Homework 4

Lane Departure Warning System Implementation With Using OpenCV

Submitted by
Taylor Whitaker
Ying Fan
Md Jubaer Hossain Pantho
Pankaj Bhowmik

Objective:

The main object of this project is to :
-To implement a lane departure application using Hough transform functions of OpenCV.
-A warning is raised whenever the vehicle crosses one of the lines delimiting the lane
-Using the Hough Transform to identity the lines and write a procedure to recognize line crossing and produce warning.

Lane Departure Warning System:

Basic lane departure warning is implemented on vehicles driving in lanes separated by two continuous or interrupted lines. [1]In road-transport warning system is a mechanism designed to warn the driver when the vehicle begins to move out of its lane on freeways and arterial roads, these system are designed to minimize accidents by addressing the main cause of collisions: drive error, distractions and drowsiness.

Hough Transform:

[2]Is a feature extraction technique used in image analysis computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

System Overview:

AvitivityBOT360 + Raspberry Pi
[3]Contains multicore propeller control board, high speed servo motors with optical encoders for consistent maneuvers, as well as breadboard and 3-pin headers.
The Raspberry Pi is running the native OS, Raspbian, in order to host OpenCV and utilize other drivers for camera interaction and serial communication with the ActivityBot.

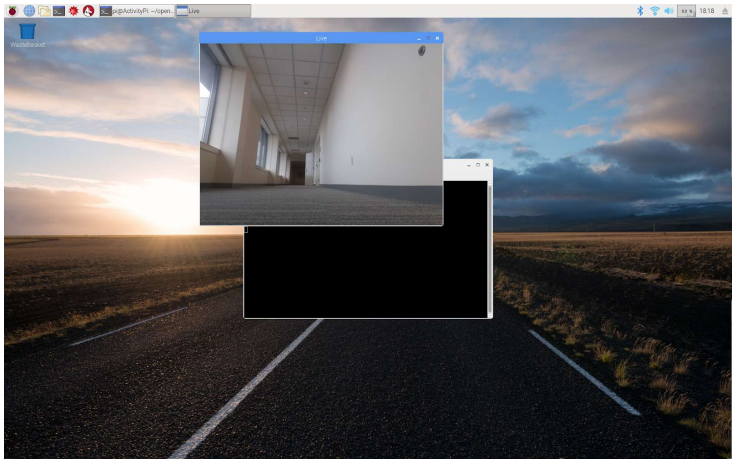

Figure 1: Raspberry Pi OpenCV Environment

Implementation:

First we use the contain lane departure video file to test if the edge detection work or not, this program include canopy algorithm and Hough transform function by using OpenCV. Video link provided in appendix A.

[4]The Hough transform is a global method help us to find the straight line that hidden in larger amount of other data. Each point in Hough space corresponds to a line at angle T and distance d from the origin in the original data space. The value of a function in Hough space gives the point density along a line in the data space. For each point in the original space consider all the lines which go through that point at a particular discrete set of angles, in our case, we want to detect two lines where the car is driving between. In order to do so, we first compute the value of the slope that the line might be, then narrow the range of the candidate lines. After which, we find the optimal range of the slope to detect. For each line we consider above, we increment a count in the Hough accumulator at point(d,T). After considering all the lines through all the points, a Hough accumulator with a high value will probably correspond to a line of points.

In the last step, instead of using video file as input for our program, we port our application to the Raspberry Pi camera attached to the ActivityBot and test on the track where we put on the white tape on the floor, imitating the conditions of a road.

Result:

By using the lane departure implementation we descripted above, we found an interesting result, which the car can do the self-correction(avoid the further approach to the lane). This signal us our program is working correctly for give the robot warning if the robot try not to stay on the line.
To prove our implementation, video file attached below Appendix B.

Graduate Section:

The graduate section was assigned an implementation of the Activity Bot - Raspberry Pi for following a wall and detecting doorways along the way. We utilized the original lane departure code with a simple modification to detect doorways. The added bit of code simply looks at the rightmost third of the camera feed and identifies vertical lines. The flow chart below demonstrates the algorithm we used to detect the doors.

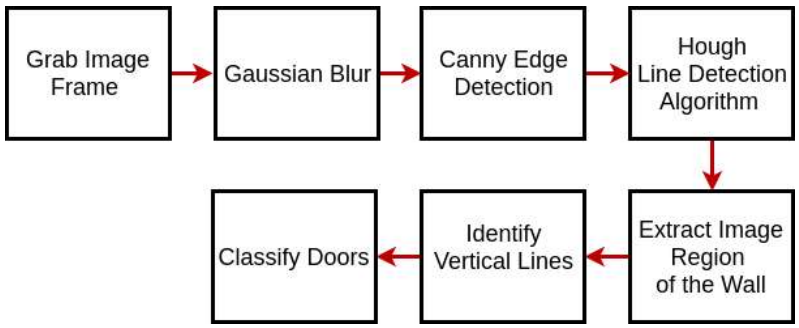

Figure 2: Flow Chart Describing Algorithm to Detect Doors

First, the pi on the activity bot grabs the video frame from the camera and perform gaussian blur. This step is important since it helped us remove majority of the noise from the image. Then we performed canny edge detection and hough transform to obtain the image with detected lines. In this test the wall was on our left. This means we only need to examine the left part of the image to detect doors. The algorithm extracts the image region where the wall resides. Afterwards, we examine the line equations within this region to identify vertical lines. We calculated the slope of the line equation to perform this task. A set of vertical lines within a small region of the wall was identified as doors.

Conclusion:

A Lane Departure warning system is an important mechanism for safe road transportation. It gives the warning for the car/driver when moving out of its lane, on freeways and arterial roads. It not only helps the driver to correct behavior when it is making an error, but also minimizes the accidents by addressing the causes of collisions. Hough transform and canny algorithm help us to finding straight lines and detect the edge in large amounts of visual data. In our example, the environment we have to test has large amounts of gaussian noise, potentially caused by the carpet. By including a Gaussian blur to our camera's output, we improved the accuracy for our detection test. Our experimental result show that we have successfully developed the system to help vehicles remain in their lane.

Appendix A: Test Video Files

1. https://drive.google.com/open?id=1t1SwX2wsAxUPYmH3yhlVxcaF3827nxll

Appendix B: Lane Departure and Wall Following Demonstrations

1. https://drive.google.com/open?id=1t1SwX2wsAxUPYmH3yhlVxcaF3827nxll
2. https://drive.google.com/open?id=0B4YGp5aFXSX2czBXZ0tpZTU5ajg3T1BSUS1TUjRrNlNZQnZR

Appendix C: C++ Code for Lane Departure Implementation

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "iostream"
#include <opencv2/opencv.hpp>
//#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "iostream"
#include <time.h>

using namespace std;
using namespace cv;

double tStart=clock();
void help()
{
 cout << "\nThis program demonstrates line finding with the Hough transform.\n"
        "Usage:\n"
        "./houghlines <image_name>, Default is pic1.jpg\n" << endl;
}
int main( int argc, char** argv )
{
 clock_t tStart = clock();
 Mat src;
 cout << "\nThis program demonstrates line finding with the Hough transform.\n";
      // Open a video file:
      cv::VideoCapture cap(0);
      if (cap.isOpened() == false)  //  To check if object was associated to webcam successfully
      {
       std::cout << "error: Webcam connect unsuccessful\n"; // if not then print error message
       return(0);            // and exit program
      }
 cv::Mat dst, cdst, image2;
 GaussianBlur( image, image2, Size( 3, 3 ), 3, 0 );
 Canny(image2, dst, 80, 130, 3);
 cvtColor(dst, cdst, COLOR_GRAY2BGR);

 #if 0
  vector<Vec2f> lines;
```

```
 HoughLines(dst, lines, 1, CV_PI/180, 100, 0, 0 );

 for( size_t i = 0; i < lines.size(); i++ )
 {
   float rho = lines[i][0], theta = lines[i][1];
   Point pt1, pt2;
   double a = cos(theta), b = sin(theta);
   double x0 = a*rho, y0 = b*rho;
   pt1.x = cvRound(x0 + 1000*(-b));
   pt1.y = cvRound(y0 + 1000*(a));
   pt2.x = cvRound(x0 - 1000*(-b));
   pt2.y = cvRound(y0 - 1000*(a));
   line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
 }
#else
 vector<Vec4i> lines;
 HoughLinesP(dst, lines, 1, CV_PI/180, 50, 50, 15 );
for( size_t i = 0; i <lines.size(); i++ )
 {
 cout << lines.size()<<"\n";
Vec4i l = lines[i];
float slope= abs((float)((l[1]-l[3])/(float)(l[0]-l[2])));
if((slope >0.3))
line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,255,0), 3, CV_AA);
else
{
            line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV_AA);
            std::cout <<"slop:" << slope << "\n";
}
 }
#endif
// imshow("source", dst);
 imshow("detected lines", cdst);

 for( int i = width/3; i <(width*2)/3; i++ ){
            for (int j = height*0.9; j <height; j++) {
                        Vec3b color = cdst.at<Vec3b>(Point(i,j));
                         if(color[1] =255){
                                    std::cout <<"Lane departer";
                          }
              }
 }
 // Save frame to video
        out << image;
        //cv::imshow("Modified video", image);
        // Stop the camera if the user presses the "ESC" key
        if(cv::waitKey(1000.0/FPS) == 27) break;
        waitKey(200);
    }
 return 0;
```

```
}
```

Appendix D: C++ Code for Wall Following Implementation

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "iostream"
#include <opencv2/opencv.hpp>
//#include "opencv2/imgcodecs.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "iostream"
#include <time.h>

using namespace std;
using namespace cv;

double tStart=clock();
void help()
{
 cout << "\nThis program demonstrates line finding with the Hough transform.\n"
        "Usage:\n"
        "./houghlines <image_name>, Default is pic1.jpg\n" << endl;
}
int main( int argc, char** argv )
{
 clock_t tStart = clock();
 Mat src;
 cout << "\nThis program demonstrates line finding with the Hough transform.\n";
      // Open a video file:
      cv::VideoCapture cap(0);
      if (cap.isOpened() == false)  //  To check if object was associated to webcam successfully
      {
       std::cout << "error: Webcam connect unsuccessful\n"; // if not then print error message
       return(0);            // and exit program
      }
 cv::Mat dst, cdst, image2;
 GaussianBlur( image, image2, Size( 3, 3 ), 3, 0 );
 Canny(image2, dst, 80, 130, 3);
 cvtColor(dst, cdst, COLOR_GRAY2BGR);

 #if 0
  vector<Vec2f> lines;
  HoughLines(dst, lines, 1, CV_PI/180, 100, 0, 0 );

  for( size_t i = 0; i < lines.size(); i++ )
  {
    float rho = lines[i][0], theta = lines[i][1];
    Point pt1, pt2;
```

```
    double a = cos(theta), b = sin(theta);
    double x0 = a*rho, y0 = b*rho;
    pt1.x = cvRound(x0 + 1000*(-b));
    pt1.y = cvRound(y0 + 1000*(a));
    pt2.x = cvRound(x0 - 1000*(-b));
    pt2.y = cvRound(y0 - 1000*(a));
    line( cdst, pt1, pt2, Scalar(0,0,255), 3, LINE_AA);
  }
 #else
  vector<Vec4i> lines;
  HoughLinesP(dst, lines, 1, CV_PI/180, 50, 50, 15 );
 for( size_t i = 0; i <lines.size(); i++ )
  {
  cout << lines.size()<<"\n";
Vec4i l = lines[i];
float slope= abs((float)((l[1]-l[3])/(float)(l[0]-l[2])));
if((slope >0.3))
line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,255,0), 3, CV_AA);
else
{
line( cdst, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3, CV_AA);
std::cout <<"slop:" << slope << "\n";

}
  }
 #endif
// imshow("source", dst);
 imshow("detected lines", cdst);
 for( int i = width/3; i <(width*2)/3; i++ ){
            for (int j = height*0.9; j <height; j++) {
                        Vec3b color = cdst.at<Vec3b>(Point(i,j));
                         if(color[1] =255){
                                    //std::cout <<"Lane departer";
                          }
              }
 }
                         if(slope>2.75)
                         {
                                    if(l[2]<width/3 && l[0] <width/3)
                                    {
                                                cout << "we have a door";
                                                cout << "we have a door";
                                                cout << "we have a door";
                                                cout << "we have a door";
                                    }
                         }

 // Save frame to video
        out << image;
        //cv::imshow("Modified video", image);
        // Stop the camera if the user presses the "ESC" key
```

```
        if(cv::waitKey(1000.0/FPS) == 27) break;
        waitKey(200);
    }
 return 0;
}
```


Reference:
[1] https://en.wikipedia.org/wiki/Lane_departure_warning_system
[2] https://en.wikipedia.org/wiki/Hough_transform
[3] https://www.amazon.com/Parallax-32500-ActivityBot-Education-Programmable/dp/B00F1P0004
[4] http://vision.cs.arizona.edu/nvs/research/image_analysis/hough.html