Marcos Nieto's Blog
Computer vision, research and more!

Lane markings detection and vanishing point detection with OpenCV

Posted on December 27, 2011 by marcosnietodoncel

Hi guys! The vanishing point detection topic has occupied a good part of my (research) life. Indeed I spent quite a long time to finish my PhD whose title was "Detection and tracking of vanishing points in dynamic environments".

In this post I would like to show a simple yet robust solution for the detection of a single vanishing point in road scenes.

Original grayscale image

The vanishing point in this scenario can be very usefull to retrieve the camera calibration, to perform some planar homography transformation, to determine a ROI inside the image, etc. Although there are several vanishing points defined by the elements of this scenario (the vertical and horizontal directions of the panels), we want to focus on the vanishing point defined by the lane markings. Note that for curvy roads, the vanishing point does not exist, although you can thought of it as the direction of the tangent on the car position on the curve.

For that reason, we first need to extract the lane markings, which can be done in many, many different ways (thresholding the intensity, connected components, edges, etc). In this post I share one of the fastest I've used in my works. It is published in my Springer MVAP paper "Road environment modeling using robust perspective analysis and recursive Bayesian segmentation", and the code in C++/OpenCV I share here (sorry it's an image because the <code> </code> html commands seem not to work fine in WordPress):

```
void laneMarkingsDetector(cv::Mat &srcGRAY, cv::Mat &dstGRAY, int tau)
{
    dstGRAY.setTo(0);

    int aux = 0;
    for(int j=0; j<srcGRAY.rows; ++j)
    {
        unsigned char *ptRowSrc = srcGRAY.ptr<uchar>(j);
        unsigned char *ptRowDst = dstGRAY.ptr<uchar>(j);

        for(int i=tau; i<srcGRAY.cols - tau; ++i)
        {
            if( ptRowSrc[i]!= 0)
            {
                aux = 2*ptRowSrc[i];
                aux += -ptRowSrc[i-tau];
                aux += -ptRowSrc[i+tau];
                aux += -abs((int)(ptRowSrc[i-tau] - ptRowSrc[i+tau]));

                aux = (aux<0)?(0):(aux);
                aux = (aux>255)?(255):(aux);

                ptRowDst[i] = (unsigned char)aux;
            }
        }
    }
}
```

Applying this filter we get images like the following (note that I have set to black the upper half of the image), where tau is the expected width (in pixels) of the lane markings. For a better performance, this value can be adapted to the perspective of the road, although for this special case this is exactly what we do not have!):

Detected lane markings

After a proper thresholding we can get something like this:

Binarized lane markings

Although we do have a lot of false positive pixels (the vehicle or lateral elements of the scenario), the following robust stages will find the correct vanishing point.

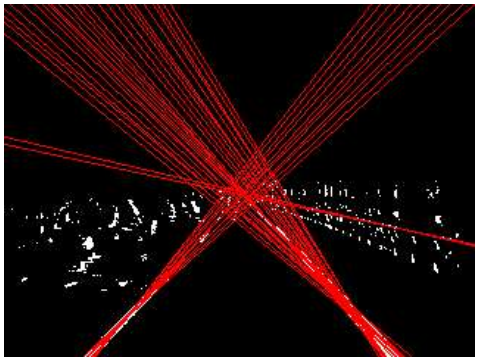
Using OpenCV, I have found that a quite reliable solution is based on (i) the use of the Hough transform, and (ii) the computation of the intersection of the lines we get.

For the first part, OpenCV has two main options, the Standard Hough Transform (SHT), and the Progressive Probabilistic Hough Transform (PPHT). I use the first because it returns lines and not pairs of points or line segments and although it is a little bit slower, it requires the user to set less parameters and it works fine in most cases. The Hough transform can be applied as:

```
// USE STANDARD HOUGH TRANSFORM
vector<Vec2f> lines_;
HoughLines( __lmGRAY, lines_, 1, CV_PI/180, __houghMinLength );

for( size_t i = 0; i < lines_.size(); i++ )
{
    float rho = lines_[i][0];
    float theta = lines_[i][1];

    double a = cos(theta), b = sin(theta);
    double x0 = a*rho, y0 = b*rho;


    Point pt1(cvRound(x0 + 1000*(-b)),
              cvRound(y0 + 1000*(a)));
    Point pt2(cvRound(x0 - 1000*(-b)),
              cvRound(y0 - 1000*(a)));

    cv::clipLine(srcGRAY.size(), pt1, pt2);

    if(!dstBGR.empty())
        line( dstBGR, pt1, pt2, Scalar(0,0,255), 1, 8 );

    cv::imwrite("HOUGH.bmp", dstBGR);
}
```

Where __lmGRAY is the image we obtain from laneMarkingsDetector, and __houghMinLength is the minimum length we require (it should be set according to the image dimensions, something like 30 should work for small images 320 x 240).

The result is a set of lines that visually converge on a small region of the image:



Detected Hough lines

In this simple case there are no strong outliers, i.e. lines that clearly do not intersect at the vanishing point, although we do have a non-negligible intersection error. For cases like this or with more outliers, we can use a RANSAC-like method to find the more likely vanishing point.

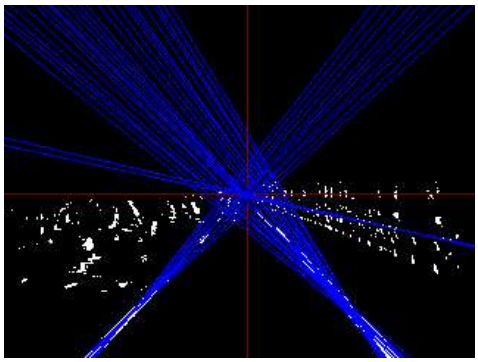
(UPDATE: The MSAC class is no longer available as it was, instead, you can download the new MSAC class with a full sample capturing images or video and computing as many vanishing point as desired, both finite or infinite. Please refer to the specific post for more details).

For that purpose I use a variation of RANSAC called MSAC which simply weights inliers according to their cost function (instead of just counting 1 for inliers and 0 for outliers as RANSAC does). I have programmed a very simple version of it, in a C++ class, which only needs two steps:

```
// Initialization

__msac.init(IMG_WIDTH, IMG_HEIGHT, 1);

// Execution (passing as argument the lines obtained with the Hough transform)

__msac.singleVPEstimation(lines, &number_of_inliers, vanishing_point);
```

Where ___msac is an object of class MSAC, number_of_inliers is an output int that contains the number of inliers MSAC has found to compute the vanishing_point (if you want to play with this, you can go to the Code page from my website although it is not optimized nor commented).

The result is normally a good vanishing point (I have tested it in many, many type of road sequences, and it works fine as long as there are some painted lane markings).



Detected vanishing point

Additionally, I usually compute the vanishing point for a set of time instants, and check if the vanishing point is coherent and steady in time. In case not, I restart the procedure until I find something reliable.

That's all for today!

This entry was posted in Computer vision, OpenCV and tagged C++, Hough, lane markings, MSAC, OpenCV, RANSAC, source code, vanishing point. Bookmark the permalink.

46 Responses to *Lane markings detection and vanishing point detection with OpenCV*

Omkar Kulkarni *says:*
January 10, 2012 at 13:32

here is the main function where i retrieve frames and display...before using msac object for class msac i sent these frames to the two functions:
int main(int argc , char** argv)
{

IplImage* frame;
IplImage * histImage;
float ranges[]={0,255};
float* Range[1]={&ranges[0]};
int Bin=256;

CvCapture* capture=cvCreateFileCapture("test.avi");
cvNamedWindow("Video",1);
cvNamedWindow("Histogram", 1);

cout<<capture<<endl;
if(capture==NULL)

```
{
cout<<"NO capture"<origin = 1;

for(int i=0;i<Bin;i++)
{
cvLine(histImage,cvPoint(i,0),cvPoint(i,int(cvQueryHistValue_1D(R_hist,i)/50)),CV_RGB(i,0,0));
cvLine(histImage,cvPoint(i,150),cvPoint(i,int(cvQueryHistValue_1D(G_hist,i)/50)+150),CV_RGB(0,i,0));
cvLine(histImage,cvPoint(i,300),cvPoint(i,int(cvQueryHistValue_1D(B_hist,i)/50)+300),CV_RGB(0,0,i));
cvLine(histImage,cvPoint(i,450),cvPoint(i,int(cvQueryHistValue_1D(Grey_hist,i)/50)+450),CV_RGB(i,i,i));
}

cvShowImage("Video",frame);
cvShowImage( "Histogram", histImage );

char c=cvWaitKey(33);
if(c==27) break;
}

cvReleaseCapture(&capture);
cvReleaseImage(&frame);
cvReleaseImage(&histImage);
cvDestroyWindow("video");
cvDestroyWindow("Histogram");

return 1;

}
```

Reply

Omkar Kulkarni *says:*
January 10, 2012 at 13:38

I am sorry!...the full main function isnt displayed! this is the main function where i am retrieving frames:

```
#include “stdafx.h”

#include “cv.h”
#include “highgui.h”

int main(int argc,char** argv)
{
cvNamedWindow(“Canny Edges”,CV_WINDOW_AUTOSIZE);
cvNamedWindow(“Original View”,CV_WINDOW_AUTOSIZE);

CvCapture* capture;
if(argc==0)
{
capture=cvCreateCameraCapture(0);
}
else
{
capture=cvCreateFileCapture(“new.avi”);
}

IplImage* frameRGB;

while(1)
{
frameRGB=cvQueryFrame(capture);
```

```
/*IplImage* frameG=cvCreateImage(cvGetSize(frameRGB),IPL_DEPTH_8U,0);
cvConvertImage(frameRGB,frameG,0);

CvSize sz=cvGetSize(frameRGB);
sz.width=sz.width;
sz.height=sz.height;

IplImage* frameRG=cvCreateImage(sz,IPL_DEPTH_8U,0);
cvResize(frameG, frameRG,CV_INTER_LINEAR);

IplImage* framec=cvCreateImage(sz,IPL_DEPTH_8U,0);

cvCanny(frameRG, framec, 50, 5, 3);*/
if(!frameRGB)break;
cvShowImage("Original View",frameRGB);
/*cvShowImage("Canny Edges",framec);*/

char c=cvWaitKey(33);
if(c==27)break;

}
cvReleaseCapture(&capture);
cvDestroyWindow("Original View");
/* cvDestroyWindow("Canny Edges");*/
return 0;
}
```

here where should i add the calls using object __msac?? i am not getting which variables to pass too! plz plz help!

Reply

marcosnietodoncel *says:*
January 10, 2012 at 14:08

Hi!
You can use the Hough transform to get the lines, for instance after applying the Canny edge detector as you are doing now.
The piece of code is in image in this post where it reads // USE STANDARD HOUGH TRANSFORM, you only have to substitute __lmGRAY which is the image I used for the one you are using, framec, and you will get a set of lines in the variable vector lines_.
After that you should convert vector lines_ into vector < vector < Vec2f > > lines before passing it to the MSAC object, for instance like this:

```
vector < vector < Point > > lines;
vector < Point > aux;
for(size_t i=0; i < lines_.size(); ++i)
{
aux.clear();
// Get the two end-points of current line segment
float rho = lines_[i][0];
float theta = lines_[i][1];

double a = cos(theta), b = sin(theta);
double x0 = a*rho, y0 = b*rho;

CvPoint pt1, pt2;
pt1.x = cvRound(x0 + 1000*(-b));
pt1.y = cvRound(y0 + 1000*(a));
pt2.x = cvRound(x0 – 1000*(-b));
pt2.y = cvRound(y0 – 1000*(a));
```

```
aux.push_back(pt1);
aux.push_back(pt2);
lines.push_back(aux);
}
}
```

Then you can call the __msac like:

```
int number_of_inliers = 0;
CvMat *vanishing_point = cvCreateMat(3,1,CV_32F);
__msac.singleVPEstimation(lines, *number_of_inliers, vanishing_point);
```

The resulting vanishing_point is in homogeneous coordinates, so make sure the third coordinate is one!
Do not forget to create and init the the MSAC object before!

Omkar Kulkarni *says:*
January 10, 2012 at 17:19

So,after deliberate efforts, i came up with the main(), still giving lots of error functions, plz Sir rectify if any!

```
#include “stdafx.h”
#include “cv.h”
#include “highgui.h”

int _tmain(int argc, _TCHAR* argv[])
{
cvNamedWindow(“Canny Edges”,CV_WINDOW_AUTOSIZE);
cvNamedWindow(“Original View”,CV_WINDOW_AUTOSIZE);

CvCapture* capture;
if(argc==1)
{
capture=cvCreateCameraCapture(0);
}
else
{
capture=cvCreateFileCapture(“new.avi”);
}

IplImage* frameRGB;

while(1)
{
frameRGB=cvQueryFrame(capture);

IplImage* frameG=cvCreateImage(cvGetSize(frameRGB),IPL_DEPTH_8U,0);
cvConvertImage(frameRGB,frameG,0);

CvSize sz=cvGetSize(frameRGB);
sz.width=sz.width;
sz.height=sz.height;

IplImage* frameRG=cvCreateImage(sz,IPL_DEPTH_8U,0);
cvResize(frameG, frameRG,CV_INTER_LINEAR);

IplImage* __lmGRAY=cvCreateImage(sz,IPL_DEPTH_8U,0);

cvCanny(frameRG, __lmGRAY, 50, 5, 3);
if(!frameRGB)break;
```

```
vector lines_;
HoughLines(__lmGRAY,lines_,CV_PI/180,__houghMinLength);
for(size_t i=0;i<lines_.size();i++)
{
float rho=lines_[i][0];
float theta=lines_[i][l];
double a=cos(theta),b=sin(theta);
double x0=a*rho,y0=b*rho;
Point pt1(cvRound(x0+1000*(-b)),cvRound(y0+1000*(a)));
Point pt1(cvRound(x0-1000*(-b)),cvRound(y0-1000*(a)));
cv::clipLine(srcGRAY.size(),pt1,pt2);
if(!dstBGR.empty())
line(dstBGR,pt1,pt2,Scalar(0,0,255),1,8);
cv::imwrite("HOUGH.bmp",dstBGR);
}

vector < vector > lines;
vector aux;
for(size_t i=0; i < lines_.size(); ++i)
{
aux.clear();
// Get the two end-points of current line segment
float rho = lines_[i][0];
float theta = lines_[i][1];

double a = cos(theta), b = sin(theta);
double x0 = a*rho, y0 = b*rho;

CvPoint pt1, pt2;
pt1.x = cvRound(x0 + 1000*(-b));
pt1.y = cvRound(y0 + 1000*(a));
pt2.x = cvRound(x0 – 1000*(-b));
pt2.y = cvRound(y0 – 1000*(a));

aux.push_back(pt1);
aux.push_back(pt2);
lines.push_back(aux);
}
int number_of_inliers = 0;
CvMat *vanishing_point = cvCreateMat(3,1,CV_32F);
__msac.singleVPEstimation(lines, *number_of_inliers, vanishing_point);

cvShowImage("Original View",frameRGB);
cvShowImage("Canny Edges",framec);

char c=cvWaitKey(33);
if(c==27)break;

}
cvReleaseCapture(&capture);
cvDestroyWindow("Original View");
cvDestroyWindow("Canny Edges");
return 0;
}
```

Reply

Omkar Kulkarni *says:*

January 11, 2012 at 11:12

Sir, i am waiting for ur reply...thank you!

Reply

qurban124 *says:*
March 4, 2012 at 21:44

please send me complete code for road detection in opencv

Reply

marcosnietodoncel *says:*
March 4, 2012 at 23:46

Hi!
For the moment I can't share more than I currently do. Anyway, please keep visiting the blog because I will probably add more hints and useful code examples in the future.
Regards,

Marcos

Reply

spring *says:*
March 6, 2012 at 09:37

Hi,
Thanks a lot for your posting, if u can share your complete code it will be excellent
Thank u again
Good luck

dp *says:*
March 5, 2012 at 04:55

Hey, just wanted to say thanks for posting this. Its hard to find examples of this kind of stuff on the net, so thanks for sharing. I'm working on an small scale autonomous car and was trying to figure out whats the next step after hough transform. This answers my question, thanks!

Reply

spring *says:*
March 6, 2012 at 09:31

Hi,
I 'm working on the same field too, can u help me? I'm in the beginning of this

Reply

Vitamin A *says:*
April 9, 2012 at 23:33

Greeting from across the sea. excellent blog I shall return for more.

Reply

Marek *says:*
May 18, 2012 at 02:03

this laneMarkingsDetector, doesnt work for me.... after running, error apears every sing time :/

Reply

marcosnietodoncel *says:*
May 18, 2012 at 21:49

Send me or post your code if you want, otherwise, I cannot help :)

Reply

Marek *says:*
May 26, 2012 at 15:38

It was my mistake. I fixed it ;)
Recently i read your publication “Road environment modeling using robust perspective analysis and recursive Bayesian segmentation” and according to it i was trying to write my own lane markings detection algorithm.

I Use:
1) Inverse PerspectiveMapping
2) Sobel edge detector
3) then i’m binarizing the image
4) and use my algorithm to detect road stripes (only vertical like ones)

this is one of the images i worked on:
Before:
http://imageshack.us/photo/my-images/109/83464702.jpg/
and After:
http://imageshack.us/photo/my-images/403/zapisz1.jpg/
http://imageshack.us/photo/my-images/836/zapisz2.jpg/

I would like to ask you, as a specialist, if you could give me some tips to help me find the best way to write an algorithm for Lane tracking like this one, you presented on YT:

I am beginner in working with Opencv (i downloaded it 2 weeks ago), so I would be very thankful for any help. This is very important for me cause it is a part of work at my university to build an autonomous cat.

best regards
Marek Kotewicz

Marek *says:*
May 26, 2012 at 15:43

car*, not cat* :)

marcosnietodoncel *says:*
May 28, 2012 at 08:35

Hi Marek,

Good job! It looks you're doing the right way.
Nevertheless, if you are planning to jump into an autonomous vehicle, you should consider the extra difficulties a real scenario poses. The first one (and the most significant for the inverse perspective mapping) is the vibration of the camera, and the motion of the vehicle. This will make your IPM to be very unsteady. Typically you can only trust to have reliable straight-vertical lines in the very close distance.
Other aspects to consider: absence of lane markings for a while, rain, shadows, occlusions due to other vehicles...
Welcome to the road environment!

Regards,

Marcos

hb2012 *says:*
June 13, 2012 at 15:21

slt mes amis,
j'ai l'honneur je vous contacter une autre fois dans ce fameux forum,bon j'ai besoin d'une petit correction au niveau ce code en opencv
hello marcos,
well, I need a small correction to this code in opencv:

```
CvPoint meas_x1,meas_y1,cord_x1,cord_y1;
cord_x1.x=230;
cord_x1.y=100;
cord_y1.x=550;
cord_y1.y=500;
for (int l=0;l<10;l++)
{
std::string varimg;
char format[] = "franck_000%d.jpg";
char filename[sizeof format+100];
sprintf(filename,format,l);
varimg = filename ;
IplImage*imgw = cvLoadImage( varimg.c_str() );
cvNamedWindow( "Example1", CV_WINDOW_AUTOSIZE );
meas_x1.x=cord_x1.x;
meas_x1.y=cord_x1.y;
meas_y1.x=cord_y1.x;
meas_y1.y=cord_y1.y;
```

it is a part of the main program, but the most important is that you can help me to get a solution allows me to access the next position of the object to follow that is the measure of each point varies as they are developing a rectangle.this code used to display a series of images (though making a sequence) and in these images there is an object (eg the face of a person) that makes the mouvement.In this case, I just measured the new position of this object so I do the tracking of this object. In the first place I framed by a rectangle object and consequently I have (hopefully) receive each time the measurement of the position of the object to correct it.

Reply

marcosnietodoncel *says:*
June 13, 2012 at 15:58

Hi,

I am sorry I can not really understand what's the goal of your program (by the way, unless you post the complete code I can not figure out where the problem can be).
Nevertheless, you talk about tracking an object, such as a face. In case so, OpenCV has a lot of tools for that. For instance, there are a couple of samples about face detection with cascade classifiers, and some tutorials have been published as well (http://opencv.itseez.com/doc/tutorials/objdetect/cascade_classifier/cascade_classifier.html#cascade-classifier).
If you are thinking more on the tracking process, maybe you can use Camshift (http://opencv.itseez.com/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=camshift#cv.CamShift), or apply template matching with Kalman filter (http://opencv.itseez.com/modules/imgproc/doc/object_detection.html?highlight=matchtemplate#cv.MatchTemplate, http://opencv.itseez.com/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=kalman#KalmanFilter).

In summary, I recommend you to download the OpenCV from the svn and compile the samples. Play one or two related to the topics I've mentioned and probably you can find the solution.

Best regards,

Marcos

Reply

---

hb2012 *says:*
June 13, 2012 at 18:13

thank you for this earlier answer,but the goal of my project is use the opencv only with a simple fonction to realize a tracking object with kalman filter .For this i haven't used this fonction predefined in opencv from kamlan filter because i have a some image to configure at a sequence for tracking object. Therefore i must work with the exact place of subject .

Reply

---

ardi *says:*
July 20, 2012 at 07:27

Hi Mr Marcos..

I have sent you an email, please reply my email.

Reply

marcosnietodoncel *says:*
July 20, 2012 at 07:33

Done :)

Reply

---

ardi *says:*
July 20, 2012 at 08:59

Thanks for your reply.. :)
I ask you via email, please check it sir.

Reply

---

nehad *says:*
December 19, 2012 at 05:19

I just want to ask if this code can use with a maps image or no?

Reply

marcosnietodoncel *says:*
December 19, 2012 at 09:12

Hi!
What do you mean with a “maps image”?
Actually the code can be used with images as cv::Mat objects, type CV_8U.
Best Regards,
Marcos

Reply

Nehad *says:*
February 14, 2013 at 17:23

hi Marco,
I am looking for complete code of lane marking road detection.I need it urgently.Can you please provide me. I will be thankful to you.

Reply

Pingback: *A First Attempt at Lane Detection | Jay Chakravarty*

Asif *says:*
May 9, 2014 at 11:53

Can you share the link of sample video for lane marking detection?

Reply

marcosnietodoncel *says:*
June 1, 2014 at 10:31

Hi!
I am sorry I can not distribute video sources since they do not belong to me.
However, you can probably find road videos elsewhere.
Kind regards,

Marcos

Reply

Soju T Varghese *says:*
June 18, 2014 at 14:49

Hello, i have got a working code for lane and vehicle detection in opencv(version 2.3, based on C). Everything is fine, excpet that in the lane detection output window, the lanes detected get overlayed over the previous, thus filling the window with lines, subsequently. I do not know how to delete the previous drawn lane-lines from the window. The normal cvLine() function is used for drawing the lane lines. Your help would be highly appreciated.

Regards,

Soju.

Reply

marcosnietodoncel *says:*
June 18, 2014 at 18:54

Hi!
You probably just need to reset the image on which you are drawing using something like:

```
image.setTo( 0 );
```

Assuming that image is a cv::Mat and that has been created before that.
Kind regards,
Marcos

Reply

Ivan *says:*
March 2, 2015 at 09:59

I want to use RANSAC to detect a line? Do you have any source code to share?

regards,
Ivan

Reply

marcosnietodoncel *says:*
March 7, 2015 at 10:56

Hi,
I think so!
The vanishing point detection project (https://sourceforge.net/projects/vanishingpoint/) includes a basic RANSAC implementation.
Kind regards,
Marcos

Reply

Ravichandra *says:*
March 4, 2015 at 07:41

Hi sir..
I am using the same lane detection concept in vision based aircraft runway detection for auto landing system.
Please send the code of this project..
It may helps lot.
Thanks in advance...

Reply

marcosnietodoncel *says:*
March 7, 2015 at 10:53

Hi,
Unfortunately, I don't have an entire sample for lane detection available to share. Moreover, I don't think it would be useful in your case, since aircrafts have more degree of freedom than cars, so the assumptions I use for lane detection will not hold (basically, constant roll angle and preferrably equal to zero).
Regards,
Marcos

Reply

Ravi *says:*
March 12, 2015 at 09:59

Yes sir.
The roll and pitch can be controlled by the Horizon Detection based on the Vanishing point i can adjust the roll by height of the vanishing point

Moveh *says:*
June 27, 2016 at 04:19

Hi
this is a brilliant work, although i am working on a different platform as a beginner and i was a bit confused when i was going through the way you applied your hough transform.I am presently working on lane detection using matlab as my image processing platform...i have presently captured and processed the image by simply capturing,converting to grayscale and applying edge detectors operators and i am presently stuck on applying the houghline so as to get my lane boundries....i would really appreciate your guidance in this please. below are the image processing techniques i used

```
a = imread('roadlane.jpg');
a = rgb2gray(a);
imshow(a)
% h = [1 1 1; 1 1 1; 1 1 1]/9;
% c = imfilter(a,h);
% % imshow(c);
% for sobel edge detector
% h = [1 0 -1; 2 0 -2; 1 0 -1];
% c = imfilter(a,h);
% imshow(c);
c = edge(a);
imshow(c);
% for canny edge detector
c = edge(a,'canny');
imshow(c);
```

Reply

marcosnietodoncel *says:*
July 27, 2016 at 11:29

Hi Moveh,

Apologies for answering this late, I was terribly busy with other duties.
So, in my opinion your idea is just fine. You can use any type of lane marking detector, such as the Canny or Edge detectors in Matlab, because at the end, what you need is a set of points in the image that belong to the lane markings. The Hough transform takes these points and finds the dominant lines (as clusters of points). Matlab comes with a nice set of Hough implementations, you have probably already found a solution for this.

Regards,

Marcos

Reply

Moveh Samuel *says:*
July 27, 2016 at 11:51

Hi
I really appreciate your reply,although as u mentioned i have found a way out after tresholding the image and applied d hough transform and obtained an excellent result.i am presently trying to analyze which of the edge detector is more suitable.
Thanks and best regards.

marcosnietodoncel *says:*
August 18, 2016 at 09:16

Great!
Sometimes it is a good idea to keep using simpler edge detectors (e.g. Sobel), which may run faster afterwards if migrated to embedded platforms.
Regards,
Marcos

Anuj *says:*
July 26, 2016 at 20:44

Hello,
I saw your video on lane tracking and was simply amazed by it. I have just started of with my project which involves lane detection and tracking. Just to begin with it, what I have done is the following:

– For each incoming frame from a video or a camera placed on a moving vehicle, I detect edges, apply Hough Transform (OpenCV implementation) to get a bunch of lines, filter out lines based on the slope criteria to get only those lines which corresponds to the lane.
– Now, in many of the frames, I am not getting the hough lines so I applied Kalman Filter but it is not working.
– I take out the slope and intercept of the line and model them as my state vector, I use a 2×2 identity matrix as the state transition matrix (F). Why I am taking it as an identity matrix is because I don't see this model as a constant velocity model. But I know that Kalman filter will only be able to predict the new state of the system if we assume that it is a constant velocity or a constant acceleration model.

So all together, I am not able to think how should I model my state vector and transition matrix when I have to track lanes based on hough lines. The only property of the these hough lines I can think about are the slope and intercept which somewhat change with each frame but that change is not constant.
Could you please suggest me the steps how to go about lane tracking?

Reply

marcosnietodoncel *says:*
July 27, 2016 at 11:26

Hi Anuj,

Your approach is correct, although you may need to better fine tune your parameters. The road scenario is vastly dynamic which makes fixed thresholds and assumptions to hold only for certain situations or during small periods of time.
For instance, you are using edge detection and Hough transform. Fine, but probably you need to dynamically adjust the parameters as the scene evolves (what happens if the road is suddenly not well painted? or if you enter a tunnel?).
Once you have your lane markings detected (as Hough lines or other type of detections), you probably want to fit a lane model. In my case I use multi-lanes and parabolic fitting in the bird's-eye view. Of course, this is up to you. The simplest approach is to model a single lane, without curvature, which can be basically be defined by a fixed vanishing point (that you can compute at the beginning and keep it fixed, or update it online with the observations) and two points at the bottom of the image.

Then you can apply a Kalman filter to provide smoothness to your tracking. Using constant-velocity is probably a good option.
All I can say is that if you plan to use this in a real environment, you probably also want to avoid costly operations (to run your sw in embedded platforms) such as the Hough transform, and also avoid OpenCV implementations, which are great, but general.

Good luck with your project!

Marcos

Reply

strangelyhuman *says:*
November 9, 2016 at 02:11

stumbled onto your post while I was researching the Vanishing point. Helped a lot. Thanks a ton!

Reply

Ian Min *says:*
March 14, 2017 at 18:14

Hi Marco!

Out of curiosity, can it detect yellow line?

I think it'll get too much noise when you try to get a yellow line with this algorithm

Sorry if it sounds aggressive. Thank you :)

Reply

marcosnietodoncel *says:*
May 24, 2017 at 07:50

Hi,
Thanks for your comment, it's all right!
The algorithm is designed to detect high contrast at row level. It works on grayscale images. Therefore, if the yellow line is well contrasted against the pavement, it will probably look bright in the grayscale image, and the system will work.
But, if the lines are not well painted, no matter what color they have, the system won't detect them!
Regards,
Marcos

Reply

Amol vagad *says:*
May 22, 2017 at 18:24

Hi Marcos,

I am also working on lane detection currently. I am successfully detecting the lanes. The last thing I want to do is fill the space between the lanes with a color. I am using probabilistic hough transform to detect the lanes. I tried the opencv fillpoly function but had no success with it. Can you please guide me for the same ?
Thanks

Reply

marcosnietodoncel *says:*
May 24, 2017 at 07:47

Hi,
Once you have your lane markings detected with Hough, you need one extra step to create (and track if you wish) a polygon from them. Normally you should be able to locate the vanishing point as the point where the lines meet (probably you need to stabilize that with a Kalman filter). Then, you can select some row below the vanishing point and cut your lines there, and another cut at the bottom row. Then you have 4 points you can fill in with color.

Regards,
Marcos

Reply
