Autonomous Golf Cart Vision Using HSV Image Processing and Commercial Webcam

John D. Fulton *Student, Electrical Engineering Department California Polytechnic State University,San Luis Obispo June 2011*

Abstract—

Using openCV I was able to use a store bought webcam and my laptop to build code that could detect the lane lines and the curb on the road and use them to calculate a trajectory for an autonomous vehicle to follow. The code does have some flaws, as discovered in testing, and definitely room for improvement but it still functions as a useful basis for development. Developed primarily for the Autonomous Golf Car Project at Cal Poly the principles of how it functions could be applied to other projects that want to use computer vision.

I. Introduction

The project hopes to demonstrate that with the proper tools and guidance it is possible for even a novice user to produce useful computer vision code for relevant real world projects and products.

Using openCV and associated guides and tutorials I developed code that was capable of looking at a frame of the road and then locate the curb and lane lines within the frame. After that processing it then used the information gathered to chart an appropriate trajectory to try and keep the vehicle in the center of the road. It is now possible for another group to take this code and integrate it into the Autonomous Golf Cart Project currently under the direction of Christopher Clark.

I have discovered that the code does have some limitations including the distance from guiding marks like lane lines and curbs, not having either one or the other, and the available contrast between these marks and the surrounding environment. I feel these problems could be corrected with more time and effort but unfortunately I have run out.

In the end I have developed useful code that can help to navigate an autonomous vehicle. I hope that it will be of use to future Autonomous Golf Cart teams.

II. Background on Computer Vision

RGB vs HSV

If you refer to figure one, you can see a rather elegant diagram of how an RGB image is formed. Each individual pixel, the building blocks of an image, is determined by three independent values based on the different colors of light, red, green, and blue. It is the combination of these lights that make up the visible spectrum of color. Similarly it is a combination of these values that determine the color of a pixel.

In openCV each pixel of an image is given a one byte per color so 3 bytes of information total stored as an array. These value thus range from 0 to 255 with 255 being the highest value. For a pure red pixel we might expect something like (255,0,0). In openCV the image is stored in a one dimensional array along with a lot of information about the image like pixels wide and tall, and other useful information. The program takes that information and renders the image to a window.

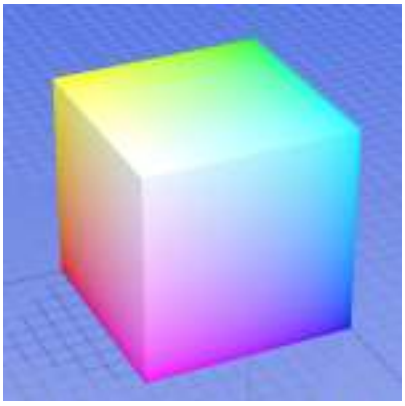

Figure 1: RGB Cube

In contrast to RGB, HSV is an image format that reduces the color to a single byte, the hue, and then uses the other two bytes to express how much coloration is present, saturation, and how bright or dark the pixel is, value. While this doesn't save any space on the hard drive it does provide the user with slightly more intuitive method of isolating particular colors than trying to find a good combination of red, green, and blue.

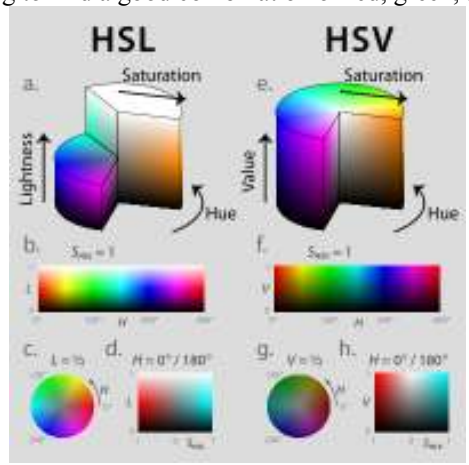

Figure 2: HSV Diagram

Image Processing

In general image processing would be any action we take to modify the original image. In our case we are particularly interested in a process called Gaussian Smoothing.

This removes a lot of the sharpness to the image by averaging neighbors together to create a smoother or blurred image. Doing so removes some of the variability in the image that can come from how the light hit the object(s) your interested in. For instance we are looking for curbs and lane lines, both tending to be rough surfaces mean that as light strikes them it tends to scatter in an unpredictable pattern, from the viewpoint of our camera. By taking an average of all the pixels in a neighborhood and then setting a pixel's value based on this average means that we can force or rather encourage a cluster of pixels that should go together, such as a lane line, to have less distinction from each other while maintaining their distinction from the rest of the environment, such as the asphalt. So instead of some bright yellow, medium yellow, dark yellow, we can get a much less diverse distribution of yellow that more clearly varies from the asphalt around it.

Figure 3: Contour of the Curb

Filtering

Filtering is a sub category of image processing, separated here to draw special attention to the approach taken in this project specifically. In general this is an evaluative process done on a pixel by pixel basis. In this case I generated a secondary image that was a monochromatic representation of my vision. Normally this would produce a world that was shades of grey, however by applying a binary filter I can make it so the entire image is either white or black.

Stepping through each pixel of the original image, I can choose to set the corresponding pixel in the monochrome image to either balck or white. If the original pixel falls within the range I am interested in then I set the monochrome pixel to white, otherwise it is black.

Prior Research and Work

I am not the first person to use openCV for color detection and image filtering. There is a full community dedicated to using this code library and doing development and tutorials for this tool.

One such resource I borrowed heavily from was http://www.shervinemami.co.cc/openCV.html which had wonderful information on using HSV for blob and skin detection. Of particular value was his RGB to HSV conversion code that made it possible to use the full 8 bytes (0-255) rather than the more limited (0-180) that openCV normally used when converting from RGB to HSV.

In addition to the color conversion I needed a method of extracting the colors I was interested in from the surrounding image. Through my research into the problem I came across a Youtube video and the corresponding website http://myopencv.wordpress.com/ which became crucial to developing my project. Using the principles of conversion and thresholds I was able to isolate colors of the curb and the lane lines and then using the functions built into openCV to find the contours of these shapes and pull geometry from them.

III. Reasons for this Project

Large Research into Autonomous Vehicles

Currently there seems to be a growth in research in autonomous vehicles, the most notable of which are projects like the Darpa Grand Challenge, the Google Car, and the various military drones currently in operation. Each of these relies on a host of sensors and the computational power to do something useful with them but vision can become a greater part of this arsenal.

One method is to develop the road and car in tandem. By either making the road have channels for the cars to travel in or by embedding special sensors in car and road for communication the hope is to reduce the number of accidents that occur each year and make the roadways more efficient. However this would leave the question of what to do with the current roads and how can you standardize something like this on a large scale.

However if a system could be developed that used vision, in combination with other sensors, then is could begin to process the roads in the same way that we do now, arguably in the way for which they were designed.

Increased use of vision in robotics

Already we have seen the use of vision in robotic manufacturing. Cameras are used to verify components are properly aligned in assembly lines and for the inspection of final products. I would thus argue the use of vision in robotics is well established and the boundaries worth pushing.

IV. Approach

(Graphic of Image Pipeline towards the bottom of the page)

Choosing Method of Filtering

Filtering became a unique challenge as several options were available to me. I knew that the use of openCV's built in functions would be valuable and began by experimenting with the threshold functions, converting the color image to black and white and then trying to isolate the values of interest based on a single threshold. It was only after further evaluation that I determined more fine control was necessary.

I had chosen to position the camera high on my vehicle looking out hoping to get the longest view possible but this meant also capturing the horizon. In addition the lanes and the curb required separate processing. For this reason the first level of filtering was to isolate the lower left quadrant and the lower right quadrants for separate processing.

The filtering process would step through each pixel one at a time using two for loops, by setting those for loops with the right start and end values. Since the pixels are stored in a one dimensional array with three places to one pixel I used the qualities of the image such as width of each row to isolate one property of the pixel value at a time and if it passed the threshold values then I set the pixel in a monochromatic version of the image to 255 or white while the others were made black.

Choice of Threshold Value

Given how my code works a crucial step was the selection of appropriate threshold values. I found mostly through trial and error that I needed to filter on hue, saturation, and value to properly isolate the values that I was interested in. This was made easier with a few lines of code that allowed me to click on an area that I was interested in and print the HSV values of that pixel. By clicking both the desired area and the area that I wished to isolate it from I was able to choose values that were appropriate. In addition these threshold values can be found in the header file so they can be tweaked as needed.

Computation of Trajectory

Once I could reliably find the pixels I was interested in it was a relatively simple matter to use the tools at my disposal to calculate a trajectory for the vehicle.

The first tool is the use of finding contours in openCV. If I understand it correctly the function looks for contrast in given image and then proceeds to outline the contrast based on some threshold value. Since we have filtered images that are either black or white then the choice of threshold is arbitrary and this is simply a tool, or rather step, to the next part of bounding boxes.

Once the contours have been located I can isolate them one at a time and put a bounding box around each one. This bounding box is a rectangle which is as small as possible but still contains the full contour within its boundaries. Associate with the bounding box are 3 automatically generated values, the center point (X Y in terms of pixels from the top left going to the bottom right), the size (length and width in pixels), and most important and angle (computed between the horizontal and the first side of the rectangle that it becomes parallel with. For the trajectory I primarily used this angle in my calculations.

Since I can usually rely on the lanes on my left and the curb on my right computing an average of the angles between them turns out to be a fairly reliable method for computing trajectory. So by taking these two angles and computing the average and giving the start point as the bottom center of the image I could compute a goal point or rather an angle of trajectory using simple trigonometry.

This provided a small problems, primarily the angle would sometimes jump dramatically when an error occurs in processing. For this reason I compare the new value calculated with the last value. If the new calculated value is higher than the old value, I increase the value by one degree, if lower than I decrease by the same amount. This means that brief spikes in values or fluctuations are less harmful to the final calculation. Unfortunately this means that when the trajectory goes far off track it takes time for the program to recover it if does at all.

Handling Artifacts and Erroneous data

Most generated artifacts are already handled by the process of Gaussian Smoothing at the very beginning, however it seems to be that in areas of particular brightness seem to generate false positives for the algorithm.

In addition with dashed lane lines on the left hand side it is inevitable that I should lose track of the lane lines every few frames. However this problem was more or less solved by giving the algorithm a sort of memory as if it finds no new lane line then it simply uses its last value. This allows us to leap over the few frames that the lane disappears until we can reacquire a new one.

CONTINUES ON NEXT PAGE:

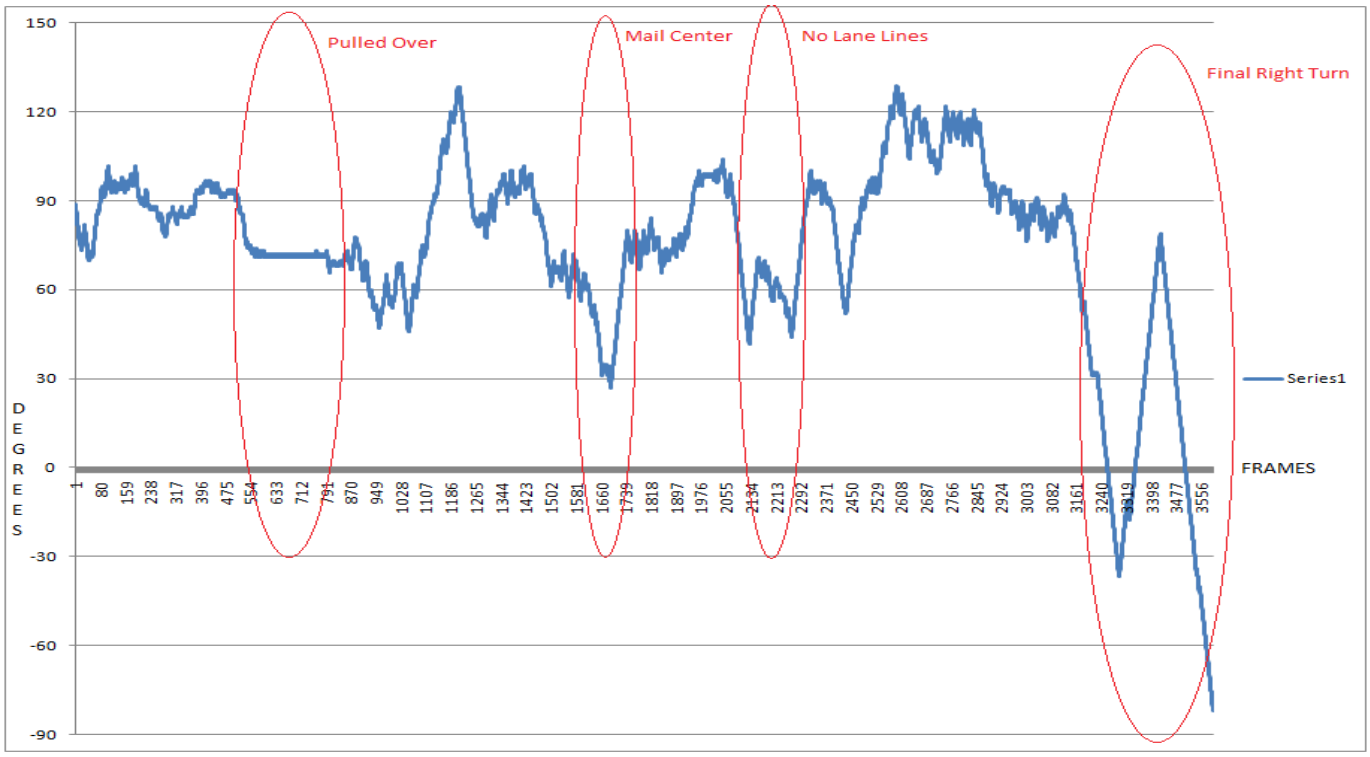

Figure 4: Trajectory Angle vs Frame Number

V. Results

(Plot of angle of travel over time)

Testing Environment

For testing purposes I elected to use ideal conditions to serve as a base line and establish that at the very least the principles of my code work.

Using Village Drive from the H14 Parking lot up to building 71 on the Cal Poly property seemed to be the ideal location for testing. It offered numerous turns, well painted lines, and the curb was clearly distinguishable from the asphalt. Testing during the early afternoon also allowed for excellent lighting.

Figure 5: Map of Test Route

Analysis

Refering to the above graph you can see the trajectory plotted with respect to the frame. I have highlighted four crucial points that were of particular interest, mostly because they were parts where my code had some significant deviation from the norm.

It is important to note that the displayed angle is actually 180 degrees from the actual angle. The program believes that straight up, or forward, from our perspective is 270 degrees most likely because the (0,0) point is the top left corner. However to make things easier for graphing purposes and for ease of explanation I edited the displayed data. Also take not e that the boxes angles are in degrees, the calculated angles are in radians, and the graphs angles are in degrees. These means there is a chance of losing some data between calculations. The only thing left is that for our purposes above 90 is a left turn and below 90 is a right turn.

Figure 6: Right Turn

Analyzing the above graph and comparing it to the map we could almost reproduce the map from the graph. Note that we begin with a fairly gentle right turn, then we straighten out followed by the pause and then a more dramatic right turn. The most dramatic changes are likely errors but on the whole it seems to be fairly consistent in giving clear directions.

The first circled area is the point where I pulled over just before the first dramatic right turn. The problem is that it should be telling the car to turn left but is instead telling it to turn right. I believe this is because when the car is not near the center then the influence of the opposing side gets stronger due to perspective while the current side it is on appears straight.

The second area circled, which occurred near the mail center of poly canyon apartments, was an interesting point because the sidewalk area appeared to have the same values as the lane lines and so while my code still functioned in terms of giving appropriate directions it was receiving interference from the additional signals..

When the lane lines disappeared, the third area circled, I suffered from the same problem as being too close to the curb. However in this case it was combined with the mail room in that an area in front of me was being picked up by the lane lines side of the code. What was strange though was the ellipse that normally circles the lane lines appeared to be perpendicular to the side walk it seemed to be centered on. I am at a loss to explain why this occurred.

Finally, after the last curve in the road, the program seemed to lose its mind completely as the angle became completely irrelevant to the direction the vehicle was supposed to be traveling. The ellipses seemed to be in the right places but the angle calculated where way off base.

I would say that overall my code seems to function fairly well. It consistently directed the vehicle towards the center of the road, even around sharp and gentle curves. The cases already listed I believe are fixable but that requires a higher level of knowledge and skill. In the end I would say that it is ready for integration and further testing, but the code will require a stronger programmer to polish it and perhaps add some features that will be mentioned later.

Limitations

This program currently requires fairly good conditions to operate properly. While it was part of my intention to make it more flexible, I found that the challenges of getting the program to work under near perfect conditions was fairly challenging and took most of my time. Good conditions consist of bright lighting and a strong contrast between road, curb, and lane lines.

In addition I found that perspective can cause problems with my program. As the vehicle moves close the curb or the lanes then they begin to appear straight and the angles of the opposing side becomes more dramatic and the car is directed to go more off course rather than correct. This is especially evident when I pulled over to check that my camera was running properly and when I lost the left lane line coming around the corner near the R3 Parking structure. This means that the cart needs to start well centered and has less room for error.

Figure 7: Pulled Over Next to Curb

VI. Future Work

Integration with the Golf Cart

Integration with the golf cart should be a fairly straight forward feature. There are two variables of interest for each frame. A goal location in X and Y on the frame in pixels, with top left being the (0,0) point. The first step will be to translate this goal location, and your current location, into appropriate control signals for whatever is driving the robot.

Aside from the code, a camera mount will need to be made for the webcam. I suggest that it is mounted high and looks out in such a way that the horizon is just above the middle of the frame. However you can make some quick modifications to the code and make it so the full frame is the road. This would involve changing the limits of the "for loops" in the filter function to accommodate the new frame.

```
//for every "row" in curb data
  for(i=height/2;i<height;i++) {
    //for evey pixel in every "row" in
curb data
    for(j=width/2;j<width;j++) {

//for every "row" in lane data
  for(i=height/2;i<height;i++) {
    //for evey pixel in every "row" in
lane data
    for(j=0;j<width/4;j++) {
```

Improving the program

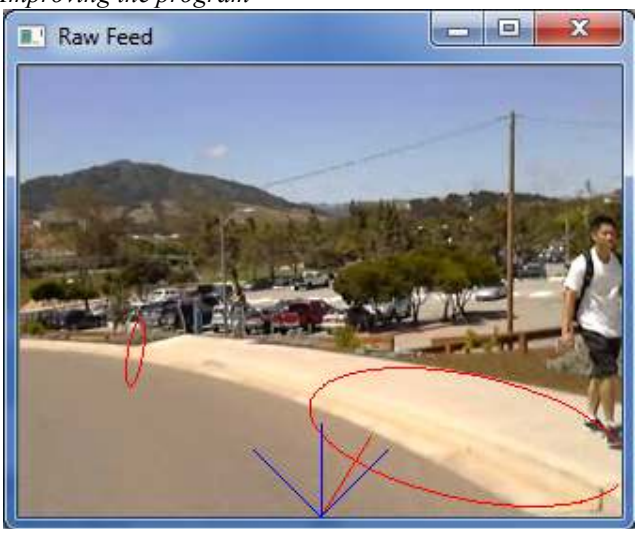

Figure 8: Lane Line Disappeared

The primary thing that I would like to add is the ability to quickly adapt to changing conditions. By making the thresholds more interactive this program could adapt to whatever condition it is in by a few mouse inputs. For instance you could press one button and then click a few spots on the lane line and the code would determine the appropriate threshold values for the current environment. If someone is adventurous then another button could allow you to click on errors within the code and the thresholds would be intelligently reassigned to exclude those points. The only reason these features were not included was I had difficulty on how I would create and intelligent algorithm that could determine good threshold values especially in conditions of poor contrast.

Another thing I would like to have added was better handling of perspective. Using some kind of comparison system perhaps using the center points of the bounding rectangles could be used to determine if we may be too close to a boundary and maybe even a weighting system that makes the closer boundary more important and even use this distance to help calculate trajectory.

CONTINUES ON NEXT PAGE:

VII. CODE?

```
/* Header File for John Fulton Senior Project
 */
#include"math.h"
#include"conio.h"
#include"cv.h"
#include"highgui.h"
#include"stdio.h"

//Reference Files
#define IMAGE_NAME1 "Blobs1.jpg"
#define IMAGE_NAME2 "Blobs2.jpg"
#define IMAGE_NAME3 "Blobs3.jpg"
#define VIDEO_NAME1 "CPV_Curb1.avi"
#define VIDEO_NAME2 "CPV_Road1.avi"
#define VIDEO_NAME3 "Long_Road.avi"

//windows
#define INPUT_WINDOW  "Raw Feed"
#define SMOOTH_WINDOW   "Smooth Feed"
#define HSV_WINDOW    "HSV Feed"
#define LANE_WINDOW     "Lane Lines Feed"
#define CURB_WINDOW     "Curb Feed"
#define CONTOUR_WINDOW    "Contours Display"
#define DISPLAY_WINDOW    "Display"

//Various Constants for Functions
#define CHANNELS    1
#define DEPTH      8
#define EXTERNAL_COLOR    cvScalarAll(255)
#define HOLE_COLOR    cvScalarAll(255)
#define LINE_TYPE     8
#define MAX_VALUE     255
#define MAX_LEVEL     100
#define OFFSET       cvPoint(0,0)
#define THICKNESS     1
#define COLOR_CHANNELS    3
#define MONO_CHANNELS   1

//Thresholds
#define LANE_HUE_LOWER    5
#define LANE_HUE_UPPER    100
#define LANE_SAT_LOWER    70
#define LANE_SAT_UPPER    150
#define LANE_VAL_LOWER    200
#define LANE_VAL_UPPER    255

#define CURB_HUE_LOWER    5
#define CURB_HUE_UPPER    100
#define CURB_SAT_LOWER    40
#define CURB_SAT_UPPER    255
#define CURB_VAL_LOWER    200
#define CURB_VAL_UPPER    255

#define MIN_CURB_SIZE   1500.0
#define MIN_LANE_SIZE   50.0

//macros
#define WAIT_FOR_ESC char c = cvWaitKey(33); if(c == 27) { break; }
#define PAUSE if(c == 'p') { cvWaitKey(); }
```

```
//prototypes
IplImage* convertImageRGBtoHSV(const IplImage *imageRGB);
IplImage* convertImageHSVtoRGB(const IplImage *imageHSV);
void filterImage();
void processImage();

/* Main Source File for John Fulton Senior Project
 */

#include "main.h"

//Globals
IplImage    *g_frame, *g_smoothImg, *g_hsvImg, *g_laneImg, *g_curbImg, *g_contourImg,
*g_displayImg;
CvCapture     *g_capture = cvCaptureFromFile(VIDEO_NAME3);
CvMemStorage   *g_laneStorage, *g_curbStorage;
CvBox2D       g_laneBox, g_curbBox;
bool       g_laneFlag, g_curbFlag;
double      g_angle = 3*3.1415926/2;

void mouseHandler(int event, int x, int y, int flags, void* param) {
  switch(event) {
    case CV_EVENT_LBUTTONDOWN:
      fprintf(stdout, "MOUSE CLICK\n");
      fprintf(stdout, "X: %d, Y: %d\n", x, y);
      if(g_hsvImg != NULL) {
        int height  = g_hsvImg->height;
        int width   = g_hsvImg->width;
        int step  = g_hsvImg->widthStep;
        int channels  = g_hsvImg->nChannels;
        uchar* data   = (uchar*)g_hsvImg->imageData;
        fprintf(stdout, "HUE: %d, SAT: %d, VALUE: %d\n",
          data[y*step+x*channels],
          data[y*step+x*channels+1],
          data[y*step+x*channels+2]);
      }
      break;
    case CV_EVENT_RBUTTONDOWN:
      fprintf(stdout, "MOUSE CLICK\n");
      fprintf(stdout, "Lane Angle: %.2f, Curb Angle: %.2f\n",
        g_laneBox.angle, g_curbBox.angle);
      break;
    default:
      break;
  }
}

void filterImage() {
  if( g_frame == NULL ) {
    fprintf(stderr, "No image\n");
    return;
  }

  cvReleaseImage(&g_hsvImg);
  cvSmooth(g_frame,g_smoothImg,CV_GAUSSIAN);
  g_hsvImg = convertImageRGBtoHSV(g_smoothImg);

  int mouseParam=5;
  cvSetMouseCallback(INPUT_WINDOW,mouseHandler,&mouseParam);
  cvSetMouseCallback(CURB_WINDOW,mouseHandler,&mouseParam);
```

```
cvSetMouseCallback(CONTOUR_WINDOW,mouseHandler,&mouseParam);

cvZero(g_laneImg);
cvZero(g_curbImg);

int height  = g_hsvImg->height;
int width   = g_hsvImg->width;
int step  = g_hsvImg->widthStep;
int channels  = g_hsvImg->nChannels;
uchar* data   = (uchar*)g_hsvImg->imageData;

int laneHeight   = g_laneImg->height;
int laneWidth  = g_laneImg->width;
int laneStep   = g_laneImg->widthStep;
int laneChannels = g_laneImg->nChannels;
uchar* laneData  = (uchar*)g_laneImg->imageData;

int curbHeight   = g_curbImg->height;
int curbWidth  = g_curbImg->width;
int curbStep   = g_curbImg->widthStep;
int curbChannels = g_curbImg->nChannels;
uchar* curbData  = (uchar*)g_curbImg->imageData;

//Find the lanes
int i,j;
//for every "row" in lane data
for(i=height/2;i<height;i++) {
  //for evey pixel in every "row" in lane data
  for(j=0;j<width/4;j++) {
    //if the hue is within our bounds
    if( (data[(i)*step+j*channels] >= LANE_HUE_LOWER) &&
      (data[(i)*step+j*channels] <= LANE_HUE_UPPER) ) {
        //if the saturation is within our bounds
        if( (data[(i)*step+j*channels+1] >= LANE_SAT_LOWER) &&
          (data[(i)*step+j*channels+1] <= LANE_SAT_UPPER) ) {
          //if the saturation is high within our bounds
          if( (data[(i)*step+j*(channels)+2]) >= LANE_VAL_LOWER &&
            (data[(i)*step+j*(channels)+2]) <= LANE_VAL_UPPER) {
            //highlight the curb
            laneData[i*laneStep+j*laneChannels] = 255;
          } else {
            laneData[i*laneStep+j*laneChannels] = 0;
          }
        } else {
          laneData[i*laneStep+j*laneChannels] = 0;
        }
    } else {
      laneData[i*laneStep+j*laneChannels] = 0;
    }
  }
}
//for every "row" in curb data
for(i=height/2;i<height;i++) {
  //for evey pixel in every "row" in curb data
  for(j=width/2;j<width;j++) {
    //if the hue is within our bounds
    if( (data[(i)*step+j*channels] >= CURB_HUE_LOWER) &&
      (data[(i)*step+j*channels] <= CURB_HUE_UPPER) ) {
        //if the saturation is within our bounds
        if( (data[(i)*step+j*channels+1] >= CURB_SAT_LOWER) &&
          (data[(i)*step+j*channels+1] <= CURB_SAT_UPPER) ) {
```

```
          //if the saturation is high within our bounds
          if( (data[(i)*step+j*(channels)+2]) >= CURB_VAL_LOWER &&
            (data[(i)*step+j*(channels)+2]) <= CURB_VAL_UPPER) {
            //highlight the curb
            curbData[i*curbStep+j*curbChannels] = 255;
          } else {
            curbData[i*curbStep+j*curbChannels] = 0;
          }
        } else {
          curbData[i*curbStep+j*curbChannels] = 0;
        }
      } else {
        curbData[i*curbStep+j*curbChannels] = 0;
      }
    }
  }
}
void processCurb() {
  if( g_curbStorage==NULL ) {
    g_curbStorage = cvCreateMemStorage(0);
  } else {
    cvClearMemStorage(g_curbStorage);
  }

  g_curbFlag = false;
  CvSeq*curbContours = 0;
  CvContourScanner curbScanner = cvStartFindContours(g_curbImg,
                          g_curbStorage,
                           sizeof(CvContour),
                           CV_RETR_EXTERNAL,
                        CV_CHAIN_APPROX_SIMPLE
                        );

  while( (curbContours = cvFindNextContour(curbScanner)) != NULL ) {
    if(curbContours) {
      CvBox2D box = cvMinAreaRect2(curbContours,
                  g_curbStorage
                  );
      double area = box.size.height * box.size.width;
      if( area >= MIN_CURB_SIZE) {
        g_curbFlag = true;
        cvZero(g_contourImg);
        g_curbBox = box;

        if(g_curbBox.size.height < g_curbBox.size.width) {
          g_curbBox.angle += 90;
        }
        cvDrawContours(g_contourImg,
              curbContours,
              cvScalarAll(255),
              cvScalarAll(255),
              255
              );
      }
    }
    cvClearSeq(curbContours);
  }
}

void processLane() {
  if( g_laneStorage==NULL ) {
```

```
    g_laneStorage = cvCreateMemStorage(0);
  } else {
    cvClearMemStorage(g_laneStorage);
  }

  g_laneFlag = false;
  CvSeq*laneContours = 0;
  CvContourScanner laneScanner = cvStartFindContours(g_laneImg,
                          g_laneStorage,
                           sizeof(CvContour),
                           CV_RETR_EXTERNAL,
                         CV_CHAIN_APPROX_SIMPLE
                         );

  while( (laneContours = cvFindNextContour(laneScanner)) != NULL ) {
    if(laneContours) {
      CvBox2D box = cvMinAreaRect2(laneContours,
                  g_laneStorage
                  );
      double area = box.size.height * box.size.width;
      if( area >= MIN_LANE_SIZE) {
        g_laneFlag = true;
        cvZero(g_contourImg);
        g_laneBox = box;

        if(g_laneBox.size.height > g_laneBox.size.width) {
          g_laneBox.angle += 90;
        }
        cvDrawContours(g_contourImg,
               laneContours,
               cvScalarAll(255),
               cvScalarAll(255),
               255
               );
      }
    }
    cvClearSeq(laneContours);
  }
}

int main(int argc, char** argv)
{
  g_frame = cvQueryFrame( g_capture );

  g_smoothImg = cvCreateImage(cvGetSize(g_frame), DEPTH, COLOR_CHANNELS );
  g_hsvImg = cvCreateImage(cvGetSize(g_frame), DEPTH, COLOR_CHANNELS );
  g_laneImg = cvCreateImage(cvGetSize(g_frame), DEPTH, MONO_CHANNELS );
  g_curbImg = cvCreateImage(cvGetSize(g_frame), DEPTH, MONO_CHANNELS );
  g_contourImg = cvCreateImage(cvGetSize(g_frame), DEPTH, MONO_CHANNELS );

  cvNamedWindow(INPUT_WINDOW, CV_WINDOW_AUTOSIZE);
  cvNamedWindow(SMOOTH_WINDOW, CV_WINDOW_AUTOSIZE);
  //cvNamedWindow(HSV_WINDOW, CV_WINDOW_AUTOSIZE);
  //cvNamedWindow(LANE_WINDOW, CV_WINDOW_AUTOSIZE);
  cvNamedWindow(CURB_WINDOW, CV_WINDOW_AUTOSIZE);
  cvNamedWindow(CONTOUR_WINDOW, CV_WINDOW_AUTOSIZE);

  CvPoint camera = cvPoint(g_frame->width/2,g_frame->height);

  FILE* pOut = fopen("OutputAngle5.txt", "w");
  int i = 0;
```

```
int frameCount = 0;

while(1) {
  //load frame from the camera
  g_frame = cvQueryFrame( g_capture );
  if(g_frame == NULL) {
    break;
  }

  frameCount++;

  //filter and process the image to extract important data
  filterImage();
  processCurb();
  processLane();

  double tempAngle = g_curbBox.angle+g_laneBox.angle;
  double lastAngle = g_angle;

  if(g_laneFlag && g_curbFlag) {
    lastAngle = 2.0*3.14159/360*tempAngle;
    cvEllipseBox(g_frame, g_curbBox, cvScalar(0.0,0.0,255.0,1.0));
    cvEllipseBox(g_frame, g_laneBox, cvScalar(0.0,0.0,255.0,1.0));
  } else if(g_laneFlag && !g_curbFlag){
    lastAngle = 2.0*3.14159/360*tempAngle;
    cvEllipseBox(g_frame, g_laneBox, cvScalar(0.0,0.0,255.0,1.0));
  } else if(!g_laneFlag && g_curbFlag){
    lastAngle = 2.0*3.14159/360*tempAngle;
    cvEllipseBox(g_frame, g_curbBox, cvScalar(0.0,0.0,255.0,1.0));
  } else {
    //printf("No contours\n");
  }

  if(lastAngle > g_angle) {
    g_angle += 2.0*3.14159/360;
  } else if(lastAngle < g_angle) {
    g_angle -= 2.0*3.14159/360;
  }

  double cosine = cos(g_angle);
  double sine = sin(g_angle);

  double goalX = camera.x-50*cosine;
  double goalY = camera.y+50*sine;
  CvPoint goal = cvPoint((int)goalX,(int)goalY);

  cvLine(g_frame,camera,goal,cvScalar(0.0,0.0,255.0,1.0));

  cvLine(g_frame,
       camera,
       cvPoint((int)(g_frame->width/2+50*cos(3.14159/4)),
           (int)(g_frame->height-50*sin(3.14159/4))),
       cvScalar(255.0,0.0,0.0,1.0)
       );

  cvLine(g_frame,
       camera,
       cvPoint((int)(g_frame->width/2+50*cos(3.14159/2)),
           (int)(g_frame->height-50*sin(3.14159/2))),
       cvScalar(255.0,0.0,0.0,1.0)
       );
```

```
    cvLine(g_frame,
         camera,
         cvPoint((int)(g_frame->width/2+50*cos(3*3.14159/4)),
             (int)(g_frame->height-50*sin(3*3.14159/4))),
         cvScalar(255.0,0.0,0.0,1.0)
         );

    //show images in window
    cvShowImage(INPUT_WINDOW, g_frame);
    cvShowImage(SMOOTH_WINDOW, g_smoothImg);
    //cvShowImage(HSV_WINDOW, g_hsvImg);
    cvShowImage(LANE_WINDOW, g_laneImg);
    cvShowImage(CURB_WINDOW, g_curbImg);
    cvShowImage(CONTOUR_WINDOW, g_contourImg);

    //Once ESC is pressed exit
    WAIT_FOR_ESC
    PAUSE;

    i++;
    if(i>=1) {
      fprintf(pOut, "%d, %.2f, %.2f, %.2f\n", frameCount, goalX, goalY, g_angle);
      i = 0;
    }
    printf("%d\n", frameCount);
  }

  //realease memory
  cvReleaseImage(&g_hsvImg);
  cvReleaseImage(&g_laneImg);
  cvReleaseImage(&g_curbImg);
  cvReleaseImage(&g_contourImg);

  cvDestroyWindow(INPUT_WINDOW);
  //cvDestroyWindow(HSV_WINDOW);
  //cvDestroyWindow(LANE_WINDOW);
  cvDestroyWindow(CURB_WINDOW);
  cvDestroyWindow(CONTOUR_WINDOW);

  cvReleaseCapture(&g_capture);

  exit(0);
}
```

CONTINUED ON NEXT PAGE:

ACKNOWLEDGEMENTS

I want to thank Chris Clark for giving me a lot of guidance on this project. Under his direction I was able to break up my overall goal into more manageable and achievable steps and with his encouragement take on a project that was beyond my skill level.

I want to thank the Cal Poly Robotics Club's Intelligent Ground Vehicle Competition. Specifically Joel for helping solve my initial problem with getting OpenCV to function on my computer and Kerry whose work with OpenCV for this project gave me the inspiration to attempt it myself.

Finally I want to thank the OpenCV community for providing such well documented projects, tutorials, and libraries that made it possible for a relatively novice programmer such as myself to put together this project.

REFERENCES

1) *http://myopencv.wordpress.com/2008/12/20/red-blob-detection-from-video/*
2) *http://www.shervinemami.co.cc/openCV.html*
3) *Bradski, Gary R., and Adrian Kaehler.* Learning OpenCV: [computer Vision with the OpenCV Library]. *Sebastopol: O'Reilly, 2008. Print.*
4) *http://opencv.willowgarage.com/wiki/*