Máster Universitari en Enginyeria Industrial (MUEI)
Escola Tècnica Superior d'Enginyeria Industrial de Barcelona (ETSEIB)

Sistemas de Percepción. Mini-proyecto

Diseño e implementación de la transformada de Hough para guiar coches en una carretera.

Jaume Cartró Benavides

Albert Costa Ruiz

Fecha: 15/5/2018

Profesor: Alberto Sanfeliu Cortés

Índice

1. Resumen del proyecto

En el siguiente proyecto se va a describir la técnica de la transformada de Hough aplicada a la detección de líneas rectas que en nuestro caso serán las líneas de una carretera. Con esta información se pretende ser capaz de guiar un coche autónomo. Todas las imágenes y videos que se usaran han sido realizados por los integrantes del trabajo. Con este material recolectado, se pretende tener una muestra fiable de las situaciones que se quieren resolver.

El procedimiento se resume en tres fases: una de preprocesado de imagen, otra de aplicación de la transformada de Hough y por último la obtención de las líneas a añadir a la imagen original. Los resultados muestran que el método es eficaz pero sin un buen preprocesado de las imágenes no siempre se consigue detección correcta de las líneas de los carriles. Al final del proyecto se comentaran en más profundidad los resultados y la fiabilidad del proceso.

2. Introducción

Entre las diversas técnicas utilizadas para la detección de las líneas de calles y carreteras, este trabajo se centra un método concreto: la transformada de Hough. Esta técnica permite detectar muchas formas geométricas a partir de varios parámetros. Por ejemplo, para la más simple, líneas rectas, necesita de dos parámetros; para esferas tres y a medida que la geometría se complica aumenta el número de parámetros. Sin embargo, el método pierde fiabilidad al usar muchos parámetros y por eso solo se va a aplicar en líneas rectas. Además, en nuestro caso de estudio es una aproximación más que suficiente incluso en tramos con curvas.

2.1 Motivación

El tema ha sido escogido con el objetivo de adentrar-nos en los sistemas usados en la conducción autónoma, más concretamente en la detección de los carriles de carretera para permitir su guiado. Con la realización de este proyecto, se espera aprender a usar el método que interviene y, sobretodo, lo que conlleva su implementación en un entorno real.

2.2 Método empleado

Con el objetivo de localizar y extraer las líneas que configuran los carriles de la carretera, se usa una técnica de detección de línea llamada Transformada de Hough. La principal idea de esta técnica es realizar agrupaciones de puntos que sean candidatos a formar entre ellos una recta. Las rectas que sean formadas por más puntos serán elegidas como verdaderas rectas según un valor límite (*'threshold'*).

Las rectas candidatas se describen en forma de coordenadas polares denotadas como ρ y Θ. Donde ρ es la distancia del origen a la recta tratada y Θ es el ángulo que forma la recta perpendicular a la recta tratada que pasa por el origen con el eje vertical. La imagen izquierda de la Figura 1 muestra dicha configuración.

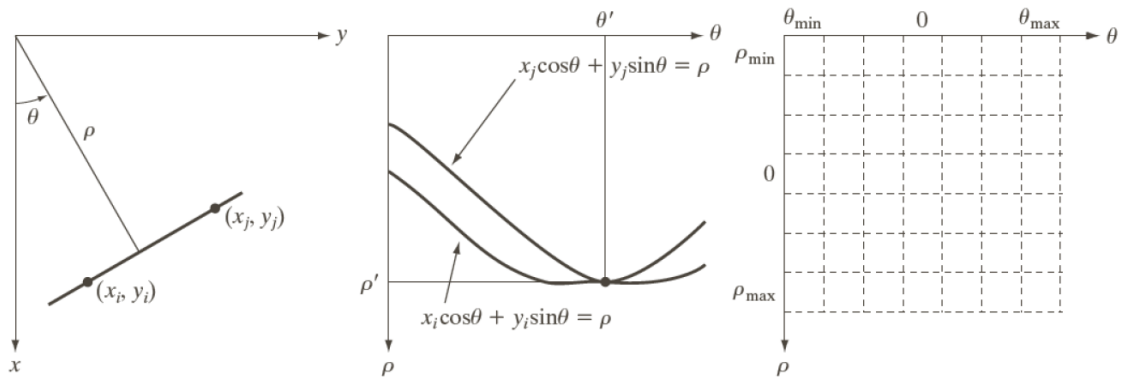

Figura 1: Izquierda) Espacio cartesiano Central)Espacio de Hough Derecha)Matriz Acumulador

De este modo, las posibles rectas que puedan pasar por un punto de la imagen de coordenadas (x, y) cumplirán la siguiente expresión:

$$\rho = x \cdot \cos\Theta + y \cdot \sin\Theta \;\; \text{con}\; \Theta \in [0, \pi] \; \text{y} \; \rho \in \mathbb{R}$$

El espacio de pares de valores ρ y Θ que representan las rectas que pasan por los puntos de la imagen, se denomina espacio de Hough. Con esta connotación, se pude decir que para cada punto de la imagen tendremos una recta sinusoidal que representa el conjunto de rectas que pasan por un punto en cuestión. La imagen central de la Figura 1muestra este espacio de Hough para dos puntos determinados de la imagen los cuales formaran una recta de parámetros ρ' y Θ' que correcponden al punto donde intersecan en este espacio. De esta manera si sinusoides de diferentes puntos intersequen en un punto determinado del espacio de Hough, indicará que hay muchos puntos colineales que forman una recta con tales parámetros.

Para el caso tratado, los puntos de los que se extraerán rectas serán aquellos que sean susceptibles de formar las líneas del carril de la carretera (tono suficientemente blanco de la imagen que corresponda al tono de los carriles). El algoritmo basado en la transformada de Hough construirá una matriz llamada Acumulador que contabilizará el número de pares ρ y Θ que coinciden entre puntos analizados (imagen derecha de la Figura 1). Una vez analizada toda la imagen, los pares de valores que hayan dado más concurrencias serán los considerados como rectas.

3. Implementación del método para un fotograma en Matlab

Matlab incorpora en sus librerías el método en su totalidad. Sin embargo, se ha programado manualmente para ver paso a paso su funcionamiento. En concreto se ha seguido el esquema propuesto por la documentación disponible en Wikipedia, Figura 2. Ésta describe el procedimiento a seguir mediante unos puntos que se explicaran a continuación. En nuestro código la función se llama 'Hough_Complet' y se encuentra en el anexo.

```
1:    cargar imagen
2:    detectar los bordes en la imagen
3:    por cada punto en la imagen:
4:       si el punto (x, y) esta en un borde:
5:          por todos los posibles ángulos θ:
6:             calcular ρ para el punto (x, y) con un ángulo θ
7:             incrementar la posición (ρ, θ) en el acumulador
8:    buscar las posiciones con los mayores valores en el acumulador
9:    devolver las rectas cuyos valores son los mayores en el acumulador.
```

Figura 2: Pasos a seguir para aplicar la transformada de Hough para la detección de rectas

1,2: Los dos primeros pasos son que conforman el pre-pocesado y como resultado se obtiene una imagen binarizada donde idealmente los píxeles que forman parte de los bordes toman valor 1 y el resto 0.

Tanto el punto 1 como el 2 se han implementado en una función aparte, Annexo 1.2. Dada su gran importancia se vera con mas detenimiento en un apartado aparte posteriormente.

3,4: Se hace un recorrido por todos los píxeles de la imagen y, si éste es "cierto" (al tratarse de una matriz lógica significa que su valor es 1), se procede a tratarlo.

5,6: Se calcula el parámetro $\boldsymbol{\rho}$ para cada ángulo $\boldsymbol{\theta}$. Cabe decir que es necesario discretizar ambos valores ya que por un pixel pueden pasar infinitas rectas. También se ha aplicado un filtro que consiste en recorrer los ángulos de $\pi/5$ a $4\pi/5$ en lugar de 0 a π para dejar fuera a las rectas que se acerquen a una horizontal, las cuales no interesan para nuestro propósito.

7: Para cada imagen se crea una matriz de ceros llamada Acumulador donde cada celda representa una recta. La fila de la celda marcara el parámetro $\boldsymbol{\rho}$ y la columna que ocupe el parámetro $\boldsymbol{\theta}$. Entonces para cada pixel seleccionado y para cada ángulo se incrementara la posición ($\boldsymbol{\rho}$,$\boldsymbol{\theta}$) en una unidad.

Los puntos que van del 3 al 7 están implementados en la función 'trans_hogh_casera' mostrada en el anexo, la cual retorna la Matriz acumulador a partir de una imagen binarizada.

8: Dentro de la matriz Acumulador se buscarán los valores que superen un límite. Se ha optado por normalizar la matriz de manera que si el límite es 0.8, se cogerán los valores que sean un 80% el valor máximo de la Matriz acumulador.

Para entender mejor como escoger las rectas, vamos a ver gráficamente la matriz Acumulador en un ejemplo y que rectas se han de coger. Para graficarla se tomará una escala de grises donde las celdas 0 serán de color negro, las de valor 1 blanco, y escala de grises entremedio.

Dada la Figura 3,

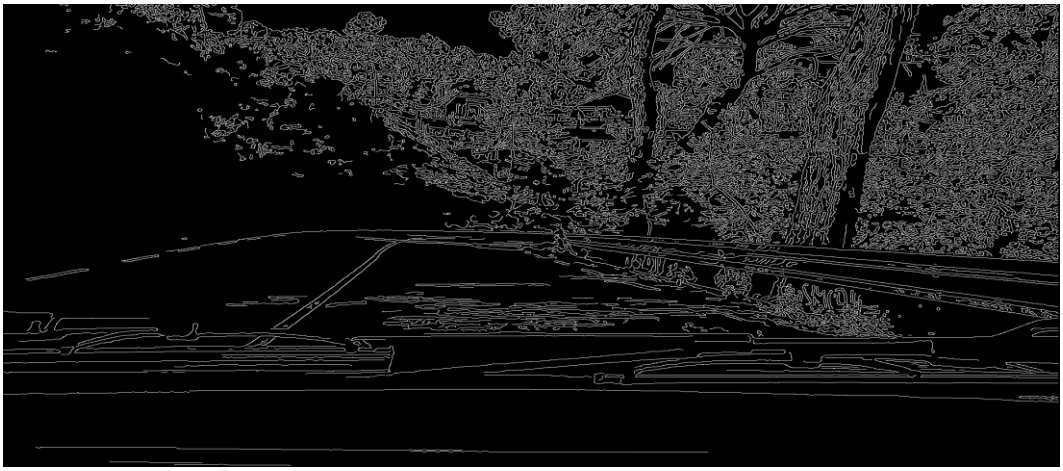

Figura 3: Imagen binarizda

Se obtine la matriz acumulador siguiente:

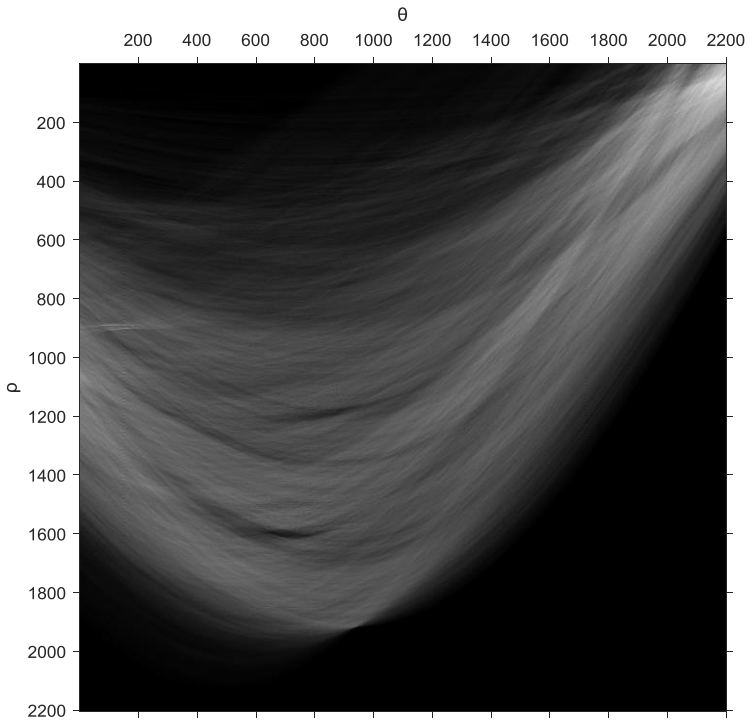

Figura 4: Representación de la matriz Acumulador

Donde las posiciones con tonos más blancos seran escogidos para reprensentar la recta correspondiente.

9: De las rectas seleccionadas se transforman los parámetros en el espacio de Hough a los parámetros de una recta en el espacio cartesiano para representarlas.

Los puntos 8 i 9 se han implementado mediante la función 'Plot_hough_lines', mostrada en el annexo. Esta función tiene varios propósitos:

- Para empezar, se seleccionará las celdas que superen el límite marcado. No obstante, para evitar tener muchas rectas muy parecidas solo se coge a una en un cierto espacio. Es decir, se cogerán los máximos locales. Para ello se ha usado el concepto de máscara: se itera buscando el máximo de la matriz Acumulador para cada iteración y se guarda los parámetros de la recta en otra matriz. Luego todas las celdas cercanas se ponen a zero para que no puedan ser seleccionadas.
  (Se podría imponer por ejemplo que solo iterara dos veces si se está seguro que se obtendrán las dos rectas a cada lado de la calzada.
- Para esta matriz se realizan unos cambios a los valores de los ángulos: primero se pasa de la codificación de columnas al ángulo en el espacio de Hough y luego, se hace una conversión de los ángulos para facilitar la representación de la recta pasando su valor nulo al eje horizontal.
- Con los dos parámetros de Hough se calculan los parámetros ***m*** i ***b*** de una recta en el espacio cartesiano ($y = mx + b$) y se calculan dos puntos con los cuales se representa la recta que los une.

Como ejemplo, en la Figura 5, los parámetros (***ρ,θ***)=(900,pi/4) son los que definen la recta de color naranja. La recta azul es la perpendicular que pasa por el origen de la imagen y en rojo está marcado el punto de intersección.

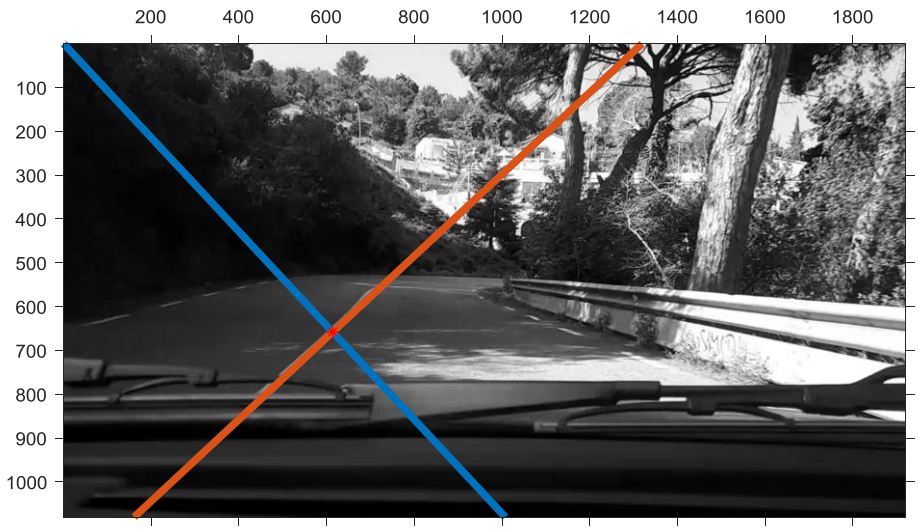

Figura 5: Ejemplo de representación de una recta

3.1 Pre-procesado

El principal objetivo del procesado, es tratar la imagen de tal manera que se facilite la posterior detección de las líneas buscadas. Se tomarán medidas tanto para eliminar todo aquello que sea susceptible de ser captado como línea de carril pero que no lo sea, como para resaltar las líneas que configuren el carril y así su posterior detección resulte más fácil.

Para empezar, comentar que los videos grabados fueron a color con fotogramas constituidos por píxeles de valores enteros de 8 bits sin signo (uint8). La Figura 6 muestra un ejemplo.

Figura 6: Fotograma de un video grabado a color

El primer paso que se realiza es pasar la imagen de color a escala de grises (con `rgb2gray()`) y a continuación representar sus píxeles como valores de tipo coma flotante de 8 bytes tomando valores comprendidos entre 0 y 1 (con `im2double(I)`). La Figura 7 muestra el resultado después de este primer tratamiento.

Figura 7: Imagen en escala de grises y en formato double

El segundo paso que se lleva a cabo es un recorte de la imagen que elimina parte del cielo y el salpicadero del coche para así dejar solo visible la parte que conforma la carretera (con `imcrop()`). La Figura 8 muestra la ventana de la imagen que se deja.

Figura 8: Imagen recortada por la parte de interés

El tercer paso consiste en el binarizado de la imagen. Como las líneas que conforman los carriles suelen ser de un tono blanco o muy cercano a él, el umbral de binarizado usado es próximo al

valor de blanco con tal que queden el mínimo de elementos en ese color, siempre y cuando no se eliminen las líneas a identificar (se usa `imbinarize()`).

Figura 9: Imagen binarizada

En el siguiente paso se hace una limpieza de elementos blancos que son lo suficientemente pequeños como para asegurar que no son rectas. Primero se realiza una limpieza de los bordes de la imagen mediante la función `imclearborder()`. Con esta función suprimimos estructuras blancas que están tocando los marcos de la imagen menores a una determinada magnitud de estructura. Después se usa una función de 'area opening' con `bwareaopen()`, la cual permite eliminar todos los píxeles blancos que conforman un elemento conectado de dimensión inferior a un número indicado de píxeles. La Figura 10 muestra el resultado de aplicar estas dos operaciones.

Figura 10: Limpieza de las estructuras blancas de menor tamaño

Pera finalizar con el preprocesado, se realiza la detección de contornos mediante la función `edge()` y el método Canny, como se observa en la Figura 11. En los anexos se muestra la función de preprocesado que se acostumbra a usar en los videos.

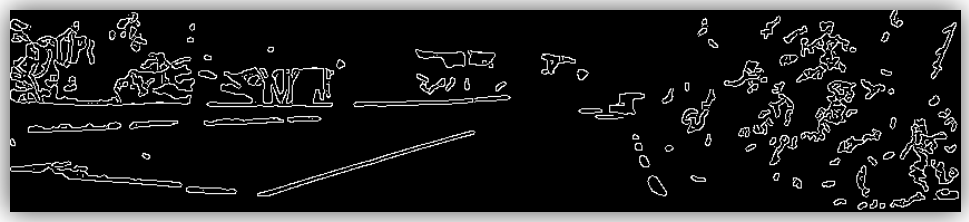

Figura 11: Detección de contornos de todas las estructuras blancas

3.2 Función añadida para indicar dirección

El objetivo principal del proyecto es el de guiar un coche durante su funcionamiento. Para ello no solamente hace falta ver las líneas de la carretera, sino que también la dirección que ha de tomar el coche para ir por el medio del carril.

Para implementar esta característica, se ha creado una función, Anexo A.4, que añade una flecha la cual marca la dirección a seguir. Entre todas las opciones posibles, debido tiempo de dedicación al proyecto, se ha decido que solo muestre la flecha cuando el sistema detecte dos líneas las cuales se consideraran como buenas (como trabajo futuro sería pertinente comprobar más rigurosamente si realmente son líneas correctas).

3.3 Tratamiento de un video

Para poder utilizar el programa en su entorno, durante la conducción, es necesario que procese muchas imágenes in situ. Para simular esta acción se ha hecho un programa (Anexo A.6) el cual dado un video va leyendo cada *frame* y aplica la función 'Metode_complet', que obtiene la imagen ya procesada, y va generando otro video para su posterior visualización.

Además, se ha añadido a la función 'Metode_complet' una característica que permite tratar por ejemplo un *frame* de cada tres y de esta manera agiliza el proceso enormemente (ha sido de gran utilidad ya no se tenía suficiente potencia de cálculo en el computador para procesar cada video de manera fluida y no afecta cualitativamente al resultado final). Los *frames* que no hacían el proceso completo mostraban la misma información que el último frame tratado. Para ello se ha creado unas variables locales a la función en sí para que se guardaran cálculos anteriores.

4. Resultados

El método ha sido probado con videos realizados en carreteras de diferentes tipos, con diferentes entornos y en diversos momentos del día. La principal problemática que se ha ido encontrado ha sido la robustez del método frente a los diversos tipos de video. En casi todos había que variar los parámetros y funciones a usar del preprocesado para realizar una buena captura de las líneas.

Primero se presentarán los factores que han dado lugar a más problemáticas. Los videos en los que aparecían cambios de luminosidad, como sombras y reflejos en la carretera, producían la detección de formas colineales extras o erróneas. La Figura 12 es un buen ejemplo en el que se da este suceso. Donde el conjunto de sombras y luz dificulta realizar un preprocesado en el que solo resalten las líneas deseadas.

Figura 12: Detección de líneas en un lugar de luminosidad irregular

También han surgido problemas en las curvas muy cerradas en las que las líneas de los carriles tenían tal curvatura que no eran detectadas como más de una recta. La Figura 13 muestra un claro ejemplo.

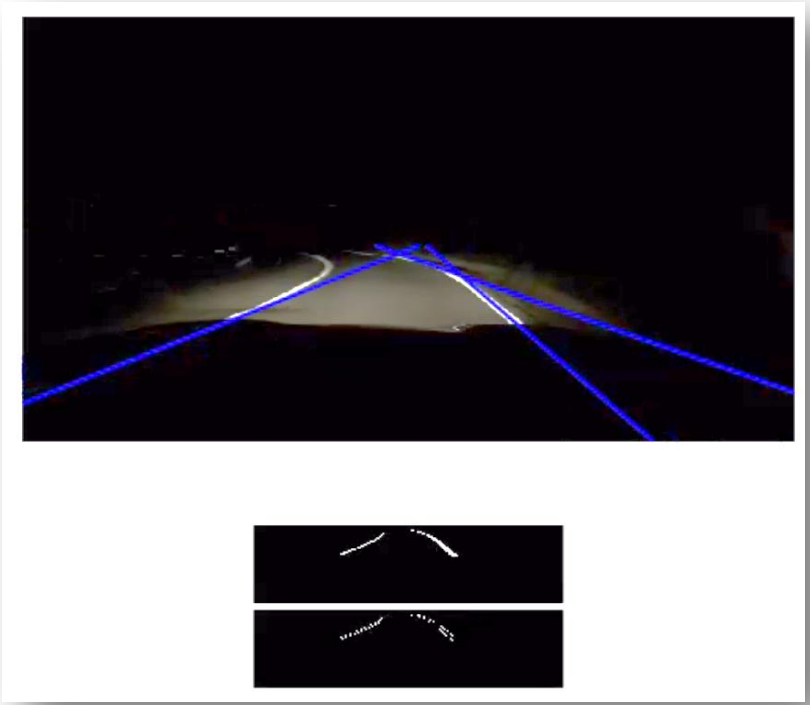

Figura 13: Detección de un carril formado por rectas de gran curvatura

En resumen, los principales problemas son:

- De día: las sombras en la carretera donde se genera un importante gradiente de color y acaban detectándose como rectas (en la Figura 12 se puede ver su efecto) .
- De noche: una gran iluminación repentina como por ejemplo la que se crea al encontrarse con un vehículo que circula en sentido contrario. En la siguiente figura se puede apreciar este efecto.

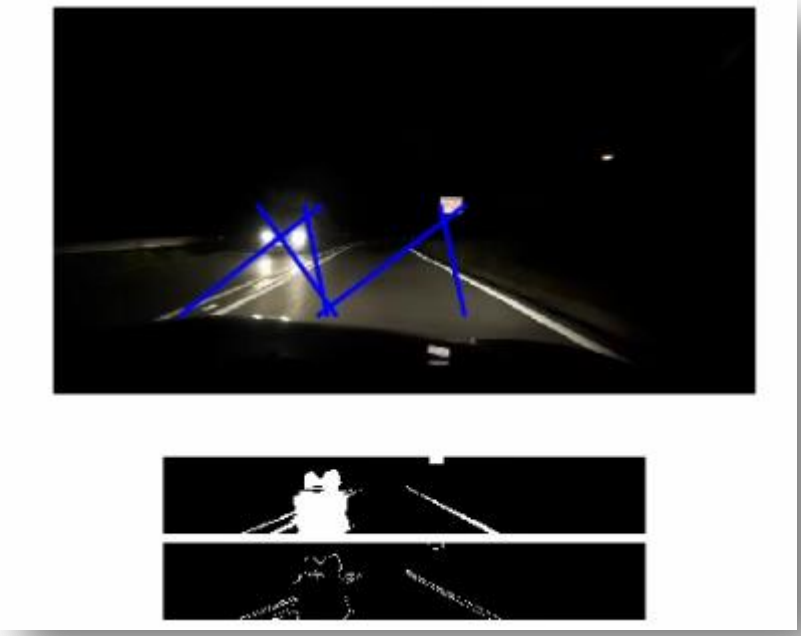

Figura 14: Situación de iluminación repentina por los faros de un coche en la noche

Las situaciones idóneas de detección de las dos rectas se han dado cuando la luminosidad de las imágenes permanecía más o menos constante a lo largo del video en todos los pixeles de los fotogramas, y cuando los carriles mantenían una dirección más menos constante sin cambios bruscos de dirección. Las siguientes dos figuras muestran videos en los que se dieron estas características. Uno fue realizado de noche y el otro durante el día.

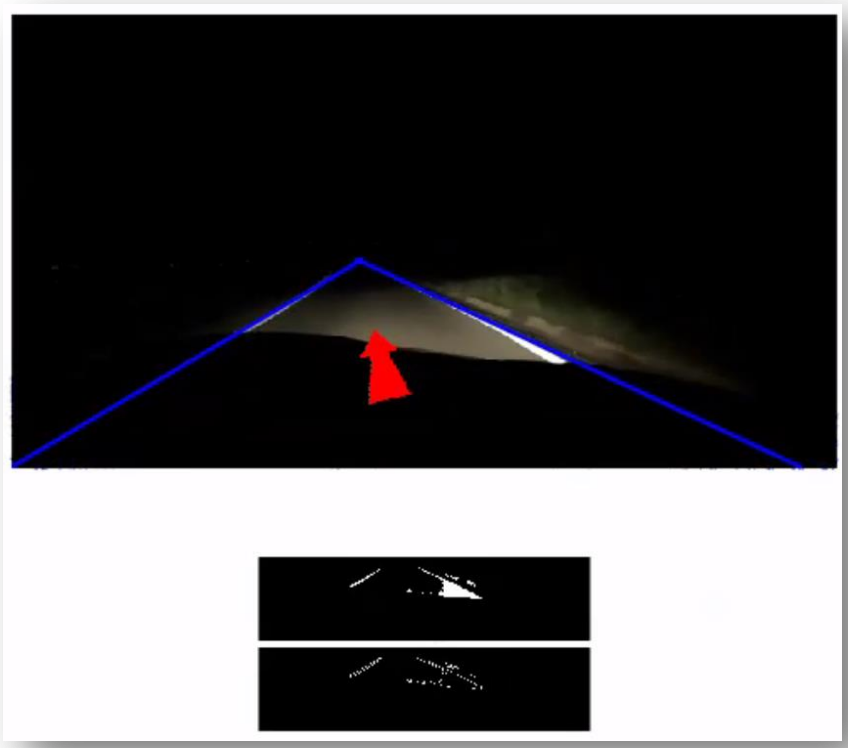

Figura 15: Detección de un carril de líneas rectas por la noche

Figura 16: Detección de líneas rectas discontinuas durante el día

5. Mejoras futuras

El método de Hough como tal funciona correctamente. El preprocesado en cambio, que condiciona la aplicación de la transformada de Hough posterior, puede mejorarse bastante y por eso la mayoría de mejoras van enfocadas en ese sentido. Entre muchas otras estarían las de:

- Mejorar el tratamiento de las líneas discontinuas con tal de mejorar su detección como rectas.
- Realizar un Filtro de Kalman para que, a cada iteración del proceso, las nuevas rectas ilustradas sean aquellas que se asemejen a las anteriores (restricción para el resto de rectas que se puedan crear). También serviría para estimar las nuevas rectas en caso que no se detecten.
- Detectar inicio y final de las líneas que conforman los carriles. Se podría realizar a partir de un método de búsqueda de características o descriptores como HoG o SIFT.
- Realizar un sistema de preprocesado inteligente, de forma que según la luminosidad de la imagen y su histograma detecte la zona en la que se halle el carril y que tome las técnicas idóneas para tratar las imágenes lo mejor posible.
- Ilustrar la flecha que indica la dirección que ha seguir el coche solo en los casos en que se hayan encontrado bien las líneas de los carriles.
- A lo que se refiere a la captación de los videos, implementar en el coche una cámara fija y así evitar los movimientos y descuadres del plano que ahora se tienen.
- Desarrollar funciones y métodos de preprocesado más especializados que realicen los mismos cálculos pero de manera más eficiente.

6. Conclusiones

Se ha logrado el objetivo principal de implementar satisfactoriamente el método de Hough. A pesar de su buena implementación, la detección de las líneas no siempre ha sido la adecuada. Se ha visto de la gran importancia de hacer un buen preprocesado de la imagen el cual condiciona sin ninguna duda los resultados finales.

También se querría destacar que gran parte de la implementación del proceso, sobre todo la parte del método y el post-procesado, ha sido realizado con código personal evitando la ayuda de funciones predeterminadas. De esta manera se ha entendido el método con profundidad y se ha podido adecuar y personalizar para nuestros intereses.

Pensamos que con el tiempo dado y los recursos de los que se disponía, se han conseguido unos resultados bastante satisfactorios. Ha resultado enriquecedor realizar este proyecto, sobre todo por la experiencia de usar este tipo de técnicas de visión por computador en un caso práctico, siendo así conscientes de toda la problemática y factores envolventes que entran en juego.

Annexo: Scripts/Funciones Matlab

A.1 Implementación completa para un frame

```
function y = Metode_complet(Frame,cont,F)
persistent x_plot;persistent y_plot;persistent I;persistent Iv;
Tall_sup = 150;  %Vid9:180
Tall_inf = 45;   %Vid9:45
if cont == 0 % es tracta el pixel
    subplot(3,1,[1,2])
    [Iv,I] = preprocesat_1(Frame,Tall_sup,Tall_inf);
    imshow(Frame)
    Discret_thetas=(pi*0.1:0.005:pi*0.9);
    [Acumulador,~] = trans_Hough_casera(Iv,Discret_thetas);
    [x_plot,y_plot]=Plot_hough_lines(Frame,Acumulador,0.4,Discret_thetas,..
        Tall_sup,Tall_inf);
    % Frame (+ direccio quan calgui)
    imshow(Frame)
    Frame = afegir_direccio(Frame,x_plot,y_plot,Tall_sup,F);
    imshow(Frame);hold on;axis off;
    % Afegir lineas carretera
    c8 = 1;
    for e = x_plot'
        plot(e,y_plot(c8,:)+Tall_sup,'linewidth',1.5,'color','b');
        c8 = c8 +1;
    end
else % no es tracta, s'agafa dades anteriors
    subplot(3,1,[1,2])
    [Frame,cc,cc2] = afegir_direccio(Frame,x_plot,y_plot,Tall_sup,F);
    imshow(Frame);hold on;axis off;
%     scatter(cc,Tall_sup,80,'g');scatter(cc2,61+Tall_sup,80,'g')
    try
        c8 = 1;
        for e = x_plot'
            plot(e,y_plot(c8,:)+Tall_sup,'linewidth',1.5,'color','b');
            c8 = c8 +1;
        end
    end
end
subplot(3,1,3)
[~,amp] = size(I);
II = [I;ones(10,amp);Iv];
imshow(II);axis off;
set(gcf,'color','white')
set(gca,'XAxisLocation','top','YAxisLocation','left','ydir','reverse');
end
```

A.2 Función de preprocesado

```
function [y1,y2] = preprocesat(Frame,Tall_sup,Tall_inf)
I = rgb2gray(Frame);
I = im2double(I);

I = imcrop(I,[0 Tall_sup c (f-Tall_sup-Tall_inf)]);
%segun luminosidad binarizo una parte
%de la imagen distinta que la  otra
thresh = 175;
thresh2=130;
I(:,1:c/2)= imbinarize(I(:,1:c/2),thresh/255);
I(:,c/2:c)= imbinarize(I(:,c/2:c),thresh2/255);

I(:,1:c/2) = bwareaopen(I(:,1:c/2),20);
I = imclearborder(I,8);

Iv = edge(I,'canny');

y1 = Iv;
y2 = I;

end
```

A.3 Función de la Transformada de Hough

```
function [y,Rho_max] = trans_Hough_casera(I,Discret_thetas)
[f,c] = size(I);
contador = length(Discret_thetas);
Rho_max = ceil(norm([f,c]));
Acumulador = zeros(Rho_max,contador);
for x_i = 1:c
    for y_j = 1:f
        if I(y_j,x_i)
            for theta = 1:contador
                Theta = Discret_thetas(theta);
                rho = round(1*(y_j*cos(Theta)+x_i*sin(Theta)));
                if rho > 0
                    Acumulador(rho,theta) = Acumulador(rho,theta) + 1;
                end
            end
        end
    end
end
% Normalitzacio
valor_max = max(Acumulador(:));
y = Acumulador/valor_max;

end
```

A.4 Función para añadir dirección

```
function [y,cc,cc2] = afegir_direccio(Frame,x_plot,y_plot,Tall_sup,F)
% Afegir fletxa quan es te dos lineas bones
    [s1,s2] = size(x_plot);
    [~,s4] = size(Frame);
    if (s1 == 2) && (s2 == 2)
        centre_x = round(mean(mean(x_plot)));
        centre_y = round(mean(mean(y_plot)));
        x_mitjal = mean(x_plot);
        x_mitja = (x_mitjal(1)-x_mitjal(2));
        y_mitja = mean(y_plot);
        if (x_mitja > s4*0.4) && (x_mitja < s4*0.6) % nomes si esta al centre
            cc=x_mitjal(1);  % per representar punts que marquen la direccio
            cc2=x_mitjal(2);  % i veure que s'orienta be la fletxa
            if x_mitja > 0
                angle = -(atan(abs(x_mitja)/(y_mitja(2)-y_mitja(1)))*(180/pi));
            else
                angle = (atan(abs(x_mitja)/(y_mitja(2)-y_mitja(1)))*(180/pi));
            end
            F = imrotate(F,angle);  % orientar fletxa
            [f1,f2,~] = size(F);
            ill = 0;i22=0;
            for il = (centre_y - round(f1/2)+Tall_sup):1:(centre_y - round(f1/2)+f1+Tall_sup-1)
                ill = ill+1;
                for i2 = (centre_x - round(f2/2)):1:(centre_x - round(f2/2) + f2-1)
                    i22 = i22 +1;
                    if F(ill,i22,1) == 1  % afegir pixels de la fletxa al frame original
                        Frame(il,i2,1)=255;
                        Frame(il,i2,2)=0;
                        Frame(il,i2,3)=0;
                    end
                    if i22 == f2
                        i22 = 0;
                    end
                end
            end
        else
            cc=0;cc2=0;
        end
    else
        cc=0;cc2=0;
    end
y = Frame;
end
```

A.5 Función que encuentra puntos que definen las rectas a mostrar

```
function [x_plot,y_plot] = Plot_hough_lines(Frame,Acumulador,llindar,...
    Discret_thetas,Tall_sup,Tall_inf)
[cl,~] = size(Frame);
%% Agafar les rectes importants
M_lines = [];
[f,c] = size(Acumulador);
[max_num, max_idx]=max(Acumulador(:));  % valor max Acumulador
[e1,e2]=ind2sub([f,c],max_idx);         % index valor max
while max_num > llindar
    M_lines = [M_lines;e1,e2];
    for i1 = (e1-120):1:(e1+120)        % es posen a 0 rectes properes
        for i2 = (e2-120):1:(e2+120)    % amb menys punts i l'escollida
            try                         % evita error al sobrepassar marcs
                Acumulador(i1,i2)=0;
            end
        end
    end
    [max_num, max_idx]=max(Acumulador(:));   % es recalcula valor max
    [e1,e2]=ind2sub([f,c],max_idx);
end
if ~isempty(M_lines)                        % si s'ha trobat alguna recta
    Rhos = M_lines(:,1);
    Thetas = M_lines(:,2);
    %% Conversio de index a angle
    for e = 1:length(Thetas)
        Theta_convertit = Discret_thetas(Thetas(e));
        if Theta_convertit <= pi/2
            Thetas(e) = pi/2 - Theta_convertit;
        else
            Thetas(e) = -(Theta_convertit - pi/2);
        end
    end
    %% Trobar l'equació de les rectes   (y = m·x + b)
    punt_recta = [Rhos.*cos(Thetas),Rhos.*sin(Thetas)];
    m_perpen = punt_recta(:,2)./punt_recta(:,1);
    b = punt_recta(:,2)+(m_perpen.^-1).*punt_recta(:,1);
    %% Pintar les rectes completes
    % x = 0:0.1:c2;
    % y = -(m_perpen.^-1).*x+b;
    % y = y + Tall_sup; % correccio retall d'una imatge a laltre
    %% Trobar 2 punts de cada recta dins del retall de la imatge
    x_plot = [];
    y_plot = [];
    for e = 1: length(Rhos)
        y_plot = [y_plot;0,cl-Tall_inf-Tall_sup];
        x_plot = [x_plot;(y_plot(e,:)-b(e))./(-m_perpen(e)^-1)];
    end
else     % si no s'ha trobat cap recta
    x_plot = [];
    y_plot = [];
    Rhos = [];
end
%yp = m_perpen.*x;    % rectes perpendiculars
%plot(x,y,'linewidth',2,'color','r');  %pintar rectes escollides
%plot(x,yp,'linewidth',1,'color','g');  % pintar rectes perpendiculars
%plot(punt_recta(:,1),punt_recta(:,2),'r*');  % pintar punts per on passa
                        %recta per origen i es perpendicular a la recta y
end
```

A.6 Script para tratar un vídeo

```
close all
clear all
clc
%%
F=llegir_fletxa(0.2);
v = VideoReader('Video_1.mp4');
vid = VideoWriter('Video_1_mod','MPEG-4');
vid.FrameRate = v.FrameRate;
cont = 0;
c_max = 50;
while hasFrame(v)
    vidFrame = readFrame(v);
    vidFrame = imresize(vidFrame,1);
    Hough_complet(vidFrame,cont,F);
    cont = cont +1;
    if cont > c_max
        cont = 0;
    end
    pause(1/v.FrameRate);
    open(vid);
    f = getframe(gcf);
    writeVideo(vid,f)
end
close(vid);
```