Detección de Lineas 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*—En este documento, se pueden encontrar las diferentes explicaciones de cómo hemos formalizado nuestro proyecto para el final de la asignatura *Visión por Computador* [1] de la Universidad de las Islas Balears. La explicación de la tecnología utilizada, problemas que esta nos ha dado, sistemas de calibración y tecnología hardware que se ha utilizado así como nuevas técnicas que hemos utilizado para intentar mejorar la detección de los carriles con un código que sea por encima de todo rápido.**

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 nuestro proyecto vamos 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 esta establecida, configurada y calibrada en un coche común. La figura 4 es una muestra de una de las imágenes que deberíamos tratar.

Entonces encontraremos tanto la linea que queda tanto en el lado izquierdo como la linea que queda al lado derecho del coche, ya sean continuas o discontinuas). Esto nos ayudará a discriminar aquellos casos en que el coche este realizando un cambio de carril bien se esté saliendo de la calzada.

I-B. Tecnología Utilizada

Hemos utilizado el lenguaje de programación `C++` en una plataforma "Ubuntu GNU/Linux 10.04". Por otro lado hemos utilizado las librerías gráficas Open CV 2.1, con las cuales hemos 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, de un video con compresión *DV-PAL* progresivo y con una resolución de 720 por 576 píxeles.

II. Implementación

Vamos a explicar cuales han sido los pasos que hemos seguido para la detección de la lineas así como para la detección de los cambios o salidas de carril. A continuación presentamos un *pseudo-código* para explicar la secuencia de acciones que se realizan así como todas las técnicas que utilizamos con el fin de explicarlas mejor a continuación.

```
mientras podamos capturar {

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

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

II-A. Perspectiva / Calibración

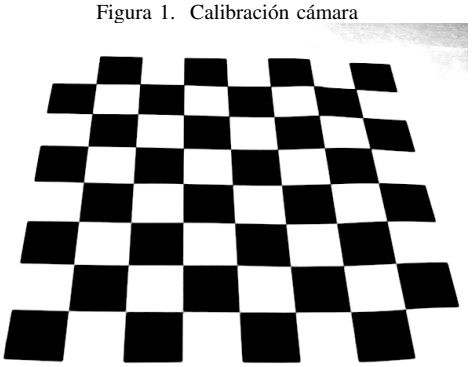
Una vez una dispositivo de captura de imágenes ha sido montado en el coche debemos llevar a cabo su calibración. Existen muchas maneras diferentes de llevar a cabo la calibración, podemos saber la posición de la cámara con parámetros como su altura, inclinación, etc ... Se puede llevar a cabo una calibración manual (como hemos hecho nosotros en este caso), pero estamos seguros de que la mejor forma de calibrar una imagen es mediante el sistema que se propone en casi todos los artículos que hablan sobre detección de lineas y carriles.

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 (en este caso el carril) donde es tomada la imagen. Con otra imagen de un tablero con sus lineas rectas (horizontales y verticales) mediante funciones de OpenCV como :

```
cv::calibrateCamera();
```

[1] Profesor: Paco Perales.

Figura 1. Calibración cámara



Conseguimos la calibración de forma automática hasta nuevo movimiento o cambio de perspectiva de la cámara. El sistema que hemos utilizado nosotros, es mucho más artesano pero nos ha funcionado. En la figura 2 se muestran puntos de diferentes colores. Estos son los puntos que definen una área en forma de trapezoide que nosotros queremos trasladar y hacer su perspectiva de forma que podamos ver esta imagen con cada uno de sus puntos colocados en una de las cuatro esquinas que tengan más cercana.

De esta forma conseguimos, en nuestro caso dos de los objetivos que necesitamos :

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

Figura 2. Perspectivas Probadas

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 imágen y de la lente.

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 lineas del carril para conseguir detectar las fronteras de estos. Podemos observar también en la imagen, y en el pseudocódigo de esta sección, que antes de aplicar este filtro la imagen ha sido transformada a una escala de grises, por lo que la información que disponemos en la matriz de la imagen y en la que vamos a estudiar a fondo las lineas tenemos un menor espectro de colores con los que trabajar.

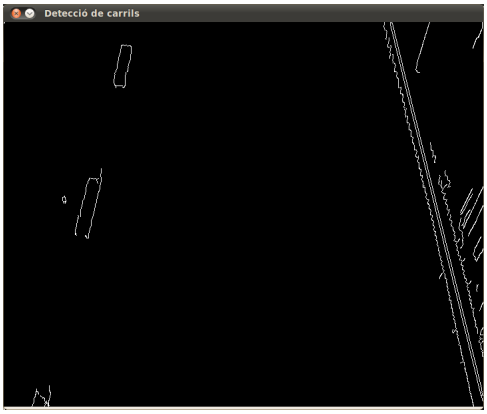


Figura 3. Filtro de Canny

Las instrucciones que se muestran a continuación son las que hemos 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.

```
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 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 nos 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. Las detección de lineas de “Hough” es muy conocida por su eficacia y por la facilidad de representación parametrizada de las lineas que encuentra. Cada linea que encuentra es representada por dos valores, *tetha* y *rho*, 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 lineas. En nuestro caso en particular no tuvimos ningún problema per la detección de las lineas continuas tanto izquierda como derecha, pero no obstante, tuvimos algunos problemas para mantener una linea de referencia con la linea discontinua. Por esa razón decidimos aplicar dos veces el algoritmo de “Hough”.

Gracias a ello, conseguimos que la segunda vez que aplicamos el algoritmo, ya aparecieran las primeras lineas y tuviera una mayor referencia para detectar las lineas, aunque debido a que la transformación de la perspectiva no es la óptima, hay veces que perdemos la referencia de la línea discontínua.

```
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 trozo de código anterior podemos observar como, una vez la linea ha sido encontrada, llevamos a cabo una discriminación. El ángulo de la linea nos ayudara a determinar si la linea es la del lado izquierdo o la del lado derecho, así como también nos ayudará a que no tengamos en cuenta aquellas lineas horizontales que puedan alterar el resultado del algoritmo.

Por otro lado, también tenemos en cuenta la posición que ocupan las lineas. Aquellas lineas que creemos que por su ángulo son del lado izquierdo, deben además estar posicionadas en este lado en la imagen. Lo mismo pasa con las lineas 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.

II-C2. Media Lineas: También se puede observar en el código como llevamos a cabo un media de todas las lineas resultantes. En nuestro caso, y además aplicando dos veces el algoritmo de “Hough” es muy grande el número de lineas que se encuentra. Por esa razón llevamos a cabo una media de todos los valores de *tetha* y *rho* detectados para formar una nueva linea (más gruesa) con los valores medios de todas las encontradas.

III. Mejoras Posibles

Una de las mejoras que hemos añadido al algoritmo, además de aplicar dos veces el algoritmo de “Hough” es la utilización de lo que hemos llamado *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, conseguimos que las lineas discontinuas no desaparezcan y aparezcan a gran velocidad y puedan ser visibles.

IV. Detección de cambio de carril

Una vez hemos realizado dos pasadas de la transformada de “Hough”, tenemos 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 hemos establecido para que se produzca el cambio de carril es que las dos pasadas del algoritmo no hayan encontrado ninguna línea. Esto funciona en la mayoría de los casos, aunque al ser una solución muy sencilla, podemos hacerla más compleja para que no incluya *falsos positivos*, es decir, que no detecte

cambios de carril cuando en realidad no se producen. En la figura 5 podemos ver el funcionamiento de la detección de cambio de carril.

Figura 5. Detección de cambio de carril

V. Conclusión

En conclusión hemos conseguido que la práctica funcione con un rendimiento muy bueno, y creemos que aplicando nuestro código a un sistema real, la detección de lineas puede ser muy buena. Creemos que la calibración puede mejorar si la llevamos a cabo mediante el sistema de calibrado avanzado explicada en apartados anteriores. Por lo demás, creemos que la integración con una interficie gráfico puede hacer que sea una práctica más completa. Para ver un vídeo de la ejecución de la práctica, puede seguirse el siguiente enlace: http://www.youtube.com/watch?v=3H2ecmD4Pwc.

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 Road-linesDetection* 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