Detección de Líneas y Sistema de Estabilidad de Carril basado en cámara frontal

Álvaro Medina Ballester
Ingeniería Informatica Superior
Universidad de las Islas Baleares
Visión por computador
Email: alvaro@comiendolimones.com

Xavier Leal Meseguer
Ingeniería Informática Superior
Universidad de las Islas Baleares
Visión por computador
Email: lealxavi@gmail.com

***Resumen*—Un gran porcentaje de los accidentes de tráfico son debidos a factores humanos. De estos, la salida de carril es uno de los más comunes. En el presente artículo se propone un sistema de detección de salidas de carril basado en visión por computador. Se obtienen imágenes mediante una camara de vídeo situada en el coche y posteriormente se testea el algoritmo de detección fuera de la ejecución en tiempo real. El algoritmo consiste en el uso del filtro de *Canny* seguido de una transformada de *Hough* de forma repetida. Con dos pasadas de estas dos funciones conseguimos resaltar más las características de la carretera y posteriormente obtenemos las dos líneas del carril. En ausencia de las dos líneas en las zonas laterales del campo de visión de la cámara, se da un aviso de salida de carril. Este trabajo académico se ha elaborado para la asignatura de Visión por Computador, impartida por el doctor Francisco** J. PERALES**.**

I. INTRODUCCIÓN

En este apartado vamos a explicar en qué consiste esta aplicación así como las tecnologías que han sido utilizadas.

I-A. Problema a resolver

En el presente proyecto se va a estudiar la detección de las lineas (continuas y discontinuas) de una calzada o carretera. Se supone la recepción de una secuencia de imágenes, captadas por una cámara que está montada, configurada y calibrada en un coche común. La figura 4 es una muestra de una de las imágenes que se tratan.

Una vez captada la imagen, se pasa a buscar tanto la linea que queda en el lado izquierdo como la linea que queda al lado derecho del coche (ya sean continuas o discontinuas). Esta detección es la base para discriminar aquellos casos en que los que el coche esté realizando un cambio de carril o bien esté saliendo de la calzada.

I-B. Tecnología Utilizada

Se ha utilizado el lenguaje de programación `C++` en una plataforma "Ubuntu GNU/Linux 10.04". Por otro lado se han utilizado las librerías gráficas Open CV 2.1, con las cuales se ha llevado a cabo la implementación de todos los filtros y técnicas que se explicarán a continuación. Por último se ha utilizado un video de entrada para las diferentes pruebas, con compresión *DV-PAL* progresivo y con una resolución de 720 por 576 pixeles.

II. IMPLEMENTACIÓN

A continuación se explican cuales han sido los pasos que se han seguido para la detección de la lineas así como para la detección de los cambios o salidas de carril. Presentamos el *pseudo-código* mostrado en el algoritmo 1 para explicar la secuencia de acciones que se realizan. Como se puede comprobar, primero se realiza el cálculo de perspectiva para obtener líneas de carriles prácticamente verticales; tras ello se aplican dos veces los filtros de *Canny* y la transformada de *Hough* —buscando únicamente las líneas que nos son útiles—, y al final se comprueba si estamos fuera del carril.

Listing 1. Algoritmo de detección de carriles

```
mientras podamos capturar {

    realizarPerspectiva();
    pasamosAEsacalaDeGrises();
    aplicamosFiltroCanny();
    buscamosHoughLines() {
        discriminamosLineas();
    }
    calculamosMedia();
    aplicamosFiltroCanny() // segunda vez
    buscamosHoughLines() {
        discriminamosLineas();
    }

    calculamosMedia();
    deshacemosPerspectiva();
    solapamosImagen();
    comprobamosSalidaCarril();
}
```

II-A. Perspectiva / Calibración

Una vez el dispositivo de captura de imágenes ha sido montado en el coche, se lleva a cabo su calibración. Existen muchas maneras diferentes de calibrar la cámara; se puede saber la posición de la cámara con parámetros como su altura, inclinación, etc. o se puede llevar a cabo una calibración manual (como se ha realizado en este caso), aunque una de las mejores formas de calibrar la cámara es la que se expone a continuación, comentada en las sesiones teóricas de la asignatura de Visión por Computador.

Figura 1. Calibración cámara

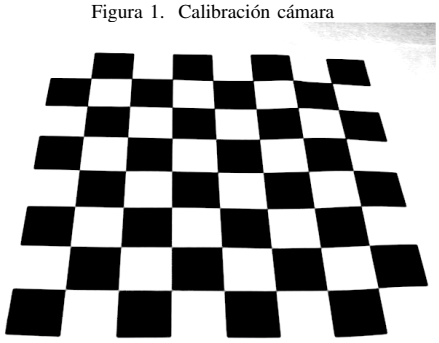

Figura 2. Perspectivas Probadas

El método se muestra en forma de ejemplo en la figura 1 y consiste en la colocación de un tablero de ajedrez en el suelo, dentro del campo de visión de la cámara. Una vez capturada la imagen del tablero, y suponiendo que la cámara no cambia de posición, gracias a la función cv::calibrateCamera(); [5] se logra obtener un tablero donde todas las casillas formen un cuadrado perfecto y la imagen no aparezca deformada. Los parámetros que se obtienen se corresponden a la deformación que se produce en la perspectiva de la cámara para que el tablero pueda verse correctamente.

El sistema que se ha utilizado es mucho más artesano pero ha funcionado en términos aceptables. En la figura 2 se muestran puntos de diferentes colores. Estos son los puntos que definen una área en forma de trapezoide, donde cada punto de él se traslada a la esquina más próxima de la imagen, transformándolo en un cuadrado que ocupa toda el área de la matriz sobre la cual trabaja el algoritmo. Con esta transformación manual se consigue un resultado similar a la calibración automática comentada anteriormente, aunque para conseguir un resultado óptimo se han probado diferentes trapezoides, representados cada uno de ellos con puntos de un mismo color.

De esta forma se consiguen dos de los objetivos que se necesitan:

- El horizonte queda acotado, y la imagen a tratar es inferior (se descartan todas aquellas partes de la imagen que no forman parte de la calzada).
- Se consigue que todas aquellas lineas que aparecen distorsionadas por la imagen perspectiva que coge la cámara aparezcan como líneas “casi rectas” y por lo tanto funcionen mejor los diferentes algoritmos.

Podemos ver a continuación un fragmento del código donde se especifica una de las matrices de perspectiva que han sido utilizadas en este proyecto:

```
org[0] = Point2f(126, 158);
dst[0] = Point2f(0, 0);

org[1] = Point2f(521, 158);
dst[1] = Point2f(720, 0);

org[2] = Point2f(2, 317);
dst[2] = Point2f(0, 576);

org[3] = Point2f(718, 313);
dst[3] = Point2f(720, 576);
```

Cabe destacar que el algoritmo presentado puede funcionar de forma correcta en otros ejemplos, aunque para que funcione de forma óptima deberíamos volver a ajustar las variables de transformación de perspectiva. Para ello, lo más recomendable es realizar la calibración de la imagen y de la lente, disponiendo de una cámara fijada en el coche.

II-B. Canny Filter

El filtro de *Canny* es la segunda de las técnicas que utilizamos para conseguir la detección de los carriles. Como puede verse en la figura 4 se utiliza el cambio de contraste que existe entre el fondo (carril) y las líneas del carril para conseguir detectar las fronteras entre ambos. Podemos observar también en la imagen, y en el pseudocódigo 2, que antes de aplicar este filtro la imagen ha sido transformada a escala de grises, por lo que la información que disponemos en la matriz de la imagen y sobre la que vamos a estudiar a fondo las líneas, tenemos un menor espectro de colores con los que trabajar. El filtro de Canny requiere esta conversión.

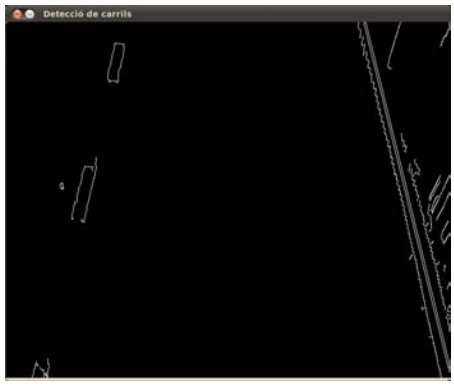

Figura 3. Filtro de Canny

Las instrucciones que se muestran a continuación son las que se han utilizado para llevar a cabo la aplicación de los

dos filtros (transformada a escala de grises y *Canny*). Los parámetros 3 y 4 de la función de *Canny* indican el umbral (*threshold*) inferior y superior, respectivamente, de este filtro y nivel de *hysteresis* con el que se trabaja [6].

Listing 2. Filtro de Canny

```
cvtColor(bordes, filtros, CV_BGR2GRAY);
Canny(filtros, filtros, 30, 120);
// Sobel(aux, aux, IPL_DEPTH_16S, 1, 0, 3);
```

Por último podemos ver como en el código 2 se encuentra "comentada" la instrucción `Sobel`. Esta instrucción pertenece a otro filtro de "detección de fronteras", que produce un efecto parecido al filtro que realmente aplicamos. En cambio esta función ha dado dos problemas:

1. Disminución de la velocidad de tratamiento de cada imagen
2. Aumento de la sensibilidad en la obtención de fronteras

II-C. *Houghline Doble*

La siguiente técnica o algoritmo es uno de los más importantes en este proyecto. La detección de líneas mediante la transformada de "Hough" es muy conocida por su eficacia y por la facilidad de representación parametrizada de las líneas que encuentra. Cada linea que encuentra es representada por dos valores, *theta* Θ y *rho* ρ, que representan la distancia de la perpendicular a la recta al punto de origen y el ángulo de rotación de la perpendicular, respectivamente.

Figura 4. Doble aplicación de la transformada de Hough

De esta forma se consigue una gran velocidad y rendimiento en la detección de líneas. En el caso que se trata no ha aparecido ningún problema en la detección de las líneas continuas tanto izquierda como derecha, pero no obstante, sí han habido algunos problemas para mantener una linea de referencia con la linea discontinua. Por esa razón se ha decidido aplicar dos veces el algoritmo de "Hough" [3].

Gracias a ello, se ha conseguido que la segunda vez que se aplica el algoritmo, ya aparezcan las primeras líneas y se tenga una mayor referencia para detectarlas, aunque debido a que la transformación de la perspectiva comentada en la sección II-A no es la óptima, hay veces que se pierde la referencia de la línea discontinua.

Listing 3. Criterio de discriminación de líneas encontradas

```
FLOAT theta = linies[i][1];
FLOAT rho = linies[i][0];
DOUBLE a = cos(theta), b = sin(theta);
DOUBLE x0 = a*rho, y0 = b*rho;

/* Discriminacion de lineas */

FLOAT graus = (theta*180)/CV_PI;
IF ((graus >= 0.00) && (graus <= 20.00)) {
  IF (x0 >= 50 && x0 <= 150) {
    rho_esq += linies[i][0];
    theta_esq += linies[i][1];
    idx_esq += 1;
  }
} ELSE IF ((graus <= 180.00) &&
           (graus >= 160.00)) {
  IF (x0 >= 500 && x0<=650) {
    rho_drt += linies[i][0];
    theta_drt += linies[i][1];
    idx_drt += 1;
  }
}
```

II-C1. Discriminación: En el fragmento de código 3 se puede observar como, una vez la linea ha sido encontrada, se lleva a cabo una discriminación. El ángulo de la linea ayudará a determinar si la linea es la del lado izquierdo o la del lado derecho, así como también ayudará a que no se tengan en cuenta aquellas líneas horizontales que puedan alterar el resultado del algoritmo.

Por otro lado, también se ha tenido en cuenta la posición que ocupan las líneas. Aquellas líneas que por su ángulo son del lado izquierdo, deben además estar posicionadas en este lado en la imagen. Lo mismo pasa con las líneas del lado derecho. Esto hace que muchas línas encontradas que no nos sirven sean descartadas y no afecten al siguiente paso, la media de líneas. Las líneas que se descartan son, por ejemplo, las líneas horizontales que aparecen en la imagen y que no representan ninguna característica que ayude a conocer si se ha producido el cambio de carril.

II-C2. Media Líneas: También se puede observar en el código como se lleva a cabo una media de todas las líneas resultantes. En este caso, y además aplicando dos veces el algoritmo de "Hough", el numero de líneas encontrado es muy grande. Por esa razón se lleva a cabo una media de todos los valores de *tetha* y *rho* encontrados para formar una nueva linea (más gruesa) con dichos valores.

III. Mejoras Posibles

Una de las mejoras que se han añadido al algoritmo, además de aplicar dos veces el algoritmo de "Hough" es la utilización de lo que se ha nombrado como *puntos de restauración*. Cada linea media, y final, de cada fotograma, es guardada, y si en el siguiente fotograma no se encuentra, se utiliza la anterior. Con esta mejora, se consigue que las líneas discontinuas no desaparezcan y aparezcan a gran velocidad y puedan seguir siendo visibles.

IV. Detección de cambio de carril

Una vez se han realizado dos pasadas de la transformada de "Hough", se dispone dos contadores de numero de líneas encontradas, uno para las líneas situadas a la izquierda y otro para las líneas situadas a la derecha. La condición que se ha establecido para decidir si se produce el cambio de carril es que las dos pasadas del algoritmo no hayan encontrado ninguna línea. Esto funciona en la mayoría de los casos, aún siendo una solución muy simple, lo que permite mantener un coste computacional bajo. En caso de tratar con problemas más complejos como trazados irregulares o con peores condiciones de visibilidad, es muy fácil incluir más condiciones a la detección del cambio de carril. En la figura 5 podemos ver el funcionamiento de la solución propuesta.

Figura 5. Detección de cambio de carril

V. Conclusión

Se ha conseguido desarrollar un algoritmo para la detección de cambio de carril sencillo y efectivo en un entorno controlado, con una calzada bien marcada y con buena visibilidad. El hecho de que sea una solución que incorpore una condición de detección de cambio de carril sencilla hace que funcione con un coste computacional bajo, lo que da pie a que pueda ser considerado para su uso en un entorno de ejecución en tiempo real. El aspecto negativo de ser una solución sencilla hace que fuera de un entorno controlado el sistema no funcione correctamente. Una mejor calibración de la cámara, un algoritmo de detección de cambio de carril más complejo y el uso de técnicas más avanzadas de detección de características —variación de los parámetros de la transformada de "Hough" según el número de líneas detectadas, por ejemplo— son aspectos que pueden mejorar el sistema.

Referencias

[1] Fengyuan Wang, Zonghe Guo, Daolin Zhang, *Detection of Road Guiding Lines with Computer Image Processing* Automobile School Shandong Institute of Technology Zibo City, Shandong 255012, CHINA

[2] Barna Saha *Bidirectional Fuzzy-Regression Model for RoadlinesDetection* Indian Institute of Technology Kanpur

[3] ZhanweiWu, BinKong, FeiZheng'andJunGao *Detection and Extraction of Discontinuous Lines*

[4] *Knowledge-based Power Line Detection for UAV* Li Yuee Liu ,Ross Hayward, Jinglan Zhang, Jinhai Cai Surveillance and Inspection Systems Zhengrong

[5] *Camera calibration and 3d reconstruction* OpenCV Reference manual. http://opencv.willowgarage.com/documentation/cpp/camera_calibration_and_3d_reconstruction.html?highlight=calibratecamera#calibrateCamera

[6] *Feature detection: canny filter* OpenCV Reference manual. http://opencv.willowgarage.com/documentation/cpp/feature_detection.html?highlight=canny#Canny