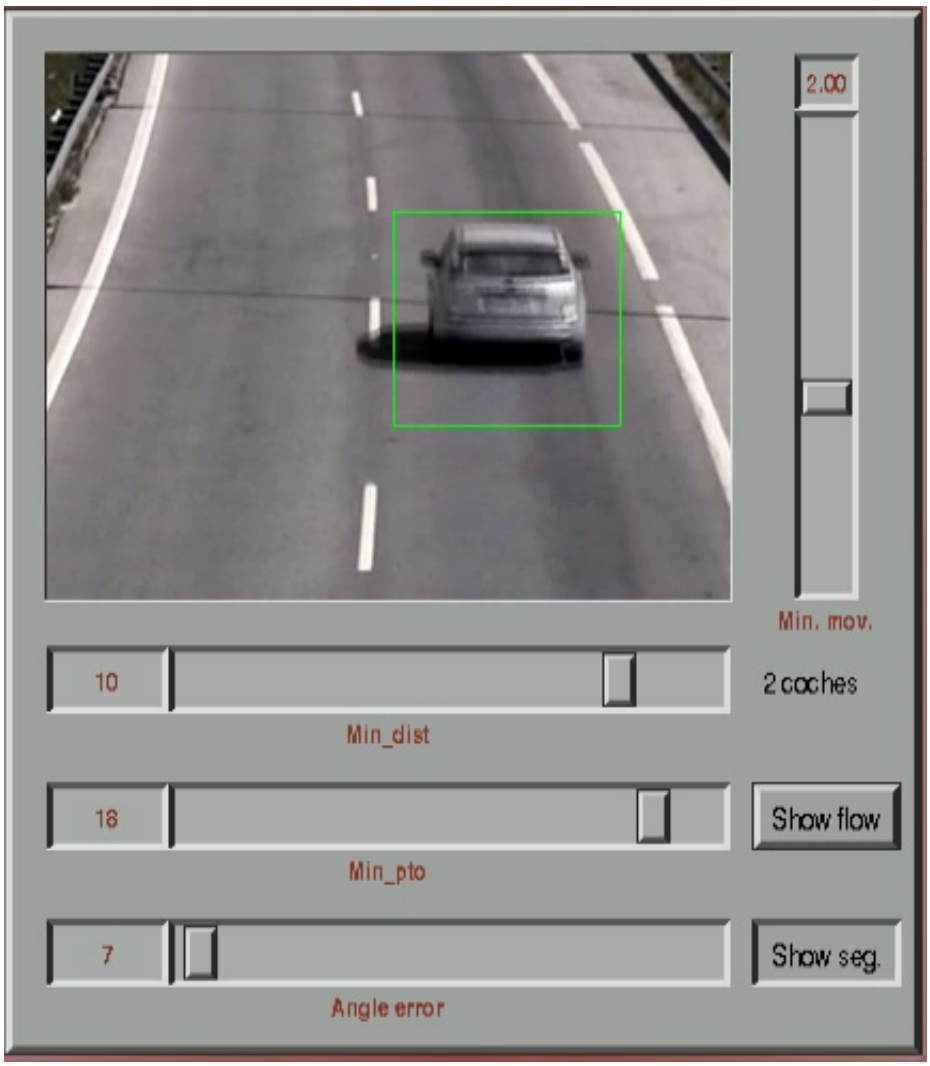


Figura 5.11: Interfaz del esquema *cuenta-coches*

Los parámetros que puede controlar el usuario son:

- Mediante **Min. mov**: el movimiento mínimo que se debe percibir mediante el flujo óptico (en píxeles) para que el programa tenga en cuenta el movimiento de dicho píxel.
- **Min_dist** controla la distancia mínima entre dos segmentos que pertenecen al mismo grupo de ángulos. También se puede entender como el espacio máximo (horizontal o vertical) que se puede tener sin que aparezca ningún punto con flujo en esa dirección.
- **Min_pto** indica el mínimo número de puntos con flujo óptico que tiene que haber dentro de un segmento para considerarlo válido.
- **Angle error** ajusta el intervalo que se toma en torno a un máximo local a la hora de realizar grupos de ángulos.

Los botones *Show seg.* y *Show flow* sirven para mostrar los segmentos y el flujo óptico que los soporta respectivamente. Los segmentos se muestran como recuadros de diferentes colores, y el flujo óptico como flechas del mismo color que el segmento al que soportan.

La interfaz incorpora además un cuadro de texto en el que puede ver el número de coches que ha pasado por la imagen hasta el momento.

5.2.6. Experimentos

Una vez más, es necesario comprobar si los resultados obtenidos son satisfactorios y cumplen los objetivos marcados. Para ello, se han realizado pruebas de funcionamiento y experimentos para ajustar los parámetros. Habrá que comprobar que la aplicación es capaz de contar los coches que pasan por debajo de la cámara.

En los experimentos se han utilizado vídeos grabados desde diferentes puentes de la N-V, y algunos grabados en la EX-A2. Ha sido necesaria la creación y utilización de un elemento que permita tomar vídeos como señal de entrada para las imágenes de la plataforma *jdec* (ver apartado 3.1). Ha sido realmente útil disponer de esta herramienta, ya que simplifica en gran medida las pruebas, no teniendo que desplazarse continuamente para poderlas realizar y además permite realizar pruebas con la misma

entrada varias veces, lo que permite observar las mejoras en los algoritmos.

Parametrización

La parametrización es un aspecto peliagudo de esta aplicación. Cada vídeo de entrada requiere una parametrización diferente, dependiendo de varios factores. La posición de la cámara (altitud e inclinación), así como el zoom utilizado hacen que la forma en que se percibe un coche y su velocidad sean distintas.

El primer elemento que hay que cambiar al utilizar otro vídeo de entrada es la máscara. Hay que ocultar al esquema *opflow* las regiones de la imagen que no interesan, así la segmentación se centrará sólo en las áreas de la imagen que interesen. Generalmente, habrá que poner todo lo que no sea carretera en negro y la carretera en blanco, pero si la imagen de entrada dispone de mucho horizonte, habrá que ocultarlo también para evitar que los segmentos se junten y producir mal funcionamiento (ver figura 5.12).

En cuanto al parámetro *Min. mov*, elimina de la segmentación los puntos con flujo óptico pequeño. Habrá que darle más valor cuando haya mucho zoom, y bajarlo cuando haya poco, así se consigue evitar ruido. Un buen valor que funciona bien en la mayoría de los entornos es 2.0.

Con el *Min_dist* se le indica al algoritmo qué distancia hay entre los segmentos. Dependiendo del tamaño de los coches en la imagen, habrá que poner un valor más alto o más bajo. Generalmente están entre los 8 (para coches lejanos) y 12 (para coches cercanos).

De la misma forma varía *Min_pto*, que indica la mínima cantidad de puntos con flujo que debe de haber en un segmento. Sus valores oscilan entre 7 y 20 según la distancia a la que se encuentren los coches (aunque con otros valores también funciona, aparecerán más segmentos fantasma (si se baja demasiado) o algunos segmentos no se darán por buenos (si se aumenta demasiado).

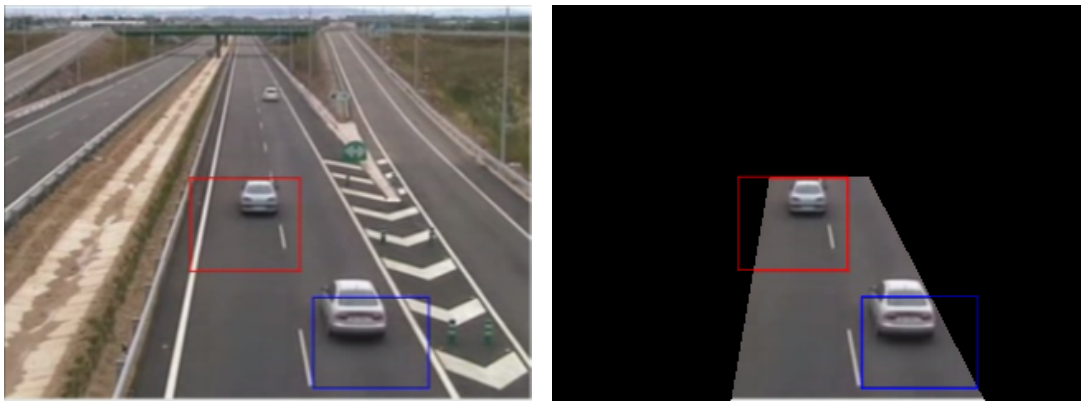
Por último *Angle error*, que indica la amplitud de los grupos de orientaciones relevantes, depende de la dispersión del flujo óptico. Para los experimentos que se han realizado, siempre con coches alejándose, un valor a partir de 7 proporciona buenos

resultados.

Ejecución típica

En cuanto a los experimentos, cabe destacar que se hicieron pruebas de funcionamiento (tanto del seguimiento como de la capacidad de contar los coches que pasan bajo la cámara) con diferentes vídeos de entrada.

Los mejores resultados se alcanzan en vídeos en los que los coches aparecen más horizontales, porque en este caso es más difícil que se produzcan confusiones y se tomen dos coches como uno. El problema que tiene este tipo de plano es que la velocidad a la que aparecen los coches es mucho mayor y esto hace que el seguimiento sea más difícil, ya que aumenta la dificultad de estimar la posición actual de la población de segmentos anterior. No obstante, si se consigue mitigar este efecto no tomando un plano cenital, sino un poco más inclinado, los resultados serán buenos.



(a) Imagen sin máscara (b) Imagen con máscara

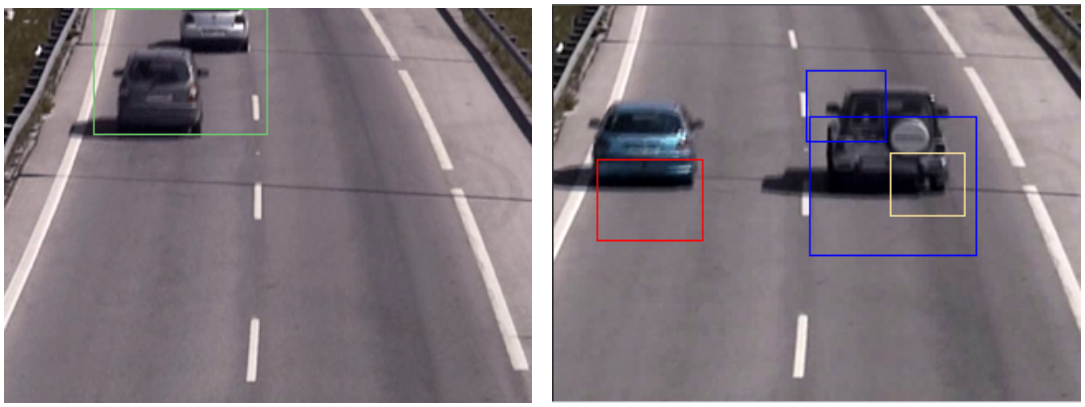
Figura 5.12: Uso de la máscara para evitar la segmentación en el horizonte

Cuando la toma es demasiado paralela a la carretera, al fondo de la imagen los coches comienzan a juntarse y puede llegar a ocurrir que se perciban varios como uno solo. En este caso, la mejor solución es ocultar con la máscara todas las zonas en las que se produzca este efecto; aplicando este método, los resultados fueron muy satisfactorios, ya que dejaban de juntarse segmentos en el horizonte (lo que hacía perder muchos coches). En la figura 5.12 se puede observar el modo de colocar la máscara para evitar este efecto.

Con los parámetros y consideraciones comentadas se ha conseguido un funcionamiento razonablemente bueno de la aplicaicon. No obstante se han identificado algunos casos de fallo que originan problemas a la hora de contar los coches.

En el primer caso, cuando la imagen no es cenital y dos coches vienen muy juntos, generalmente el flujo que se percibe hace que la segmentación sólo encuentre un único vehículo, en consecuencia cuente sólo uno en lugar de dos (ver figura 5.13(a)). En la mayoría de los casos esto se produce porque no hay una distancia mínima entre un coche y otro que permita al algoritmo percibirlos como diferentes.

Otra curiosidad observada en los experimentos, es que de vez en cuando, junto algún segmento que representa un coche, aparecen algunos más pequeños que no cubren la totalidad del mismo. Aparecen durante una o dos iteraciones, y no son contados porque no han tenido correspondencia el mínimo número de veces. Las causas de este problema son dos: la primera de ellas es porque la característica que define el coche (grupo de ángulos) no abarca todo el flujo que está generando. En este caso, se genera otro segmento que cubre el resto, pero en una o dos iteraciones se vuelve a cubrir con un solo grupo todas las direcciones del flujo óptico que genera el vehículo y en consecuencia, el segmento desaparece. La otra causa, es que en una iteración no se genera flujo óptico en todas las partes del vehículo, por este motivo al segmentar aparece un segmento más pequeño que no encuentra correspondencia con el anterior ya que los perímetros no son semejantes. En la interacción siguiente, se vuelve a generar el flujo óptico en todo el vehículo y el segmento más grande vuelve a encontrar correspondencia, el otro más pequeño permanecerá varias iteraciones a la espera de encontrar correspondencia y luego desaparecerá sin incrementar el contador. En la figura 5.13(b) se puede observar este fenómeno.



(a) Dos coches segmentados como uno (b) Segmentos pequeños falsos alrededor de uno válido

Figura 5.13: Casos de fallo de la aplicación

Capítulo 6
Conclusiones y trabajos futuros

Una vez descritas las soluciones adoptadas para resolver los problemas planteados al principio de esta memoria, en este último capítulo se quieren recapitular todas la experiencias que se han vivido a lo largo del desarrollo del proyecto, así como proponer mejoras y líneas futuras que se pueden plantear a raíz de las aplicaciones desarrolladas.

6.1. Conclusiones

Los objetivos del proyecto (descritos en el capítulo 2) han sido abordados con éxito. En primer lugar creando el esquema *opflow*, descrito en el apartado 4.2. Este esquema cumple las características descritas, ya que:

- Calcula el flujo óptico en tiempo real y consumiendo pocos recursos (ver resultados en el apartado 4.3), gracias a que el diseño del algoritmo, utilizando recursos como el doble *buffer* (ver apartado 4.2.2), así como la utilización de la biblioteca IPP (ver apartado 3.3) permite ahorrar mucho tiempo de cómputo. Estas características cumplen el primer objetivo propuesto para la aplicación.

- En segundo lugar, los experimentos han demostrado que el error que presenta la aplicación es muy bajo, eliminando casi en su totalidad los falsos positivos, además los datos que proporciona como salida indican el nivel de error del cálculo del flujo óptico para cada punto en el que se calcula. Esta propiedad amplía uno de los objetivos, ya que además de ser fiable, indica en qué grado lo está siendo, lo que permite al usuario decidir el grado de fiabilidad con el que quiere los datos.

- Por último, y también de forma experimental, se ha demostrado que el algoritmo funciona bien en prácticamente todos los entornos de iluminación, si bien no se comporta de forma idónea cuando los cambios de iluminación se producen de forma brusca. También se ha observado que frente a movimiento demasiado

rápidos (que en una iteración van de un lado a otro de la imagen) los resultados no siempre son los esperados.

Del mismo modo, hay que hacer referencia al problema de la apertura (apartado 4.1.1) que se presenta cuando en la imágen hay algún borde muy marcado y movimiento en su entorno, en este caso se detecta un flujo paralelo al borde. A pesar de estas particularidades, se puede considerar que los objetivos se han cumplido, ya que estos casos son extremos y solo producen el error durante una o dos iteraciones, recuperando la fiabilidad de forma instantánea.

En definitiva, con el esquema *opflow* se ha obtenido un componente capaz de calcular el flujo óptico que puede ser utilizado por otras aplicaciones integrándose perfectamente con ellas a través de la plataforma *jdec*.

En cuanto a las aplicaciones utilizadas como ejemplo del uso del flujo óptico, el resultado ha sido satisfactorio, demostrando que el flujo óptico es una señal de la que se puede extraer gran cantidad de información. En este caso, se puede decir que los objetivos han sido cumplidos:

- Para la aplicación *cuenta-coches*, se ha conseguido realizar un seguimiento basado únicamente el el flujo óptico, que ha redundado en el suncionamiento exigido, poder contar los coches que pasan bajo la cámara. La fiabilidad de esta aplicación no ha sido del 100 % (ver apartado 5.2.6), si bien esto no es problemático, puesto que el objetivo pricipal de estas aplicaciones era ilustrar cómo se puede utilizar la señal de flujo óptico.

- En cuanto a las aplicaciones *eyeoperator*, se ha logrado, de dos formas diferentes, una interfaz que permite manejar el robot de forma sencilla y más o menos natural. Se ha conseguido además que la respuesta del robot sea adecuada a los movimientos que se realizan delante de la cámara como se muestran los experimentos descritos en el apartado 5.1.4.

Valorando las experiencias, haber trabajado con *IPP* ha supuesto un gran reto, puesto que actualmente se dispone de muy poca información sobre esta biblioteca. Además, presenta ciertos aspectos que la hacen un tanto difícil de comprender en un principio[1]. Aún así, y pese a que estas peculiaridades retrasaron el aprendizaje en un

[1]La representación de las imágenes, por ejemplo, está dividida en varias variables separadas que el programador tiene que manejar conjuntamente para trabajar con ellas, algo muy diferente a la forma

principio, una vez acostumbrado a ellas el trabajo con *IPP* es muy satisfactorio, ya que se puede percibir un aumento considerable del rendimiento utilizando las funciones que ofrece, no solo por la utilidad de las funciones, sino porque la implementación está muy depurada y aprovecha toda la potencia del procesador.

El flujo óptico es una señal con muchas posibilidades, la experiencia con ella también ha sido interesante. Es una señal completamente diferente a las que se utilizan habitualmente en visión computacional y no debe de ser tratada como tal. Uno de los principales problemas que se presentaron durante la realización del proyecto fue el intentar realizar una segmentación basada en flujo óptico. Los primeros intentos se centraron en realizar la segmentación exactamente del mismo modo en que se haría con el color, pero resultó ser un problema, porque la señal de flujo óptico es no-densa (no se pude calcular para todos los puntos porque generaría un coste computacional muy alto, además existe el problema de que todos los puntos no son igual de aptos para establecer correspondencias). Otra característica de esta señal, es que contiene gran cantidad de información, y hay que buscar una buena forma de procesarla.

La experiencia demuestra que los mejores usos del flujo óptico, y los más naturales, son aquellos en los que se utiliza el movimiento medio de una zona o región de la imagen. Esta forma de uso se puede utilizar para disparar otros mecanismos de detección o segmentación, como forma de prever el movimiento y realizar seguimientos de forma sencilla, o en el caso de un robot móvil navegación (en el apartado 6.2 se explican algunas de estos usos). No hay que olvidar que, pese a que el flujo óptico es una señal muy útil, hay que buscar los momentos en los que conviene utilizarla. No siempre es la mejor opción, incluso se puede aumentar el trabajo por utilizarla de forma errónea. En la aplicación *cuenta-coches* (ver 5.2), por ejemplo, la segmentación ha sido más costosa por realizarla basada en el flujo óptico en lugar de basarla en color.

6.2. Líneas futuras

6.2.1. Mejoras de las aplicaciones actuales

En el caso de las aplicaciones de ejemplo, se podrían realizar mejoras en la aplicación *cuenta-coches* implementado una segmentación basada en el color, pero disparada por el flujo óptico. Esto mantendría las ventajas de utilizar el flujo óptico, permitiendo

de trabajo con imágenes en OpenCV, dónde las imágenes son una única variable que engloba todos los datos.

conocer el movimiento del coche, y evitaría las posibles carencias que tiene la segmentación basada en el flujo óptico, como por ejemplo que se pierde el objeto cuando éste se detiene. En la aplicación *eyeoperator* cabe la posibilidad de mejorar el control en la aplicación que utiliza sólo una región. Utilizando para ello el método comentado más arriba que realiza una segmentación de la mano disparada por el flujo óptico, utilizando los datos del flujo óptico para caracterizar el movimiento de la mano y transmitirlo al robot. Esta forma de utilizarlo, sería aún más parecida a el *touchpad* de un ordenador.

También en la aplicación *cuenta-coches*, se puede hacer una mejora para poder calcular la velocidad de cada coche. Ya que el flujo óptico nos indica cuánto se han desplazado en la imagen y, utilizando algún dato más, se puede trasportar esta velocidad en la imagen el mundo en 3D y calcular su velocidad.

6.2.2. Nuevas aplicaciones

Al incluir en la plataforma *jdec* un módulo que calcule la señal de flujo óptico, se permite a los programadores utilizarla sin tener que preocuparse por calcularla.

De esos posibles usos cabe destacar dos de ellos muy interesantes:

- Navegación del robot basada en flujo óptico. En este caso, un robot deberá ir evitando obstáculos a medida que avanza, utilizando para ello simplemente el flujo óptico. La forma, aparentemente, más sencilla es hacerlo como lo hacen algunos animales ([Duchon *et al.*, 1998] cita algunos de estos ejemplos), por ejemplo igualando el flujo óptico que se percibe a la derecha y a la izquierda de la imagen (como hacen las abejas). Cuando el robot se encuentra cerca de un obstáculo por alguno de los dos lados, el flujo óptico será mayor por ese lado, entonces, para hacerlo disminuir e igualarlo al del otro lado, tendrá que girar hacia el lado contrario para evitar el elemento que produce ese flujo óptico. En [Dev *et al.*, 2004] se realiza una implementación similar a la propuesta, utilizando el flujo óptico para avanzar centrado en un pasillo, pero sin evitar obstáculos.

- Representación en 3D de una escena estática utilizando el flujo óptico y una sola cámara móvil. Para poder realizar la representación hay que haber calibrado la cámara previamente, de esta manera se puede conocer la distancia a un objeto utilizando geometría proyectiva y el flujo óptico que generan los objetos. Al

tratarse de un entorno estático, la única forma de generar flujo óptico es mover la cámara; esto proporciona ciertas ventajas porque al saber cuánto se mueve la cámara, es más fácil calcular la distancia.

Bibliografía

[Cañas *et al.*, 2007] José M. Cañas, Antonio Pineda, Jesús Ruíz-Ayucar, Javier Martín, y José A. Santos. *Programación de robots con la plataforma Jdec*, 2007.

[Cañas, 2003] José M. Cañas. *Jerarquía Dinámica de Esquemas para la generación de comportamiento autónomo*. PhD thesis, Universidad Politécnica de Madrid, 2003.

[Dev *et al.*, 2004] Anuj Dev, Ben Krose, y Frans Groen. Navigation of a mobile robot on the temporal development of the optic flow. Technical report, RWCP Novel Functions SNN*Laboratory, Department of Computer Science, University of Amsterdam, 2004.

[Duchon *et al.*, 1998] Andrew P. Duchon, William H. Harren, y Leslie Pack Kaelbling. Ecological robotics. *Adaptive Behavior*, 6(3/4):473–507, 1998. Special Issue on Biologically Inspired Models of Spatial Navigation.

[Díaz Peña, 2005] Pedro Miguel Díaz Peña. *Navegación visual del robot Pioneer.* Proyecto fin de carrera, Universidad Rey Juan Carlos, 2005.

[Gonzalez y Woods, 1993] R.C. Gonzalez y R.E. Woods. *Digital Image Processing*. Addison-Wesley, 1993.

[Horn y Schunck, 1981] B.K.P. Horn y B.G. Schunck. *Artificial Intelligence*, chapter Determining optical flow, 17, pages 185–204. 1981.

[Horswill, 1993] Ian Horswill. *Polly: A vision-based Artifial agent*. AAAI, 1993.

[Ipp, 2002] *Intel® Integrated Performance Primitives v5.1.1 for Linux on IA-32 Intel® Architecture, Release Notes*, 2002.

[Ipp, 2006] *Intel® Integrated Performance Primitives for Intel® Architecture, Reference Manual; Volume 2: Image and Video Processing*, 2006.

[Lucas y Kanade, 1981] B. Lucas y T. Kanade. *Proceedings of DARPA Image Understanding Workshop*, chapter An iterative image registration technique with an application to stero vision, pages 121–130. 1981.

[Martínez de la Casa Puebla, 2003] Marta Martínez de la Casa Puebla. *Comportamiento sigue pelota con visión cenital.* Proyecto fin de carrera, Universidad Rey Juan Carlos, 2003.

[Sánchez *et al.*, 2006] Ángel Sánchez, Ana Belén Moreno, José F. Vélez, y Antonio Sanz. Visión artificial: Tema 7 visión dinámica. Guía de clase, Ingeniería Informática URJC, 2006.