VISIÓN ARTIFICIAL
APLICADA A VEHÍCULOS INTELIGENTES

Memoria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per

David Gerónimo Gómez

i dirigit per

Joan Serrat Gual

Bellaterra, Juny del 2004

El sotasignat, *Dr. Joan Serrat Gual*
Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en *David Gerónimo Gómez*.

I per tal que consti firma la present.

Signat:.....

Bellaterra, Juny del 2004

A mi padre

Índice

Capítulo 1

Introducción

Durante los últimos veinte años, la visión artificial ha pasado de ser un campo científico encerrado en un laboratorio, con más hambre de investigación que objetivos prácticos, a convertirse en una de las áreas más estudiadas de la ciencia e ingeniería, ésta vez con objetivos y metodologías concretas.

Uno de los campos donde la visión artificial está llamada a ser la piedra angular de la innovación tecnológica del siglo XXI es en los vehículos inteligentes, donde los presupuestos para investigación por parte de compañías y universidades en la última década han crecido exponencialmente.

En éste proyecto se han analizado, diseñado e implementado tres métodos para detectar el horizonte, pieza fundamental en el cálculo fiable de distancias a otros vehículos, como veremos. Si bien algunas compañías han comercializado productos orientados al cálculo de distancias, éste sigue siendo un problema sin solución completa. Su abanico de aplicaciones es extremadamente amplio, tan sólo hemos de pensar en la información que extrae nuestro sistema visual de un cálculo muy aproximado de las distancias, de las cuales la estimación de velocidad o el tiempo de colisión serían los resultados más inmediatos.

A continuación comentamos la estructura de ésta memoria.

- Introducción. Breve descripción histórica de los vehículos inteligentes, explicando las alternativas que se dan en las soluciones empleadas. Explicación específica acerca del cálculo de distancias y sus problemas, y finalmente los objetivos del presente proyecto.
- Distancias a partir del horizonte. En éste capítulo se explicará la base matemática que sustenta nuestro trabajo, empezando por una sección muy básica de geometría proyectiva, pasando por el modelo de nuestro problema y acabando en la razón de ser del proyecto: las consecuencias que nos provoca el error en la localización del horizonte.

- Proyección horizontal. El primer método estudiado se basa en el desplazamiento de la proyección horizontal de la imagen, bajo distintos parámetros, filtros y algoritmos, con la finalidad de estimar el horizonte actual partiendo de un horizonte fijado inicialmente. Además, se ha implementado el cálculo de distancias para demostrar la facilidad de obtenerlas una vez tenemos el horizonte.

- Geometría epipolar. Otra manera de enfocar la solución del problema, ésta vez por el lado más matemático, es la geometría. Explicaremos los pasos necesarios hasta llegar a nuestra solución.

- Foco de expansión. Partiendo de un resultado intermedio de la geometría epipolar, desarrollaremos otra solución más adaptada al problema, que nos aportará mejores resultados finales.

- Resultados globales. Evaluación global de los tres métodos, prestando atención a las faltas y puntos positivos que hemos extraído de cada uno de ellos, enfocando las ideas a posibles usos en el campo práctico.

- Conclusiones. Se expondrán cuales han sido los objetivos alcanzados, qué dificultades hemos encontrado a lo largo de la realización del proyecto, posibles mejoras y ampliaciones de éste, y cual sería el camino a seguir partiendo de nuestros resultados.

1.1. Vehículos inteligentes

Pese a que el concepto de *vehículo inteligente* pueda sonar a visión futurista, y aunque a los usuarios, incluso a principios del nuevo siglo, les resulte difícil imaginar las posibilidades reales y factibles de los sistemas de transporte inteligentes, ésta idea surgió hace ya dos décadas y su evolución no ha parado desde entonces.

Ya en 1986, el equipo del Doctor Ernst Dickmanns -Universität der Bundeswehr, Munich- consiguió desarrollar un prototipo de vehículo automático capaz de conducir por autopistas a 100Km/h, usando filtros de Kalman [7] para discriminar el conjunto de interpretaciones de la escena mediante dos procesadores 08086. En 1988, en la Universidad de Carnegie-Mellon, se implementaron algoritmos basados en el color para seguir carreteras, y se desarrollaron técnicas de detección de obstáculos mediante visión 3D. Todo esto fue integrado en una furgoneta llamada *NavLab*. Uno de los problemas que encontraron, que siempre aparece en los proyectos de visión e inteligencia artificial, fue que el ancho de banda necesario era altísimo, por lo que la velocidad máxima de *NavLab* era de 10cm/s [3]. Como último ejemplo, en la misma universidad pero siete años más tarde, encontramos *RALPH* (*Rapidly Adapting Lateral Position Handler*), un sistema que ayuda al conductor a girar el volante. La idea era detectar el desplazamiento lateral, relativo al centro del carril por el que se circula, entre dos

fotogramas [4].

Pero, ¿por qué surgió éste campo de investigación, y cuál es su finalidad real? La motivación principal es que la mayoría de los accidentes de tráfico se producen por errores humanos. Un gran número de los accidentes mortales son debidos al exceso de velocidad, a la pérdida de atención sobre la carretera, o a la poca precisión en el cálculo del tiempo de impacto a un vehículo precedente por parte del conductor. Además de la clara intención de incrementar el nivel de seguridad en las carreteras, que cada año suponen miles de pérdidas tanto humanas como económicas, los *sistemas de transportación inteligente* pretenden mejorar desde el consumo de energía, reduciendo así la polución emitida y el gasto en carburante, hasta la optimización del flujo de tránsito las redes viales.

Hoy en día el término *ordenador de abordo* no resulta nada extraño, un turismo consta de una serie de controladores que regulan desde el comportamiento del motor hasta la climatización, pasando por la dirección asistida del volante o los frenos antibloqueo, los conocidos *ABS*. El siguiente paso en la integración vehículo-computadora será seguir la carretera y mantener el vehículo en el carril correcto, mantener una distancia segura respecto a los demás, regular la velocidad conforme a las condiciones del tráfico, evitar obstáculos, proponer rutas óptimas, y en general facilitar la conducción y optimizar las capacidades del vehículo.

Figura 1.1: **Último modelo de *Ford Fusion* con un sistema de protección inteligente.** *Incorpora airbags delanteros de dos tiempos y sensores que calculan la severidad del impacto. (Fuente: Ford España).*

Existen dos etapas en la evolución de la investigación en el campo de los vehículos inteligentes. La primera de ellas, como ya se ha comentado, surgió en los años 80 tras la concienciación de los gobiernos de la necesidad de invertir dinero en el sector.

Se crearon varios proyectos a nivel internacional, como por ejemplo *PROMETHEUS* (*Programme for a European Traffic of Highest Efficiency and Unprecedented*) en Europa, puesto en marcha en 1986 con la colaboración entre fabricantes y universidades. En Estados Unidos se creó el *NAHSC* (*National Automated Highway System Consortium*) en 1995, y en Japón el *AHSRA* (*Advanced Cruise-Assit Highway System Research Association*) en 1996 [1]. Éstas últimas con el mismo propósito: desarrollar no sólo vehículos sino también las infraestructuras necesarias para conseguir los objetivos numerados en la sección anterior.

La segunda fase, en la cual aún estamos inmersos, parte de ese profundo análisis conseguido por parte de empresas automovilísticas y universidades, y su objetivo es llegar a soluciones robustas e implementables en vehículos comerciales, nutriéndose de campos tan diversos como la inteligencia artificial, la robótica, la automática o la visión. Los requisitos para éstos sistemas de transporte principalmente son el bajo coste, la integración y la robustez. Por ejemplo, un sistema para calcular la distancia a los dos vehículos más cercanos en un aparcamiento en fila, comercializado recientemente en diversos modelos de turismos de gama alta, no puede superar cierto porcentaje del precio final, pues hemos conducido sin éste tipo de ayuda durante más de cien años, y podríamos permitirnos prescindir de ella. Respecto a la integración, se hace imprescindible un interfaz "'user-friendly"' con el sistema, lo cual no es un problema viendo que la informática y toda la tecnología en general tiende a éste modelo más amigable de intercomunicación hombre-máquina. Finalmente, la robustez es uno de los problemas más importantes hoy en día, ya que adaptar el sistema a todas las condiciones meteorológicas posibles, diferentes tipos de iluminación, o casos peculiares del tráfico, como por ejemplo los atascos, es una tarea complicada.

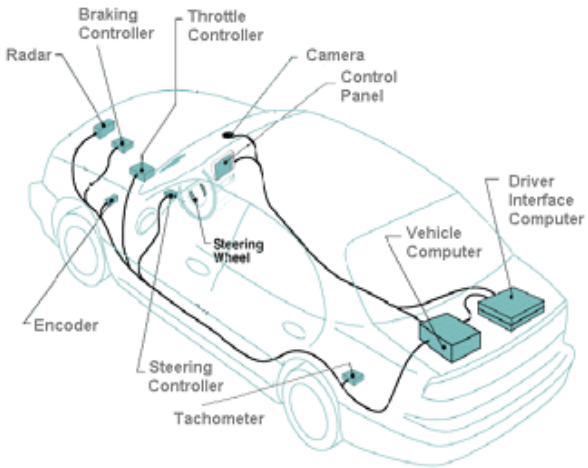

Figura 1.2: **Esquema del hardware utilizado en un vehículo inteligente.** *Diagrama del UA ATLAS lab, prototipo de vehículo inteligente desarrollado por la Universidad de Arizona. (Fuente: Arizona Engineer).*

1.2. Estado del arte

La investigación actual sigue dos líneas, en muchos casos conectadas y utilizadas al mismo tiempo. Primeramente encontramos la de *sensores activos*, basada en láser o radar, y de naturaleza intrusiva. Su funcionamiento básico consiste en lanzar una señal al exterior del vehículo, y calcular cuanto tarda la onda en volver al sensor después de la reflexión contra los objetos. Encontramos por ejemplo el sistema de aparcamiento antes mencionado: en el momento de estacionar el turismo, podemos dejarlo justo en el centro de los dos vehículos más cercanos. Funciona con dos láseres, uno posterior y otro anterior, que una vez reflectados sobre las superficies de los otros vehículos, permiten calcular la distancia gracias al tiempo que ha tardado la onda en volver. Su principal ventaja es que el cálculo es prácticamente directo, por lo que no requiere apenas procesamiento; además, usando radares de onda milimétrica, la precisión es muy grande. El inconveniente, además de su carácter invasivo -imaginemos el caso en que todos los vehículos de una autopista lanzasen láseres, las interferencias serían más que importantes-, es la imposibilidad de saber qué es lo que hay tras la distancia calculada. Pensemos en que nuestro vehículo está apunto de entrar en una curva, y calcula que a 5 metros hay un objeto, al cual nos dirigimos a 50Km/h. ¿Cuál debería ser la respuesta? Podría ser otro vehículo que hemos calculado estático -ya que la distancia a ese objeto no ha hecho más que disminuir-, por tanto activar una alarma o un mecanismo de reducción de velocidad. O podría ser un árbol, que evitaremos tranquilamente al seguir la carretera girando la curva. Éste ejemplo ilustra la problemática principal en éste tipo de sensores, por lo que normalmente se usan en situaciones muy concretas, y como apoyo a otro tipo de sistemas.

La otra línea, realmente activa y en crecimiento, es la de los *sensores pasivos*. Éste tipo de sensores, no invasivos, permiten abrir el abanico de posibilidades sin influir en el entorno. En éste caso se trabaja por ejemplo con una o más cámaras que envían fotogramas a cierta velocidad al computador para ser procesados, obteniendo información de todo un espectro de frecuencias incluso mayor que el de la luz visible. Aquí no tenemos el problema de que ciertas reflectancias peculiares en los objetos nos den información equivocada, y eliminamos la ambigüedad resultado de no saber qué objetos están a ciertas distancias. El número de aplicaciones que nos ofrece la visión artificial en éste aspecto es muy amplio, algunas de imposible implementación con sensores activos. La detección de líneas de carril, por ejemplo, imprescindible en muchas de las técnicas de proceso a alto nivel, necesita de la visión artificial. Por otro lado, éste área también tiene sus problemas, quizás más que la rama de los sensores activos: a diferentes iluminaciones necesitaremos diferente tratamiento, la noche y el día son casos totalmente diferentes; condiciones de niebla o lluvia producirán artefactos en el parabrisas o en el asfalto, producto de las reflexiones de las luces de los demás vehículos; el sol proyectará sombras de vehículos, árboles y señales que dificultarán los cálculos e incrementarán la complejidad de los algoritmos; la detección deberá prever un enorme número de tipos de vehículos posibles, etc.

Figura 1.3: **Sensores.** *A la izquierda, radar (sensor activo). A la derecha, sistema de video de rango dinámico (sensor pasivo)* [9].

Finalmente, mencionar el gran número de artículos que se publican al año dedicados a aspectos muy concretos de la visión por computador aplicada a vehículos, como la detección en imágenes del espectro infra-rojo [11], aplicadas a conducción nocturna, la detección de peatones -una de las áreas más investigadas [15] [2]-, o la detección de líneas de carril [14] en trazados con curvas. Además, anualmente se realizan diversos congresos dedicados a éste tema, como el *International Conference on Intelligent Transportation Systems*, con su 6ª edición en 2003, o el *IEEE Intelligent Vehicles Symposium*. La investigación sobre vehículos inteligentes está en pleno crecimiento, y está llamada a ser unas de las principales piezas del marketing en la empresa automovilística éste nuevo siglo.

1.3. Cálculo de distancias

Una de las lesiones más graves en un accidente de tráfico viene predecida por una colisión frontal. Se denomina lesión por desgarramiento, y es producto de la desaceleración del tórax -sujetado por el cinturón de seguridad- y el movimiento hacia adelante del cuello -debido a la inercia-. Las fibras cervicales, vasos sanguíneos y el tallo cerebral se estiran por las dos fuerzas opuestas, y pueden llegar a desgarrarse. Éste es uno de los escenarios más comunes de los accidentes actuales, las secuelas son evidentes, y la industria es consciente de ello. Como ejemplo, en Fórmula 1 se introdujo el *HANS*, unas sujeciones entre el casco y el asiento, que mantienen la cabeza y el torso en la misma posición en caso de impacto. Implantar ésta solución fuera de los circuitos es efectivamente inviable.

Dada ésta imposibilidad, en el campo de los vehículos inteligentes éste problema se intenta solucionar mediante uno de los aspectos más complejos: el cálculo de distancias. Controlando la distancia a los vehículos precedentes podemos mantener un margen de seguridad, realizar un seguimiento, incluso predecir comportamientos, y lo que es más importante, actuar sobre el vehículo de forma inteligente y segura, evitando desaceleraciones y colisiones bruscas, difícilmente controlables por un humano. Y éste es sólo un caso concreto de las aplicaciones que tiene el cálculo de distancias,

como veremos a lo largo del capítulo.

Partiendo de ésta necesidad, encontramos múltiples tipos de propuesta al atacar la cuestión. En [6] encontramos un buen análisis introductorio sobre como enfocar el problema. Como hemos explicado anteriormente, el uso de sensores activos, pese a ser los más precisos, no es suficiente a la hora de plantear la solución, pero además sabemos que el uso de las leyes de la perspectiva pueden ayudarnos mucho. La visión humana es capaz de realizar cálculos aproximados de distancias a partir de un sistema binocular, aunque éstos dejan de ser precisos en distancias superiores a lo que podemos alcanzar con nuestras manos. Aún así, el sistema visual humano es capaz de realizar estimaciones bastante correctas de tiempo de contacto usando divergencia retinal, es decir, mediante el cambio de escala de los objetos según su lejanía.

Figura 1.4: **Detección de líneas de carril para calcular la anchura de éste.** *El tracking en éste ejemplo es robusto a sombras, obstáculos e incluso a líneas parciales o inexistentes sobre la carretera [12].*

Pero iremos más allá, ¿por qué no usar un sistema monocular? El introducir dos cámaras nos produce un gasto en consumo, sobretodo cuando la disparidad es grande, por lo que el gasto de procesador será más alto. Además, el problema de mantener calibradas dos cámaras no es de solución trivial, y menos si pensamos en la producción en serie de éste tipo de sistemas.

En éste punto, tenemos dos posibles vías a partir de las cuales extraer distancias, ambas resultado de la geometría proyectiva. Una primera idea sería la siguiente: si conocemos la anchura de un vehículo precedente, la llamaremos m, y conocemos los parámetros de calibración de nuestra cámara, entre ellos la distancia focal f, podemos calcular la distancia partiendo de cuanto mide ese mismo vehículo en nuestra imagen. Pese a ser tan simple y evidente como veremos en próximos capítulos, no es nada precisa. La razón es que un vehículo cualquiera puede medir entre 1.5m y 3m, por lo que los resultados que calculemos tendrán tan sólo una precisión del 30 %. Ésta precisión es insuficiente para sacar conclusiones y actuar correctamente [6].

La segunda aproximación pasa por usar la geometría de la carretera para detectar a qué distancia están las ruedas (parte más cercana al suelo) del vehículo precedente respecto a la cámara.

1.4. Objetivos del proyecto

Éste proyecto se propone llegar a una solución de la detección del horizonte, para después realizar el cálculo de distancias, utilizando geometría proyectiva. Como veremos más adelante, el problema de las distancias puede resolverse una vez hemos localizado el horizonte (uno de los métodos para localizarlo es mediante la intersección de las líneas de carril en el infinito), y conociendo la longitud de un segmento en la carretera, como por ejemplo una marca vial (e.g. flecha, separador de carriles, etc). Veremos también que ésta localización del horizonte es crucial, y que a medida que aumenta la distancia al objeto, el error en el cálculo crece exponencialmente. Las deformaciones de la carretera, el balanceo de la cámara debido a aceleración o desaceleración del vehículo, o el hecho de que la carretera no sea plana, dispararán los errores muy gravemente, por eso el obtener una buena aproximación en la detección del horizonte es tan importante.

En nuestro proyecto se estudian e implementan tres métodos para la localización del horizonte:

- En primer lugar, la **proyección horizontal**, basada en los desplazamientos que se producen en la proyección de la imagen cuando encontramos baches o sufrimos aceleraciones, como antes hemos comentado.
- En el segundo método, usamos la **geometría epipolar** y el algoritmo RANSAC [17][8] para extraer una *matriz fundamental* y localizar el epipolo de la imagen. En éste método se utilizan algunos algoritmos ya implementados por investigadores, y de comprobada robustez en sus aplicaciones originales.
- Finalmente, y como complemento al segundo método, detectamos el **foco de expansión** de la imagen, pero no mediante geometría epipolar, sino calculando directamente la intersección de las líneas creadas entre correspondencias en cada par de frames.

Así mismo, también enumeraremos los aspectos que NO se buscará resolver:

- No se implementará ningún tipo de **detector**, ni de vehículos ni de líneas de carril. El campo de la detección en visión es extremadamente amplio, por lo que es inabarcable en el presente proyecto.
- El **tiempo real** no será un aspecto prioritario. Si bien siempre se buscará la optimización en los algoritmos, tanto temporal como espacial (cuando sea posible), no se tratará de conseguir algoritmos en tiempo real. Las razones son varias, podemos argumentar que el entorno *MATLAB*, pese a ser una de las mejores herramientas a nivel matemático, no se caracteriza por sus altas prestaciones en cuanto a tiempos de ejecución -pese a estar basado en principios de localidad espacial de datos-. Además, el concepto de tiempo real puede ser alcanzado sólo

una vez se han encontrado buenas soluciones a los problemas, no durante su investigación.

Algunos conceptos y objetivos pueden parecer confusos en éste momento, sin haber visto la teoría necesaria, por lo que simplemente se han de tomar como ideas generales que serán explicadas en los capítulos siguientes.

Capítulo 2

Distancias a partir del horizonte

Antes de entrar en el funcionamiento de los métodos propuestos, es necesario explicar la teoría básica en la que están fundamentados. La mayor parte de los teoremas aquí expuestos están basados en [18], por lo que animamos a acudir a la fuente para obtener mayor detalle y explicaciones más amplias sobre éste capítulo.

2.1. Geometría proyectiva

Empezaremos definiendo un punto en el plano: $(x, y)^T$ en $\mathbb{R}^2$. Si consideramos $\mathbb{R}^2$ como un espacio de vectores, la pareja de coordenadas $(x, y)^T$ es un vector, luego un punto se identifica por un vector.[1]

Como sabemos, una línea está representada por la ecuación $ax+by+c = 0$, por lo que diferentes valores de a,b y c nos definen una línea diferente. Así pues representaremos una línea con el vector $(a, b, c)^T$. Mencionar también que todos los productos de un vector de la forma $k(a, b, c)^T$ por cualquier k diferente de cero, representan la misma recta, son por tanto equivalentes. El conjunto de todos éstos vectores en $\mathbb{R}^3 - (0, 0, 0)^T$ forman el *espacio proyectivo* $\mathbb{P}^2$.

Un punto también puede representarse por un vector de tres elementos: $(x_1, x_2, x_3)^T$. A ésta representación la llamaremos forma *homogénea*. Así mismo, éste punto en su forma *no homogénea* se expresa como $(x_1/x_3, x_2/x_3)^T$. Por ejemplo, el punto en forma *homogénea* $(8, -6, 2)$ corresponde al punto $(8/2, -6/2, 2/2) = (4, -3)$ en coordenadas

[1] Escribiremos $\mathbf{v} = (x, y)^T$ para referirnos al vector columna formado por las coordenadas x e y. Ésta nomenclatura se usa para facilitar el trabajo con matrices, es decir, si multiplicamos una matriz (por ejemplo de homografía) por un vector columna, obtenemos otro vector columna. Por tanto cuando nos refiramos a v, estaremos hablando del vector columna $(x, y)^T$, y cuando digamos v^T, estaremos hablando de su transpuesto, en éste caso el vector fila (x, y).

no homogéneas (no anotamos el tercer elemento).

Una vez definidos el punto y la recta, sabemos que un punto $\mathbf{x} = (x, y)^T$ pertenece a la recta $\mathbf{l} = (a, b, c)^T$ si y sólo si $ax + bx + c = 0$, lo cual puede escribirse como $(x, y, 1)(a, b, c)^T = 0$. Así pues, obtenemos el resultado

Resultado 1. *Un punto* $\boldsymbol{x}$ *pertenece a la línea* $\boldsymbol{l}$ *si y sólo si* $\boldsymbol{x}^T\boldsymbol{l} = 0$.

Vayamos más allá. Supongamos que tenemos dos líneas $\mathbf{l} = (a, b, c)^T$ y $\mathbf{l'} = (a', b', c')^T$. El vector $\mathbf{x} = \mathbf{l}\times\mathbf{l'}$, que representa el producto vectorial entre las líneas, nos sirve para llegar a que el producto escalar $\mathbf{l}.\mathbf{x} = 0$, siempre que $\mathbf{x}$ se encuentre sobre la línea $\mathbf{l}$. Por consiguiente, $\mathbf{l}.(\mathbf{l}\times\mathbf{l'}) = \mathbf{l'}.(\mathbf{l}\times\mathbf{l'}) = 0$, lo cual nos lleva a definir que

Resultado 2. *La intersección entre dos líneas* $\boldsymbol{l}$ *y* $\boldsymbol{l'}$ *es el punto* $\boldsymbol{x} = \boldsymbol{l}\times\boldsymbol{l'}$.

Finalmente, nos queda definir una línea mediante dos puntos siguiendo el mismo argumento.

Resultado 3. *La línea entre dos puntos* $\boldsymbol{x}$ *y* $\boldsymbol{x'}$ *es la línea* $\boldsymbol{l} = \boldsymbol{x}\times\boldsymbol{x'}$.

PUNTOS IDEALES Y LÍNEAS EN EL INFINITO

Ahora consideremos dos rectas paralelas $ax + bx + c = 0$ y $ax + bx + c'x = 0$, representadas por los vectores $\mathbf{l} = (a, b, c)^T$ y $\mathbf{l'} = (a, b, c')^T$. Si calculamos su intersección (usando el Resultado 2), obtendremos el punto $(b, -a, 0)^T$. Si tratamos de encontrar su representación *no homogénea* obtenemos $(b/0, a/0)^T$, lo cual significa que las coordenadas son infinitas, resultado de intentar intersectar dos líneas paralelas. Éstos puntos, cuya coordenada $x_3 = 0$, son llamados puntos ideales, o puntos en el infinito.

Igualmente, podemos definir la línea en el infinito como $\mathbf{l}_\infty = (0, 0, 1)^T$, ya que $(0, 0, 1)(x_1, x_2, 0)^T = 0$.

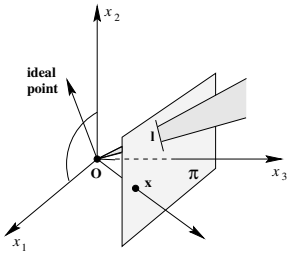

Figura 2.1: **Modelo del plano proyectivo.** *Las líneas en el plano* $x_1 - x_2$ *representan puntos ideales, y el plano* $x_1 - x_2$ *representa* l_∞. *Además, un punto y una línea de* $\mathbb{P}^2$ *se representan como un rayo y un plano en* $\mathbb{R}^3$, *respectivamente.* π *es el plano proyectivo, y en la sección siguiente veremos como representa al plano imagen.* Fuente: [18]

La explicación de los puntos en el infinito nos ha servido para simplificar el concepto de la intersección entre líneas y puntos en el infinito. En el estudio de la geometría proyectiva no se hacen distinciones entre puntos ideales (en el infinito) y puntos ordinarios, ésto no es así en la geometría Euclidiana estándar de $\mathbb{R}^2$.

HOMOGRAFÍAS

Una vez hemos hablado de $\mathbb{P}^2$, haremos la siguiente definición:

Definición 1. *Una homografía es una transformación lineal invertible h, desde $\mathbb{P}^2$ hacia él mismo, tal que tres puntos x_1, x_2 y x_3 yacen sobre la misma línea si y sólo si $h(x_1)$, $h(x_2)$ y $h(x_3)$ también lo hacen.*

Así pues, una homografía es un *mapping* $h : \mathbb{P}^2 \to \mathbb{P}^2$, donde se utiliza una matriz $\mathbf{H}$ 3×3 no-singular. Para utilizar éste teorema, tomaremos un punto $\mathbf{x}$ en coordenadas homogéneas (i.e. vector de 3 elementos), y le aplicaremos la matriz $\mathbf{H}$, obteniendo un nuevo punto $\mathbf{x'=Hx}$.

Ilustremos éstos conceptos con una aplicación práctica.

Figura 2.2: **Eliminando la distorsión producto de la perspectiva.** *En la imagen de la izquierda las líneas de la base de las ventanas convergen en un punto finito, luego la imagen sufre distorsión perspectiva. La imagen de la derecha es una imagen sintetizada aplicando una homografía, ahora las líneas paralelas no convergen.* Fuente: [18]

En la figura 2.2 encontramos un ejemplo de lo que nos permite hacer, a nivel práctico, el concepto de homografía. Podemos eliminar la distorsión debida a la perspectiva aplicando una matriz de homografía $\mathbf{H}$ a cada punto de la imagen. Para conseguir la matriz $\mathbf{H}$, tomaremos cuatro puntos -donde como máximo 2 de ellos pueden ser colineales-, por ejemplo las cuatro esquinas de una ventana, y sus correspondientes en un rectángulo de tamaño conocido. Como hemos comentado anteriormente, la matriz $\mathbf{H}$ es invertible, por tanto podremos pasar de la primera imagen a la segunda, deshaciendo la perspectiva, y viceversa.

Otra cosa que nos permiten las homografías es el encadenado de transformaciones. Como veremos en la próxima sección, existen diferentes tipos de transformaciones, por lo que las homografías nos serán útiles para encadenar éstas transformaciones. Por ejemplo, una homografía se puede descomponer en el producto vectorial de tres matrices de transformación, cada una de un tipo diferente.

2.2. Modelo matemático del problema

RECTIFICACIÓN AFÍN

Una transformación afín (*afinidad*), a diferencia de una transformación proyectiva (*homografía*), mantiene invariable el paralelismo de las líneas, el ratio entre áreas y la línea en el infinito l_∞. En transformaciones de similaridad (*similaridades*) se mantiene, además de eso, el ratio entre la longitud de las líneas y los ángulos; y en transformaciones euclídeas (*isometrías*), además se mantiene invariable el área de los cuerpos y las longitudes de las líneas.

Dentro de ésta jerarquía de transformaciones, donde la proyectiva es la que menos propiedades mantiene invariables, la transformación afín presenta unas características peculiares. Por ejemplo, en una transformación proyectiva los puntos ideales pueden convertirse en puntos finitos, y por tanto l_∞ puede pasar a ser una línea finita. Ésto no pasa en una *afinidad*, donde l_∞ permanece en el infinito. Por tanto podemos decir que

Resultado 4. *Una transformación afín es la transformación lineal más general que mantiene las líneas en el infinito (l_∞), pero no necesariamente mantiene invariables los puntos ideales.*

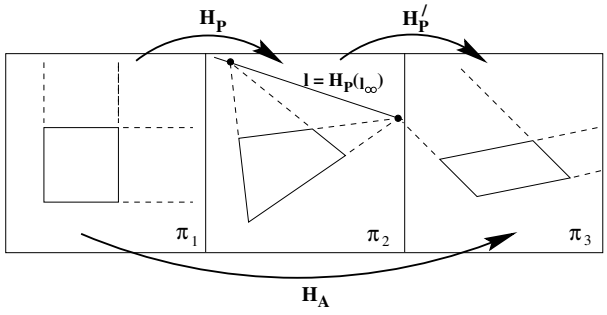

Figura 2.3: **Rectificación afín.** *H_P transforma l_∞ en el plano euclídeo π_1 en una línea l finita en el plano π_2. Al transformar con H'_P ésta línea l de nuevo en una línea en el infinito, podemos decir que H'_P es una afinidad. Por tanto, las propiedades afines del primer plano pueden ser medidas en el tercero.* Fuente: [18]

En la figura 2.3 vemos un ejemplo de como recuperar las propiedades afines de

una imagen. A priori es una idea un poco confusa de entender, pero sabemos que una línea en el infinito, l_∞, es una línea finita después de aplicar una homografía **H**si y sólo si **H** es una afinidad.

En la rectificación afín que nos interesa, los planos en perspectiva, podemos utilizar éstas ideas. La línea l_∞ en el mundo, se transforma en una línea de fuga l (del inglés *vanishing line*) en nuestra imagen. Más adelante trataremos éste tema, pero como avance, podemos decir que si encontramos la intersección de dos pares de líneas paralelas en la imagen, y unimos esos dos puntos, encontraremos esa línea l. Después, esa imagen puede ser rectificada (como ya hemos visto en la figura 2.3) de modo que l pase a ser $l_\infty = (0, 0, 1)^T$.

MODELO DE UNA CÁMARA

A efectos matemáticos, una cámara es un *mapping* entre el mundo 3D y una imagen 2D. El modelo más básico de cámara se llama ***modelo de pinhole***, también conocido como *cámara puntual*. La construcción práctica de éste modelo se basa en una caja cerrada en cuyo interior hay una placa de papel fotográfico, sensible a la luz, y en el otro extremo de la caja existe un pequeño agujero, que idealmente sería infinitamente pequeño. La luz del exterior pasará por el agujero formando una imagen en la placa del interior. Éste ejemplo es muy ilustrativo, y ayuda a entender rápidamente el modelo. Dado que el punto es infinitamente pequeño, podemos asumir que cada punto de la superficie visible del objeto observado estará representado por un solo punto en la imagen (ya que los rayos de luz viajan en línea recta) [10]. Éste modelo también es conocido como *proyección central*.

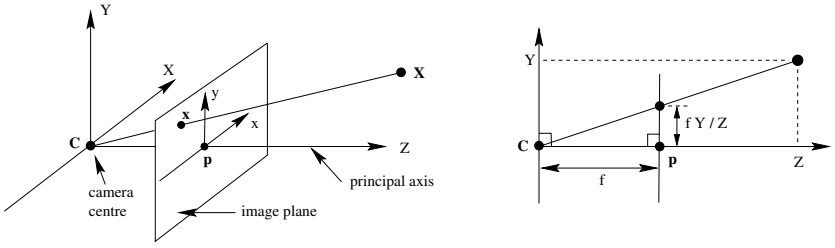

Figura 2.4: **Geometría de una cámara puntual.** ***C*** *es el centro de la cámara (en el origen de coordenadas), y P el punto principal (centro del plano imagen).* ***X*** *es un punto en el mundo 3D, cuyo rayo de luz acaba en el punto* ***x*** *de la imagen.* Fuente: [18]

¿Pero qué relación hay entre los puntos del mundo, y los puntos de nuestra imagen? Observemos la imagen de la derecha, en la figura 2.4. Un punto **X** en el mundo (*world*), a cierta altura y_{world}, se proyecta en la imagen (*image*) en el punto **x**, situado en $y_{image} = y_{world}/z_{world}$. Es decir, si mantenemos constante su altura en el mundo, y lo alejamos de la cámara (i.e. aumentamos z_{world}), su altura en la imagen (y_{image}) disminuirá. Encontramos el mismo comportamiento en x, por tanto $x_{image} =$

x_{world}/z_{world}. Así pues, el mapping en coordenadas homogéneas es

$$(x, y, z)^T \mapsto (fx/z, fy/z)^T$$

dónde f es la distancia entre el centro de la cámara C y el punto principal p.

Una vez expuesto el funcionamiento de la cámara más básica, en la siguiente sección nos centraremos en los aspectos que nos interesan a la hora de resolver nuestro problema. Explicaremos como, partiendo de las ideas descritas, llegaremos a calcular la posición del horizonte usando líneas de carril. Además, rectificaremos la imagen para poder calcular distancias con ella. Y finalmente, analizaremos los errores que se producen después de utilizar el método que expondremos, los cuales nos conducirán a diseñar las soluciones investigadas en éste proyecto.

SOLUCIÓN DEL PROBLEMA

¿Qué es lo que nos interesa extraer de la imagen 2D para obtener distancias en el mundo 3D? La respuesta: el plano de la carretera. Si logramos transformar el plano de la carretera -conociendo cuál es la línea del horizonte- desde la imagen en perspectiva, hasta obtener un plano imaginario -visto desde arriba, ya no en perspectiva- donde poder medir directamente distancias, habremos solucionado el problema.

Pero vayamos más allá en la pregunta. ¿Qué concepto anteriormente expuesto nos permite pasar de un plano en perspectiva -formalmente *plano proyectivo*- a un plano visto desde arriba -mejor dicho *plano afín*-? Efectivamente se trata de la homografía.

Necesitamos localizar la línea en el infinito l_∞ de alguna manera, y a continuación transformar el plano a un plano afín donde poder hacer los cálculos. En [13] se explican algunos métodos para encontrar la matriz H de homografía necesaria.

Partiendo de la imagen de la figura 2.5, explicaremos como conseguir la homografía que relacione la imagen con el plano real de la carretera. Los puntos P_a y P_b pertenecen a una línea ortogonal al eje X [2]. Pasa lo mismo con P_c y P_d. Por el resultado 3, si calculamos el producto vectorial de P_a por P_b, obtendremos la línea $l_1 = P_a \times P_b$. Igualmente con los otros dos puntos: $l_2 = P_c \times P_d$.

Ahora calculamos el punto de fuga $F_1 = l_1 \times l_2$, obteniendo el vector $F_1 = (f_1, f_2, 1)$. Además, podemos suponer que tenemos otro punto de fuga en el infinito, de coordenadas $F_2 = (1, 0, 0)$, que junto a nuestro punto de fuga F_1 en la dirección de la carretera, nos proporciona la línea en el infinito $l_\infty = F_1 \times F_2$ que necesitábamos para recuperar las propiedades afines comentadas anteriormente.

[2] Pensaremos que el eje X atraviesa la imagen horizontalmente, el Y verticalmente, y Z se adentra en la imagen. Ver la figura 2.4.

Figura 2.5: **Horizonte mediante la intersección de líneas de carril.** *La línea l_1 pasa por P_a y P_b y la línea l_2 por P_c y P_d. Ambas líneas son paralelas, luego intersectan en un punto en el infinito en el mundo real, que en proyección perspectiva es el punto finito F, conocido como* ***punto de fuga*** *o* ***vanishing point****.*

Finalmente, asumiendo que:

- el plano imagen es prácticamente paralelo al plano $X-Y$ real (la cámara apunta al infinito, paralela al eje Z), y que
- la carretera es prácticamente plana,

sabemos que la matriz de homografía es

$$\mathbf{H} = \begin{pmatrix} 1 & -f_1/f_2 & 0 \\ 0 & 1 & 0 \\ 0 & -1/f_2 & 1 \end{pmatrix}$$

Ahora ya estaríamos en condiciones de medir la distancia **de cualquier segmento paralelo a las líneas l_1 y l_2**. Si además quisiésemos medir otras distancias que no cumplan esa característica, podríamos encontrar otro punto de fuga no en el infinito [13], pero para nuestro problema no nos interesa.

EJEMPLO PRÁCTICO

A continuación detallaremos paso a paso un ejemplo del cálculo de distancias que hemos explicado antes.

Figura 2.6: **Ejecución de ProvaDistancia.** *Calcularemos distancias sobre ésta imagen. Éste es el primer paso del programa, donde se marcan los extremos de dos líneas de carril para calcular el punto de fuga. Hemos añadido las letras para referenciarlas en la explicación.*

Aplicaremos el algoritmo sobre la imagen de la figura 2.6, de 360×288. Lo primero que haremos será marcar los extremos de dos segmentos paralelos, y en nuestra dirección. Las líneas de carril nos servirán. Así pues las coordenadas de los puntos serán [3]:

$$A = (49, 127),\ B = (110, 89),\ C = (356, 137) \text{ y } D = (307, 100).$$

A continuación calculamos los vectores de las líneas (producto vectorial de los puntos):

$$l_1 = A \times B = (49, 127, 1) \times (110, 89, 1) = (38, 61, -9609)\ l_2 = D \times C = (-37, 49, 6459)$$

con las cuales ya podemos calcular su intersección, es decir, el punto de fuga (*vanishing point*) F:

$$F = l_1 \times l_2 = (38, 61, -9609) \times (-37, 49, 6459) = (864840, 110091, 4119)$$

el cual en coordenadas no homogéneas (dividiendo por su tercera componente) es el punto:

$$\mathbf{F} = (f_1, f_2) = (209'96, 26'72)$$

[3] Al ser un programa en Matlab, el origen $(0, 0)$ se encuentra en la esquina superior izquierda de la imagen)

Con éste punto, ya podemos tener nuestra matriz de homografía de la imagen:

$$\mathbf{H} = \begin{pmatrix} 1 & -f_1/f_2 & 0 \\ 0 & 1 & 0 \\ 0 & -1/f_2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & -209'96/26'72 & 0 \\ 0 & 1 & 0 \\ 0 & -1/26'72 & 1 \end{pmatrix}$$

A continuación marcamos los dos extremos de un segmento de longitud conocida y en la misma dirección, por ejemplo la flecha. Sabemos, porque las medidas de marcas viales son estándar, que la flecha mide *7.5 metros*. Además, marcando sus extremos obtenemos sus coordenadas homogéneas:

$$extremo_1 = (202, 235, 1) \text{ y } extremo_2 = (205, 70, 1)$$

Las dispondremos en vertical para poder trabajar con ellas.

$$\mathbf{segmento} = \begin{pmatrix} 202 & 205 \\ 235 & 70 \\ 1 & 1 \end{pmatrix}$$

Ahora transformaremos éstos extremos del *plano proyectivo* al plano *afín mediante* la homografía H.

$$\mathbf{segmentoAfin} = H \cdot segmento = \begin{pmatrix} 1 & -209'96/26'72 & 0 \\ 0 & 1 & 0 \\ 0 & -1/26'72 & 1 \end{pmatrix} \cdot \begin{pmatrix} 202 & 205 \\ 235 & 70 \\ 1 & 1 \end{pmatrix}$$

Nuestro segmento de longitud conocida, en coordenadas coordenadas afines, es

$$\mathbf{segmentoAfin} = \begin{pmatrix} 211{,}92 & 213{,}02 \\ -33{,}32 & -43{,}23 \\ 1 & 1 \end{pmatrix}$$

Ahora ya estamos en disposición de calcular el *factor de escalado*[4]:

$$length = \text{longitud real del segmento}$$
$$\boldsymbol{factorEscalado} = length/(-33{,}32 - (-43{,}23)) = -0{,}7568$$

Finalmente, para calcular distancias sólo tenemos que transformar las coordenadas de los puntos marcados y de la base de nuestra imagen a coordenadas afines de la misma manera (multiplicando por la matriz de homografía). Así pues, las nuevas coordenadas son:

[4]El factor de escalado se define como la relación de escala entre longitudes en el plano afín y el plano real de la carretera.

$$baseImagen = (1, 288, 1) \implies \textbf{baseImagenAfin} = (1, -29{,}46, 1)$$
$$punto_1 = (137, 46, 1) \implies puntoAfin_1 = (311'15, -63'79, 1)$$
$$punto_2 = (233, 48, 1) \implies puntoAfin_2 = (181'01, -60'30, 1)$$

Ahora ya podemos calcular la distancia real a cada punto, a la que añadiremos lo que se llama *distancia base*, es decir, la distancia desde la cámara hasta la primera fila que vemos en la imagen, que nos queda oculta.

$$dist(punto_n) = distBase + (puntoAfin - baseImagenAfin) * factorEscalado$$

Si distanciaBase = 5.0 metros

$$dist(punto_1) = 5{,}0 + (-63'79 + 29'46) * -0{,}7568 = \textbf{30'98} \text{ metros}$$
$$dist(punto_2) = 5{,}0 + (-60'30 + 29'46) * -0{,}7568 = \textbf{28'34} \text{ metros}$$

Figura 2.7: **Ejecución de ProvaDistancia.** *Éste es el resultado del programa. La cruz en negro marca el punto de fuga, y las cruces blancas marcan dos puntos de los que se ha calculado la distancia (entre llaves)*

2.3. Consecuencias del error en la localización del horizonte

En el ejemplo anterior hemos explicado la solución al problema de calcular distancias. Éste método da muy buenos resultados cuando como en nuestro caso lo ejecutamos sobre un fotograma concreto. El objetivo de marcar las líneas de carril simplemente es obtener el punto de fuga (*vanishing point*), que a su vez se encuentra en el horizonte. Una vez tenemos el horizonte, podemos empezar a rectificar la imagen. Si además queremos calcular las distancias, podemos tener un segmento que vaya en la dirección del punto de fuga, y del cual sepamos su longitud. La longitud de éste segmento puede ser fijada inicialmente -es un estándar de tráfico-.

¿Cuál es el problema entonces? El problema es la precisión del cálculo del horizonte. Podríamos suponer un horizonte fijo, que se mantenga constante desde el principio según los parámetros de la cámara. A continuación expondremos los errores producidos al suponer un punto de fuga fijo.

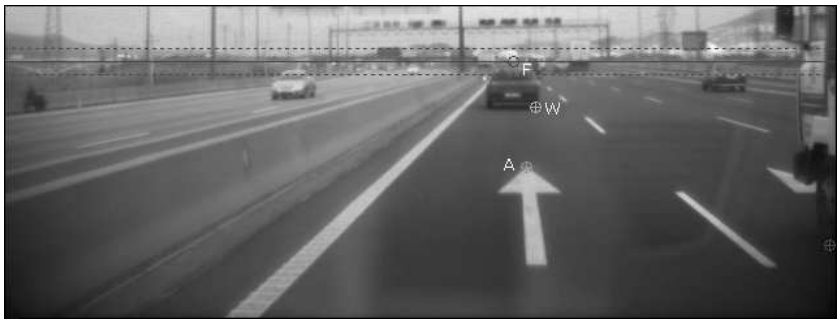

Figura 2.8: **Detección del horizonte.** *La línea continua marca señala el horizonte real, y las discontinuas el horizonte nominal (dentro de que margen se encuentra el real).* ***F*** *marca el punto de fuga,* ***W*** *marca un punto del que queremos calcular distancias (una rueda trasera), y* ***A*** *señala un extremo de nuestra flecha de carril.*

En la figura 2.8 observamos una fotografía de una carretera, de la cual hemos calculado el horizonte, por ejemplo usando el método explicado en la sección anterior. Las aceleraciones, desaceleraciones después de frenar, baches del carril, etc. mueven todos los puntos marcados en la figura, y muy especialmente el que más nos interesa: el punto de fuga.

Como vemos en las figuras 2.9 y 2.10, cuanto más cercano es el punto, mayor será el error debido a desplazamientos del horizonte. Es por ello que resulta determinante obtener una buena aproximación del horizonte. Errores de 10 píxeles hacia arriba en la detección, nos producen hasta 35 metros de error al calcular la distancia al vehículo precedente en éste ejemplo.

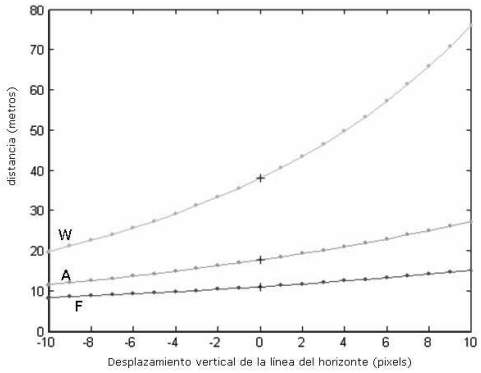

Figura 2.9: **Variación de la distancia debido al desplazamiento del horizonte.** ***F*** *es el punto de fuga,* ***A*** *es la flecha y* ***W*** *la rueda de un coche precedente. A medida que el horizonte detectado en nuestra imagen varía, la distancia a éstos puntos en el mundo recibe un error.*

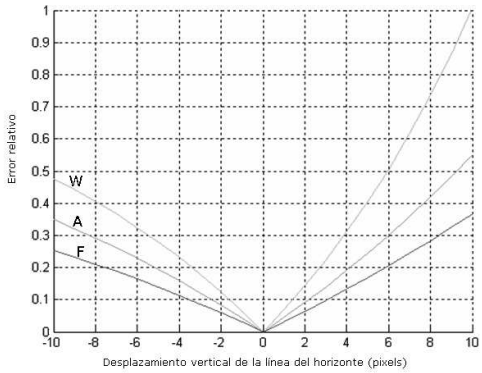

Figura 2.10: **Error relativo debido al desplazamiento del horizonte.** *Encontramos el mismo comportamiento, pero ahora presentado en porcentajes.*

Finalmente, podemos concluir que **el cálculo correcto de la posición horizonte es el paso más importante en el problema de calcular distancias**.

A continuación entramos en materia, con tres capítulos explicando los tres métodos implementados para detectar el horizonte. Después, dos capítulos de resultados globales y conclusiones, en las que expondremos las impresiones y aspectos que hemos extraído de nuestro trabajo.

Capítulo 3

Proyección horizontal

El primer método para detectar el horizonte que vamos a estudiar es la *proyección horizontal*. Se trata del método que menos teoría matemática tiene, pero el que mejores resultados aporta en nuestro proyecto.

A grandes rasgos, la idea es utilizar la proyección horizontal de la imagen -ésto es, la suma de los valores de cada fila- para conseguir un histograma de intensidad. Entonces, mediante varios métodos analizados e implementados, calcular el desplazamiento vertical de un horizonte fijado inicialmente.

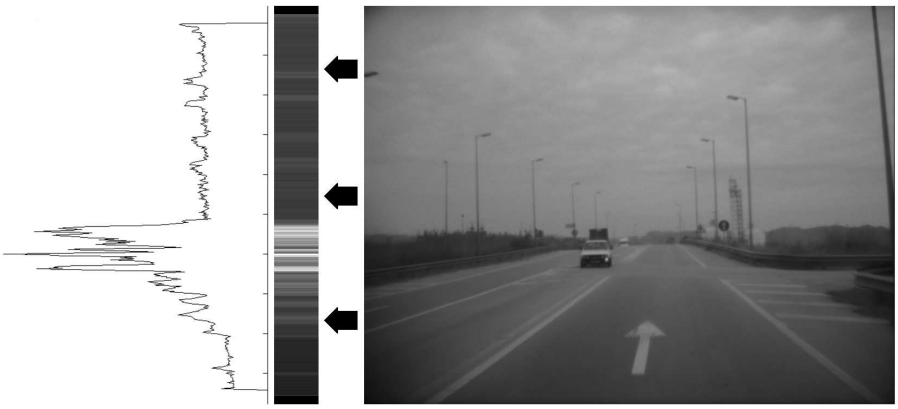

Figura 3.1: **Proyección horizontal de los contornos de la imagen e histograma de intensidades.** *Aquí tenemos la idea general de éste método. Partiendo de la imagen de la derecha, un frame cualquiera de nuestra visión frontal de la carretera, calculamos los contornos y después sumamos los valores de cada una de sus filas. A medida que avance el tiempo, ésta proyección horizontal se desplazará verticalmente.*

3.1. Análisis

Para iniciar el estudio del problema, hemos tomado como ejemplo de pruebas un fragmento de un video de 200 fotogramas [1]. Se ha escogido éste fragmento por contener los casos más significativos que nos desplazan el horizonte, que son los que nos interesa especialmente analizar:

- **Badenes y agujeros** en el pavimento producen desplazamientos positivos. El vehículo baja, la cámara baja con él, y por tanto el horizonte subirá.
- Los **resaltos** producen desplazamientos negativos. La cámara sube con el vehículo por tanto el horizonte bajará en nuestra imagen.
- **Resaltos o badenes sólo en un lado** del coche producen rotaciones en Z de la cámara. Son bastante raros de encontrar, y a menos que vayan acompañados de uno de los casos anteriores, no tienen demasiado impacto en nuestro problema.
- **Saltos en el vídeo** producen desplazamientos de todo tipo. En éste caso existen pequeños saltos entre ciertos frames del vídeo -faltan algunos fotogramas-, producidos por los tiempos de escritura en cámaras con disco duro por ejemplo. Su impacto es el mismo que el de los dos primeros casos, y como veremos es indistinguible de los demás.
- **Señales de tráfico** cercanas o aéreas, o que desaparecen de nuestro campo de visión de un fotograma al siguiente, propiciarán la aparición de errores ya que suelen contener multitud de contornos.

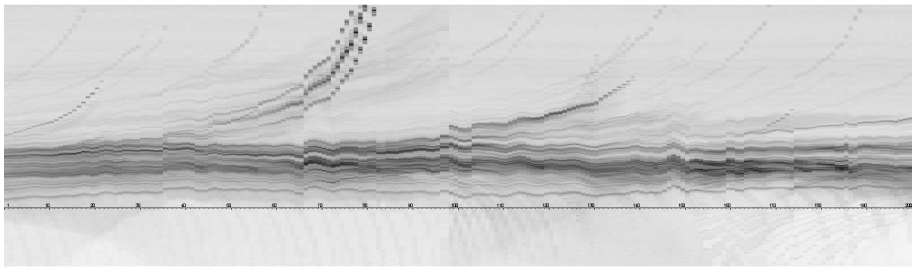

Figura 3.2: **Proyecciones horizontales de contornos en cada frame del vídeo, en escala de grises invertida.** *Cada una de las columnas que vemos es la proyección horizontal de los contornos de un fotograma. Podemos apreciar el desplazamiento que sufren a medida que avanzamos en el tiempo, lo cual nos provocará errores en el cálculo de distancias.*

[1] Éste vídeo puede encontrarse en el CD-ROM anexo, en el directorio `/videos/video_analysis`, en formato de una imagen ".tif" por frame.

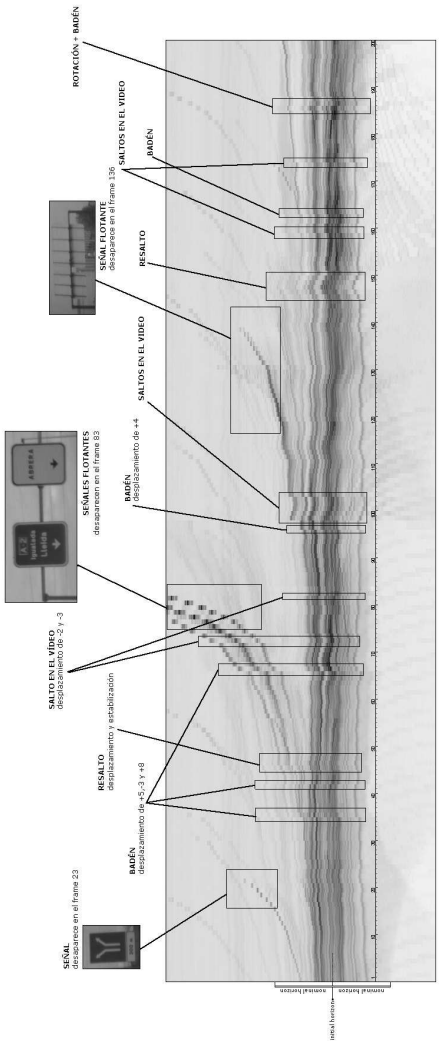

Figura 3.3: **Análisis de las causas de los desplazamientos del horizonte.** *La misma figura anterior ha sido analizada para encontrar las causas de los desplazamientos. Encontramos los puntos enumerados en ésta sección, y además el desplazamiento de pixels calculado manualmente.*

3.2. Diseño del programa

Antes de empezar a hablar sobre las aproximaciones realizadas en éste método, explicaremos el funcionamiento global de éste, y especialmente el algoritmo empleado para testear los resultados rápidamente.

En principio, la creación de una interfaz gráfica no era ni siquiera una idea, mucho menos un objetivo que alcanzar. Aún así, se consideró que podría ayudar a dinamizar y facilitar las pruebas, y que además, la interacción final con un interfaz siempre es más rápida e intuitiva que con un programa por línea de comandos basado en parámetros.

Figura 3.4: **Interfaz de nuestro programa *(Horizontal Projection)*.** .

En la figura podemos ver una captura de la aplicación. A la izquierda encontramos la lista de parámetros: el horizonte inicial, el tamaño de la ventana nominal, la ventana de correlación, el desplazamiento máximo permitido y el *step* -cada cuantos fotogramas realizaremos un nuevo cálculo-. Éstas constantes serán explicadas en detalle más adelante [2].

Una vez escogidos éstos parámetros, ejecutaremos el programa escogiendo el tipo de filtrado inicial de la imagen, la ventana sobre la que se realizarán los cálculos, y el método empleado

[2]Para conocer el funcionamiento del programa, consultar la ayuda del mismo en el menú `Help ->Horizon Detection Help`

A continuación tenemos el pseudo-código del algoritmo principal.

```
ParaTodo i en [1..numFrames] {
  imagen = leerImagen(frame[i]);
  Si frame[i] ha de ser procesado {
    proyeccion = suma de las filas de la imagen
    caso {
      <diferencia>            : desplazamiento = diferenciaEntreProyeciones
      <correlacion>           : desplazamiento = correlacion            "
      <correlacion normal>    : desplazamiento = correlacion_normal     "
      <programacion dinamica> : desplazamiento = programacion dinamica  "
      }
      horizonte = horizonteAnterior + desplazamiento

      horizonteAnterior = horizonte
      proyeccionAnterior = proyeccion
  }
}
```

El bucle principal procesa el video cargando un nuevo fotograma por iteración. En una fase más avanzada del diseño incluimos un parámetro llamado *step* -en español *paso*-, el cual nos permite acelerar la ejecución del programa no calculando ciertos frames intermedios. Es decir, si $step = 5$ se realizarán los cálculos entre los frames 1-5, 5-10, 10-15, etc. De ésta manera los desplazamientos inter-frame serán más significativos [3]. Por tanto, si en la iteración actual hemos de realizar cálculos, entramos en el cuerpo de la condición, en caso contrario, arrastraremos la última proyección que tengamos.

Una vez dentro del cuerpo del *Si*, hacemos la proyección horizontal de la imagen, es decir, sumamos los elementos de cada fila, con lo cual obtendremos una columna. Es con esa proyección con la que calcularemos el desplazamiento usando los distintos tipos de correlaciones. Finalmente, corregimos el horizonte gracias al desplazamiento que hemos calculado.

En éste ejemplo vemos como se realiza la proyección horizontal por si quedaba algún tipo de duda. Como vemos es bastante simple:

$$Sum \begin{pmatrix} 5 & 1 & 4 & 2 & 3 & 4 & 6 \\ 2 & 7 & 6 & 0 & 5 & 2 & 3 \\ 1 & 4 & 3 & 3 & 9 & 6 & 3 \\ 5 & 2 & 4 & 2 & 5 & 5 & 0 \end{pmatrix} \Rightarrow \begin{pmatrix} 25 \\ 25 \\ 29 \\ 23 \end{pmatrix} = \textbf{Proyección Horizontal}$$

[3] En todo caso, en el método de proyección horizontal el mejor valor de step, empíricamente hablando, es 1, ya que nos interesan más los pequeños desplazamientos que los grandes. Será en los dos siguientes métodos donde ésta variable será verdaderamente importante.

3.3. Aproximación 1: Diferencia de proyecciones

La primera aproximación que veremos es el cálculo del desplazamiento utilizando la diferencia entre dos proyecciones consecutivas. Se trata de la aproximación más básica.

El algoritmo es el siguiente:

```
Fun diferenciaEntreProyecciones(proyeccion,proyeccionAnterior) {
  ParaTodo pixeles en [-10..10] {
    proyeccionDesplazada = desplazarProyeccion(proyeccion,pixels)
    diferencia = sumaTotal(abs(proyeccionDesplazada - proyeccionAnterior))
    Si (diferencia < diferenciaMinima) {
     diferenciaMinima = diferencia
     desplazamiento = -pixeles
    }
  }
}
```

Su funcionamiento empieza por un bucle que reseguirá una ventana de correlación, y que en el algoritmo hemos expresado con el rango de -10 a 10, pero que en la implementación realmente es variable. Dependiendo del valor de *pixeles* en la iteración del bucle, desplazaremos nuestra proyección actual a más o menos altura. Después, calculamos el valor absoluto de la resta de las dos proyecciones (no es más que una resta entre dos vectores), y finalmente sumaremos todos los elementos del vector resultante. El desplazamiento final será el valor de pixeles (i.e. cuánto hemos movido nuestra proyección) que produzca la *sumaTotal* más baja, es decir, la diferencia mínima entre dos proyecciones). Llanamente, podríamos decir que gana el desplazamiento que mejor haga encajar las dos proyecciones.

EJEMPLO TEÓRICO

Veamos un ejemplo:

$$\mathbf{Proyeccion} = \begin{pmatrix} 35 \\ 12 \\ 21 \\ 55 \\ 28 \end{pmatrix} \qquad \mathbf{ProyeccionAnterior} = \begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 16 \end{pmatrix}$$

Si nos encontramos en la iteración donde $pixeles = 0$, es decir, estamos comparando directamente las dos proyecciones, sin desplazar ninguna, obtenemos el siguiente desplazamiento:

$$\textbf{diferencia} = sumaTotal \left(abs \left(\begin{pmatrix} 35 \\ 12 \\ 21 \\ 55 \\ 28 \end{pmatrix} - \begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 16 \end{pmatrix} \right) \right) = sumaTotal \begin{pmatrix} 23 \\ 9 \\ 34 \\ 27 \\ 12 \end{pmatrix} = \textbf{105}$$

En la siguiente iteración del bucle, tendremos $pixeles = 1$, por lo que antes de realizar el cálculo tendremos que desplazar la proyección actual una unidad [4].

$$\textbf{diferencia} = sumaTotal \left(abs \left(\begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 0 \end{pmatrix} - \begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 16 \end{pmatrix} \right) \right) = sumaTotal \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 16 \end{pmatrix} = \textbf{16}$$

El desplazamiento del horizonte en el fotograma actual respecto al anterior será el número de pixeles (negado) para el cual el resultado de la diferencia sea mínima. Por tanto $min(105, 16) = 16$, que se daba en el caso que $pixeles = 1$, por tanto $desplazamiento = -pixeles = -1$.

Éste es un caso ideal, pues los valores de las proyecciones coinciden exactamente, ésto no pasará en proyecciones extraidas de fotogramas reales, pero la tendencia y el funcionamiento del algoritmo será el mismo.

EJEMPLO PRÁCTICO

Ahora apliquemos el algoritmo sobre fotogramas reales de nuestro vídeo de ejemplo.

La figura 3.5 nos muestra la proyección horizontal de dos fotogramas consecutivos. Se trata de un badén, en el cual nuestro vehículo bajará, por tanto el horizonte se desplazará hacia arriba. Efectivamente, si observamos detalladamente las imágenes, veremos como los objetos en el fotograma 67 están ligeramente más arriba en la imagen que en el fotograma 66. Éste movimiento se reflejará en la proyección horizontal, que también estará ligeramente más hacia arriba que la anterior.

[4] En nuestra implementación los espacios vacíos que deja un desplazamiento son rellenados por el valor que pasemos mediante un parámetro, en éste caso concreto rellenaremos con el valor 0

Figura 3.5: **Cálculo de la diferencia de proyecciones entre dos frames.** *A la izquierda, frame 66 del vídeo de pruebas junto a su proyección horizontal. A la derecha, el fotograma siguiente junto a su proyección.*

En la figura 3.6 vemos el cálculo entre los dos fotogramas de la anterior figura. Hemos utilizado el rango [-5..+5] en lugar de [-10...10] para que se aprecien bien las imágenes.

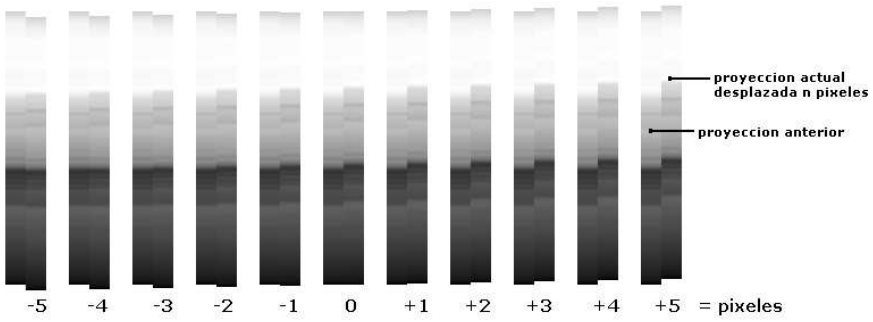

Figura 3.6: **Bucle de correlación.** *Aquí vemos el bucle de correlación con pixeles* $= [-5.,5]$*, la proyección anterior se compara con la proyección actual desplazada mediante el cálculo explicado, y gana la correlación cuya suma total sea mínima.*

Aplicando el algoritmo sobre el vídeo de pruebas completo obtenemos el resultado mostrado en \results\horizontalProjection\difference\originalImage-difference-fullImage.

A primera vista lo observamos que el horizonte calculado (*computedHorizon*) tiende a irse hacia abajo en la imagen (hacia donde va la carretera), si bien normalmente mantiene el mismo comportamiento que un horizonte fijado (*initialHorizon*), es decir, no corrige los desplazamientos del horizonte causados por baches, señales, etc.

CONCLUSIONES

Veamos nuestras conclusiones:

- **Diferencia.** El uso de la diferencia entre proyecciones es poco sensible a resaltos o badenes. La razón es que es linear, y pequeñas diferencias de intensidad en zonas no importantes de la imagen ocultan características que deberían ser las que verdaderamente deberíamos tener en cuenta.
- **Uso de la imagen original.** El emplear la imagen original para realizar las proyecciones nos produce resultados muy erróneos. En realidad ésta es la primera consideración que se debería tener en cuenta, ya que hacer los cálculos sin filtrar características previamente nos hace tener en cuenta todos los puntos de la imagen, sean útiles o no. Además, si no filtramos estaremos trabajando con el ruido implicito de la imagen, y que por ejemplo en ésta aproximación nos provoca resultados impredecibles.

3.4. Aproximación 2: Correlación de contornos

En nuestra segunda aproximación utilizaremos la correlación, entendida como la suma de los elementos del producto entre proyecciones. En éste caso se introduce un factor no lineal que nos ayuda a distinguir entre zonas importantes con mucha intensidad (muchos contornos) y zonas menos significativas para el cálculo.

Primeramente, calculamos los contornos de la imagen, y hacemos la proyección:

```
contornosImagen = detectarContornos(imagen)
proyeccion = suma de las filas de la imagen
```

El algoritmo del cálculo, muy parecido a la aproximación que hemos explicado antes, es el siguiente:

```
Fun correlacionEntreProyecciones(proyeccion,proyeccionAnterior) {
  ParaTodo pixels en [-10..10] {
    proyeccionDesplazada = desplazarProyeccion(proyeccion,pixels)
    correlacion = sumaTotal(proyeccionDesplazada * proyeccionAnterior)
    Si (correlacion > correlacionMaxima) {
     correlacionMaxima = correlacion
     desplazamiento = -pixels
    }
  }
}
```

La explicación es también bastante similar a la diferencia de proyecciones. Desplazamos la proyección actual en vertical para lograr encajar de la mejor forma con la proyección anterior. Ésta vez, en lugar de calcular el valor absoluto de la resta, multiplicamos los dos vectores. De ésta manera, zonas con niveles de intensidad muy altos -por ejemplo la zona del horizonte- serán multiplicadas por zonas con niveles de intensidad también altos en caso de correspondencia. Finalmente sumamos los elementos del vector final, y escojemos la mayor de las sumas. Al igual que en la diferencia, el desplazamiento será el número de pixeles negado para el cual la correlación sea máxima.

PORQUÉ UTILIZAR CONTORNOS

Antes de entrar en un ejemplo teórico responderemos a una pregunta importante: ¿Por qué no utilizamos la imagen original en lugar de los contornos? Además de la razón explicada en la aproximación anterior -hay puntos (zonas) en la imagen más importantes que otros, y una manera de discriminarlos es mediante filtros de contornos, de esquinas, de crestas, de líneas, etc.-, existe otra razón a nivel de implementación.

Tengamos presente el algoritmo: para hacer la correlación desplazamos la proyección actual desde -10 a +10, por ejemplo, y vamos multiplicando los elementos por el vector de la proyección anterior. Cuando movemos la proyección 10 unidades hacia arriba, rellenamos los 10 últimos elementos del vector (columna) con ceros. Después, multiplicamos los dos vectores elemento a elemento, y hacemos la suma global. Está claro que el producto entre esos nuevos elementos que nos han aparecido y su correspondiente en la proyección anterior siempre dará 0 (porqué el primero de ellos será 0). Por tanto, para cualquier desplazamiento $\neq 0$ de la proyección tendremos elementos con valor 0, que no aportarán nada a la suma final.

Si intentamos solucionar éste problema rellenando los nuevos espacios del vector con el máximo valor posible, por ejemplo 255, acabaremos "'dando ventaja"' a los desplazamientos -10 y +10 (máximos desplazamientos posibles en nuestro ejemplo), ya que se estarán multiplicando elementos de la proyección anterior por el máximo valor posible (e.g. 255), con lo cual en la suma siempre ganarán los desplazamientos extremos. ¿Por qué no usar un valor intermedio, por ejemplo la media entre los valores de intensidad del vector? Si escogiésemos un valor "'neutral"', por ejemplo la media, estaríamos influyendo en el comportamiento del algoritmo, y además en la mayoría de los casos tendríamos el mismo problema que si rellenásemos los nuevos elementos con 0.

Así pues, hemos llegado a la conclusión de que en métodos de correlación, o en general métodos que utilicen producto o división en sus cálculos, no se han de usar imágenes originales, sino imágenes filtradas, ya que éstas tienen la propiedad de que sus extremos tienen valores cercanos a 0 (negro).

EJEMPLO TEÓRICO

Tomemos de nuevo los dos vectores usados en la diferencia de proyecciones.

$$\textbf{Proyeccion} = \begin{pmatrix} 35 \\ 12 \\ 21 \\ 55 \\ 28 \end{pmatrix} \qquad \textbf{ProyeccionAnterior} = \begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 16 \end{pmatrix}$$

Para $pixeles = 0$:

$$\textbf{correlacion} = sumaTotal \left(\begin{pmatrix} 35 \\ 12 \\ 21 \\ 55 \\ 28 \end{pmatrix} * \begin{pmatrix} 12 \\ 21 \\ 55 \\ 28 \\ 16 \end{pmatrix} \right) = sumaTotal \begin{pmatrix} 420 \\ 252 \\ 1155 \\ 1540 \\ 448 \end{pmatrix} = \textbf{3815}$$

Y ahora para $pixeles = 1$:

$$\textbf{correlacion} = sumaTotal\left(\begin{pmatrix}12\\21\\55\\28\\0\end{pmatrix} * \begin{pmatrix}12\\21\\55\\28\\16\end{pmatrix}\right) = sumaTotal\begin{pmatrix}144\\441\\3025\\784\\0\end{pmatrix} = \textbf{4394}$$

El máximo entre las dos correlaciones es 4394, por lo que $desplazamiento = -pixeles = -1$.

EJEMPLO PRÁCTICO

En éste caso, el algoritmo es el mismo, sólo que antes de calcular el desplazamiento, detectamos los contornos horizontales de la imagen, y después hacemos la proyección horizontal. En la siguiente figura se ilustra ésta idea.

Figura 3.7: **Proyección horizontal de los contornos horizontales de la imagen.** *Partiendo de la imagen original, calculamos sus contornos convolucionándola con la derivada de la gaussiana, para después proyectar.*

En la figura 3.7 vemos como la proyección horizontal sólo de los contornos nos aporta más datos que la proyección de la imagen original. Vemos un aumento de la intensidad en el horizonte nominal, lo cual se refleja en un cálculo más preciso del desplazamiento.

El resultado de la correlación de contornos sobre nuestro vídeo de ejemplo lo podrán encontrar en \results\horizontalProjection\correlation \contours-correlation-fullImage.

Viendo el resultado, podemos apreciar que ahora la línea del horizonte es correcta en sus primeros pasos, cuando encuentra resaltos o baches los compensa bien, pero al encontrar señales aereas como las comentadas en el análisis de éste capítulo, el horizonte se desvía y tiende a acercarse a ellas.

CONCLUSIONES

Finalmente, las conclusiones que extraemos:

- **Contornos.** Tan sólo tenemos en cuenta las partes importantes de la imagen, que es lo que necesitábamos.
- **Correlación.** Realiza un cálculo no lineal, puesto que los puntos con intensidad alta tienen más importancia que los de intensidad baja, por tanto el horizonte (i.e. el que tiene más contornos) influirá más en el resultado que todo lo demás.
- **Cálculo sobre la imagen completa.** El problema ahora es que elementos que no nos interesan, como son las señales de tráfico, que se mueven a mucha velocidad y contienen muchos contornos, atraen el horizonte hacia ellas. La solución a éste problema, en la siguiente aproximación.

3.5. Aproximación 3: Correlación de contornos en el horizonte nominal

Ésta vez utilizaremos el mismo algoritmo que en la aproximación anterior, ya que se sigue tratando de correlación, pero ahora en lugar de aplicar los cálculos a la proyección de los contornos de la imagen entera, sólo tendremos en cuenta una ventana de cierto tamaño alrededor de nuestro horizonte, que llamaremos horizonte nominal.

EJEMPLO PRÁCTICO

En \results\horizontalProjection\correlation\contours-correlation - nominalHorizon tenemos el resultado.

Ahora las características que no nos interesan quedan fuera de nuestra proyección, por lo que el desplazamiento se calculará sobre datos útiles y fiables. El tamaño de la ventana no es algo fijado que pueda definirse como constante para imágenes de todos los tamaños, pero aproximadamente una ventana de un 25 % de alto respecto a la altura de la imagen, alrededor del horizonte inicial, dará resultados aceptables. Ventanas más pequeñas podrían producir errores en las cuestas, por ejemplo, donde el horizonte sube o baja muy significativamente respecto al horizonte inicial.

CONCLUSIONES

Las conclusiones son:

- **Contornos.** Ya se ha comentado anteriormente, pero como mejoras podríamos introducir detección de líneas, esquinas o crestas en lugar de contornos.
- **Correlación.** Como mejoras podríamos desarrollar un método más adaptativo y dinámico, como la correlación normalizada.
- **Cálculo sobre el horizonte nominal.** Se trata de una gran ventaja, ya que además de evitar errores, acelera la velocidad del cálculo.

3.6. Mejoras

De entre las posibles mejoras, hemos analizado e implementado dos: la correlación normalizada y la programación dinámica.

CORRELACIÓN NORMALIZADA

Se trata de un cálculo más adaptativo que la simple correlación. Utilizamos la función $corrcoef()$ de Matlab, que calcula el coeficiente de correlación entre nuestros dos vectores de datos (nuestras dos proyecciones). Se basa en conceptos de estadística, y la fórmula es la siguiente:

$$r = \frac{\sum_1^N z_X z_Y}{N-1}$$

donde

N es el número de variables (i.e. altura en píxeles de la proyección).
z_X es la transformación z de X, definida por $z_X = \frac{X-\bar{X}}{s_X}$
z_Y es la transformación z de Y, definida por $z_Y = \frac{Y-\bar{Y}}{s_Y}$
en ambos casos s_X y s_Y es la desviación estándar.

El resultado de utilizar el coeficiente de correlación para calcular el desplazamiento es que, por ejemplo, ahora se adapta mejor a diferentes iluminaciones en dos frames consecutivos, porque primero normalizamos las muestras y usamos su desviación, en lugar de realizar el cálculo con los datos sin tratar.

En el directorio \RESULTS\HORIZONTALPROJECTION\NORMALIZEDCORRELATION se pueden ver los resultados de ésta mejora.

PROGRAMACIÓN DINÁMICA

Ésta última mejora intenta obtener un método aún más adaptativo, partiendo de la idea de que la diferencia entre las proyecciones horizontales de distintos frames no se basa sólo en desplazamientos lineales, sino que también se aplica un escalado.

Basándonos en la excelente explicación del funcionamiento de la programación dinámica aplicada a la alineación de proteínas en cadenas de ADN en [5], hemos desarrollado un algoritmo apropiado para nuestro problema. Aconsejamos revisar la fuente original, que contiene una explicación muy ilustrativa y fácil de entender, antes de continuar leyendo nuestra implementación, ya que pasamos por alto detalles en los que no podemos entrar para no alargarnos demasiado en la explicación.

Dispondremos los vectores de manera que tengamos el primero en forma de columna y el segundo en forma de fila, así nos enmarcarán una matriz que luego rellenaremos.

El algoritmo es:

```
Fun programacionDinamica(vector1,vector2) {
  Filtrado de los vectores (suavizar, normalizar, etc.)

  Inicialización de cada elemento de la matriz M así:

    Si M(i,j) = M(i-1,j-1) {  // Si coinciden los valores
      S=2                     // de los dos elementos puntuaremos
    } Sino {                  // sino penalizaremos
      S=-1
    }

    M(i,j) = MAX(
                 M(i-1,j-1) + S,
                 M(i,j-1) + w,   // w es una penalización (e.g. -2)
                 M(i-1,j) + w
                )

    (Además guardamos cual de las nuevas posiciones escogemos)

  Fase de traceback:
    Desde el último elemento de la matriz (última columna y fila)
    volvemos por el camino que habíamos guardado

  desplazamiento = Nº de desplazamientos horizontales realizados
                   en la etapa de traceback

}
```

Usando éste algoritmo queremos que nuestra nueva proyección se adapte al histograma de la proyección anterior, así podremos contabilizar en qué sitios ha tenido que estirarse y moverse, con lo cual obtendremos el desplazamiento.

Llegados a éste punto, encontramos dos problemas:

- el contabilizar los movimientos de la primera forma (proyección) necesarios para adaptarse a la segunda forma no nos dice nada sobre el desplazamiento entre dos frames, ya que simplemente trata de adaptarse, no de medir una correlación.
- aunque consiguiésemos medir cual es el desplazamiento, no tenemos forma de ver si éste ha sido hacia arriba o hacia abajo en la imagen.

Pensemos pues en un refinamiento del algoritmo. En la figura 3.8 vemos las dos proyecciones, donde claramente una está desplazada respecto la otra. La solución que hemos pensado es recortar la primera por los dos primeros máximos significativos de los extremos, y después adaptarla a la otra forma. El punto en el que acabemos la fase de *traceback* en el algoritmo (i.e. el primer máximo del vector1 se pose sobre el primer máximo del vector2) menos el punto original del primer máximo será el desplazamiento buscado.

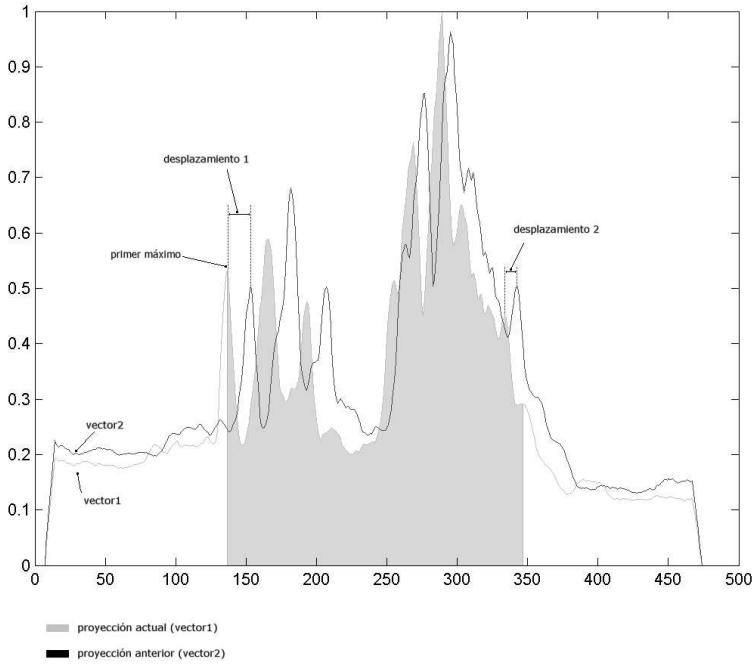

Figura 3.8: **Representación de las dos proyecciones en forma de histograma.** *Claramente existe un desplazamiento entre las dos formas, hemos de encontrar la manera de medirlo de una manera adaptativa.*

Llegados a éste punto, encontramos otro problema más dentro del problema inicial: hemos de detectar el primer máximo significativo, que supere cierto umbral. Una solución muy barroca sería seguir la función y detectar como máximo local todo aquel punto x que cumpla la condición $(f(x) > f(x-1))$ y $(f(x) > f(x+1))$. Ésta solución tan simple no es nada efectiva, ya que detecta cualquier pico como un máximo local. Tendremos que buscar una solución más sofisticada.

Ésta nueva solución pasa por implementar una máquina de estados que detecte cuando y cuanto se está subiendo, y cuando empiece a bajar señale como máximo todo aquel punto que se encuentre por encima de una altura desde el último mínimo. Para hacer el método de la manera general, primero normalizamos a [0..1] la forma,

y pasaremos el umbral como parámetro (e.g. 0.1 o 0.2).

Una vez hemos recortado el primer vector por el primer y último máximos locales, aplicaremos el algoritmo de programación dinámica explicado antes. Los resultados tampoco son satisfactorios.

Si tenemos en cuenta el desplazamiento del primer máximo, el resultado es 15, y si tenemos en cuenta el del máximo global, el resultado es 9, cuando manualmente hemos calculado que el desplazamiento real está sobre las 7 u 8 unidades.

La conclusión es que no tenemos manera de saber cual es el máximo local que hemos de tener en cuenta. Además, viendo los resultados, no está del todo claro que necesitemos adaptarnos también a transformaciones de escala, además de a las translaciones. Por tanto, hemos de decir que los resultados obtenidos con la programación dinámica no han sido positivos.

3.7. Localización de distancias

Una vez tenemos el horizonte fijado, usando cualquiera de los métodos anteriores, podemos calcular distancias. En nuestro proyecto, al no implementar detección de vehículos, no calcularemos distancias a puntos, pero sí podremos hacer el cálculo inverso, es decir, calcular en qué puntos de la imagen se encuentran tales distancias.

Partiendo del subapartado *ejemplo práctico* de la sección 2.2, invertiremos las ecuaciones para conseguir nuestra solución.

Contamos con un punto de fuga, ya que teníamos un segmento de longitud conocida en la dirección de nuestra carretera -lo habíamos marcado en la imagen junto a los parámetros iniciales-. Por tanto, podremos calcular la matriz H de homografía. Con ésta matriz calcularemos el factor de escalado igual que en el ejemplo antes comentado.

A continuación, partiendo de la ecuación

$$dist(punto_n) = distanciaBase + (punto - baseImagen) \times factorEscalado$$

derivamos

$$punto = \left(\frac{distancia(punto) - distanciaBase}{factorEscalado} \right) + baseImagen$$

Sabiendo la distancia[5] y los demás parámetros, calculamos cual es el punto (recordemos que será en coordenadas afines).

Ahora sólo nos queda pasar el punto a coordenadas de nuestra imagen, cosa que haremos invirtiendo la matriz de homografía H.

$$puntoImagen = H^{-1} \times puntoAfin$$

De ésta manera hemos demostrado que el cálculo de distancias habiendo localizado el horizonte es muy sencillo.

[5] En nuestra implementación hemos señalado donde se encuentran los 10, 25, 50, 75 y 100 metros.

3.8. Análisis de los resultados

Para concluir éste capítulo, hemos realizado unas pruebas finales utilizando la aproximación más fiable: correlación sobre los contornos de la imagen en el horizonte nominal. Las figuras a continuación muestran algunos fotogramas del vídeo de pruebas que hemos utilizado durante toda la explicación.

Figura 3.9: **Frames 100 (recta) y 218 (cambio de rasante)**

Figura 3.10: **Frames 250 (bajada) y 280 (cambio de rasante)**

Figura 3.11: **Frames 298 (subida) y 448 (subida)**

Hemos seleccionado fotogramas donde se pudiese estimar manualmente la línea del horizonte partiendo de dos líneas paralelas en nuestra dirección, e intentando intuir un comportamiento de nuestro programa.

He aquí las impresiones y conclusiones que hemos sacado de éste método, *proyección horizontal*, viendo los resultados globales:

- Utilizando correlación o correlación normalizada, sobre los contornos de la imagen, y aplicando los cálculos sólo a la ventana del horizonte nominal, obtenemos resultados muy satisfactorios, que realmente se ajustan a la detección manual del horizonte.
- Se comporta muy bien sobre zonas sin cambios de rasante, ya sean rectas, subidas o bajadas (ver las figuras).
- Las curvas no parecen afectar negativamente al método.
- Es robusto a señales aereas, otros vehículos y demás interferencias que podrían hacernos errar en los cálculos.
- En cambio, la estimación del horizonte en los cambios de rasante (frames 218 y 280) es muy mala, e incluso en ocasiones es muy difícil intentar localizar el horizonte manualmente.
- Además en cambios de iluminación o de paisaje repentinos, por ejemplo en la entrada y salida de túneles, el horizonte se pierde y arrastra el error producido en cada uno de éstos puntos.

Capítulo 4

Geometría epipolar

Al segundo método lo hemos llamado *geometría epipolar*. Su funcionamiento teórico parte de los conceptos que hemos explicado en la sección de *geometría proyectiva* dentro del capítulo 2.

4.1. Análisis

La geometría epipolar entre dos vistas es esencialmente la geometría de la intersección de los planos imagen con el conjunto de planos, que tienen la línea de base (i.e. línea que une los centros de las dos cámaras) como eje [18].

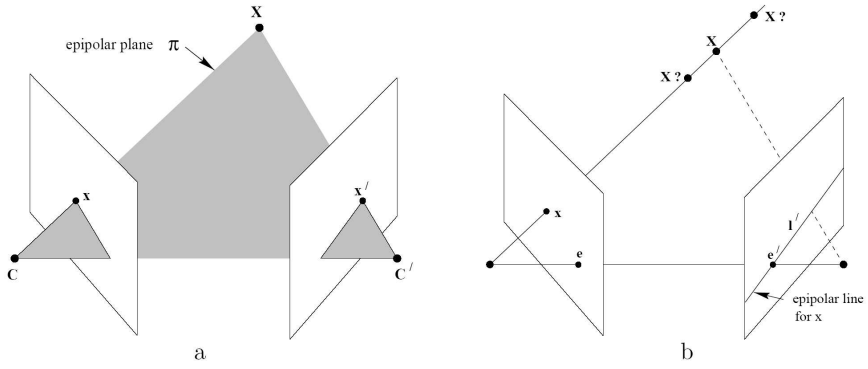

Figura 4.1: **Geometría epipolar.** ***a)*** *El punto X del mundo se proyecta en el punto x de la imagen de la izquierda (cuya cámara tiene centro C), y en el punto x' de la imagen de la derecha (con centro de cámara C'). El plano epipolar es π, y es el que une X, x y x'.* ***b)*** *Los puntos e y e' son los epipolos, y l' la línea epipolar, que señala donde puede encontrarse el punto x de la primera imagen, en el segundo plano imagen.*

En la figura 4.1 observamos los fundamentos de la geometría epipolar. Si tenemos un punto X en el mundo 3D, que se representa en la primera imagen como x, y en

la segunda como x', sabemos (por la teoría explicada en el capítulo 2) que hay una matriz de homografía H que cumple $xH = x'$. Ahora podremos introducir una serie de nuevos conceptos. La línea que une los centros de las cámaras C y C' es la línea de base (*baseline*), que corta los planos imagen por los puntos e y e', llamados epipolos. De la figura anterior, también extraemos una importante observación: **Un punto x de la primera imagen yace sobre la línea epipolar l' en la segunda imagen, por lo cual X en la segunda imagen también está sobre l'.**

Ésta correspondencia entre un punto x y su línea epipolar l' en la segunda imagen está representada por la **matriz fundamental** F. La propiedad principal de ésta matriz es:

$$x'^T F x = 0 \quad \forall\ x, x' \ \ tal\ que\ x \leftrightarrow x'$$

No entraremos ni en los pasos seguidos para hallar F ni en las características de ésta para no alargar la explicación previa del método, por lo que enumeraremos las propiedades que nos interesan a medida que las necesitemos en el capítulo.

Además, en nuestro problema ésta matriz tiene unas características especiales, ya que usaremos la geometría epipolar aplicada a un posicionamiento muy peculiar entre las dos cámaras, conocido como *pure translation*. La cámara, estacionaria, se mueve en una sola dirección $-t$, y todos los puntos de el mundo se mueven siguiendo líneas paralelas a t. La intersección de éstas líneas es el ya conocido *punto de fuga* (*vanishing point*) en la dirección de t.

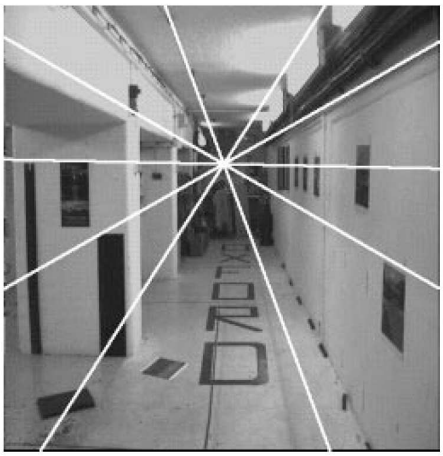

Figura 4.2: **Punto de fuga mediante las líneas epipolares.** *En la imagen podemos ver las líneas epipolares, resultado de un movimiento en $+Z$, es decir, moviendo la cámara hacia adelante. La intersección de todas ellas señala el epipolo -el centro de nuestro horizonte-.*

Nuestra solución por tanto tratará de encontrar primeramente los puntos que definen las líneas epipolares, para obtener la matriz fundamental F y finalmente extraer de ésta el punto de fuga que nos localizará el horizonte.

4.2. Implementación

Para la implementación de éste método hemos diseñado otra interfície gráfica (figura 4.3) que nos permita hacer pruebas rápidamente y observar los resultados de una manera cómoda.

Figura 4.3: **Interfaz de nuestro programa *(Epipolar Geometry)*** .

En la implementación interna del método está basada en el código y algoritmos desarrollados en Matlab por Peter Kovesi, de The University of Western Australia [16]. La implementación de Kovesi, básicamente, trata de solucionar un problema de estimación del desplazamiento de una imagen. Es decir, primero sacamos una fotografía de un escenario, después movemos la cámara una distancia -no rotarla ni trasladarla en el eje Z (girarla o alejarla)-, y aplicando el algoritmo, sabremos cuál ha sido el desplazamiento de cada punto de la imagen original.

Nuestro problema es algo diferente, ya que aunque tampoco rotamos sobre Z (o por lo menos suponemos que no lo hacemos), sí nos desplazamos en Z, por tanto la imagen sufre un escalado, lo cual provocará que el *matching* entre puntos no sea igual de robusto tal y como está implementado. Así mismo, la matriz F es utilizada de una manera diferente, por lo cual se han tenido que analizar y adaptar gran parte de las funciones. En las secciones siguientes explicaremos el funcionamiento de cada una de las partes del algoritmo.

4.3. Detector de Harris

Lo primero que tenemos que hacer es encontrar las líneas epipolares. Éstas líneas están definidas por dos puntos correspondientes, uno en cada imagen. El detector implementado para después calcular la correspondencia de éstos puntos característicos ha sido un detector de esquinas, más concretamente el *detector de Harris*.

Se trata de un detector publicado por C.Harris y M.Stephens en 1988, y está basado en reconocer esquinas a través de una ventana pequeña. A continuación movemos la ventana en cualquier dirección, si en hay una esquina el cambio de intensidad será grande. Por el contrario, si hay un borde o simplemente una textura plana, no habrá apenas cambio de intensidad.

Figura 4.4: Detección de esquinas en dos fotogramas.

4.4. Matching por correlación

Una vez tenemos las esquinas de los dos fotogramas, tenemos que calcular la correspondencia de cada una de ellas en las dos imágenes.

Primeramente se aplica un filtro en las dos imágenes, para que las zonas brillantes no desvíen los cálculos. Después, para cada esquina del primer fotograma buscaremos su correspondiente en la segunda imagen. Directamente desecharemos los puntos que se encuentren fuera de una cierta distancia permitida, es decir, si tenemos un punto x en la primera imagen, su punto x' correspondiente en la segunda imagen deberá de estar a una distancia d menor de *maxPointDisplacement* píxeles, 30 por ejemplo. Ésta restricción, está basada en la idea de que en nuestro problema los puntos (esquinas) no pueden desplazarse más allá de cierta distancia en nuestro intervalo de tiempo (directamente proporcional al parámetro *step*). Además, discriminaremos también las distancias menores de *minPointDisplacement*, 5 por ejemplo. En éste caso, si el punto se ha movido muy poco, definirá una línea epipolar demasiado corta, que podría introducirnos errores más tarde.

Para encontrar la mejor correlación usaremos dos ventanas, una para el punto de la primera imagen (w_1), y otra para el de la segunda imagen (w_2). A continuación extraeremos el coeficiente de correlación usando:

$$C = \frac{\sum(w_1 \times w_2)}{\sqrt{\sum(w_1 \times w_1) \cdot \sum(w_2 \times w_2)}}$$

El punto x' con mayor coeficiente C será el que corresponda al punto x de la primera imagen. Una de las diferencias pequeñas pero cruciales, entre el comportamiento de nuestro programa y la implementación de Kovesi, es que en su caso cuando obtiene la correspondencia entre dos puntos, éstos ya no pueden usarse más adelante. Ésto es así porque la seguridad de encontrar dos ventanas idénticas en las dos imágenes es muy grande, siempre y cuando el punto se encuentre en las dos imágenes, y no haya sido ocultado. En nuestro caso, las ventanas no son idénticas, ya que la imagen sufre escalado, por tanto en iteraciones sucesivas podríamos encontrar mejores correlaciones para nuestro punto.

La operación de búsqueda de correspondencias entre una imagen y la otra se hace dos veces: una para buscar correspondencias partiendo del primer fotograma, y otra partiendo del segundo fotograma. Finalmente, se escogen sólo las correspondencias consistentes en las dos direcciones, lo cual elimina puntos que están en una imagen, pero no en la otra.

Figura 4.5: **Correspondencia de puntos en dos fotogramas superpuestos.**

4.5. RANSAC

Una vez tenemos la correspondencia entre puntos, ¿podríamos decir que ya tenemos las líneas epipolares que buscábamos? La respuesta es que no. Lo que tenemos es una serie de líneas que son producto de más de un tipo movimiento en la imagen. Es

decir, además del movimiento de nuestro vehículo, que genera unas líneas epipolares con el epipolo en nuestro horizonte, también se habrán detectado muchas más líneas epipolares correspondientes al movimiento individual de cada uno de los vehículos de nuestro campo de visión, con su correspondiente epipolo.

Para llegar a nuestra solución, lo que nos interesa es conseguir un modelo que se adapte a las líneas epipolares de nuestro propio movimiento, pero que deseche las demás muestras. El algoritmo que nos encontramos ahora se llama *Random Sample Consensus*, más conocido como *RANSAC*.

El algoritmo RANSAC [8][17] es un algoritmo para la estimación robusta de modelos, partiendo de datos muy dispares y tolerante a los outliers (i.e. muestras que no pertenecen a nuestro modelo). Fue desarrollado en 1981 por M.Fischler y R.Bolles, y se aplica a un gran número de problemas de estimación de modelos en el campo de la visión artificial. El algoritmo es bastante simple, pero muy eficiente:

1. *Selección de las muestras.* Escogemos el mínimo número de muestras necesario para definir nuestro modelo. En nuestro caso necesitaremos 8 líneas (8 pares de puntos correspondientes) para definir nuestro modelo (i.e. la matriz fundamental F). La elección es aleatoria, con la misma probabilidad de selección para cada muestra.

2. *Evaluación de hipótesis.* A continuación se genera un modelo (hipótesis) de acuerdo a las muestras escogidas, y se mira el soporte a ese modelo por parte del resto de muestras. A las muestras que no cumplan el modelo las llamaremos *outliers*, mientras que a las que estén dentro del modelo las llamaremos *inliers*. Cuanto mayor número de *inliers* encontremos, más apoyo tendrá nuestro modelo y más posibilidades de tratarse del que nos interesa.

3. *Resultados.* Finalmente, el modelo que más apoyo tenga ganará, los inliers serán muestras que cumplan el modelo, y los *outliers* que no estén dentro del modelo serán desechados.

Nuestra implementación sigue éstos mismos pasos. Primero se **seleccionan** aleatoriamente 8 pares de puntos correspondientes, que serán las 8 muestras necesarias para definir nuestro modelo, una matriz 3x3 con ciertas restricciones (ver [18], sección 8.3.1 y 10.7.1).

Después **normalizaremos** éstos puntos: descartaremos puntos en el infinito, situaremos el centroide de la nube de puntos en el origen, y finalmente escalaremos el *dataset* para que la distancia media al centroide sea $\sqrt{2}$.

Una vez tenemos normalizado el conjunto de puntos, crearemos el **modelo** M para éste, es decir, construiremos una matriz 3x3 sabiendo que cada correspondencia es una restricción lineal.

A continuación buscaremos los inliers del modelo M, comprobando la distancia a la que están de éste modelo.

El resultado final será el mejor conjunto de *inliers*, que corresponderán a las líneas epipolares de nuestro movimiento, y una **matriz fundamental** $F = M$. De ésta matriz F extraeremos el **epipolo** e, que será nuestro punto de fuga, utilizando la propiedad:

$$Fe = 0$$

$$F^T e' = 0$$

4.6. Análisis de los resultados

Al igual que en el primer método, hemos ejecutado el vídeo de pruebas en nuestra aplicación. A continuación mostramos los fotogramas más significativos, de los cuales extraeremos algunas conclusiones:

Figura 4.6: **Frames 22 (desplazamiento horizontal) y frame 41 (recta)**

Figura 4.7: **Frames 70 (badén) y 73 (subida post-badén)**

Figura 4.8: **Frames 85 (recta) y 198 (recta)**

Examinando el vídeo que nos ha resultado la ejecución rápidamente nos damos cuenta de que al método le falta mucho para llegar a ser de utilidad real en el mundo de los vehículos inteligentes. Primeramente sorprende que el epipolo se desplace tanto. Ésto es así porque en cada frame se detectan esquinas, independientemente de los anteriores, y además el algoritmo RANSAC es no determinista, por lo cual no tenemos

porque obtener el mismo resultado en dos ejecuciones con los mismos parámetros. Ésto nos hace entender porqué el epipolo actual no necesariamente ha de estar cercano al del último cálculo.

En la figura 4.6 vemos como afecta un desplazamiento horizontal del vehículo: el epipolo se desplaza hacia los lados. Si además tenemos en cuenta que al girar el volante, en éste caso a la izquierda, el vehículo sufre una pequeña rotación en Z, apoyándose más sobre las ruedas de la parte izquierda, entenderemos el resultado. En el frame 41 tenemos una recta, y un foco de expansión más o menos correcto.

En la figura 4.7 tenemos el problema del badén ya visto anteriormente. En ésta ocasión el comportamiento del horizonte será desplazarse hacia abajo al entrar en el desnivel, y subir al salir del hueco en la carretera.

En la última figura, 4.8, vemos la influencia de otros vehículos y del paisaje.

El mayor problema que se ve a simple vista es el cálculo erróneo de correspondencias, que nos hará errar en la localización del epipolo. Vemos como las matrículas y focos, los *quita-miedos*, los pequeños contornos del paisaje e incluso las líneas de carril más alejadas influyen muy negativamente en el resultado, al haberse hecho una mala correspondencia de puntos.

Las conclusiones son las siguientes:

- La correspondencia errónea entre puntos es muy grande, lo que deriva en un modelo erróneo, que finalmente nos devuelve un epipolo muy variable e inútil.
- Los desniveles en la carretera desplazan el epipolo según su dirección, que es lo contrario que buscábamos.
- Los desplazamientos horizontales del vehículo no producen errores directos, pero sí la pequeña rotación de cámara que se deriva de ellos.
- La localización del epipolo en tramos rectos con o sin señales, incluso con puentes y túneles, paisajes y otro tipo de artefactos (e.g. sombras, manchas, árboles, etc.) es correcta siempre y cuando las correspondencias entre puntos se hayan calculado bien.
- El uso de una ventana (horizonte nominal) hace que no tengamos en cuenta líneas de carril muy válidas para éste método, pero por otra parte elimina zonas que producen puntos de correspondencia confusos, como los paisajes.
- La existencia de *inliers* que pertenecen a distintos tipos de movimiento en la escena (e.g. otros vehículos) es una de las mayores causas de errores del método.

Capítulo 5

Foco de expansión

El tercer y último método implementado en el presente proyecto hace uso del foco de expansión, en corto *FOE*, del inglés *Focus of Expansion*. Parte del método de geometría epipolar, de hecho basa prácticamente todo el código en la implementación anterior, pero intenta mejorar el comportamiento a la hora de encontrar el punto de fuga.

Como hemos visto en el anterior método, el conseguir el punto de fuga mediante la matriz fundamental F puede arrojarnos resultados poco correctos, ya que podemos tener correspondencias de puntos erróneas que desvíen el epipolo.

¿Qué podemos retocar para mejorar la aproximación del punto de fuga? Partiendo de la idea inicial de detectar líneas epipolares producto de nuestro movimiento, lo que nos interesa realmente es saber donde se encuentra el foco de expansión, es decir, el punto en la imagen del cual va surgiendo todo a medida que avanzamos en el tiempo. Si nos fijamos en la imagen de la carretera que obtenemos cuando avanzamos hacia adelante, veremos como existe un punto del cual todo parece expandirse, en forma de línea recta hacia el exterior. La idea es la misma que en la geometría epipolar de la figura 4.2.

Nuestro objetivo será pues intentar localizar la zona donde "'intersecte"' el mayor número de líneas de correspondencia. Hemos entrecomillado la palabra intersecte porque realmente no intersectarán en un sólo punto, sino que las líneas[1] intersectan dentro de una circunferencia de radio r.

[1]Ya no las llamaremos líneas epipolares, porque ahora sabemos no existe un único epipolo en la imagen.

5.1. Filtrado de inliers y localización del FOE

Al igual que en el método anterior, hemos utilizado un RANSAC, pero en éste caso no para estimar una matriz fundamental, sino para localizar el foco de expansión.

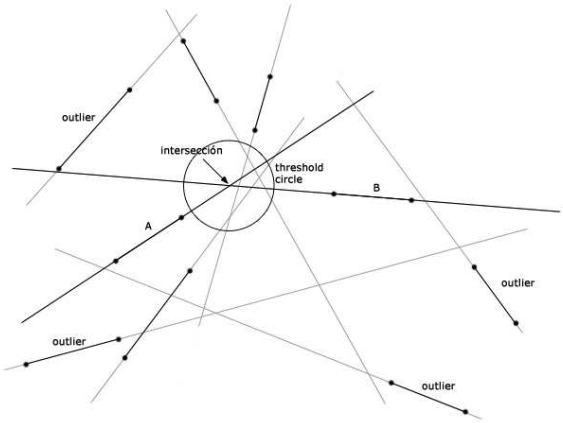

Figura 5.1: **Esquema de nuestro filtrado.** *Sólo las líneas que pasen por nuestra circunferencia umbral, obtenida de la intersección de A y B, serán inliers.*

El comportamiento del RANSAC que hemos implementado está ilustrado en la figura 5.1. Escogemos aleatoriamente dos líneas A y B, y calculamos su producto vectorial para obtener la intersección I. Alrededor de éste punto trazamos una circunferencia *threshold*, que será la que eliminará outliers. Entonces, para cada una de las líneas L de correspondencia, calcularemos su distancia a la intersección I usando

$$distancia = \frac{|L_x \cdot I_x + L_y \cdot I_y + L_z|}{\sqrt{L_x^2 + L_y^2}}$$

Si la distancia es menor a cierto umbral (que por defecto es de 15 píxeles en nuestro programa), la línea es clasificada como *inlier*, y por tanto la tendremos en cuenta para el cálculo del FOE.

Éste cálculo se repite un número de veces, por defecto 1000, y al final escogemos el mayor conjunto de *inliers* que coinciden dentro de un umbral.

Una vez tenemos el conjunto de líneas *inlier*, tenemos que calcular cual es el centroide de todas ellas. Si miramos de nuevo la figura 5.1 veremos que no intersectan en un sólo punto, por lo que nuestro punto FOE vendrá dado por la expresión:

$$\underbrace{\begin{pmatrix} L_{x1} & L_{y1} \\ L_{x2} & L_{y2} \\ \vdots & \vdots \\ L_{xn} & L_{yn} \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} FOE_x \\ FOE_y \end{pmatrix}}_{x} = \underbrace{\begin{pmatrix} L_{z1} \\ L_{z2} \\ \vdots \\ L_{zn} \end{pmatrix}}_{b}$$

Éste es un sistema de ecuaciones sobre-determinado, es decir, no existe una solución única. Lo que podemos hacer, es encontrar un vector x que esté cercano a la solución del sistema $Ax = b$, o sea un vector x tal que la norma $\|Ax - b\|$ sea mínima.

Existen dos modos de encontrar ese vector x: *Singular Value Decomposition (SVD)* y *Pseudo-Inversa*. Ambos métodos están explicados en el Apéndice 3 de [18].

A grosso modo, en SVD la matriz A se descompone como $A = SVD^T$, donde U y V son matrices ortogonales, y D una matriz diagonal de elementos no-negativos (no explicaremos como calcular ésta descomposición porque nos extenderíamos demasiado, y no atañe al presente proyecto). Después calculamos $b' = UTb$, y encontramos el vector y tal que $y_i = b'_i/d_i$, donde d_i es el iésimo elemento de la diagonal de la matriz D. Finalmente, nuestra solución es $x = Vy$.

En la pseudo-inversa, que es la que hemos implementado, el cálculo es más directo. La pseudo-inversa de A, una matriz $m \times n$ de rango n, es $A^+ = (A^T A)^{-1} AT$. Se trata de resolver las ecuaciones normales $A^T Ax = A^T b$, y si $A^T A$ es invertible, nuestra solución será $x = (A^T A)^{-1} A^T b$.

Como sabemos, x es nuestro FOE, por tanto ya hemos solucionado, de manera teórica, el problema.

Figura 5.2: **Filtrado de inliers en la práctica.** *La imagen de la izquierda representa el fotograma justo después de aplicar la correlación a los contornos detectados, es decir, antes de aplicar nuestro filtrado -el conjunto de todas las correspondencias posibles se denomina putative matches-. La imagen de la derecha representa el filtrado usando RANSAC y la pseudo-inversa.*

5.2. Análisis de los resultados

A continuación se muestran unas capturas del resultado de nuestro programa utilizando los parámetros por defecto, sobre un horizonte nominal de 300 píxeles de alto.

Figura 5.3: **Frames 29 (desplazamiento horizontal) y frame 61 (recta)**

Figura 5.4: **Frames 70 (badén) y 74 (subida post-badén)**

Figura 5.5: **Frames 113 (recta) y 189 (recta)**

Observando el resultado completo (\results \focusOfExpansion\focusOf Expansion-nominalHorizon300\) encontramos que, si bien de manera global se calculan correctamente muchos horizontes, tenemos el mismo problema de "'ruido"' que en la geometría epipolar. La correspondencia errónea de nuevo hace que el FOE varíe demasiado, y el horizonte calculado no sea fiable. Aún así, si aumentamos de

tamaño la ventana de correlación, los resultados mejoran sensiblemente. La resolución del problema pasa por escoger cada uno de los parámetros de forma precisa, sólo así conseguiremos los resultados esperados. En nuestro programa hemos inicializado por defecto los parámetros que mejor se ajustan a los vídeos que hemos utilizado, pero una sintonización de éstos para cada caso concreto nos aportarán mejores resultados finales.
Las conclusiones principales[2] son:

- El filtrar *inliers* nos elimina posibles líneas que pertenecen a otros tipos de movimientos que no nos interesa, de ahí que en secuencias de tramos rectos los resultados sean aceptables.
- Los problemas derivados de desniveles en el pavimento, así como la correspondencia errónea, afectan a nuestro método igual que afectaban a la geometría epipolar.

[2]Para extraer las conclusiones de éste método hemos analizado multitud de secuencias de vídeo, muchas de las cuales han sido incluidas en el CD-ROM anexo.

Capítulo 6

Resultados globales y aplicación práctica

Una vez estudiados los tres métodos, y vistas las soluciones y problemas que nos aporta e introduce cada uno, debemos realizar un análisis global de los resultados obtenidos. Pensemos por un momento que ésta solución se implementa directamente sobre un prototipo de vehículo provisto de cámara frontal, situada delante del retrovisor interior. Teniendo en cuenta siempre que nuestra implementación no funciona en tiempo real -no era el objetivo-, nos debemos hacer preguntas como ¿qué hemos aportado al problema? ¿qué partes del proyecto son directamente utilitzables y cuáles distan aún de llegar a ser métodos eficaces?

La proyección horizontal parece bastante robusta si introducimos un horizonte inicial correcto. El método de correlación, o correlación normalizada, aplicado a una ventana del horizonte nominal, y siempre usando el filtro de contornos, resulta en una estimación casi igual al cálculo manual del horizonte. Podríamos utilizar el método de geometría epipolar, o mejor aún, el foco de expansión, para calcular sin intervención humana un horizonte inicial no constante. Éstos dos métodos también nos servirían para reforzar las predicciones de la proyección horizontal, ya que como hemos visto, el *FOE* y el *epipolo* se desplazan justo en la dirección contraria del horizonte.

El mismo comportamiento lo podríamos usar para detectar cambios de rasante. Si detectamos uno usando FOE, desconectamos la proyección horizontal y mantenemos estático el último horizonte válido. Al salir del cambio de rasante volvemos a calcular el horizonte inicial mediante FOE o geometría epipolar, y reanudamos el cálculo por proyección.

Ésta idea también podría ser aplicada por ejemplo a túneles, donde el cambio de iluminación desvía los cálculos y aumenta el error.

Utilizando la geometría epipolar, podríamos ampliar su funcionamiento calculando distintos tipos de movimiento, al igual que hacemos con el filtro de inliers en *FOE*.

Ésto nos serviría para no tener en cuenta ciertas zonas en el método de proyección horizontal, nos aportaría heurística en la detección de vehículos, etc.

Las curvas normalmente no representan ningún problema en la proyección horizontal, pero también podrían ser detectadas por la geometría epipolar.

Figura 6.1: **Representación dual de proyección horizontal y FOE.**

Capítulo 7

Conclusiones

Para concluir, enumeraremos los objetivos alcanzados, las líneas de investigación que pueden suceder a éste proyecto, algunos problemas que hemos tenido a lo largo de la implementación, y las impresiones que hemos extraído de estos meses de trabajo:

- Primero, comentar que hemos completado todos los objetivos, y además se ha implementado una interfície gráfica para mostrar los resultados.
- El funcionamiento del método *proyección horizontal* es muy satisfactorio, e incluso se podría decir que es tiempo real. Como mejoras y posibles ampliaciones, podría incluirse un filtro más sofisticado (e.g. detector de esquinas, líneas...) e intentar buscar un tipo de solución basada en la programación dinámica que nosotros hemos analizado e implementado, y de la cual no hemos obtenido buenos resultados.
- Respecto a la *geometría epipolar* y *foco de expansión*, decir que los cálculos parecen comportarse de manera correcta, pero necesita bastante refinamiento en las fases de detección de características y correspondencia de puntos. Quizás desarrollando un detector de crestas, o de líneas diagonales (para aprovechar más la información que nos aportan las líneas de carril), el número de puntos característicos disminuiría, pero a su vez sería más acotada y eficiente la búsqueda de correspondencias. Además, el mejorar el cálculo de éstas correspondencias es vital para estabilizar los resultados que ahora mismo obtenemos, por lo que sería interesante buscar buenos algoritmos.
- En cuanto a los problemas que hemos encontrado en el proyecto, primeramente destacar la incertidumbre que representaba un proyecto tan difuso y con tantos conceptos matemáticos. Una vez superados esos pasos, y tomando la geometría como un camino más para alcanzar nuestros objetivos, y no como un obstáculo -que es como en ocasiones vemos a las matemáticas-, llegó el momento de la implementación. La idea de utilizar el entorno Matlab a la hora de codificar el

proyecto, quizás debido al regusto de haber sido ese lenguaje el empleado en las prácticas más complejas de la carrera, se presentaba más como un reto que como una ayuda. Pero una vez se avanza en el conocimiento del lenguaje y se descubren las posibilidades y su verdadera potencia, e incluso se han programado unas cuantas interfícies, se llega a la conclusión de que la elección de éste fue acertada.

- Finalmente, y ya hablando sobre el amplio campo de la visión artificial, especialmente aplicada a los vehículos inteligentes, la impresión que tenemos es que por cada problema que intentemos resolver, aparecerán unos cuantos más. Pero es éste camino el que nos permitirá refinar y solucionar cada uno de los objetivos que se plantearon allá por los años ochenta, cuyo principal lema fue conseguir una conducción más segura y fiable, a lo largo del nuevo siglo.

Bibliografía

[1] *Vision-based intelligent vehicles: State of the art and perspectives.*, volume 32, 2000.

[2] A.Fascoli A.Broggi, M.Bertozzi and M.Sechi. Shape-based pedestrian detection. *IEEE Intelligent Vehicles Symposium IV*, pages 215–220, 2000.

[3] T.Kanade C.Thorpe, M.Hebert and S.Shafer. Vision and navigation for the carnegie-mellon navlab. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(3):362–373, 1988.

[4] D.Pomerleau and T.Jochem. Rapidly adapting machine vision for automated vehicle steering. *IEEE Expert: Special Issue on Intelligent System and their Applications*, 11(2):19–27, 1996.

[5] E.C.Rouchka. *Advanced Dynamic Programming Tutorial*, 2001.

[6] O.Mano G.P.Stein and A.Shashua. Vision-based acc with a single camera: Bounds on range and range rate accuracy. *IEEE Intelligent Vehicles Symposium 2003, (Columbus, OH).*, 2003.

[7] G.Welch and G.Bishop. *An Introduction to the Kalman Filter*, 1995.

[8] H.Cantzler. *Random Sample Concensus (RANSAC).*

[9] J.Langheim. Carsense - new environment sensing for advanced driver assistance systems. *IEEE Int. Conf. On Intelligent Transportation Systems*, 2001.

[10] J.Vitrià. *Visió per computador.* Materials 14. Universitat Autònoma de Barcelona., 1995.

[11] M.Bertozzi et al. L.Andreone, P.C.Antonello. Vehicle detection and localization in infra-red images. *IEEE International Conference on Intelligent Transportation Systems*, pages 141–146, 2002.

[12] L.Petersson L.Fletcher, N.Apostoloff and A.Zelinsky. Vision in and out of vehicles. *IEEE Intelligent Systems*, 2003.

[13] G.E.Karras L.Grammatikopoulos and E.Petsa. Geometric information from single uncalibrated images of roads. *International Archives of Photogrammetry and Remote Sensing*, 34(5):21–26, 2002.

[14] A.Fascoli M.Bertozzi, A.Broggi and A.Tibaldi. An evolutionary approach to lane markings detection in road environments. *Atti del 6 Convegno dell'Associazione Italiana per l'Intelligenza Artificiale*, pages 627–636, 2002.

[15] A.Fascoli et al. M.Bertozzi, A.Broggi. I-r pedestrian detection for advanced driver assistance systems. *25th Pattern Recognition Symposium*, 2003.

[16] P.Kovesi. Matlab functions for computer vision and image analysis.

[17] R.B.Fisher. *The RANSAC (Random Sample Concensus) Algorithm.*

[18] R.Hartley and A.Zisserman. *Multiple view geometry in computer vision.* Cambridge University Press, 2000.

.....
Signat: *David Gerónimo Gómez*
Bellaterra, Juny del 2004