Lunds Tekniska Högskola

Master's Thesis

Head-Up Display Perspective Correction Using Homography Transformations

Author:
Alexander Wormbs

Supervisor:
Prof. Karl Åström

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the field of

Computer Engineering

December 12, 2017

Abstract

The purpose of this project is to research the potential of applying mathematical methods in order to aid the development of robust Head-Up Display systems for automotive vehicles. If the vehicle detects an object and the Head-Up Display should overlay some information over that object, then the information has to be displayed at some position which depends on where the driver is looking from. This has to be done keeping in mind the real-time application of driving a car. The solution proposed in this thesis is to calculate a set of homography matrices corresponding to a set of points that represent a specific position of the driver's head. Then, by tracking the head of the driver the matching homography is continuously applied to the graphical interface so that it matches the outside world.

FIGURE 1: An example Head-Up Display on a windshield. This image is constructed.

Contents

Introduction

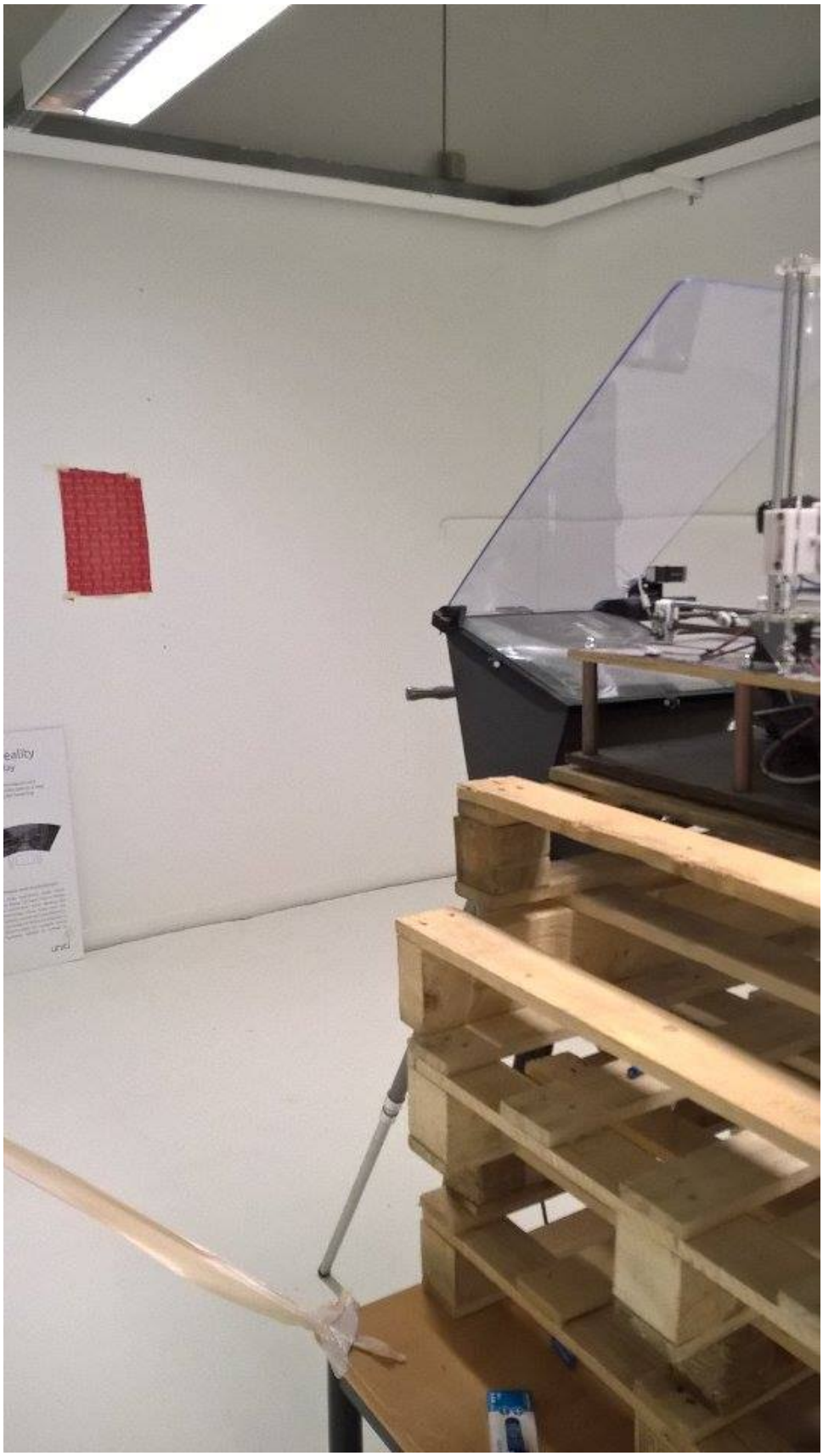
Head-Up Display technology has been around for several decades, first introduced as a way for military pilots to see crucial information without having to look down on the instrument panel while flying. It was introduced in a commercial car for the first time by General Motors in 1988, and displayed basic information such as speed and tachometer, [5]. The development has not gotten much further than that, surprisingly, and you still rarely see a Head-Up Display being used in modern cars at all except for the occasional speedometer in the corner of the windshield.

There is substantial potential with the technology available today to create a system that does much more than that. For example, the Head-Up Display can be used as a GPS navigator and project an arrow on the road that tells the driver where to go. It could also alert the driver of certain things in front of the car, and then highlight them. The applications are many, but in order to develop such a solution the underlying system needs to be robust enough.

There are a number of reasons why we have not seen this in the market yet. To fully utilize the power of a Head-Up Display, it needs to be able to cover most of the windshield area which requires large pieces of hardware that does not fit into the typical car model. An example of a proposed solution to the problem is *Gps-Based Head Up Display System For Driving Under Low Visibility Conditions*[7] where they use a combiner glass in front of the driver in order to show information. This allows for superimposed graphics on the road but is restricted to a small area and also requires big hardware in the car. Other projects has been done with the goal to show a HUD display on the whole windshield, for example *Visual Navigation System On Windshield Head-Up Display*[1] where they used a full projector and mirror to display information. They also distorted the original image in order to cancel out the distortion caused by the curved windshield, however they did not account for the head position of the driver which means they assume the driver's position to be fixed.

The goal of this thesis is to present a solution to the perspective correctness problem, i.e a system that corrects the projected image depending on from where the driver is looking. The approach taken here is to analyze the problem, discuss what methods could be utilized and then propose a system that would fit well into the environment of a car and can be used for large scale manufacturing.

For the testing of the system, a prototype Head-Up Display is used. It works by projecting a screen onto a piece of glass using a lens to parallelize the light rays. The information shown is then projected about 5 meters away from the glass. A camera is put in front of the Head-Up Display which represents the camera that would be in front of the car, and another camera is used to represent the driver's view. A third camera is also present that has an IR filter so that it could detect IR light, used for head tracking. All of the hardware is connected to a computer that runs the program.



Methodology

2.1 Finding a method

2.2 Homographies

If we want to display graphics that has to match the scene correctly from the driver's perspective, that is initially drawn in another perspective, then we can use a *homography* to describe the perspective difference. In order to correctly display the information on the windshield, a homography has to be applied that would transform the image to match the perspective of the driver. Generally, a homography is calculated from matching points in two images taken in the same scene but from different perspectives. This is also the case in this application, since the two perspectives are the driver's viewpoint and the view from the camera in front of the car. The camera recognizes objects in the scene (road, cars, pedestrians, etc...) and draws graphics around that object. Then, it transforms the graphics to the perspective of the driver and displays it on the windshield.

A homography is a matrix that can transform points from one perspective to another. It can be calculated using

$$s_i \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \; i = 1, 2, \ldots, N$$

where H is the $3x3$ transformation matrix, and s_i is a scaling factor. The known variables are x_i', y_i', x_i and y_i. These correspond to the matching points in the two images. The H matrix has 9 elements since it is a 3×3 matrix, but has 8 degrees of freedom since the scale is arbitrary. There are $3N$ equations (three for each point pair), but each pair of points introduces a new variable s_i. Therefore, we have

$$3N \geq 8 + N \implies N \geq 4 \quad ,$$

which shows that 4 pair of matching points are needed to solve the equation. Define H as

$$H = \begin{bmatrix} h_1^T \\ h_2^T \\ h_3^T \end{bmatrix}$$

where h_i^T are the rows of H. Then, the equation can be written as

$$X_i^T h_1 - s_i x_i = 0 \qquad X_i^T h_2 - s_i y_i = 0 \qquad X_i^T h_3 - s_i = 0$$

where $X_i = (x_i, y_i, 1)$. It can be written in matrix form as

$$\begin{bmatrix} X_i^T & 0 & 0 & -x_i \\ 0 & X_i^T & 0 & -y_i \\ 0 & 0 & X_i^T & -1 \end{bmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ s_i \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}.$$

Now, with four pair of points the complete system

$$\begin{bmatrix} X_1^T & 0 & 0 & -x_1 & 0 & 0 & 0 \\ 0 & X_1^T & 0 & -y_1 & 0 & 0 & 0 \\ 0 & 0 & X_1^T & -1 & 0 & 0 & 0 \\ X_2^T & 0 & 0 & 0 & -x_2 & 0 & 0 \\ 0 & X_2^T & 0 & 0 & -y_2 & 0 & 0 \\ 0 & 0 & X_2^T & 0 & -1 & 0 & 0 \\ X_3^T & 0 & 0 & 0 & 0 & -x_3 & 0 \\ 0 & X_3^T & 0 & 0 & 0 & -y_3 & 0 \\ 0 & 0 & X_3^T & 0 & 0 & -1 & 0 \\ X_4^T & 0 & 0 & 0 & 0 & 0 & -x_4 \\ 0 & X_4^T & 0 & 0 & 0 & 0 & -y_4 \\ 0 & 0 & X_4^T & 0 & 0 & 0 & -1 \end{bmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

is acquired. If it is written as

$$Ax = 0,$$

then it can be solved using Singular Value Decomposition

$$A = USV^T = \sum_{i=1}^{9} \sigma_i u_i v_i^T.$$

The singular values, σ, will be sorted in descending order. In this case the homography will be exactly determined by four points, which means that σ_9 will be zero. The homography then fits the points exactly.

The "right singular vector" is used which is a column in V that corresponds to the singular value σ_9. This includes all the coefficients for the homography matrix.

2.3 Finding matching points

The next step is to find the corresponding points in both views. In this project, several methods are considered, for example algorithms such as *Scale Invariant Feature Transform*[6]. SIFT is a well known and established method to find interesting points in a scene, or so called *features*. Then, other algorithms can be applied that match these features between different perspectives looking at the same scene, such as *Brute-Force Matcher*. However, using SIFT for this application could be problematic. SIFT generally cannot run in real-time, even with some implementations where it is optimized to run on the GPU[9]. There are some alternatives to SIFT that can run faster but with less accuracy[2], however, in this application, accuracy is of great importance. The images will also probably be very cluttered and complicated, so to rely on a feature matching

algorithm in real-time is not the best approach.

An important observation can be made here. The homographies do not depend on the actual scene, but only on the pairs of matching points. Therefore, we can calculate the homographies before we have to apply them, so they do not have to be calculated in real-time. The approach is then to pre-calculate all the homographies in a controlled environment, store all the data and fetch it later when it had to be applied. Since the computations can be done 'offline', simple but reliable techniques can be applied for finding matching points between the images. A new homography has to be calculated for each possible position of the driver's head, since when the driver changes his/her position, the perspective (and therefore the homography) will change. When the matching points has been acquired, the homographies that match different positions of the driver can be calculated, and all this information can be saved in a table that matches a set of coordinates to a homography. Later when driving the car, all that is needed is to track the head of the driver, read the coordinates for every frame and then apply a corresponding homography to the graphics that will be displayed on the windshield.

2.4 System Description

The system is divided into two parts, the "offline phase" and the "online phase". During the offline phase, all the calculations are done and then, while driving, the necessary pre-computed homographies are fetched depending on the position of the driver. The first phase uses two cameras, the "driver camera" and the "front camera", which denotes the camera representing the driver view and the camera in front of the car respectively. During the offline phase, the driver camera is put on a robotic arm that moves in a 2D plane to make an approximation of the view angle for the driver at a certain position. The camera moves around, and at every point in time the coordinates are read and a homography is computed. The driver camera coordinates and homographies are matched together in a hashmap, so every coordinate pair returns a homography. The homographies are calculated by continuously tracking points in each camera that matches at every frame.

2.5 Constructing the scene

As previously mentioned, the homographies do not directly depend on the scene, they just depend on the matching points. Therefore, we need to construct a simple scene where four points can be recognized by the cameras. To track the matching points in the two views given by the cameras, a reliable way to find the points was required. The approach was to simply provide the cameras with a simple scene involving a red rectangle. The aim is to segment the rectangle from the image and find the corner points, which would give four points to track.

2.6 Color Segmentation

First, color segmentation is performed to create a binary mask from the original image.

Color segmentation is done by setting static upper and lower thresholds that make up the interval of colors that will be segmented. By using HSV color space, as opposed to RGB, it is easy to specify an interval that corresponds to some color with varying lightness and intensity. The color segmentation I_dst is obtained as

FIGURE 2.1: Offline phase.

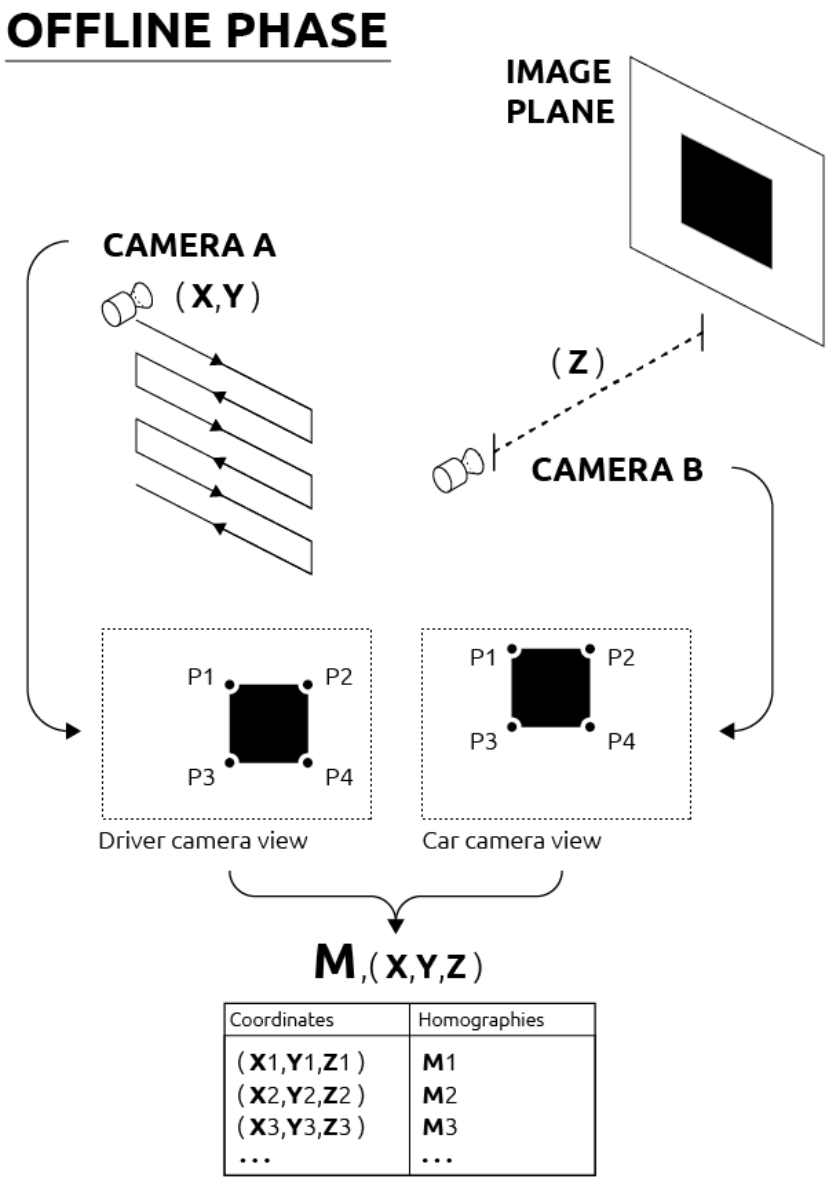
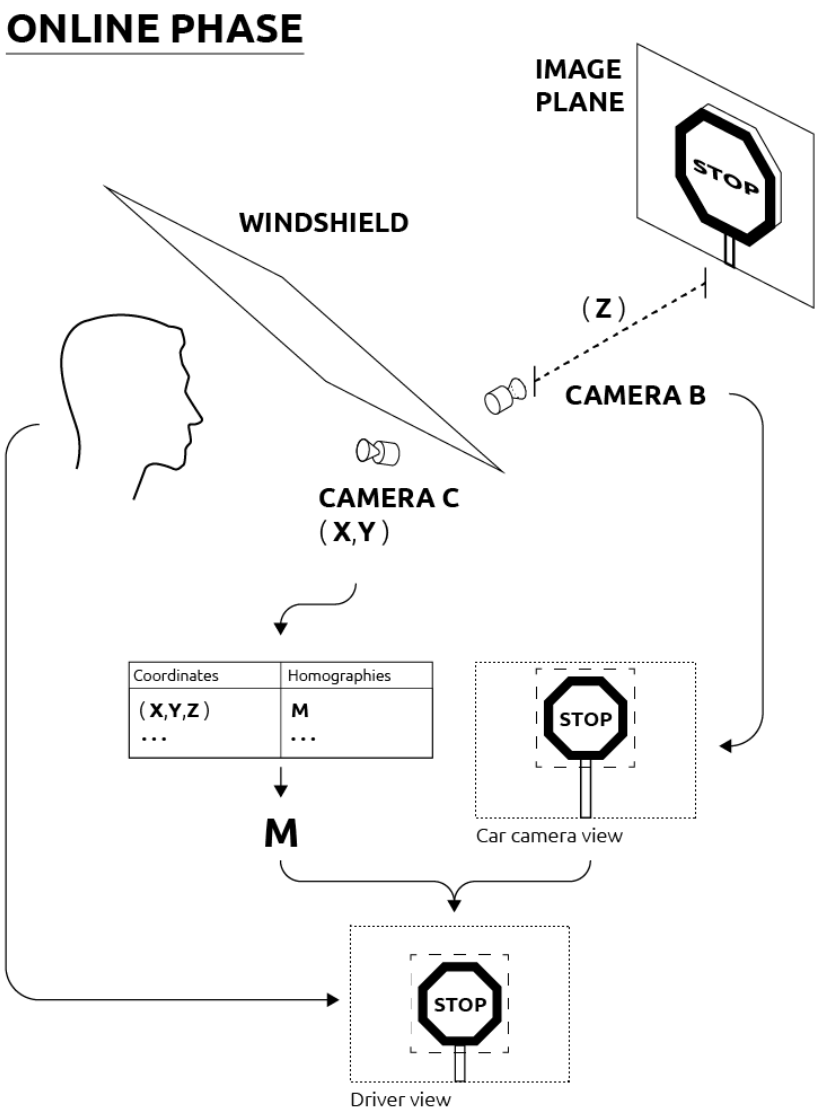


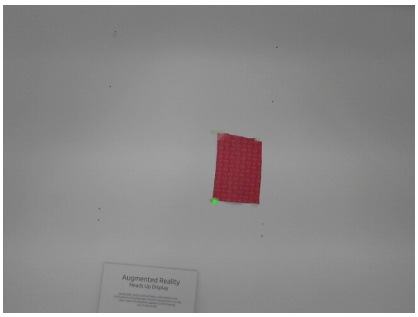
FIGURE 2.2: Driving phase.



$$I_{dst}(x_i, y_i) = \begin{cases} 255 & \text{if } B_l(k) \leq I_{src}(x_i, y_i)(k) \leq B_u(k), \\ 0 & \text{otherwise,} \end{cases}$$

where I_{src} and I_{dst} are the source and destination images, with the destination image being a binary image. Here B_l and B_u are the upper and lower bounds respectively, defined as $B = (r, g, b)$ where r, g, and b are color values.

FIGURE 2.3: Before and after color segmentation.



2.7 Edge Detection

Color segmentation is a bit limited because a specific color has to be used. This is not of great importance in this application, but another algorithm was implemented in order to test the more general case. Here follows a documentation of the method used to find any rectangular shape in an image.

In order to find the rectangle in the scene, an edge detection algorithm can be applied to localize the edges in the image. Canny Edge Detection is used for this application.

In order to reduce noise, the image is first blurred using Gaussian filtering. The image will be smoothed out and small sharp spikes caused by noise will be undetected by the algorithm. The image is convolved with the Gaussian filter

$$G_{ij} = \frac{1}{2\pi\sigma^2} exp\big(- \frac{((i-3)^2+(j-3)^2}{2\sigma^2}\big); 1 \leq i, j \leq 5 \quad .$$

The formula is convoluted with the image, making each pixel a weighted average of it's neighbors.

Next, the Sobel Edge Detector is applied to localize the edges. This is done by convolution between the image and the Sobel operator in x and y directions according to

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \text{ and } G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * I \quad ,$$

where G_x and G_y are the resulting images that contained the derivatives in the x and y directions respectively. The resulting images are then combined using

$$G = \sqrt{G_x^2 + G_y^2}$$

and the direction of the edge is computed by

$$\Theta = atan2(G_y, G_x).$$

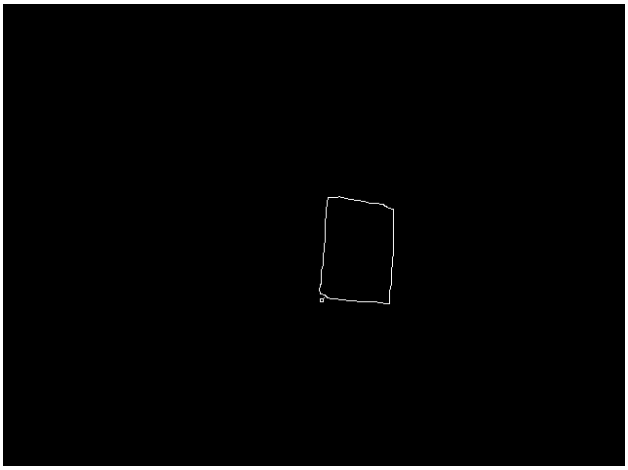
The edges are then located, and Non-maximum suppression is applied to thin the edges. Using the directions of the edges that are acquired after the Sobel operator, the intensity at the current pixel is compared with the intensities of its neighbors in the positive and negative gradient directions. If the pixel has the largest intensity compared to the other pixels on the edge, it is preserved.

In order to remove weak edges likely caused by noise, double thresholding can be used. A high and low threshold value is specified, and if an edge pixel's gradient value is above the high threshold, it is marked as a strong edge, whereas one between the threshold values are considered weak edges, and the ones below the low threshold value are discarded. The resulting edge image will then include all the strong edges, along with all the weak edges that were connected to the strong edges. The weak edges that are not connected to a strong edge are not shown in the final image.

2.8 Finding contours and contour approximation

After color segmentation has been done, the contours of the object are localized. This is done in order to use an algorithm for approximating the contours with a set of straight lines. In order to properly enclose the rectangular object, an algorithm[8] can be used to approximate a figure found in the image with 4 straight lines. If such a figure is found, then it is probably the rectangle.

FIGURE 2.4: Contours.

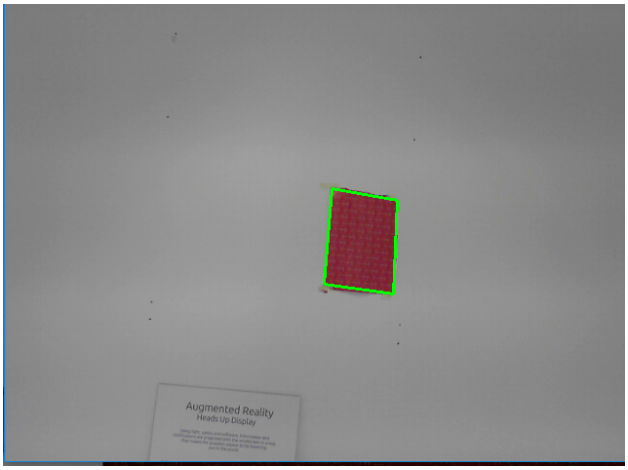


2.9 Optical Flow

After localizing corners of the object, we need to track them reliably when the driver camera is moving. In order to reliably track the points, Lucas-Kanade Optical Flow is used. The point that needs to be tracked was assumed to be within a small neighborhood and approximately constant between two frames. We have the equation

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t),$$

FIGURE 2.5: Contour approximation.



where $I(x, y, t)$ denotes the intensity of the pixel with the coordinates (x, y) at time t. Δx, Δy and Δt describe the changes between the two frames. If we assume a small movement of the pixel, we can write

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t + R.$$

Here, the change in intensity is approximated using Taylor expansion. From this, it follows that

$$\frac{\delta I}{\delta x}\Delta x + \frac{\delta I}{\delta y}\Delta y + \frac{\delta I}{\delta t}\Delta t = 0 \implies \frac{\delta I}{\delta x}\frac{\Delta x}{\Delta t} + \frac{\delta I}{\delta y}\frac{\Delta y}{\Delta t} + \frac{\delta I}{\delta t} = 0 \implies \frac{\delta I}{\delta x}V_x + \frac{\delta I}{\delta y}V_y + \frac{\delta I}{\delta t} = 0,$$

where V_x and V_y are the velocities with which the point has moved between the frames, in x and y direction respectively. $\frac{\delta I}{\delta x}$, $\frac{\delta I}{\delta y}$ and $\frac{\delta I}{\delta t}$ are the changes in intensity with respect to x,y and t. Define

$$D_x(p_i) = \frac{\delta I}{\delta x}$$

as the change in intensity in the x-direction of pixel p_i, then the system

$$\begin{aligned} D_x(p_1)V_x + D_y(p_1)V_y &= -D_t(p_1) \quad , \\ D_x(p_2)V_x + D_y(p_2)V_y &= -D_t(p_2) \quad , \\ &\vdots \\ D_x(p_n)V_x + D_y(p_n)V_y &= -D_t(p_n) \quad , \end{aligned}$$

with $p_1, p_2, \ldots, p_n$ being the pixels in the local neighborhood is acquired. This can be written in matrix form as

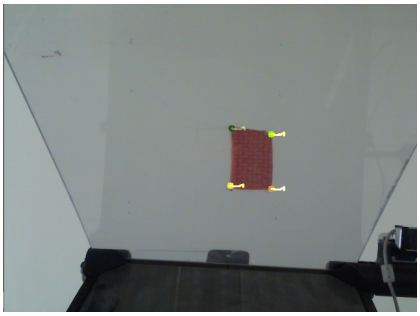
$$\begin{bmatrix} D_x(p_1) & D_y(p_1) \\ D_x(p_2) & D_y(p_2) \\ \vdots & \vdots \\ D_x(p_n) & D_y(p_n) \end{bmatrix} \begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} -D_t(p_1) \\ -D_t(p_2) \\ \vdots \\ -D_t(p_n) \end{bmatrix}.$$

Here V_x and V_y are the unknown variables. It can be solved using Least Squares, by solving the normal equations

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} \sum_i^n D_x(p_i)^2 & \sum_i^n D_x(p_i)D_y(p_i) \\ \sum_i^n D_y(p_i)D_x(p_i) & \sum_i^n D_y(p_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i^n D_x(p_i)D_t(p_i) \\ -\sum_i^n D_y(p_i)D_t(p_i) \end{bmatrix}.$$

In this application $n = 9$ is used, in other words a 3×3 neighborhood around the pixel.

FIGURE 2.6: Optical flow performed on the corners of the rectangle. Trails are drawn after the points when they move.



As the camera moves, each new position of the camera is recorded and the homography for that position is calculated using the points that are being tracked.

2.10 Interpolating homographies

While doing the calibration procedure, it is important to be aware of how many recorded coordinates that will be used. We have to choose some density of measured coordinates for the driver camera, where a higher density means more stored data in memory. Even with a high number of points, we risk having a non-smooth movement of the graphics, and each coordinate that is measured, which can't be found stored in memory, must be dealt with.

We need some way to interpolate the homography matrices. Interpolating a matrix that holds some geometric information is not a trivial task, since we might lose the geometrical meaning of the data if we just interpolate the values normally. *Ken Shoemake et al*[4] proposed a way to do this by first breaking down the matrix into meaningful components and then interpolating them. However, in this project a simple weighting function is applied, called Kernel Regression. This function is run when the camera is at a coordinate not found in the table, and then a homography for that coordinate is generated through interpolation using a multivariate normal distribution as weight function:

$$M = \frac{\sum M_i f(v_i, v)}{\sum f(v_i, v)}$$

where

$$f(x, y) = \frac{1}{\sqrt{2\pi s}} e^{-\sum \frac{(x-y)^2}{2s^2}},$$

and M is the homography matrix, v is a vector with all the coordinates and v_i the new coordinate.

2.11 Meanshift and Camshift

In order to highlight the object with some graphics that can then be shown on the HUD, the Meanshift/Camshift method is used. The Meanshift algorithm locates the maxima of a density function and can be used to track moving objects in an image. Given the gaussian kernel

$$K(x_i - x) = e^{-c\|x_i - x|^2}$$

where x is an initial estimate, there is then the weighted mean

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x)x_i}{\sum_{x_i \in N(x)} K(x_i - x)}$$

where N(x) is the neighborhood of x, a set of points for which $K(x) \neq 0$.

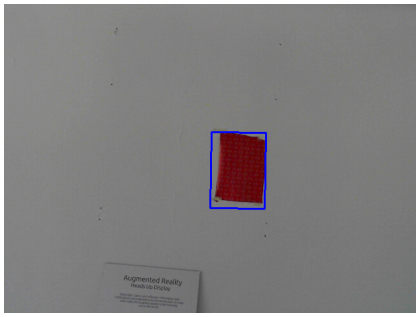
The algorithm sets $x \leftarrow m(x)$ for every iteration and repeats until $m(x)$ converges.

In order to track an object in an image, a start window is placed on the object in the image, which is the initial window for the algorithm. A confidence map in the new image is created based on the color histogram of the object in the previous image. Then, mean shift is applied to find the peak of a confidence map near the object's old position. The confidence map is a probability density function on the new image, assigning each pixel of the new image a probability, which is the probability of the pixel color occurring in the object in the previous image.

Camshift is an extension of Meanshift and uses a histogram back projection in order to generate a probability density function. An initial histogram is computed from the initial region of interest in the image. The histogram consists of the hue channel in HSV color space, and is quantized into bins. The histogram bins are then scaled between the minimum and maximum probability image intensities. The back-projection of the target histogram with any consecutive frame generates a probability image where the value of each pixel characterizes probability that the input pixel belongs to the histogram that is used. Then, the mean shift algorithm is iterated to find the center of the probability image. The window size is set to a function of the zero'th moment

$$M_{00} = \sum_x \sum_y I(x, y)[3].$$

(A) Camshift tracking the object.



Results

3.1 Object Tracking

The Camshift object tracking method seemed to work very well for this application. In this project it did not matter what kind of object was being tracked, but rather that something was being tracked reliably and without delay, and the algorithm proved to be sufficient. The test could be conducted in a controlled environment, hence the object chosen was a red rectangular piece of paper against a white wall background.

3.2 Point tracking and contours

The results of the homographies depends on the point tracking, which in turn depends on the contour approximation. The contour approximation was very accurate with an offset of maximum 4 pixels compared to human measurement. The results can also be improved by changing to an environment with better lighting conditions. The point tracking was also accurate but could be slightly improved by using IR LED light points in a dark room for example, but this would not dramatically affect the end result.

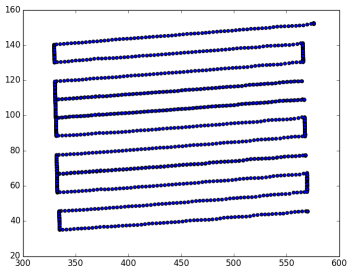
3.3 Homography Transformation

In order to analyze the results of the applied homographies after kernel regression, four tests were performed. The calibration was run and the camera coordinates were saved together with the middle point of the tracked object in every frame. The middle point of the tracking window obtained after applying the homography interpolation was also fetched and the measured error was calculated as the difference in the two middle points. The result was a set of coordinates each paired with an error. Each test was conducted with different amounts of measured camera coordinates, in order to see how well the interpolation worked with less data points.

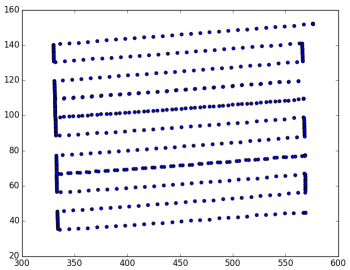
Memory usage of the different samples in the program was also measured using the python standard function $getsizeof()$ in the *sys* module. They were represented as hash maps pairing coordinate pairs with $3x3$ matrices.

- Test 1: 100 kb
- Test 2: 24 kb
- Test 3: 6 kb
- Test 4: 1.6 kb

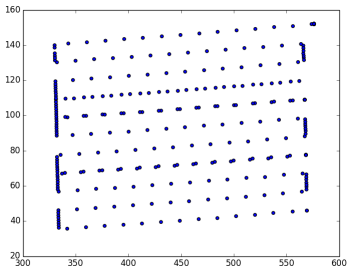
FIGURE 3.1: The measured positions of the camera during calibration in four different tests.



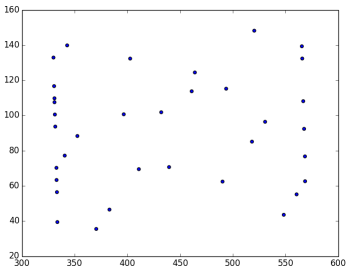
(A) Test 1: 1840 points



(B) Test 2: 743 points

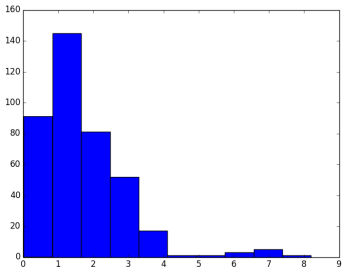


(C) Test 3: 285 points

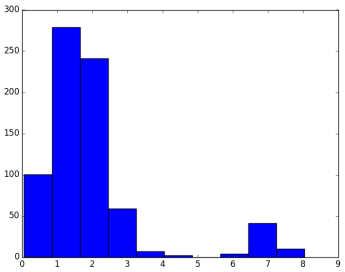


(D) Test 4: 35 points

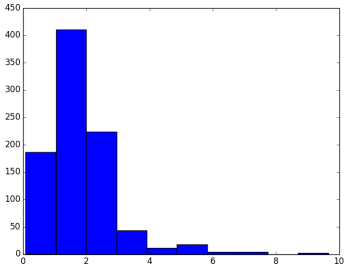
FIGURE 3.2: Block diagrams showing observed error rates in four different tests. Y-axis shows quantity and X-axis shows magnitude of error.



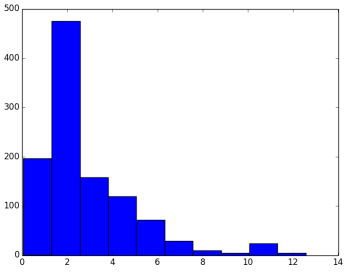
(A) Test 1



(B) Test 2

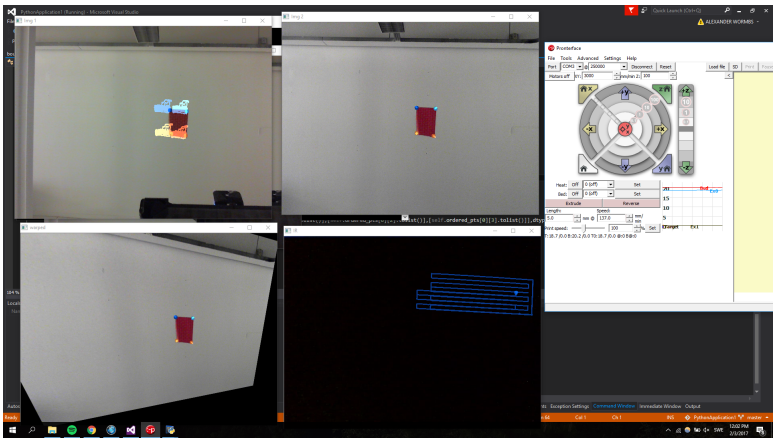


(C) Test 3



(D) Test 4

FIGURE 3.3: Program running during calibration phase. Top left image shows the point tracking and driver camera view, upper middle image shows the front camera view, bottom left image shows the applied homography transform, middle lower image shows the IR LED tracking, right image shows the 3D printer control interface.



Conclusion

There were some parts of the problem that this study did not fully address, and consequentially might compromise the results after more complete testing. The proposed system did not take the depth of the object into account, which means that the graphics will not be placed correctly for all positions of the object. A homography describes the perspective transform between two planes, and since the calculations were done for an object at one specific position, the system relied on the fact that a tracked object would reside in one plane. In other words, at a specific distance from the vehicle. If the object moves a significant distance away or toward the vehicle, the graphics will no longer match the object and be thrown off by a distance proportional to the change in depth. The solution would be to just calibrate the same way but for different depths as well, which would result in storing x times more data, where x is the amount of depth points you want to measure.

The system could undoubtedly run in a real-time environment since the program only had to make frequent lookups in a hash table to find the right homography to apply for every point, and the average time complexity for such a task is $O(1)$. The only other concern was how much memory the program might occupy since the computer running in the car might have very limited resources. As was concluded in the results, the most dense sample of points during calibration resulted in about 100 kilobytes of RAM usage. Taking the depth into account, one could assume that the depth would be measured up to 50 meters in front of the car, and the calibration would be done for every meter. Then, the RAM usage would be about 5 megabytes, which is feasible for this application. It would even take a lot less space since these calculations were done with much more points than needed.

Looking at the results from the homography transformations, visually and by interpreting the data, the approach taken seemed to be promising. While performing the tests to measure the errors, the graphics followed the object reliably in all tests except for the last one with the least data points. The tests were not perfectly performed since the points were not distributed in an optimal way. The optimal way, in terms of the least points needed for optimal results, would be to have the points equally spaced. However, it was still apparent with the tests conducted that not many points had to be used for the tracking to be reliable. The most important thing to take from the results was the max amount of error recorded during the test, which increased from about 8 pixels to 12 pixels from test 1 to test 4. This could be noticed quite clearly as the graphics started to "jump" more instead of a smooth motion.

Bibliography

[1] Yoshinari Kameda Yuichi Ohta Akihiko Sato Itaru Kitahara. *Visual Navigation System On Windshield Head-Up Display*. URL: http://www.kameda-lab.org/research/publication/2006/200610_ITSWC/200610_ITSWC_sato.pdf.

[2] Kurt Konolige Gary Bradski Ethan Rublee Vincent Rabaud. *ORB: an efficient alternative to SIFT or SURF*. URL: http://www.willowgarage.com/sites/default/files/orb_final.pdf.

[3] Jesse S. Jin John G. Allen Richard Y. D. Xu. *Object Tracking Using CamShift Algorithm and Multiple Quantized Feature Spaces*. URL: http://crpit.com/confpapers/CRPITV36Allen.pdf.

[4] Tom Duff Ken Shoemake. *Matrix Animation and Polar Decomposition*. URL: https://docs.google.com/viewer?url=http://www.cs.wisc.edu/graphics/Courses/838-s2002/Papers/polar-decomp.pdf.

[5] Donald Knuth. *How Head-up Displays Work*. URL: http://auto.howstuffworks.com/car-driving-safety/safety-regulatory-devices/head-up-display2.htm.

[6] David G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. URL: http://www.cs.ubc.ca/~lowe/papers/ijcv04.pdf.

[7] Craig Shankwitz Max Donath and Heonmin Lim. *A GPS-BASED HEAD UP DISPLAY SYSTEM FOR DRIVING UNDER LOW VISIBILITY CONDITIONS*. URL: http://conservancy.umn.edu/bitstream/handle/11299/888/1/200303.pdf.

[8] Satoshi Suzuki. *Topological structural analysis of digitized binary images by border following*. URL: http://www.sciencedirect.com/science/article/pii/0734189X85900167.

[9] Changchang Wu. *SiftGPU: A GPU Implementation of Scale Invariant Feature Transform (SIFT)*. URL: http://cs.unc.edu/~ccwu/siftgpu/.