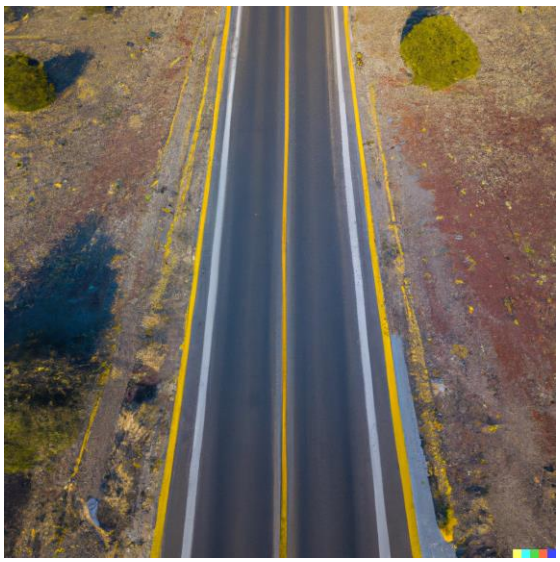
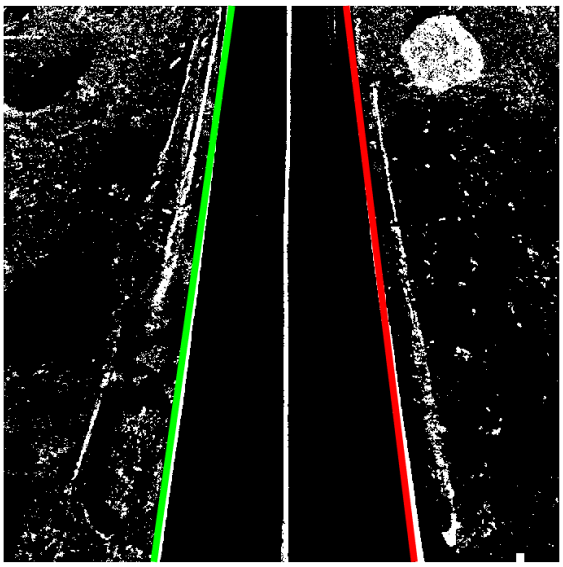
Sistemas de Percepción





Práctica 3. Segmentación de calzada en imagen aérea

Trabajar con la transformada de Hough para la detección de rectas y segmentación de regiones a partir de ellas.

Nombre: Sergio León Doncel y Álvaro García Lora

Curso: 2022/ 2023

Titulación: 4º Curso, GIERM

MEMORIA. PRÁCTICA 3

INTRODUCCIÓN

El objetivo consiste en desarrollar en Matlab un algoritmo de procesamiento de imágenes capaz de realizar la segmentación de la calzada de una carretera mediante la detección de líneas amarillas presentes en los fotogramas de un vídeo (grabado con vista aérea desde un dron) obtenido de un dataset, mostrando en una imagen sólo la parte correspondiente a esta.

VERSIÓN BÁSICA

En primer lugar, se realiza la versión básica en donde se detectan las 2 líneas con mayor número de votos, y se comprueba cuál es la exterior de la carretera (pintada de rojo) y la interior (marcada de verde). Además, en caso de detectar alguna que se corresponda con estas, se mostraría de color amarillo, y dado que se ha producido el fallo, se decide mostrar la imagen entera, ya que no es posible mostrar sólo la región de interés en ese caso sin realizar pasos extra.

Como primera medida, se implementa un bucle for para iterar el mismo procesamiento a todos los fotogramas del vídeo.

```
clear; close all; clc;

fallosTotales=0; %variable de conteo de número de fallos por detección de línea errónea

for fotograma = 260:790 %Leer y procesar de la misma manera los distintos fotogramas
para probar robustez del algoritmo ante distintas situaciones

    str_fotograma = num2str(fotograma);
    str_path = strcat('secuenciaBicicleta\000',str_fotograma,'.jpg');
    fileImageRGB = imread(str_path);
```

REDUCCIÓN DE TAMAÑO DE IMAGEN

Prosiguiendo, tal y como se pide en las especificaciones definidas en la memoria, se escala el tamaño de la imagen para reducirla, con objetivo de hacer menor el coste computacional del procesamiento.

```
    fileImageRGB = imresize(fileImageRGB,0.3); %reducir tamaño de imagen a 1/3
    figure(1);
    hold on
    subplot(3,2,1)
    imshow(fileImageRGB);
```

PREPROCESADO

Para conseguir un mejor análisis de la imagen por colores (para aprovechar el hecho de que las líneas son amarillas), pasamos la imagen a HSV y dividimos sus capas.

```
    umbralThreshold = [0.106 0.217 0.465; %fila: 1-Mínimo 2-Máximo | columna: 1-Hue 2-
Saturation 3-Value
        0.221 1.000 1.000];

    [M,N,C] = size(fileImageRGB);

    fileImage = rgb2hsv(fileImageRGB);
    HueImage  = fileImage(:,:,1);
    SatImage = fileImage(:,:,2);
    ValImage  = fileImage(:,:,3);
```

BINARIZACIÓN POR UMBRALES

Creamos una plantilla binarizada a partir de los umbrales obtenidos con Color Thresholder.

```
    if(umbralThreshold(1,1)<umbralThreshold(2,1))
        binHueImage = (umbralThreshold(1,1)<=HueImage & HueImage<=umbralThreshold(2,1));
    else
        binHueImage = (umbralThreshold(1,1)<=HueImage | HueImage<=umbralThreshold(2,1));
    end
    binSatImage = (umbralThreshold(1,2)<=SatImage & SatImage<=umbralThreshold(2,2));
    binValImage = (umbralThreshold(1,3)<=ValImage & ValImage<=umbralThreshold(2,3));
    binImage = binHueImage & binSatImage & binValImage;
    subplot(3,2,2); imshow(binImage,[]);
    subplot(3,2,4); imshow(binImage,[]);
    subplot(3,2,3); imshow(binImage,[]);
```

TRANSFORMADA DE HOUGH

Utilizamos la función hough() agregándole como entrada la imagen binarizada, para que nos devuelva la tabla de votos. Luego se recurre a houghpeaks() para obtener los valores de rho y theta de las 2 rectas con mayor número de votos.

Por otro lado, para conseguir continuidad en las variables características de la recta, ya que peaks() devuelve theta en el rango de [-90,90), se incrementa 180 su resultado para que theta pertenezca a [0,180).

Además, comprobando el parecido de las rectas obtenidas con la del primer fotograma, se guarda en la componente 1 del vector la recta correspondiente a la línea interior de la carretera, como componente 2 la correspondiente a la línea exterior, y con índice de 3 en adelante a las líneas erróneas detectadas.

```
[M_votes,tabTheta,tabRho] = hough(binImage);
peaks = houghpeaks(M_votes,2,"NHoodSize",[21 21]);
rho = tabRho(peaks(:,1));
theta = tabTheta(peaks(:,2));
for k=1:length(theta)
    if(theta(k)<0) %theta  pertenece a [-90,90) pero nos interesa de
        % [0,180) por las funciones de discretización usadas
        % posteriormente que utilizaran theta y rho
        theta(k) = theta(k)+180;
        rho(k) = -rho(k);
    end
end

%Definir en el primer fotograma color de las líneas guardando su valor
%inicialmente como valor anterior en la secuencia de fotogramas
if(fotograma==260)
    thetaG_ant=theta(1);
    thetaR_ant=theta(2);
    rhoG_ant=rho(1);
    rhoR_ant=rho(2);
end

%Comprobar línea correspodiente mediante parecido en parámetros
%característicos (theta y rho)
indiceFallosFotograma=0;
%inicializar rho y theta de las rectas del fotograma a -1 para marcar
%cuando no han sido localizadas
rhoRectasFotograma=-ones(1,length(rho));
thetaRectasFotograma=-ones(1,length(rho));
for k=1:length(rho)
    if( abs(rho(k)-rhoG_ant)<20 && abs(theta(k)-thetaG_ant)<10 ) %verde
        rhoRectasFotograma(1)=rho(k);
        thetaRectasFotograma(1)=theta(k);
        rhoG_ant=rhoRectasFotograma(1);
        thetaG_ant=thetaRectasFotograma(1);
    elseif( abs(rho(k)-rhoR_ant)<20 && abs(theta(k)-thetaR_ant)<10 ) %rojo
        rhoRectasFotograma(2)=rho(k);
        thetaRectasFotograma(2)=theta(k);
        rhoR_ant=rhoRectasFotograma(2);
        thetaR_ant=thetaRectasFotograma(2);
    else %amarillo (errónea)
        indiceFallosFotograma=indiceFallosFotograma+1;
        rhoRectasFotograma(2+indiceFallosFotograma)=rho(k);
        thetaRectasFotograma(2+indiceFallosFotograma)=theta(k);
        fallosTotales=fallosTotales+1;
    end
end
```

Tras esto, se calcula las intersecciones de esas rectas con los marcos de la imagen (teniendo en cuenta que en los vértices la intersección pueda ser doble y no debemos calcularla 2 veces), y que para k=1 se pinta de verde la línea, para k=2 de rojo, y para k distinta a estas de amarillo indicando un error.

```
    %calcular puntos de intersección con bordes
    %Orden cálculo intersecciones: recta HorizontalSuperior(HS), VerticalDerecha(VD)
    % , HorizontalIzuquierda(HI), VerticalIzquierda(VI)
    rhoMarco = [0,N,M,0];
    thetaMarco = [90,0,90,0];
    x=-ones(2,length(rhoRectasFotograma));y=-ones(2,length(rhoRectasFotograma));
    for k=1:length(rhoRectasFotograma)
        a=1;
        if(thetaRectasFotograma(k)>0) %si la recta ha sido localizada correctamente
            for i=1:4
                xIntersec=round((rhoRectasFotograma(k)*sind(thetaMarco(i))-
rhoMarco(i)*sind(thetaRectasFotograma(k)))/(cosd(thetaRectasFotograma(k))*sind(thetaMarc
o(i))-sind(thetaRectasFotograma(k))*cosd(thetaMarco(i))),2);
                yIntersec=round((rhoMarco(i)*cosd(thetaRectasFotograma(k))-
rhoRectasFotograma(k)*cosd(thetaMarco(i)))/(cosd(thetaRectasFotograma(k))*sind(thetaMarc
o(i))-sind(thetaRectasFotograma(k))*cosd(thetaMarco(i))),2);
                if(xIntersec>=0 && yIntersec>=0 && xIntersec<=N && yIntersec<=M)
                    if( ~max(x(:,k)==xIntersec & y(:,k)==yIntersec) ) % comprobar que
                        % la intersección no sea la previamente calculada ya que en los
                        % vértices la intersección es doble con los marcos de la imagen
                        x(a,k)=xIntersec;
                        y(a,k)=yIntersec;
                        a=a+1;
                    end
                end
            end

            %pintar líneas en imagen binarizada de color azul cyan
            subplot(3,2,3)
            line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','c','LineWidth',5);

            %pintar líneas en imagen binarizada marcadas con su propio
            %color invariante a lo largo de los fotogramas
            subplot(3,2,4)
            switch k
                case 1 %Verde (línea interior de carretera)
                    line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','g','LineWidth',5);
                case 2 %Rojo (línea exterior de carretera)
                    line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','r','LineWidth',5);
                otherwise %Amarillo(Fallos)
                    line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','y','LineWidth',5);
            end
        end
    end
```

EXTRACCIÓN DE REGIONES

Recurriendo a los clasificadores lineales y las funciones discriminantes, podemos utilizar las rectas obtenidas (en caso de no haber errores, lo cuál se comprueba verificando que theta no sea negativo) para quedarnos sólo con los píxeles pertenecientes a la región interior conformada por ambas rectas. Gracias a ello se crea una plantilla que sirve como máscara a la hora de mostrar la imagen con sólo la región de interés.

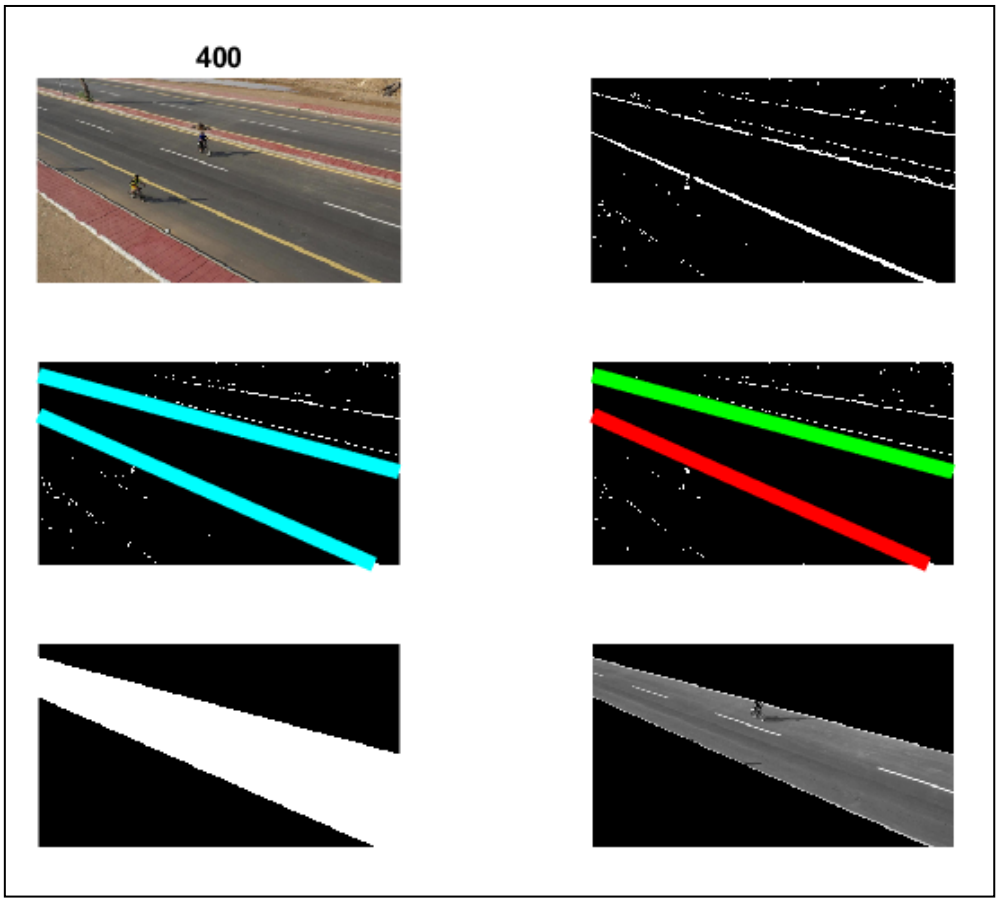
```
    if(thetaRectasFotograma(1)>0 && thetaRectasFotograma(2)>0) %comprobar si se ha
detectado correctamente las líneas
        w1_ = [cosd(thetaRectasFotograma(1)) sind(thetaRectasFotograma(1)) -
rhoRectasFotograma(1)]';
        w2_ = -[cosd(thetaRectasFotograma(2)) sind(thetaRectasFotograma(2)) -
rhoRectasFotograma(2)]';
        processedImage = uint8(zeros(M,N));

        for y1=1:M
            for x1=1:N
                x_ = [x1 y1 1]';
                fd1 = w1_'*x_; %función de discriminación 1 (línea verde)
                fd2 = w2_'*x_; %función de discriminación 2 (línea roja)
                if(fd1>0 && fd2>0) %si la región es la definida entre las 2 rectas pongo
a 1 ese bit de la máscara
                    processedImage(y1,x1) = 1;
                else %caso contrario, se deja a 0
                    processedImage(y1,x1) = 0;
                end
            end
        end
        subplot(3,2,6); imshow(processedImage.*rgb2gray(fileImageRGB),[]);
    else %caso de que no se halla detectado correctamente, mostrar toda la imagen
        subplot(3,2,6); imshow(rgb2gray(fileImageRGB),[]);
    end
        subplot(3,2,5); imshow(processedImage,[]);
    hold off

end
porcentajeFallos=(fallosTotales/530)*100 %mostrar al final de ejecución % de fallos a lo
largo de toda la secuencia de fallos
return;
```

Por último, se le muestra al usuario el porcentaje de fallos que se ha producido a lo largo de la secuencia de imágenes.

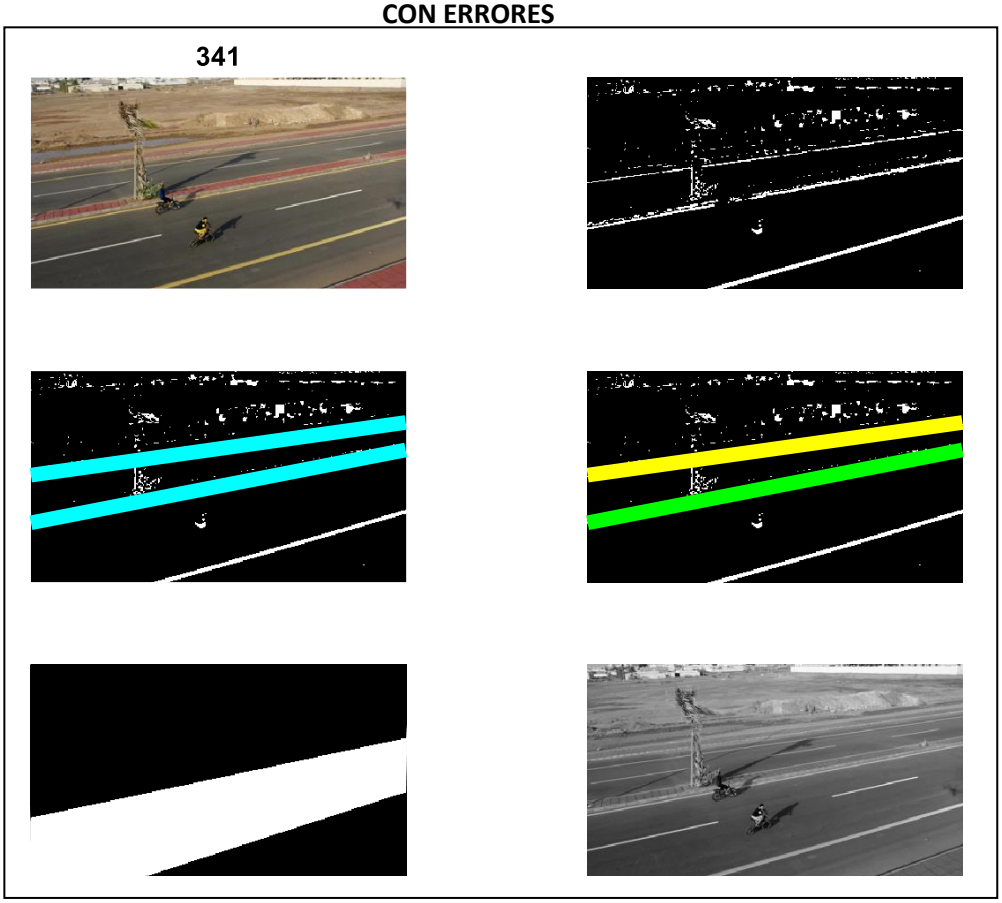
Se procede a mostrar los resultados obtenidos con alguna figura como ejemplo.

El porcentaje de fallos obtenidos para la secuencia de los ciclistas dada es: 1.3208%.

SIN ERRORES



CON ERRORES



VERSIÓN AVANZADA

Como mejora respecto a la parte básica explicada anteriormente, se procede a realizar un tratamiento de los fallos de forma transparente al usuario. Para ello, hemos realizado una serie de modificaciones respecto al código ya explicado que son las que se muestran.

Centrándonos dentro del bloque de código asociado a la transformada de Hough, en primer lugar, hemos eliminado el caso en el que al detectar que la linea recta no corresponde con la verde ni la roja se guardaban sus rho y theta asociados dentro de una tercera componente de los vectores *rhoRectasFotograma* y *thetaRectasFotograma.* De esta forma, al finalizar el bucle for que se muestra, dichos vectores tendrán dos componentes, asociadas a las rectas verde y roja respectivamente (en el caso de no existir fallo) o bien contendrán las rho y theta de la línea roja o verde mientras que la componente restante será -1 por defecto, valor al cual inicializamos los vectores.

Gracias a esto, sabemos por tanto si se ha dado un fallo (alguna de las componentes de *rhoRectasFotograma* y *thetaRectasFotograma son -1*) y cuál es la recta en cuestión (verde o roja, dependiendo de la posición en la que esté dicho -1) que falla.

TRANSFORMADA DE HOUGH

```
    (...)
```

```
for k=1:length(rho)
    if( abs(rho(k)-rhoG_ant)<20 && abs(theta(k)-thetaG_ant)<10 ) %verde
        rhoRectasFotograma(1)=rho(k);
        thetaRectasFotograma(1)=theta(k);
        rhoG_ant=rhoRectasFotograma(1);
        thetaG_ant=thetaRectasFotograma(1);
    elseif( abs(rho(k)-rhoR_ant)<20 && abs(theta(k)-thetaR_ant)<10 ) %rojo
        rhoRectasFotograma(2)=rho(k);
        thetaRectasFotograma(2)=theta(k);
        rhoR_ant=rhoRectasFotograma(2);
        thetaR_ant=thetaRectasFotograma(2);
    end
end
```

A la hora de calcular los puntos de intersección de las rectas con los límites de la imagen comprobamos previamente si tenemos para el fotograma en cuestión un fallo. La condición para que ello ocurra es que alguna componente del vector *thetaRectasFotograma* sea negativa, ya que como hemos visto, al darse un error se guardará como -1. En dicho caso, volvemos a recurrir a la función *houghpeaks()* con otros argumentos menos restrictivos para obtener un tercer pico de la matriz de votos. Las rho y theta correspondientes a este nuevo pico son en realidad las que deben ser sustituidas por las que se han detectado como fallo. Por ese motivo, sustituimos dichos valores en los vectores *rhoRectasFotograma* y *thetaRectasFotograma.* En este punto, dichos vectores contienen los parámetros correspondientes asociados a las rectas verde y roja para a partir de ellos poder obtener las intersecciones con los bordes de la imagen como se hacía en la parte básica. Además, es importante recordar el detalle de actualizar los valores de las rho y theta anteriores para el próximo fotograma.

```
%calcular puntos de intersección con bordes
%Orden cálculo intersecciones: recta HorizontalSuperior(HS), VerticalDerecha(VD)
% , HorizontalIzuquierda(HI), VerticalIzquierda(VI)
rhoMarco = [0,N,M,0];
thetaMarco = [90,0,90,0];
x=-ones(2,length(rhoRectasFotograma));y=-ones(2,length(rhoRectasFotograma));
for k=1:length(rhoRectasFotograma)

    if(thetaRectasFotograma(k)<0) %en caso de detección de línea roja
        % o verde fallida, se obtienen otra vez picos de Hough pero
        % esta vez cogiendo 3 con un área de vecindad menor
        peaks = houghpeaks(M_votes,3,"NHoodSize",[9 9]);
        rho = tabRho(peaks(3,1));
        theta = tabTheta(peaks(3,2));
        if(theta<0)
            theta=theta+180;
            rho=-rho;
        end
        rhoRectasFotograma(k)= rho;
        thetaRectasFotograma(k)= theta;
        if(k==1) %si la fallida era la verde, actualizar esta
            rhoG_ant = rhoRectasFotograma(k);
            thetaG_ant = thetaRectasFotograma(k);
        elseif(k==2) %si la fallida era la roja, actualizar esta
            rhoR_ant = rhoRectasFotograma(k);
            thetaR_ant = thetaRectasFotograma(k);
        end
    end
```

Por último, se ha eliminado el caso dentro de la sentencia switch-case asociado a representar la línea errónea en amarillo, ya que como hemos visto, ese error se ha corregido y es equivalente a como si nunca se hubiera dado dicho error. De esta forma es transparente al usuario.

```
        %pintar líneas en imagen binarizada de color azul cyan
        subplot(3,2,3)
        line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','c','LineWidth',5);

        %pintar líneas en imagen binarizada marcadas con su propio
        %color invariante a lo largo de los fotogramas
        subplot(3,2,4)
        switch k
            case 1 %Verde (línea interior de carretera)
                line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','g','LineWidth',5);
            case 2 %Rojo (línea exterior de carretera)
                line([x(1,k),x(2,k)],[y(1,k),y(2,k)],'Color','r','LineWidth',5);
        end
    end
```

Vemos a continuación los resultados obtenidos, correspondientes al mismo fotograma mostrado anteriormente donde se daba la detección de una línea incorrecta. Podemos ver como ahora ya se ha corregido, las líneas verde y roja son las correctas, y se representa en la imagen final únicamente la zona de la calzada.

CORRECCIÓN DE ERRORES