NTNU

Norwegian University of
Science and Technology

Automated Calibration of Multi-Projector Arrays

Ola Nordbryhn

Master of Science in Communication Technology
Submission date: June 2009
Supervisor: Leif Arne Rønningen, ITEM

Norwegian University of Science and Technology
Department of Telematics

Problem Description

One of the problems with video projectors is that the calibration is a manual process that takes time and may be inaccurate. This problem is magnified when setting up a multi-projector array, where several projectors are set up to work otgether, to give stereoscopic images or to increase the projected image size. By doing this process automatically, one may gain both speed and accuracy, and in turn get a more flexible projection system. One of the goals is to make a demonstrator of a system that can prove this.

Assignment given: 15. January 2009
Supervisor: Leif Arne Rønningen, ITEM

Distributed Calibration of Multi-Projector Arrays

Ola Nordbryhn

June 11, 2009

This page is intentionally left blank.

Abstract

Setting up large multi-projector arrays today usually come at a cost; manual calibration of each projector requires time. The orientation of the image from each projector must be correctly aligned in six axes to make the final projected output fit the screen. Not all aspects of calibrating projectors are possible to correct, consumer hardware usually only covers two or three of the axes, the remainder are often corrected using clever projector placement. Also, the degree of which it is possible to adjust is also limited, decreasing placement flexibility. As the collaboration surfaces in Hems lab requires a large number of projectors to work seamlessly together, good calibration techniques are required in order to keep setup and maintenance time low, while giving highly accurate calibration results.

By creating a software demonstrator that automates much of the calibration and enables quick and easy setup, I have made possible rapid prototyping, testing and demonstration of multi-projector arrays, both with single and stereoscopic views. As I will prove, this software shows a flexible approach that may be of use, not only to the Caruso lab and future Hems lab, but may also be used in other settings where projector technology up to this date still has not been widely used, by overcoming the calibration and image warping hurdles and limitations.

The software is developed with basis in the OpenCV computer vision library, and implemented in Python. Tests show that calibration time for a single projector may be cut down to a matter of seconds, regardless of the placement of the projector in relation to the screen, whereas traditional calibration often still not reach the same level of accuracy even if taking tens of minutes or require repositioning of the projector to compensate for the lack of adjustment possibilities.

Acknowledgments

I would like to thank my teachers at NTNU, especially my project tutor Leif Arne Rønningen at Department of Telematics, for guidance and advice throughout my education, and all my previous and current classmates, whose knowledge has been both inspiring and helpful. I would also like to thank my girlfriend for support during my work and for the cover art.

Disclaimer

This report is the main part of my Master's thesis at NTNU's Department of Telematics. The work was done during the spring of 2009, with a duration of 20 weeks. All sources used are properly covered and accounted for, images and figures are created myself or taken from Wikimedia Commons unless otherwise noted. The sample video used in all examples is the intro to the show Kemphanen from Belgian TV station VRT. The cover image is made by Anne Louise Morseth.

List of Figures

List of Tables

Nomenclature

CPU Central Processing Unit, the main processing unit and core of computers. Everything in a computer is connected to and through the CPU.

CRT Cathode Ray Tube, display technology used in older TV sets where a beam of electrons hit a fluorescent coating in the display screen, which then emits light.

DLP Digital Light Processing, projection technology common in home projectors today. It uses micromirrors on a chip, which is illuminated through a color wheel.

DMX512 A communication protocol and connector specification, used as an industry standard between stage light controllers and the dimmers or other equipment being controlled.

FPGA Field Programmable Gate Array, an integrated circuit where gates and wiring can be reprogrammed by the user to work in a specific manner to create a hardware implementation of a given design.

Full HD Full HD is a display specification that defines the resolution to be 1920 by 1080 pixels, displayed progressively. This gives a gross pixel count of approximately two megapixels.

GPU Graphical Processing Unit, a specialized graphics coprocessor common in most modern computers, fucusing on solving the highly parallel problem sets needed for graphics calculations.

GUI Graphical User Interface, as opposed to textual interface.

LAN Local Area Network, a small network, usually with ethernet connections. Used for connecting and communicating with nearby computers over a wired network.

MIDI Musical Instrument Digital Interface, a standard protocol for transmitting event information and synchronisation, often used to communicate between music instrument controllers and computers.

MXM Mobile PCI Express Module, a graphics interconnect standard widely used in laptops and embedded computers.

PCIe Peripheral Component Interconnect Express or PCI-express, a bus for connecting computer expansion cards on a motherboard. Intended to replace PCI, PCI-x and AGP.

PoE Power over Ethernet, a standard for transmitting power in addition to data over standard Ethernet cables.

QoS Quality of Service, measurement and service guarantees for the percepted quality, with regards to a variety of parameters like latency, bit rate and transmission errors.

RAID Redundant Array of Independent Disks, a set of hard drives configured together to provide redundancy, speedup or both.

VHDL Very High Speed Integrated Circuits Hardware Description Language, a design language to program FPGAs or to simulate the hardware behaviour of a circuit design.

Contents

1 Introduction

1.1 Motivation

Use of video projectors is widespread, both in consumer and professional markets. However, it is still a time consuming job to place and properly calibrate them. Ideally, a video projector is placed straight in front and at the same height as the screen, meaning the same spot as one would want the viewer's head. Obviously, both the physical presence of the projector and the noise it emits is not wanted in such a spot, which often also is the most wanted spot for viewing. This dilemma is often solved by mounting the projector in the ceiling, as this is a compromise with both noise and acceptable image quality. Image skew is then introduced if the projector is at an angle compared to the ideal spot. The image may suffer from becoming a trapezoid, rotated or leaking light onto surfaces other than the screen, this is often correctable to some degree by built-in calibration means.

The manual calibration process is often a tedious one, especially if the projector is ceiling mounted and one has to climb up to reach it. The resulting quality is varying, some projectors lack certain degrees of freedom when it comes to warping the image, for instance is vertical keystoning still only available on high-end equipment[6]. This means that the output image may suffer from decreased quality, geometric errors and loss of effective resolution. Finally, if the image finally fits the screen and the projector is tucked away in a not to troublesome location, one must make sure not to move the projector or screen around, as one then needs to recalibrate the projector all over again.

This calibration process needs to become more automated. Calibrating one projector is not too troublesome for most, but when faced with an array of projectors that are supposed to work together to create one big picture, the time and accuracy needed to calibrate these is a key reason not to use such an array. I want to make a solution where the projector and the computer system it is connected to calibrates and adjust the image of each projector in an array with as little as possible input from the user. The solution needs to be quick and accurate, and the result should be a stand-alone application that runs on a computer that outputs the pre-warped image to the projector, so the projector itself requires little or no manual adjustment.

The applications of such a system is plentiful. One may use this for placing a

projectors where they are most suited, where they are least troublesome and protruding, even if this is a place that is hard to reach or one needs to often move them around. One does not need to take into account the limits of the built-in image adjustment in the projector as this is done in software. This may spark more creative use of projectors, they may be used not just for video presentations, but also for lighting effects in stage productions or for projecting onto irregular surfaces normally not associated with video or visual effects, such as projecting facial and bodily features onto a static doll, giving it lifelike characteristics.

1.2 Concrete goals

In this thesis, I want to achieve the following goals

- Create a software solution that warps an image to fit the projection surface with an easy-to-use interface.
- Examine the use cameras or light sensors to automatically identify the screen surface.
- Make a solution that is both flexible and distributed, so each projector has its own keystoning computer for greater extensibility.
- Use the OpenCV framework as a basis for my solution.
- Present my findings and results in a written report.

1.3 Limitations

The goal of this work is to make a sample application, showing some possibilities with automated calibration. A finished system with a full array of projectors will not be put up, and calibration based on camera input will only be discussed, not implemented. Tests will show whether the software made has advantages compared to more traditional calibration methods and to find how this can be used in a complete system. The system is supposed to project only onto even, planar surfaces with no color correction.

1.4 Originality

The system described will differ from existing solutions in its extensibility and how it treats projection surfaces. It is not only a system designated to be used for spreadsheet presentations, but a system where flexibility is key and innovation in usage is encouraged. It shows how calibration can be done very fast and accurate using a well-known computer vision library, and its results should be important in the further development of Hems lab. I will also show how it can be used as replacement and extension of stage lighting, by projecting light and video onto both the actors, stage and stage sets.

1.5 Overview

In part 2, I will present the required background knowledge for understanding the problems, as well as highlight the elements used to solve them. Part 3 goes into detail on how the implementation works and describes how the different elements work together, part 4 describes more on how the system is thought to be used in a real-life situation and which results are to be expected. The discussion and comparison with existing systems, as well as looking at pros and cons is in part 5, while conclusions and future work is in parts 6 and 7, respectively.

2 Definitions

In this chapter, I will list and explain the technical background for the work I have done.

2.1 Hems lab and DMP

The Hems Lab, which this project is a part of, is a research project at the Department of Telematics at NTNU. Its goal is to create a networked virtual collaborative space for everything from actors practicing an opera piece, to web conferencing, with conceived near-natural quality. The goal is to make collaboration across great distances as easy and natural as actually being in the same room. It is intended that at least five surfaces should be covered with video projections with auto-stereoscopic multi-view projections in high to ultra-high resolution. Additionally, several high quality cameras and surround sound recording and playback equipment is to be fitted to provide an experience as close to real life collaboration, as is possible with state-of-the-art technology. In order to realize the Hems Lab, several difficulties have to be overcome, among others the calibration of large multi-projector arrays.

The Hems lab is a realization of the collaboration principles in Distributed Multimedia Plays, DMP. DMP is a proposal for a three-layer systems architecture for virtual networked collaboration.[51] It uses SceneProfiles that describe the characteristics with the implicit needs for all parts of a collaboration situation, they can be used for instance describing sub-objects in a scene such as actors and static objects, to give each sub-object the resources needed to be represented near-naturally while keeping bandwidth use as low as possible. Each sub-object then can be encoded and rendered with individual quality descriptions, collectively creating the entire scene.[52] Other issues that may be tested in the lab is the real-life performance testing of the AppTraNet networking protocol and the use of SceneProfiles in media coding. The AppTraNet is a combined application, transport and network layer protocol for efficient transport of video with both guaranteed QoS constraints and graceful degradation of quality when bandwidth requirements are greater than the underlying network can handle. It applies many principles from IPv6 and IPSec, while introducing new elements to the header useful for both the transport and application layer. This includes SceneProfile reference, sequence number and the physical placement of

the data in the finished output image. By replacing or modifying current protocols and customizing the header information, the protocol becomes less complex and more lightweight than current protocol stack, while being more robust and flexible when used within a DMP architecture.

2.2 Caruso lab

The research and development of this project has mostly taken place at NTNU's Caruso lab[27], the primary lab for research in 3D technology and the principles of DMP at NTNU's Department of Telematics. The lab equipment includes

- Two DLP projectors with XGA resolution
- Two LCoS projectors with SXGA+ resolution
- A full HD plasma screen with Motion+[1] technology
- Two video cameras capable of recording HD video
- Polarizing filters
- Polarity preserving silver screen
- Active shutter glasses, polarizing glasses and anaglyph glasses
- Two high-end computers for video rendering and 3D video playback

This equipment allows for extensible testing of both single and dual projector technology as well as 3D video with different technologies. The lab is widely used during tech demonstrations for guests and other students to show three dimensional video by stereoscopy and how temporal resolution increases will affect viewer experience. The lessons learned in the Caruso lab will be vital for the realization of Hems lab by proving high needs for both spatial and temporal resolution, as well as creating a testbed for early realizations of virtual collaboration.

[1] Technology that upscales temporal video resolution to 120 Hz by interpolating between the source video frames

2.3 Keystoning

Keystoning is the term coined for the effect that happens when an image is projected at an angle onto a screen. The image is skewed and becomes a trapezoid, where the parts furthest away from the projector becomes larger than the parts closest to the projector. This is an unwanted effect, as it causes lines that was supposed to be parallel now are at an angle, distorting the view. Some projectors have built-in keystone correction, either optical or electronic, but this does not always have desired adjustment range, and manually adjusting the projector to eliminate the keystone effect properly is tedious work. Therefore, it would be ideal to have an application that automated some of the work involved with calibrating an image to correct keystone issues.

The effective resolution of the image in a given axis p is given by:

$R_{eEffective} = R_{Projector} \times \frac{\cos(B_p + \frac{A_p}{2})}{\cos(B_p - \frac{A_p}{2})}$ [23]

for the x plane, where B_p is the angle between the projector and the screen, and $A_p$is the projector's beam angle, as illutrated in figure 1 on the following page.

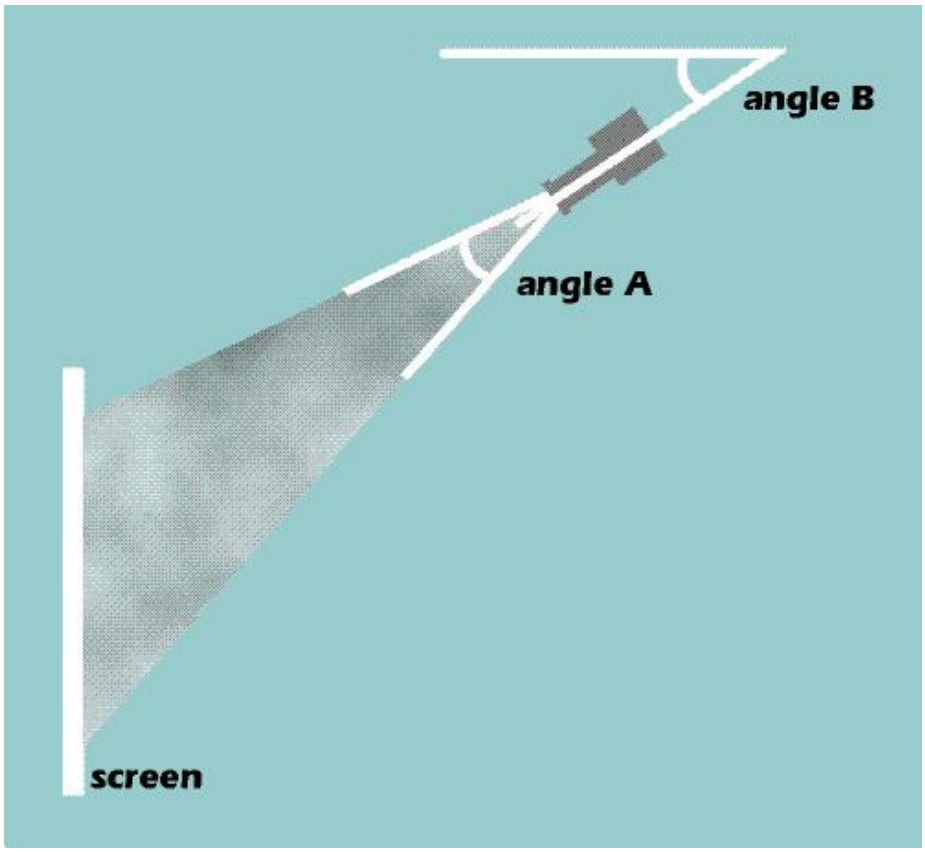

Figure 1: Illustration showing the relation between angles when calculating keystone distortion[24]

Typical beam angle values vary between projectors, the values used here are taken from Optima EX525ST[48] projector. By looking in the reference manual at screen sizes for different distances between the projector and the screen, it is found that it has beam angles of approximately 80° horizontal and 64° vertical.

By increasing the angle between the screen and projector in both x- any y-axes simultaneously, it is easy to see how much keystone affects effective resolution. This has been done in table 1, where it is possible to see what the effective resolution is for both planes as well as how much information is lost for the given angle being applied to both axes at the same time. The loss is given by

$$Loss = \frac{R_{x_{effective}} \times R_{y_{effective}}}{R_{x_{projector}} \times R_{y_{projector}}}$$

Rotation	Horizontal resolution	Vertical resolution	Total pixel loss
0°	1024	768	0 %
5°	883	688	22.8 %
10°	859	615	32.8 %
20°	591	483	63.7 %
30°	372	360	82.9 %

Table 1: Loss of resolution due to keystoning

This shows that there is a quite severe loss of information when increasing angles, this is most visible in the far edge of the keystoned image where the pixel density is lower than in the near corner because the source image is stretched.

By finding the largest rectangle inscribed in another while keeping the aspect ratio, we can approximate the loss of pixels for rotated projectors. By using the following

$$\sin\varphi = \frac{Height_{Projector} - Height_{Screen}}{Width_{Screen}}$$

where φ is the rotation angle, $Height_{Projector}$ is the projector's resolution and the $Height_{Screen}$ and $Width_{Screen}$ is the resulting resolution. By knowing the ratio between width and height of the screen, *Ratio*, we find that

$$Width_{Screen} = \frac{Height_{Projector}}{\sin\varphi - \frac{1}{Ratio}}$$

The ratio is the proportion ratio between the long and the short edge, often 4:3 or 16:9. By calculating the resulting resolution from a XGA projector where the final image has to fit within the rotated output and keep the proportions of the source, we get the following table.

Rotation	Horizontal resolution	Vertical resolution	Total pixel loss
0°	1024	768	0 %
5°	917	688	19.8 %
10°	831	623	34.2 %
15°	761	570	44.8 %
20°	703	527	52.9 %
30°	614	460	64.0 %

Table 2: Loss of resolution due to rotation

The figure 2[2] shows how total resolution is lost with increased rotation angle. This means that in order to maintain as high pixel count and density as possible, both the keystoning and rotation should be kept to a minimum. However, this is not always possible, and as one can see from the tables, the rotation can be quite acute while still maintaining a displayed resolution that is more than half the projected resolution.

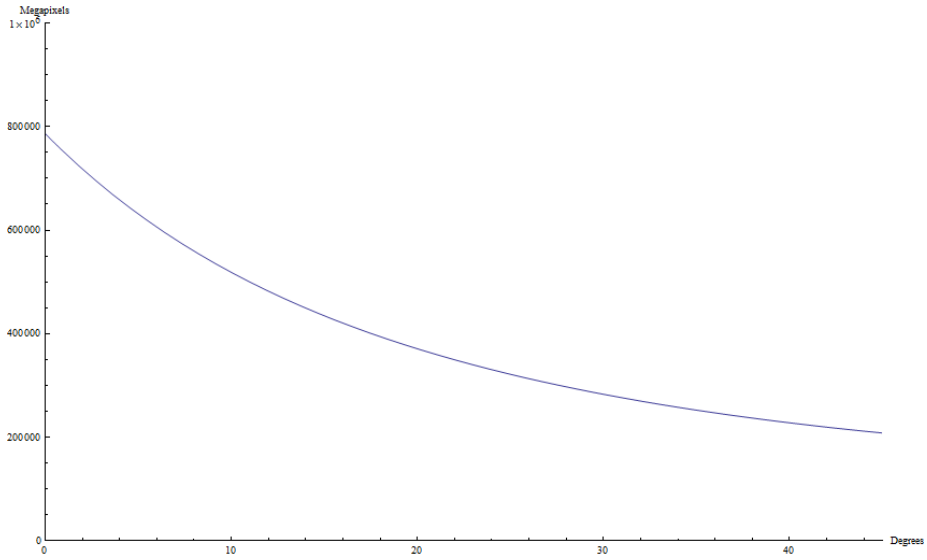

Figure 2: A plot showing total pixel count for rotated images

2.4 Optical calibration and lens shift

One of the problems with calibration that warps the output video imagery, is that it will produce a loss of information. As a projector only has a limited number of pixels, each rotation and keystoning of the image will reduce the amount one can use for projecting to the screen. Using optical lens shift, one can achieve a high degree of freedom with regards to placement of a projector without having to bear the cost of loosing image resolution. However, the optics

[2]The plot comes from Mathematica, the input is
Hpj = 768
Bpj = 1024
Bb = (Hpj/(Sin[t Degree] + (3/4)))
Hb = Bb*(3/4)
Plot[Bb*Hb , {t, 0, 45}, PlotRange -> {{0, 45}, {0, 1000000}}, AxesLabel -> {Degrees, Megapixels}]

needs to be accurate and of very high quality in order to keep image distortion to a minimum. Today, few projectors have control over three or more axes with lens shifting, as this is too costly and heavy for practical operation.

For remote control of lenses, a simple control bus like Inter-Integrated Circuits, I^2C[33], could be used. I^2C provides a low-cost, low bandwidth serial interface that is used in many simple applications, and its close integration with micro controllers will enable easy implementation.

2.5 Homography

A homography is a mathematically invertible transformation for mapping two different views together. It can be used to represent the six-axis difference between two planar surfaces in a three-dimensional space. This is a useful concept for when dealing with calibration of projected images, in order to find a mapping between what you see and what you are trying to project. Finding the homography will enable rectification of images so that lines that are straight in the source images, becomes straight when it is projected onto a screen.

Of the six axes of which to control a projector output, three of these, X, Y and Z, are often done by physically moving the projector, the other three are done either optical within the projector or by pre-warping the image before sending it to the projector. These are described in figure 3 where Θ_x, Θ_y and Θ_z describe pitch, roll and yaw, while X, Y and Z describe physical placement and the zoom or size of the image. All six axes are possible to adjust with physical movement of the projector. The keystoning directions and size adjustment may be done either with lens shifting or by calculating a three dimensional warp matrix and using this to warp the images.

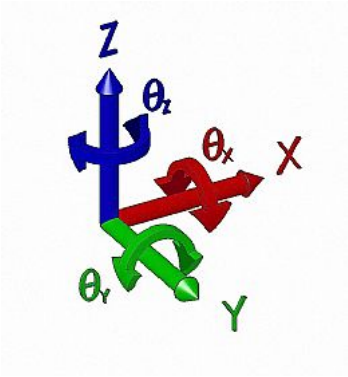

Figure 3: Figure showing the six axes[5]

By using an assumption of a pinhole camera, something that is possible because a projector can be viewed as a pinhole camera in reverse, the homography H can be calculated using the equation[38, 44]

$\widetilde{q} = sH\widetilde{Q}$

where $\widetilde{q}$ and $\widetilde{Q}$ are the points in the source image and in the projected image, respectively. The parameter s is a generic scaling factor between the images. Each point in the source image $\widetilde{q}$ can be described with $\widetilde{q} = \begin{bmatrix} x & y & 1 \end{bmatrix}^T$, correspondingly, a projected point $\widetilde{Q}$ can be described with its coordinates in three dimensions, $\widetilde{Q} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^T$. The use of the parameter Z is to describe the depth of the projected point.

The homography H is composed of two values, the physical transformation W and the camera intrisics matrix M. The physical transformation is again composed of two parameters, the three dimensional rotation and the translation vector, $W = \begin{bmatrix} R & \vec{t} \end{bmatrix}$ where $R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$and $\vec{t}$ is describes the pose of the projected point relative to the source, giving a physical relation between the source plane and image plane.

The camera intrisics matrix M is a 3x3 matrix that describes the camera's qualities with regards to focal length and the offsets of the image's center coordinates with regards to the optical axis of the projection.

$$M = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $f_x$and $f_y$is the pixel focal length, given as a product of the pixel density and the lens' physical focal length, and and $c_x$and $c_y$is the displacement of the image center point, both values are given in number of pixels. The matrix M can thus be used as a constant, as it depicts the qualities of the projector as a pinhole camera. This then gives us the following equation:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_1 & r_2 & r_3 & \vec{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

By setting the value Z for the projection to 0, it is possible reduce the expression somewhat:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = sM \begin{bmatrix} r_1 & r_2 & r_3 & \vec{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sM \begin{bmatrix} r_1 & r_2 & \vec{t} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

so that the homography is a 3x3 matrix, describable by $H = sM \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$, telling that the mapping between two views is described and determined by two dimensions of rotation and the pose of the projected point in relation to the projector. This homography is used to describe the mapping between each pixel in the source image and the resulting projected image, so that for every pixel in the source p_{src} there is a mapping to a pixel in the projected image p_{dst}.

$p_{dst} = Hp_{src}$

where p_{src} and p_{dst} is described with their coordinates $p_{src} = \begin{bmatrix} x_{src} \\ y_{src} \\ 1 \end{bmatrix}$ and $p_{dst} = \begin{bmatrix} x_{dst} \\ y_{dst} \\ 1 \end{bmatrix}$.

2.6 Perspective correction

There are basically two ways of correcting the perspective of an image projected from a projector onto a planar surface, corner pinning and perspective transformation. Both these techniques are used in the programs, and both have pros

and cons. Corner pinning is a basic transformation where the user inputs where the corners of the output is compared to where the inputs are, and the image is stretched to fit the new corner positions, regardless of proportions of the original image and whether or not the placement of the corners is logically sound. Perspective transformation takes the image and through the use of a homography manipulates any of the six axes, while having the ability to maintain the original ratio of the image. This is because all outputs are bound to be logically correct, for every manipulation done with the image, there is an orientation, as a combination of scale, rotation and pose, between the projector and the screen that will make the output fit the screen.

The best thing about corner pin calibration is its speed; by simply dragging corners to their wanted location all warp parameters are calculated. The bad thing is that the outputted image does not necessarily maintain the original ratio. This may also be used as an effect if it is wanted, by making objects in the displayed output seem warped compared to the real world. Another bad feature, unless being kept under control, is how warping is done if corners are placed incorrectly. Because the warping is done regardless of how rotation and projector placement is done in the real world, the output may be images that are impossible to project. Figure 4 a) and b) shows the inputted corners and the transformed output, it is clear to see that this transformation is of no use. Figure 4 c) shows another warped output from corner incorrect corner inputs.

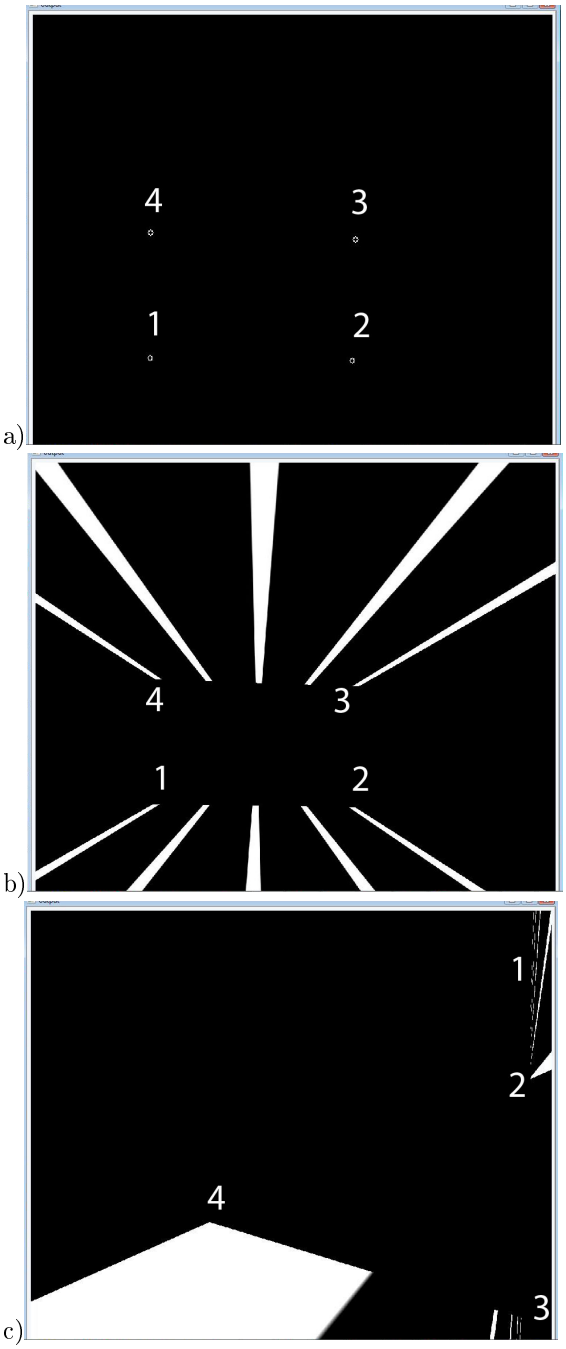

Figure 4: Corner pinning inputs and outputs that produce unwanted warping

Because such outputs are not wanted, I found a way to recognize the warp parameters that produce unwanted outputs, presented in section 3.6.1 on page 33. These are ignored, if the user tries to input such incorrect data, the default parameters are put in instead, the user may choose to input the corners again to correct the faulty corner placement. However, allowed inputs still include all possible orientations of the image, including rotation and inverting.

The perspective transformation is more logical, during a calibration and image setup, parameters are adjusted one at a time to give a correctly warped output image. As this is more detailed work to do from scratch than corner pinning, it becomes somewhat more time consuming, but after the initial calibration, the image should have correct proportions and the final output is highly accurate[3]. As the tests in parts 3.6.1 and 4.2 show, the calibration is fast, especially if the projectors are positioned close to optimal and the perspective transform just does the final few adjustments.

2.7 Calibration techniques

The automatic calibration of projectors usually uses either cameras or sensors to find the screen area. For this project, a hybrid solution is also proposed and discussed.

2.7.1 Camera based screen recognition

This approach is used by several research projects[53, 49, 39], as it is cheap, versatile and intuitive. It requires a camera, integral to the projector or external, that provides a computer with a video feed of the output screen area. It is vital that the camera covers an area greater than the projected area, that in turn needs to be bigger than the wanted screen area. Figure 5 shows that

$A_{Camera} \subseteq A_{Projector} \subseteq A_{Screen}$

where A_{Camera} is the area covered by the video camera, $A_{Projector}$ is the area covered by the projector and A_{Screen} is the screen surface area. The computer needs to get some user input on where in the projected area the desired screen is to be placed and be able to identify all corners of the projected output, so that

[3]The current implementation has some troubles with retaining original image ratio, but the use of a true homography instead of the approximated homography generated by the current calibration, would enable perfect perspective transformation.

a homography can be found and the output from the projector may be warped to fit the screen. By computing the homography between the camera and the projector, the computer can find how this warp transformation needs to be, by taking the view from the projector. Unfortunately, this approach needs a lot of computing power, an external high quality camera and often some sort of user input.

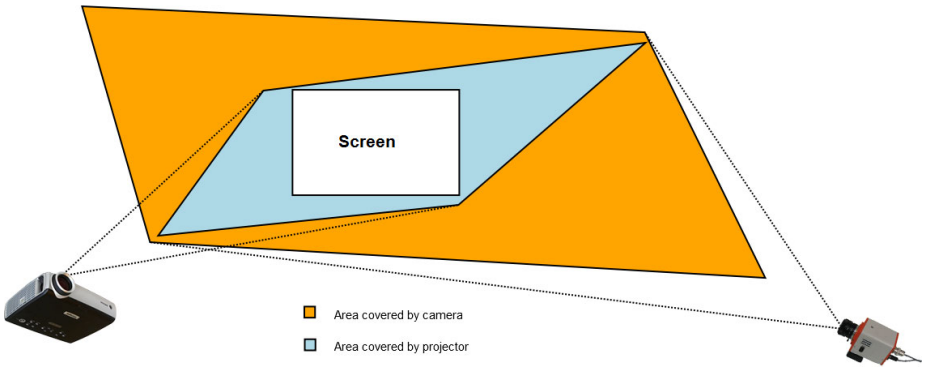

Figure 5: Illustration showing how the areas covered by the projector and camera should overlap

2.7.2 Sensor-equipped screen

Another approach used in some systems, this system relies on integral sensors in the display screen itself. Calibration patterns sent by the projector are identified by the sensors and sent back to the computer controlling the projector, so it can identify exactly where the sensors, and from that implicitly the screen, is placed in relation to the projector. From this the computer can warp the output to the projector to fit the screen. This approach is not as flexible, as it requires sensors to be placed exactly at predefined spots, so one may not make changes to the desired screen area without moving or reprogramming the sensors. However, it is fast, as showed in the demo video [32] by Johnny Lee[46] and potentially highly accurate.

2.7.3 Marked screen camera recognition

The third approach to automatic calibration takes on a somewhat hybrid approach. Most video cameras have possibilities to record IR light. This may be

utilized by putting small, movable IR LEDs at the corners of the wanted projection screen, identifying these on a camera and generating a homography matrix requires no or very little user input compared to the camera based recognition in 2.7.1. It is also more flexible than the sensor-equipped screen in 2.7.2 as the LEDs may be moved around and placed at new surfaces easily. Still, the problems from 2.7.1 with regards to camera quality and placing of the projection area within the view of the camera applies.

2.7.4 Manual calibration

The last, and most widely used, is the manual calibration. This is performed in a variety of ways, from manually moving the projector to a designated area where the projected output is acceptably close to the desired screen size and shape, to manual adjustment directly on the projector, or to manual adjustment via a web interface available on many projectors. The limitations of manual calibration vary from projector to projector, some projectors, like the Optima projectors at the Caruso lab, can only adjust vertical keystone, limiting the options on where to place it for good image quality.

To increase system flexibility, it is possible to add a physical three-axis controller to each projector. Such controllers are capable of adjusting pitch, roll and yaw of the projector, either for use for moving the output screen area or as a rough preliminary calibration. The latter case enables two-tier calibration, where the physical controls may increase actual resolution, as it decreases the need for calibrating three axes in software.

2.8 PC/104

The hardware that was imagined to be used as a basis for this project was a PC/104 computer. The PC/104 consortium[18] specifies an embedded computer standard with stackable PCB cards of compact form factor. The PCB boards, or modules, have to fit within 96x90mm, with bus interconnect to connect other boards above and/or below it. This is used to stack a number of modules on top of each other, modules may have different purposes such as motherboards, graphics, I/O or specialized math coprocessors. One of the newest additions to the standard is the PCIe/104 specification[31], which use the PCIe bus as interconnect between the modules. Modules based on this specification was

thought to be the main hardware running the calibration software, as these computers are cheap, versatile and may be placed almost anywhere. Figure 6 shows a PC/104+ stack, similar to those thought used. It is possible to see the motherboard at the top, with a CPU and a cooling fan.

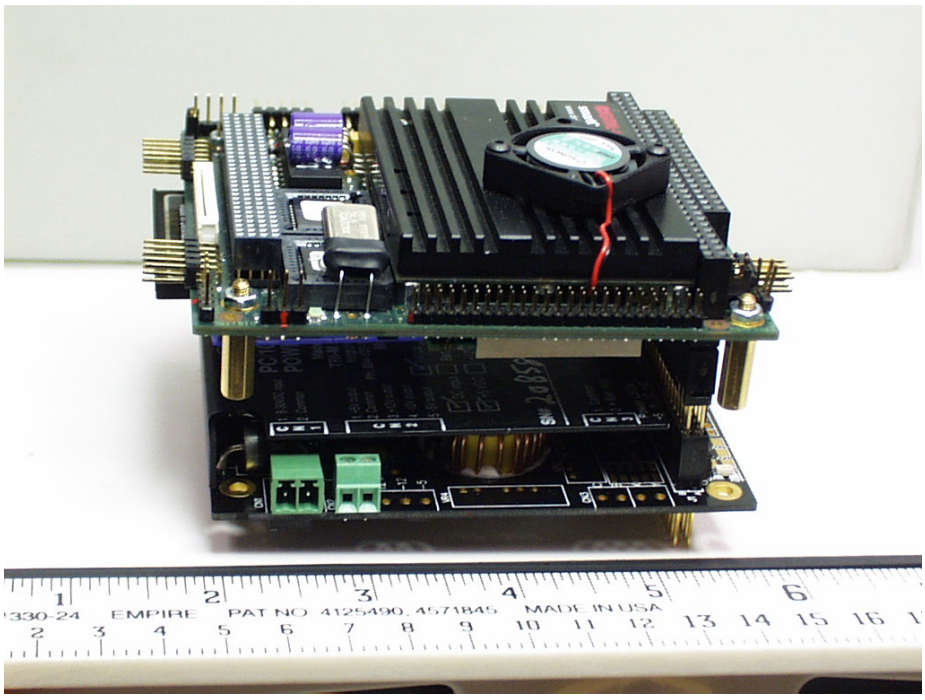

Figure 6: A PC/104+ stack[15]

The motherboard module is based on an Intel Atom processor, that despite its low clock speed and performance, has become popular within many areas, especially those with strict demands to power consumption, heat emissions or cost. The motherboard card, Digital-Logic MSM200XP[40] has in addition to an Intel Atom CPU, 2 GB of memory and several generic I/O ports such as sound, network, USB and hard disk interfaces. By connecting this card to other cards using the onboard PCIe connection, features like an ExpressCard adapter[41] or graphics modules may be added for increased functionality.

MXM, or Mobile PCI Express Module cards are the graphics extension cards for PCIe/104. There are a few different MXM modules available today, this project is intended to run on an AMD MXM card with a mobile version of one of their GPU, graphical processing units. The card intended to be used in the embedded

computer is the Digital-Logic MSMMX104EX[42] card, featuring an MXM ATi Radeon E2400 GPU, with embedded high-resolution video decoding and good 2D/3D performance, supporting both DirextX 10 and OpenGL 2.0[35].

A solution that was considered, was using the Tilera Tile64[54], a 64-core CPU for massive parallel calculations or use GPU with suiting API to accomplish specific parallel tasks. This demanded, however, that both existing libraries and new code was optimized for heavy parallel execution, and while it has been shown that OpenCV can get significant speed bumps by being optimized for many cores[3], this was considered too complex to be done at the current time. Additionally, the Tile64 is new and neither hardware or software has had the time to mature to fully utilize its potential.

2.9 Projector technology types

There exists a variety of different technologies used in projectors, each with their own pros and cons. This is a brief overview of the technologies that are most relevant at the current time

2.9.1 DLP

DLP, or Digital Light Processing is a common technology for home and office projectors. At its core is a powerful light source, an array of micro-mirrors and a color wheel. Each micro-mirror represents one or more pixels in the output image, for every color in the color wheel, each mirror either reflects light from the source through the color wheel, lens and onto the screen, or to a light dump. Lamps are today usually mercury vapor lamps with quartz arc, but some newer models have used LED as light source.

Higher-end models split the light beams into three, one for each primary color, and use a set of three individual micro-mirrors to get better color representation without having the rainbowing problems. Rainbowing is the phenomenon where one may see the individual primary colors instead of the resulting image, this is most common in scenes with a lot of movement and high contrast, higher rotation speeds of the color wheel also helps to eliminate this problem. Light wheels of today may rotate at up to ten times the frame rate of the output image, so the resulting colors may be represented quite good with very little perceptible rainbowing.

2.9.2 LCoS

LCoS, or Liquid Crystal on Silicon is a newer technology with a liquid crystal matrix on each chip instead of micro-mirrors. The chip is covered with a reflecting layer under the liquid crystal, which is similar to LCDs, only that the liquid crystals are directly on the chip. One, or three chips, if using a three-chip system, are illuminated through a color wheel or an array of colored LEDs, and where light passes through the liquid crystals, it passes through the lens and onto the screen. If the light is not to be reflected, it is absorbed on the chip.

As LCoS technology has few or no moving parts, except cooling fans, it is more robust compared to DLP, however the technology is also younger and is not as widespread in use for home users.

2.9.3 LED as light source

As the industry moves from conventional arc, incandescent or gas-discharge lamps and onto using LEDs in more and more places and devices, it is only natural that projectors receive some attention. Still, there are some major issues with using LED lighting in projectors. Firstly, the light intensity required is very high, and at the intensities required by projectors, most LEDs of today are struggling to become much more efficient or have cooler running temperatures than their more conventional counterparts. Their color spectrum may also be troublesome as neither true white or any of the basic colors are represented as accurate as wanted, and color temperature may vary with running time and device temperature.

2.9.4 Laser

Proposals of using lasers as light sources for projectors have received a lot of attention as lasers are capable of giving both the speed, high contrast, resolution, brightness and color accuracy even at very low input power. As light emitted by a laser is homogeneous, colors are always accurate, beams can be controlled fast and precise, and images can travel over great distances without loosing sharpness. The laser projectors work in much the same way as CRT beams scan across the fluorescent coatings of old television screens, beams can either be three parallel, one for each color, or a single that switches color for each

swipe. No current technology puts laser projectors onto the consumer market, although the closely related technology in laser television sets currently available gives hope that it will hit the market within a few years' time.

2.10 Operating system

The calibration software is intended to work on both Linux and Windows computers, so the programming language has to be multi-platform. One of the initial thought was to run RTLinux on the embedded computers connected to the projectors. RTLinux is an operating system optimized for real time operations, where real time interruptions are prioritized for a fully deterministic system behavior[1]. This is behavior needed for stringent real time operations and the demands that Hems lab requires. Tests show that interrupts in RTLinux are handled very fast, close to the hardware limit. As the operating system is very spartan and with few features outside the strict minimum, it is able to comply with strict real-time constraints, and tests[34] show that network jitter is reduced when playing videos in RTLinux through VLC, a free and open video playback client.

As no tests have been performed on computers running RTLinux, it is impossible to conclude that it will be the best solution, however it is likely that it is superior to most other operating systems like MS Windows and most other *NIX-distributions when it comes to handling the real-time event requirements of collaboration surfaces.

2.11 OpenCV

OpenCV[4] was originally developed by Intel to encourage advances in computer vision, giving developers a library of advanced and potentially CPU-intensive functions. The software's prowess was proven among other when the autonomous robot-car Stanley won the until-then impossible DARPA Grand Challenge with OpenCV as one of its core software elements[25, 45, 4]. Today the project is led by Willow Garage, which maintains existing code and recently released an updated pre1.1 beta version in addition to the aging 1.0 stable version[30]. The new version features improved functionality, but also

[4]CV in this case short for Computer Vision

potentially reduced stability. The components of OpenCV is split into several main parts:

- CxCore contains the core functions and data structures
- CvReference is the main section with many advanced image processing tools
- CvAux contains experimental and obsolete functionality
- HighGui contains functions for making and controlling a GUI on top of OpenCV
- MachineLearning is the component that among others is responsible for face recognition and feature tracking

OpenCV has been developed as an open source project over many years, and with a large group of more or less organized developers, it has evolved into a bit of a mess. Its documentation is insufficient in several areas, examples are few, and it makes several undocumented and often erroneous assumptions[17]. Its coordinate system puts the point (0,0) in either top-left or bottom-left corner, depending on the data structure an image is stored in, and it uses both degrees and radians as angle units. In many cases, it seems as a programmer has made a set of methods to suit his or her needs, with names that describes the general functionality, with little description internals or how to use[5]. This often leads to developers having to take the "trial and error" approach to figure out how to use a set of functions, specifically when methods are using data types that may store many types of content. Arrays (*CvArr**), which may represent as different things as images with several different encodings, matrices or a mapping between elements. The code is written in a mesh of C and C++, with a Python wrapper to some of the methods. In many cases the included Python interface causes crashes and it is generally not suited for large projects, so for my project development I used the Google Code project ctypes-opencv[2], another set of Python wrappers for OpenCV, which is far more stable and complete.

Despite of its flaws and shortcomings, it is a good resource, as it has a large library of built-in functions and a few sample applications that shows basic usage of several methods. This library is useful once the initial difficulties in using it

[5]Examples are cvFindHomography and cvPerspectiveTransform.

is solved, as it is powerful, yet quite fast. By using Python, development could be done rapidly, due to Python's ease of use.

2.12 Python

Python is an open source programming language which is based on the notion that it should be close to pseudo-code, and hence, easy to read and write[19]. Its role in this project was to enable quick growth of functionality, as it requires less organizing and lets the developer work more on functionality than debugging a large project. The Google Code ctypes-opencv has memory management integral in its code, so close to all code written for this project is directly functional and towards utilizing the power of the OpenCV library.

2.13 Stereoscopic video in Hems lab

Today, creating virtual 3D on a surface from stereoscopic images is possible with a variety of technologies. The stereoscopic images are created by using two separate sets of recorded images, one for each eye, with slightly different perspectives. Recording video with two cameras separated with approximately the space between human eyes will give the view most suited to creating realistic illusions of three dimensions, by separating the perspectives for each eye during playback, the brain will interpret the video as a 3D scene.

The most widely known technique to separate images is using anaglyph imagery, where stereo images are tinted in two different colors, one for each eye, usually red and green or red and blue. The user then wears glasses where one eye is covered with one of the colors used for tinting the images, and the other covered with the other color. This separates the images, giving each eye its correct perspective. However, because this distorts colors, images have traditionally only been in black and white. On the upside, the two images can be presented on a single screen without any modification or use of multiple projectors.

Using active shutter glasses gives a color proof rendering of the images. By showing alternating frames for the left and right perspective at twice the wanted reproduction frame rate, active shutters in the glasses open and close in order to send correct images to each eye. The glasses needs to be synchronized with the playback to be able to properly time the shutting and reopening, the shutters themselves are made of small LCD screens that alternate between being

black, closed, or clear, open. The use of these screens means that they must be powered, usually by internal batteries, the combination of the screens and their power needs results in shutter glasses being more expensive and heavy than its passive counterparts. Still, like anaglyph stereoscopy, it may be displayed with just a single projector.

The technique preferred in the Caruso lab, which is used extensively by Department of Telematics to demonstrate 3D technology, is the use of polarized glasses. Using two projectors, each with its own polarization filter, the two images are projected simultaneously at the screen, the light polarized orthogonally for the two. By using polarized glasses where the polarization for each eye corresponds to that of the projector sending the images, the two images are separated. It is more expensive to set up due to the cost of dual projectors, but the image quality is better than what is possible using anaglyph separation and the glasses are cheaper and lighter than active shutter glasses, giving the same or better user experience. Figure 7 shows two sets of glasses used for viewing stereoscopic images, anaglyph glasses on the left and shutter glasses on the right. Polarized glasses look like the anaglyph glasses with the colored filters replaced with light gray filters with orthogonal polarization.

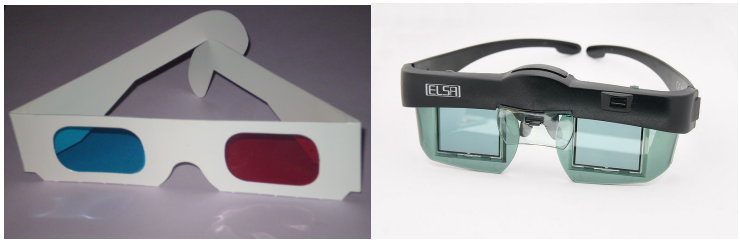

Figure 7: Anaglyph and shutter glasses used for viewing stereoscopic projections

A fourth technology for producing stereoscopic images, has gained attention from scientists and consumers. The use of a layer of lenticular lenses in front of a flat screen LCD or plasma display has the ability to separate stereoscopic images and hit the eye of the viewer with correct images for each eye, giving a stereoscopic effect without requiring the user to wear any glasses. The technology, called autostereoscopy[43], is in need of being refined, as current technology only is capable of giving good 3D effects at a limited number of sweet spots. If users moves from a sweet spot or tilt their head, the effect will diminish or be absent, only by being in one of these spots will the 3D illusion work.

3 Implementation

For the demo application in this task, the goal was to show how the use of distributed control and calibration of multi-projector arrays could be done. The implementation is done in Python, using the OpenCV library with Python wrapper classes. The choice to use this instead of C++ was mostly to be able to rapidly prototype methods and test whether or not given techniques works. Since C++ is not a programming language I am familiar with, it was easier and faster to use a scripting language like Python. There are four different versions of the main program, with different usage areas.

	Single projector	Stereo projectors
Perspective transform	SingleProject.py	StereoProject.py
Cornerpin	SingleProjectCornerpin.py	StereoProjectCornerpin.py

Table 3: The four versions of the calibration program

This chapter documents the specified functionality for the demonstrator programs, and goes into some details on how the program works. In Appendix C the full source code for one of the programs is enclosed.

3.1 Wanted result

The wanted outcome of this project is to design and implement a distributed calibration of multi-projector arrays. This will help the Hems lab in reaching the level of flexibility wanted, and may prove to have several other useful purposes. The distributed model has a purpose in the DMP architecture[47] as different projectors may correspond to different SceneProfiles, and can therefore differentiate their resource usage when it comes to bandwidth use, resolution and refresh rate. As many projectors usually requires a lot of time to calibrate these to fit correctly in accordance with each other during setup, or if equipment is replaced, having a single, easy interface to use for calibration will help in speeding up the setup and maintenance times.

Many projectors still control only a few of the six axes of the output image, in many cases they can only adjust keystone in one axis in addition to zoom. If a separate controller is set up to control the physical pitch, yaw and roll of the projector, this gives two interfaces, and neither is able to control as many

parameters as wanted. The calibration software will give a single interface to all parameters needed to make an image fit onto a screen, regardless of how the projector is placed in relation to the screen.

3.2 Development process

The development process for the calibration software can be split into four parts; Specification, design, implementation and testing. Most of the research of existing solutions were done while the system was being specified and designed, and the implementation and testing were for the most part a continuous iterating loop where newly added code was continually tested for debugging and verification of specified functionality.

3.2.1 Specification

The specification took inspiration both from the research done for the Hems lab and also the existing solutions that had similar tasks. Firstly, it was important to be able to replicate some of the previous results for projector calibration, while at the same time maintaining focus at creating solutions that could be useful for Hems lab. It should be easy to use while being able to perform many tasks like corner pinning, keystoning, rotation, moving and scaling an image, as well as being able to add masks to allow edge blending or stage effects. It should have decent performance, in order to maintain frame rates at an acceptable level, and the code should be easy to read and comprehend, in order to allow further development or let other learn from my results.

The specification called for creating a demonstrator using OpenCV that could calibrate a projector by using warped images and masks. By giving the demonstrator a simple interface, one could research into how the calibration could be done and experiment with using it for instance in stage lighting settings.

3.2.2 Design

There are four versions of the program, as shown in 3 on the preceding page, but they all are based on much the same codebase, consisting of much of the calibration, video loop and supporting methods. The main difference is that the corner pin programs have an extra step in the calibration and the stereoscopic

versions have two outputs and performs the calibration twice, one time for each of them. The programs are single class, written in imperative, functional code meant to be easy to understand, most of the code is self-documenting. Comments have been added to explain the high-level working both for and within all methods to make it easy to read and find specific functional parts. As it is written in Python and uses OpenCV, it is also cross-platform and based on open source projects, something that makes it portable, and easier to port to hardware solutions than had it been made with proprietary software.

As the focus was on creating a demonstrator, little emphasis has been put to design a system with a high degree of safety and reliability in a large scale, but as the core functionality has been extensively tested, it can be assumed that it contains few remaining errors with the existing set of functions. The possible inputs are constrained to a set of recognized keyboard and slider inputs through the GUI, as well as mouse inputs for the corner pin function. This gives enough input possibilities to be able to do all the core functionality, while restricting the input and state information to reduce the risk of undiscovered errors and faults.

3.2.3 Implementation and testing

As the implementation and testing went simultaneously, adding the specified functionality was always followed by extensive testing to verify its behavior. As this was my first work with the OpenCV library, it was also a learning process, to find out how specific parts of the library worked and explore its possibilities and limitations. Working iteratively, the requirements from the specification were accomplished and the design goals were reached. One of the problems encountered was that the image warping and video viewing was not as fast as originally planned. Playing back video in high definition proved to be troublesome, as frame rates were too low for Hems lab. Future implementations and further development can possibly change this, especially if the perspective warping algorithms is optimized to work with a massive-parallel GPU instead of running in the CPU. This could give near constant time to warp an image, given that the image dimensions are not exceedingly large.

3.3 Functionality

The program routine starts with a series of calibration tests. The cornerpin programs start with a blank image where the user specifies where the corners of the screen are located, the perspective transform version loads the latest autosaved presets. Five calibration images are shown, and the user can modify parameters, either via keyboard input or using a set of sliders in the control panels, to warp the images. Only the parameters that are relevant to the specific test are possible to modify. This is a manual calibration procedure that is fast and simple, and all image warping calibrations are done on the computer, so the projector becomes a "dumb unit" with no responsibility for image calibration.

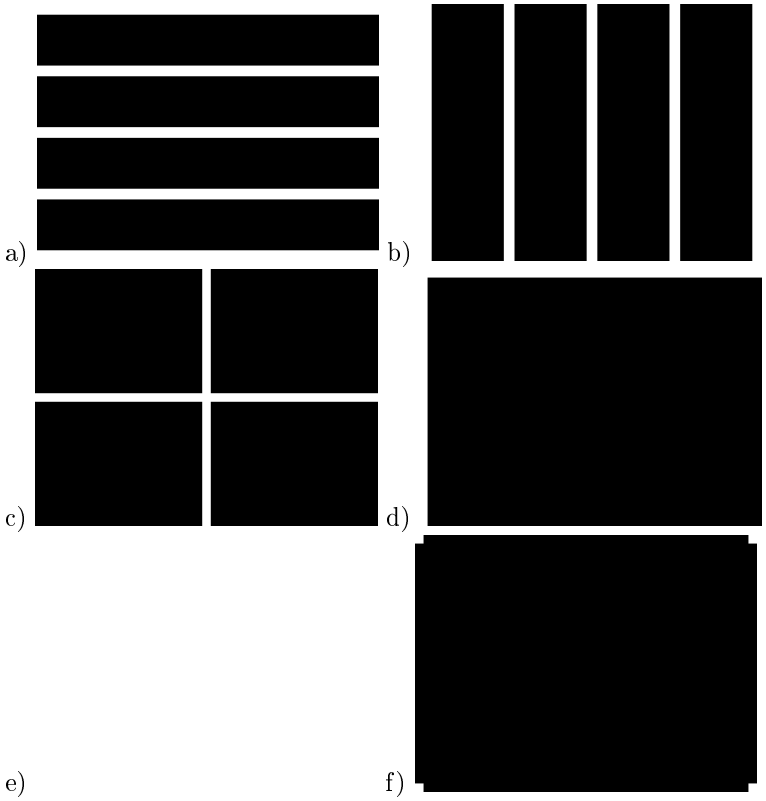

Figure 8: The calibration test images

The images are monochromatic bitmaps in VGA size, 640x480 pixels, 1 bit per pixel, the white lines are exactly 20 pixels wide. This means that the images suffer no compression artifacts and because the width and height of the images

are divisible by the width of the lines, they may be rescaled with no changes in proportions. The vertical black lines in figure 8a) are 135 pixels wide, and the horizontal black lines in figure 8b) are 95 pixels high. The need for highly accurate calibration images became apparent during some early calibration tests, where JPEG-compressed images with lines that were not completely homogeneous proved to cause problems and erroneous calibrations.

Figures 8a) and b) are used to adjust keystone, 8c) is to adjust rotation, 8d) is to place the upper left corner, 8e) is an all-white image that is used to find the correct size for the projector output. Figure 8 f) is an image that is shown by the projector that is already calibrated when using the stereo calibration programs, but it may also be put up by the single projector calibration program to be used as a template for any other projectors being calibrated. This enables accurate calibration of the second image to make it fit the first image in size and orientation.

3.4 Keyboard map

This is the map of keyboard inputs for the projector calibration programs. Unless otherwise indicated, functions apply in all versions of the program.

3.4.1 Calibration

Function	Key
Move image right	a
Move image left	d
Move image up	w
Move image down	s
Rotate image right	Shift + d
Rotate image left	Shift + a
Rotate image up	Shift + w
Rotate image down	Shift + s
Keystone image right	e
Keystone image left	q
Keystone image up	z
Keystone image down	c
Zoom in	+
Zoom out	-
Flip image[6]	f
Print transformation matrix	p
Save current settings	Ctrl + s
Skip rest of calibration	Esc
Load saved settings	Ctrl + l
Load autosaved settings	Shift + l
Jump to next calibration step	Space
Jump back to previous calibration step	Ctrl + b

Table 4: Keyboard map for the calibration part of the program

[6] only for non-cornerpin programs

3.4.2 Main loop

Function	Key
Move image right	a
Move image left	d
Move image up	w
Move image down	s
Rotate image right	Shift + d
Rotate image left	Shift + a
Rotate image up	Shift + w
Rotate image down	Shift + s
Keystone image right	e
Keystone image left	q
Keystone image up	z
Keystone image down	c
Zoom in	+
Zoom out	-
Flip image[7]	f
Print transformation matrix	p
Save current settings	Ctrl + s
Load saved settings	Ctrl + l
Load autosaved settings	Shift + l
Toggle between active frames[8]	Tab
Reset image(s) to default	Ctrl + r
Autosave and exit	Space
Exit without autosaving	Esc
Recalibrate frame / active frame	Shift + c
Putting up reference image for calibrating other instances[9]	Ctrl + b
Pause the video	Ctrl + p

Table 5: Keyboard map for the main loop of the program

3.5 Data rates

For a single wall in the Hems lab, the resolution needs to be at least between 6 to 10 megapixels in actual resolution. In addition, for the video to seem as realistic as possible, refresh rates at 120 Hz or above is needed. No current single projector is capable of displaying the spatial resolution needed, so an array of projectors must be used. The total data rate of this exceeds all current display

[7] only for non-cornerpin programs
[8] only for stereo projectors
[9] only for single projectors

standards. As seen in Table 6, there is a need for a transmission standard that well exceeds today's standards, in order to achieve these data rates today, one must bundle several InfiniBand[12], or 10-Gigabit Ethernet[7] interfaces.

Technology	Resolution	Refresh rate	Color depth	Gross bandwidth use
DVI	2 Megapixels	60 Hz	24 bits/pixel	2.9 Gbps
Dual-link DVI	4 Megapixels	60 Hz	24 bits/pixel	5.8 Gbps
HDMI 1.3	4 Megapixels	75 Hz	24 bits/pixel	7.2 Gbps
DisplayPort	4 Megapixels	60 Hz	30 bits/pixel	7.2 Gbps
Hems requirement	6-10 Megapixels	120 Hz	30-48 bits/pixel	21.6-57.6 Gbps

Table 6: Data rate for existing and needed display transmission standards

In table 6 the maximum gross bandwidth available for the most relevant transmission standards are listed. The refresh rates and color depth are maximum for the given resolution. All these data rates are gross and not effective, overhead is not included. The net bandwidth is a bit higher, this is caused by timing information, encryption and blanking, all contributing to an even higher bandwidth usage. As the resolution of 6 to 10 Megapixels should be the effective resolution, the actual resolution might be even higher, as off-center projectors lose some resolution. This then gives that the data rates would be even higher, demanding new and improved links.

If an array of projectors is used as proposed in this thesis, the data link to each one of them may be kept within current standards, as long as the output image gives the resolution in both space and time that is required for the Hems lab. Dual link DVI can be used at 120 Hz refresh rates at 1920x1080 resolution, giving effectively 2 megapixels, an array consisting of at least three, preferably five or more of these projectors may be used to give the output 6-10 effective megapixels. This then neglects the need for not-yet defined transmission specifications, and by distributing calibration to several individual embedded computers, the computational load becomes possible to handle, as each embedded computer should be able to warp a single Full HD video stream.

3.6 Code guide

The program is split into three main parts, the calibration, the video loops and the assisting functions.

3.6.1 The calibration

This part runs through a five or six-step calibration setup, the sixth is only applicable for the cornerpin programs. As each calibration step has a wanted goal, the only image parameters that are possible to modify are those that will help the user reach this goal. Disabling the use of all other functions helps speed up the process as the user does not need to spend time correcting consequences if an erroneous key is pressed. If at any point the user wants to go back one or more step, to correct any mistakes from earlier in the calibration process, this is possible. In the corner pin calibration, if the user inputs illegal corners, thereby producing outputs like those in figure 4 on page 14, I found that in the 3x3 warp matrix that used as inputs to the *cvWarpPerspective* had values of -1 or less in either position [0,1] or [1,0]. No possible sequence of correct corner inputs would produce this, thereby making it easy to detect and ignore these incorrect values.

The calibration method has its own key listener as other program features available during the calibration not necessarily are applicable during regular video playback and vice versa. The calibration is initialized using *calibrate(IplImage frame, String frameName)*, where frame is a reference to an image frame in the capture and used to find the size of the output, and frameName is the name of the output window to calibrate.

The accuracy of the calibration was tested at the Caruso lab at NTNU, which has two projectors set up to display stereoscopic video. The two projectors were aimed at a screen, and by displaying an identical test pattern on both screens, manual settings and placement was adjusted to get the most accurate settings possible. The adjustment of the projectors took close to an hour, and the result was, according to the technical staff, the most accurate calibration of the two projectors ever. By spending three minutes with the StereoProject.py, I found that one projector was off by two pixels in the x axis and one in the y axis. Some minor keystone adjustment was also needed to get the corners to overlap precisely.

3.6.2 The main video loop

The video loop is actually split in two, one with a frame limiter, the other without frame limiter. The idea is that if the video does not play back fast

enough, optimizations should be made in order to keep frame rate as high as possible. If, on the other hand, frame rate is at correct or higher than intended, one must see to it that it stays as close as possible to correct playback speed. This also means that it is possible to do some more extended calculations or add extra functionality. The accuracy of the frame rate limiter is high enough for most purposes, as shown in figure 9. With target frame rate of 60 frames per seconds and an average of 60.02 frames per second after over 10.000 frames and deviances rarely outside $\pm$ 2 frames per second, this is good enough to stop tearing and retain good synchronization with the audio.

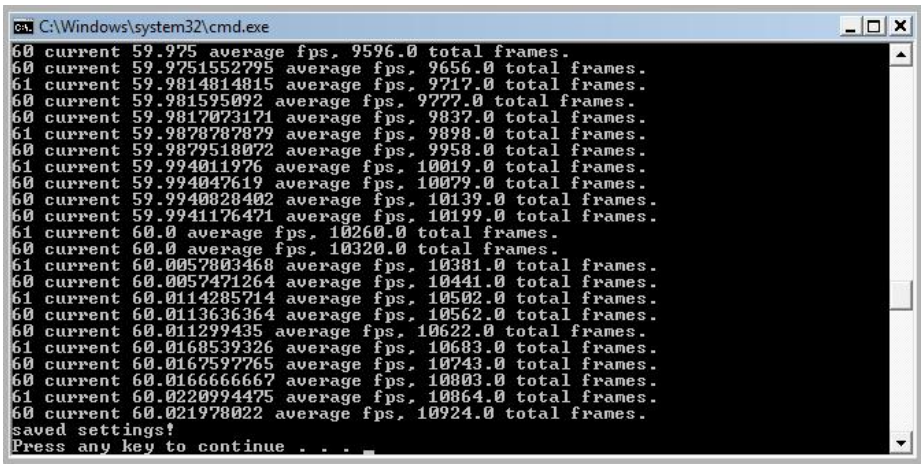

Figure 9: Average frame rate after over 10.000 frames

Within the video loop, some optimizations have been done, for instance is masking and rotation only performed when they are needed. This gives a good speed bump to the frame rate, as does putting the handlers for key input and control panel sliders in separate functions that only are called when needed. Figure 13 on page 43 shows the control panel for the two screen outputs of stereoProject.py, note that rotation, masking and flipping is only available on Control Panel 1, but are effective on both images.

3.6.3 The supporting methods

There are a number of support methods

- *update1/2(int sliderPosition)* updates the 3D transformation matrix with

position, keystoning and size, and can rotate the image ± 45º.

- *flipping(int sliderPosition)* updates the 3D transformation matrix, flipping the image horizontally and vertically. If two outputs are used, both are flipped simultaneously. As orientation comes implicit in the corner pinning programs, this is not included in them.

- *maskAndRotate(int sliderPosition)* calculates the intrinsic matrix for rotating the image 90º, 180º or 270º and can create a mask that may be added to the output video.

- *backup(boolean manualSave)* saves the current settings, either as an autosave or as a manual save. The settings are appended to a text file, so all previous settings are stored unless manually deleted in the log file. Manual saves are denoted in a log with the text "Manual save:" on the preceding line.

- *load (boolean autoSave)* gets the last autosaved configuration, which is the last line in the log if autosave is true, and gets the last manually saved configuration if autosave is false.

- *calibStringReplace(String calibString, int value, int place)* is a support method for replacing a single value in a given place, in a configuration string that contains 18 values.

- *keys(int keyPressed)* determines actions to be performed when a key is pressed in the video loop. keyPressed is the value that comes from OpenCV's key listener. If the return value is -1, the program will terminate.

- *mouse(int event, int x-value, int y-value, int flags, void* param)* is the default method being called by the OpenCV mouse listener, used in the cornerpin programs. It adds the given coordinates to an array of cvPoints. The input values 'event', 'flags' and 'param' are not used, but as the internal mouse listener is set up to do this method call by default, they must be included.

- main, program start, initializes the program, it gets the video sources, opens the control panel and video windows, starts the calibration and runs the video loops. It also destroys all windows when the program shuts down.

The position parameter in the first three methods are not used by these methods, rather they acquire the information they need by other means. The reason they are included is that the sliders in OpenCV's HighGUI classes are specified to call a given function with the position of the slider only parameter by default. As values are changed both by keyboard and sliders, and the methods are used by several different controls, the slider value is of no use, as it is not known which slider the position value belongs to.

3.7 Control latency

When calibrating and adjusting the image warp settings that are applied through the software, some delay occurs. At current, the software checks for key input events every 10 or 100 milliseconds, depending on whether it is in the loop with frame rate limitations or not. As ITU-T G.114 states, delays of up to 200 milliseconds is acceptable with voice communication and other information services with similar demands to controlling latency[14]. Tests on the programs where the polling delay for keyboard events were changed showed that a polling delay of 100 milliseconds is barely noticeable, whereas it becomes more apparent if it is increased to 200 milliseconds and beyond. This shows that control latency is far less important than maintaining frame rates at acceptable levels, at 60 frames per second, if the control latency is 100 ms, 6 images are shown in worst case before any change is done. Still, this is found to be within the limits of what is acceptable, this latency is also similar to the experienced latency in other similar tasks, like changing the TV channel.

3.8 Loss of pixels when warping image

When an image is warped, the effective resolution decreases. This is because several pixels are left blank and thus unused in the output image. The more askew the projector is compared to the screen, the more pixels are lost. Figure 10 shows how much resolution is lost, the angle between the two rectangles is 12.1°.

This leads to two different options, ideally placed projectors, or using more projectors to project the image.

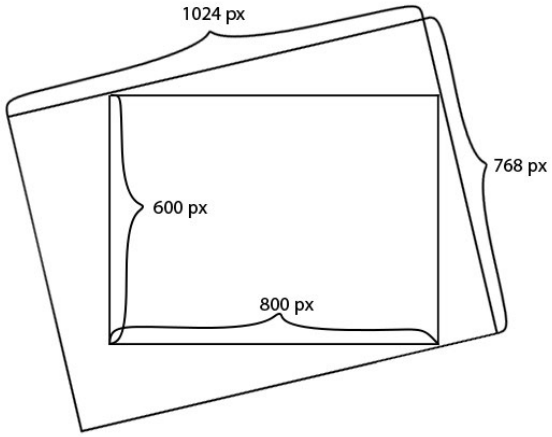

Figure 10: A SVGA resolution frame inscribing the area covered by an XGA resolution frame

3.8.1 Use of centered and ideally placed projectors

If all projectors are placed at their ideal placement, directly in front of the screen, one may fully utilize their resolution. However, this causes problems when the ideal spot for the projector is the same spot that most users would like to use as viewing spot, or would require a lot of space if the projector is placed behind the screen. In the Hems lab, the placing projectors within the room that is used for cooperation would decrease the level of realism and the usability for the lab. Placing them behind the screens is an option, but this requires a lot of space. Finding the correct placement and carefully calibrating the projectors may be time-consuming, as this is a manual process with high demands for accuracy. Figure 11 shows a projector placed in the celing directly above the viewer's head for optimal image quality.

Figure 11: A projector placed directly in front of the screen for minimum image distortion

3.8.2 Use of higher resolution projectors to compensate

As my work intends to show, using projectors that are placed in a less suited but also less disturbing spots, and using more projectors or projectors with higher resolution, it is possible to get the same image quality as ideally placed projectors, with increased flexibility and less space being used. The system will also be easier to maintain and reorganize if needed, as one simply can recalibrate the projectors to fit any new configurations. Because the pixel density in the far areas of very keystoned images is low, these areas should not be used in settings where high resolution is key, such as the Hems lab. Instead, by using masks, one could remove these from use, and use another, better placed projector to cover that area. This of course, assumes that the image is warped in a way that the pixel density of the source images in the near areas are smaller than the pixel density that the projector can provide. Figure 12 shows three projectors using the closest areas to achieve greatest pixel density total. The gray areas are blanked out by masks, while the light blue are used to project images onto

the screen.

As the overhead and loss of pixels requires that more projectors are used will lead to a greater system cost initially, but the system will require less configuration time as the calibration is automatic or semi-automatic. In this sense, the initial cost may be justified and repaid in lower maintenance time.

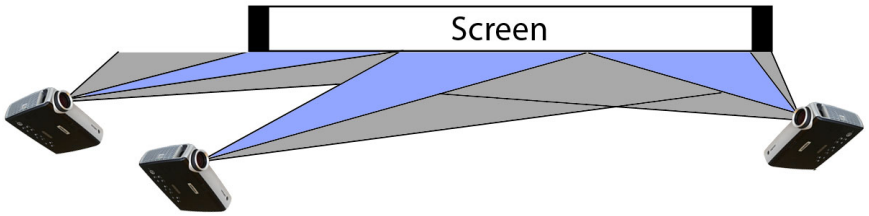

Figure 12: Top view of three projectors placed at high angles to produce a single high resolution image

3.9 Example of use

This calibration software has several uses, the most important is for Hems lab, where correctly calibrated projectors are vital to provide high enough quality for the video wall to work as a virtual collaboration surface. It may also be used to quickly and easily connect a projector to a computer, aim the projector at any suitable surface area and within seconds have correct keystoning on the output video. If two projectors are used to project stereoscopic video, this is also easy to calibrate for, by using one of the stereo versions of the program.

The work flow when setting up an example system would be

- Place a projector so the output covers the designated screen area. If stereoscopic output is wanted, place the other projector so it also covers that area.
- Repeat this step for all screen areas that should be covered by the array.
- If an external camera is used and camera calibration is developed, place the camera so it can see the entire projection surface.

- Start the calibration for each projector and run it through.
- Ensure that the calibration is correct for all projectors in the array.
- The high resolution display wall made through the distributed calibration of a multi-projector array is complete.

Appendix B has a step-by-step guide on how to set up the calibration and use the software with image examples.

3.10 Pseudo code

The demonstrator code is fairly easy to read, as it is written in imperative Python, but a short explanation in pseudo code will give a brief overview of the inner workings.

Listing 1: Pseudo code of the demonstrator program

```
Program start
    Get video stream
        either from given input or from webcam
    Find frame size
    Locate and Resize masks to frame size
    Create and Initialize ControlPanel window
    Create and Initialize Output window
    Calibrate Output window
        if Corner pin
            Get Corners
        Get keystone parameters
        Get rotation parameters
        Get size parameters
        Get location parameters
    LoopVideo in Output window
        if framerate < 60 frames per second
            LowSpeedVideoLoop
                While true
                    Get frame
                    if mask
                        Mask frame
                    if rotate
                        Rotate frame
                    Warp frame
                    Display frame

                    Get keyboardInput and perform instructions
                      accordingly
                    Get framerate
                    if framerate > 60 frames per second
                        break and enter HighSpeedVideoLoop
```

```
31
32          else
33              HighSpeedVideoLoop
34                  While true
35                      Get frame
36                      if mask
37                          Mask frame
38                      if rotate
39                          Rotate frame
40                      Warp frame
41                      Wait until current frame rate = 60 frames per
                          second
42                      Display frame
43
44                      Get keyboardInput and perform instructions
                          accordingly
45                      Get framerate
46                      if framerate < 60 frames per second
47                          break and enter LowSpeedVideoLoop
48      Destroy all windows
49  Program end
```

3.11 Speed and frame rate issue

OpenCV is quite picky when it comes to video formats. Preferably uncompressed AVI should be input video, however this is supposed to change in the 1.1 version. It is also possible to include ffmpeg, a stand-alone video codec, when building, but this is ridden with problems[20].

The transformation that warps and keystones the image is slow. This is because the mathematics behind it is quite heavy, and it may also be because of poorly optimized code for these kinds of transformations.

By doing the transformation with a graphics-optimized API like OpenGL or DirectX, the issues with maintaining the frame rate at an acceptable level would be much less severe. As many of the translations that are done when rendering 3D graphics are similar but inverse, and the transformation is invertible, using a graphics API instead of or in addition to OpenCV could increase frame rate to an acceptable level.

By using other video formats than uncompressed AVI, and a decent video decoder, it would be possible to play back more high resolution videos. At current, the data rates for the videos played are very high when playing videos in high definition, two megapixels resolution at 24 frames per seconds, 1080p@24, and using 10 bits per color, 4:4:4 subsampling reaches over 190 megabytes per

second[26], increasing the frame rate fivefold to 120 Hz would also give a fivefold increase in bit rate. At close to 1 GB/s input bit rate, not counting the audio or looking at resolutions above two megapixels, the medium access, whether using local disk array or an Internet connection needs to be much faster than what is achievable today. If this is multiplied up to the resolution wanted for the Hems lab, it is clear that some compression, either lossless or with no conceived loss of quality, should be used in order to decrease the bit rate to some extent.

If a full system is implemented, a hardware solution of the image transformation should be made. Because the core functions of OpenCV is implemented in C and C++, it should be possible to make a version in VHDL to realize it on an FPGA. The main reason the transformation is slow, is because the warping is done on a pixel-by-pixel basis. This is not optimal for being done in a CPU, which is better at doing heavy or sequential mathematical operations, not simple, but parallel calculations. By utilizing this parallel nature of the transformation, one could increase the speed and frame rate of the video to the wanted level by performing the calculations on specialized hardware implemented on an FPGA.

Tests have proven that interleaving high- and low resolution images will give close to the same user experience as using only high quality images[36]. Possibly, combining this with a good video compression algorithm as h.264 or Dirac, the combined bit rate may be possible to decrease in order to reduce the extreme bandwidth needs that Hems lab today have.

4 User manual

This section discusses the implemention and describes how the program is intended to work.

4.1 Control setup

When calibrating two monitors, having a screen to monitor the control parameters, source material and output messages will for most user be practical. This calls for the use of three screen outputs, two for the projectors that are to be calibrated and one for control. By having a separate screen for control, it is easier to maintain full control rather than having to change focus between the calibration screen and the control panel on a single screen. Figures 13 and 14 show the control panel with sliders, the source images and the text output. Having all these on a single screen gives good control of the warped outputs.

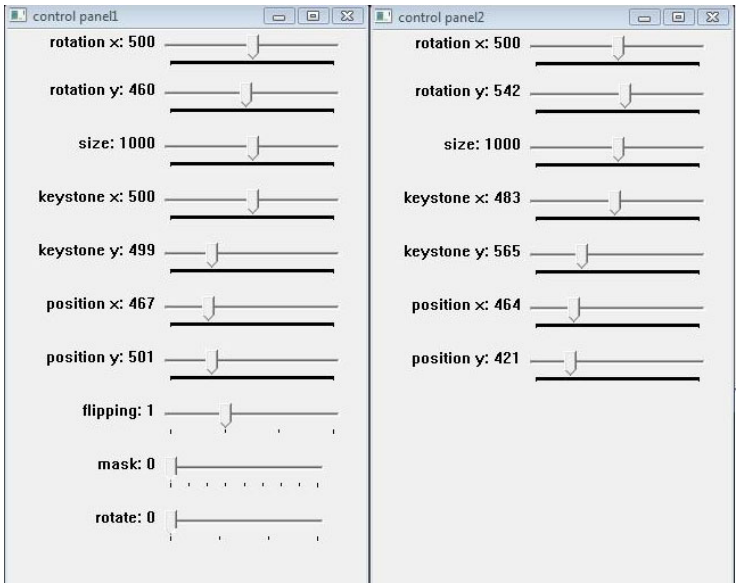

Figure 13: The control panel interface

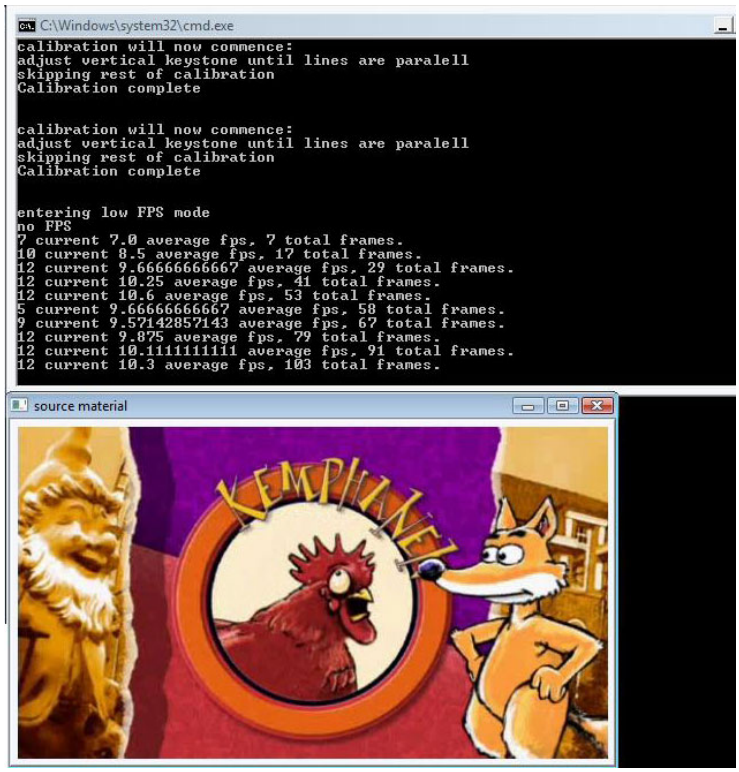

Figure 14: The source material and command output windows

4.2 Usage example

This test shows how easy setup of a single projector can be done. An XGA-resolution DLP projector was put up at an angle and pointed to a screen, and by using both perspective transformation and corner pinning, the image was calibrated to fit perfectly onto the screen. Figure 15 shows the area covered by the projector. The screen intended to be used is the rectangle formed by the three wooden sides of the standing screen and the green tape that runs across it.

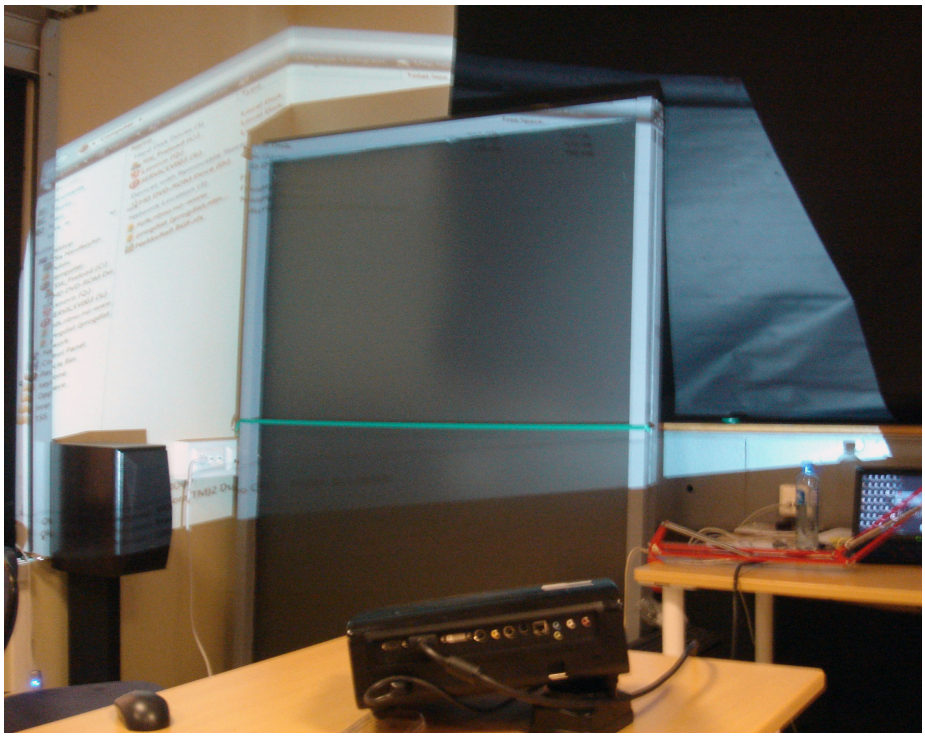

Figure 15: The area covered by the projector

By placing the projector off approximately 45° in the x plane and 20° in the y plane, and rotated approximately 10°, the result is that the projector output is askew and is for all practical purposes useless for any presentations unless it is properly calibrated. Figure 16 shows a closeup of the projector to illustrate how much it is rotated.

Figure 16: A closeup of the projector

Figure 17 shows the outline of the projector output, and shows the blank screen that is used for corner pinning. By using mouse input, the user can click on where in the blank screen the corners are to be placed in the warped images. Because corners are placed in an ordered way, the warping automatically compensates for any rotation or inverting of the image, for example if the projector is placed on its short edge or behind the screen instead of in front.

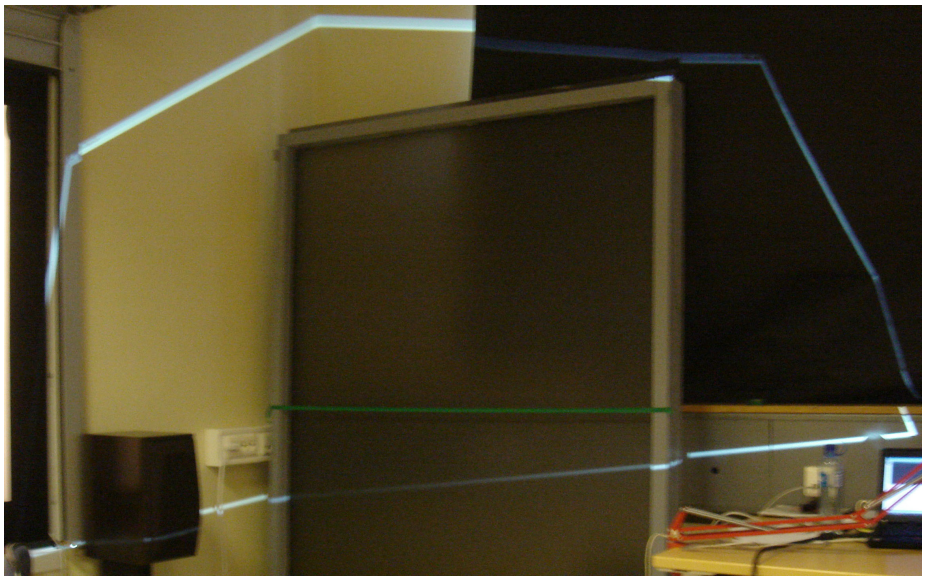

Figure 17: Beginning of the corner pinning

After defining the corners, some calibration images are shown so that the user can ensure the proper fit and orientation of the image. Figure 18 shows the first of these images, showing that the initial fit is nearly perfect. Usually, the only improvement needed is to correct for any inaccurate mouse input.

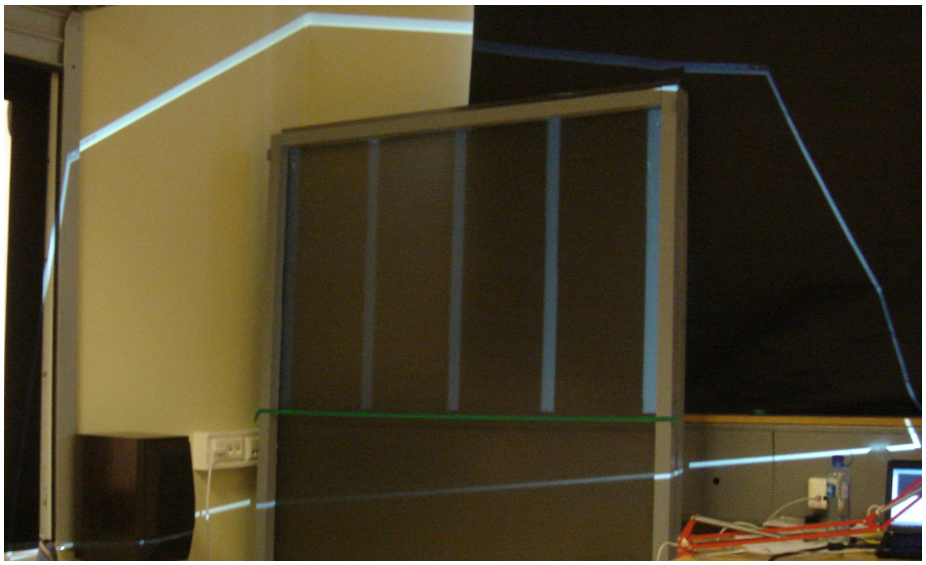

Figure 18: Standard calibration after the corner pinning

If perspective transformation is used instead of corner pinning, the calibration starts with the same image as used in Figure 18, but the starting point is the last used configuration. In Figure 19, this configuration is far away from being optimal. In this case, the user must use keyboard input or sliders on a control panel to make the lines parallel, rotating the image and finding the optimal size is the focus for later calibration screens.

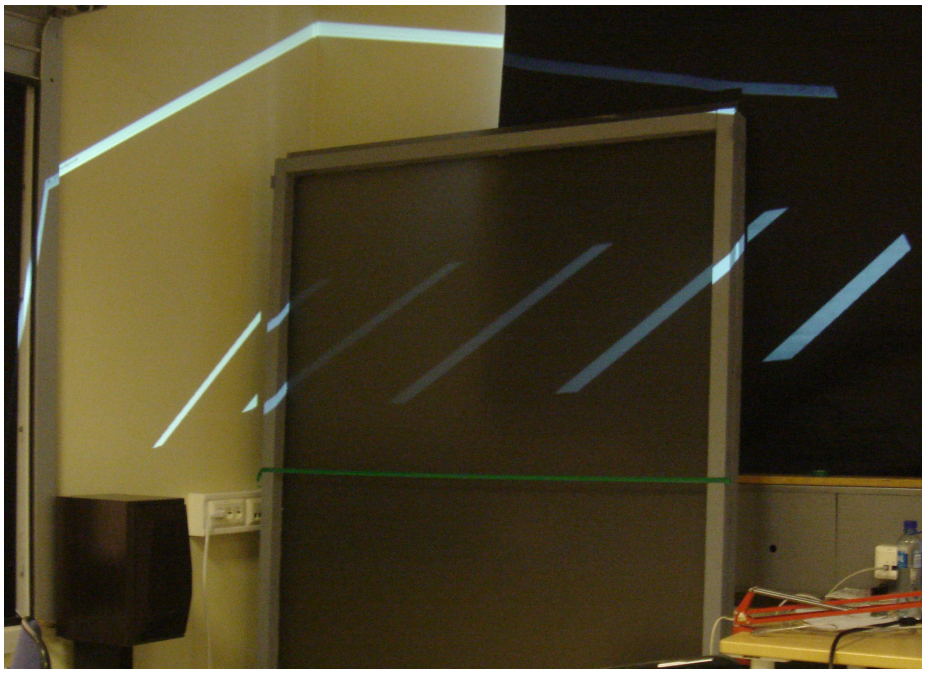

Figure 19: Start of the normal calibration without corner pinning

In Figure 20, the resulting output image fitting on the display screen is shown. The actual time spent to calibrate this image was between 20 and 30 seconds. Close inspection showed a near-perfect fit to the screen, the distance between the outer edge of the image and the frame was no more than a single pixel, except in the area at the far right where the wooden frame surrounding the screen makes it impossible to hit the screen.

Figure 20: The output video after calibration

By taking a screen shot of the warped output, it is possible to see how the image is warped. Figure 21 shows how the warped image looks when viewed on a monitor, and gives some insight to how much adjustment that is needed to correct the offset between the projector and the screen.

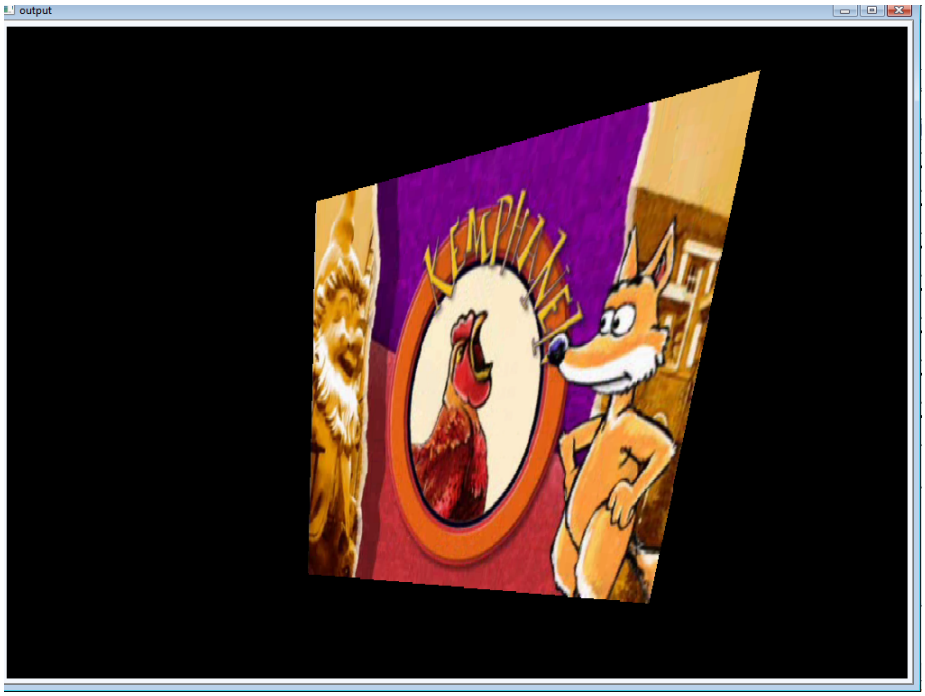

Figure 21: Screen shot of the monitor output

The output in figures 20 and 21 is created by using corner pinning. If perspective transformation and homographies had been used, the result would be much the same, but the resulting image on the screen would have retained the original aspect ratio of the video, which is 16:9. The screen area used an arbitrary ratio close to 4:3.

4.3 Calibration comparison

Running a corner pin calibration is very fast. Start the software, click on the corners, then ensure that keystoning, rotation and size is correct. In a simple test, two candidates were shown how the control works, and on first try, both managed to go through the calibration process with an accurate result in under thirty seconds. The projector was initially placed in the same spot as shown in Figure 15 on page 45, requiring adjustment in all six axes in order to properly fit the screen.

In the calibration test, the two candidates were shown how to use both corner

pinning, perspective transformation and using the built-in keystone adjustment on the projector and moving it around in the room. Then the candidates were told to calibrate the image to fit onto a screen as accurate as they could. The result showed that using corner pinning, both the candidates were able to get a satisfiable result after just 30 seconds, if using perspective transform the candidates used in average 8 minutes to finish the calibration.

When told to use the built-in keystone function on the projector, the candidates first spent some time to try to adjust the image, but after realizing that the keystone correction could not cope with the angles presented, they moved the projector around until they found a good spot to put it. One candidate concluded that it was possible, although hard, to place the projector in a good spot where it was possible to adjust the image to fit, the other ended up showing that by standing straight in front of the screen, holding the projector, one could easily make the image fit. However, when faced with the question of how practical the solution was, the candidate was quick to admit that it might not be the best permanent solution.

This proves my point, even untrained users quickly are able to calibrate a projector using corner pinning, however that the perspective transformation and manual calibration is much slower. My personal experience show that with some training, the time to achieving a good calibration result with both corner pinning and perspective transformation can be more than halved by a user familiar with the system, compared to the first-time experience of the test candidates.

4.4 Stage lighting

Using projectors to create effects on theatrical and musical stages is a field currently being explored. Current advanced lighting technology allows for moving lights with different set patterns, refracting light prisms and uniform colors, but not much more. Replacing this expensive lighting equipment with bright projectors on rotating ball joints will allow light technicians not only to recreate these effects, but also to project videos and images, warped to correct perspective, as well as giving greater opportunity to make small and accurate adjustment for each individual show. In Trondheim, the stage at Arbeiderforeningen was recently renovated, in this connection they looked into using a number of projectors to cover the stage in light, as both a compliment and partial replacement of traditional static lighting. Other stages, like Opera Company of Philadelphia

has already had some experiences with this[28], and are largely positive to using projectors to create backdrops of both still and moving images. Some of the possibilities that were looked into at Arbeiderforeningen was integrating projectors in all the surrounding walls, roof and floor of the stage, creating a much more discreet setup than the huge light rigs used for most stages. Also, by having projectors that could be moved and deployed anywhere, and having a modular stage, highly diverse stage setups could be used with rapid changes. Examples of use could be eliminating traditional stage sets, replacing them with projection surfaces, or creating effects that opens the use of new areas by reaching beyond the edge of the stage.

Depending on the need, the projectors can be used for several different tasks. Using uniform colors and adding a circular mask will simulate a static lighting fixture, in theory multiple fixtures can be replaced with one projector using masks with several circular holes that simulate several different spotlights. The major issue with this technology at current is the lack of brightness, a standard PAR64 spotlight, used as a general workhorse in most stage light setups, has a maximum output of 1 kW, on most medium-sized stages these are used by the dozen. Most consumer projectors of today still requires low ambient lighting to provide a bright and vivid picture, so in order to fully utilize the possibilities of using them as stage lights, projectors need to become much brighter that they are at current. Both LED and laser technologies can increase brightness, getting satisfactory results when using either a combined setup of projectors and fixtures or full replacement still requires some more years of maturing for the technology.

By being able to project images at stage sets, these can be very general surfaces, but still be vivid and give great flexibility. One could imagine a large, single colored wall on a stage where a play is performed, one or an array of projectors could create an image of a medieval castle wall on it. If, at a point in the story, the castle is to be attacked, the projected image could change from a normal wall to a wall with severe damage. Other possibilities is to give the audience shutter- or polarized glasses and use stereoscopic projection to create backdrops or stage sets with depth and use 3D video to give the complete illusion of the actors being in another place and time. The quick calibration and masking functions in the software I have created, allows directors and technicians to experiment and take use of such effects fast and easy, opening possibilities to unfold their creativity.

4.5 Complete system design thoughts

When designing and planning for the software made for this thesis, an outline of the finished system was made. It specified that a number of projectors should be connected to its own embedded computer with some computing power, but little or no storage. Video sources were to come either from a server with a large RAID array, from a streaming source over the Internet or a combination of these two. These embedded units were to control the calibration and warping of the images sent to the projectors, and they should be accessible either through direct keyboard input or over a robust connection, such as MIDI or DMX512 connection for calibration control and synchronization purposes. Alternatively, dedicated hardware on an FPGA chip or a purpose-built hardware implementation could provide higher frame rates and lower power use, but such hardware has not yet been designed. Figure 22 shows an overview of the elements in the imagined future system.

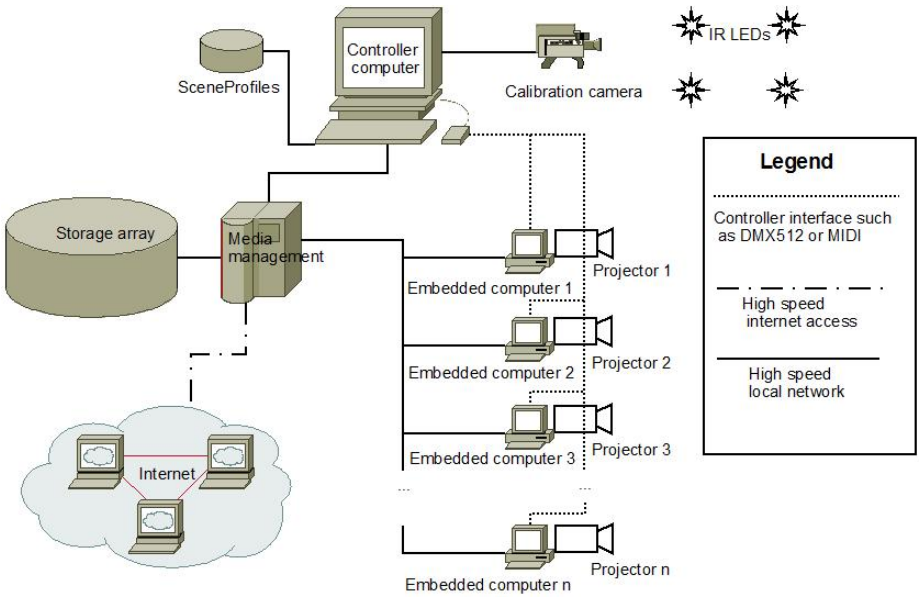

Figure 22: An overview of the system

4.5.1 The projector

In a finished system for the Hems lab or as part of stage lighting, one would want the projectors to fulfill certain demands

- Lossless keystoning and image warping
- Two inputs, image data and control information, using robust and standardized interfaces
- High to ultra high resolution
- Very high display brightness and contrast
- Embedded computer inside the projector
- Calibration camera integral to the projector
- Low power usage and heat emissions

As seen in table 1 on page 8, the digital keystone correction reduces effective resolution rapidly for increasing angles. An optical solution is therefore more suited to achieve the greatest possible resolution and realism. Using optical tilt and shift lenses, possibly controlled over an I^2C bus, one could achieve in much the same manner as now control over the keystoning and warping of the image. The control signals could be sent as a separate channel, as the demands for these signals are different to the demands for the video data. Neither timeliness or high capacity is important, latency of between 10 and 100 ms is easily enough for the control signals.

In the test application, the keyboard input was a three-digit integral, and as the total of different control inputs total are less than 32, it is possible to encode each control signal with as little as 5-bits. The minimum keyboard input latency is 10 ms, thus the maximum control bandwidth needed to transmit calibration information to a single projector is only 0.5 Kb/s. However, one would want the possibility of using common control bus to all devices, requiring means to address each individual projector. Looking at existing interfaces for controlling multiple devices with low bandwidth requirements, both the existing MIDI and DMX512 specifications provide the needed bandwidth and robustness, being capable of sending at 31.25 kbps and 250 kbps respectively, and both are widely used for controlling stage light and effects today. Devices using MIDI and DMX512 are both possible to daisy-chain together, and the signal cables themselves may be very long compared to other common control interfaces like USB or FireWire. Figure 23 shows a MIDI cable on the left, and a DMX512 connector on the right.

Figure 23: MIDI[11] and DMX512[10] cables and connectors

As the projectors should be possible to place more or less anywhere, they should be robust and produce as little heat as possible. An ideal situation would be the use of cool-running LED or laser light sources, so that both the color wheel and fans are redundant. By reducing the number of moving parts and the heat produced, the projector becomes more robust, as neither shocks nor dust poses threats the operation of the equipment. Also, by reducing the power needed by the projector, it could be possible to use a Power over Ethernet, PoE, system for power to eliminate the need for a separate power cable. At current, the PoE specification[8] allows for up to 15.4 watts, but it is likely that this is going to increase with the introduction of newer specifications.[9]

4.5.2 The controller computer

The controller computer does not need to have a lot of resources, its main job is to maintain control over the embedded computers in the projectors, keeping the different parts of the distributed system in sync. It can also be used to control the calibration of the projectors, negotiate SceneProfiles with the remote host in a collaboration setting or create homographies between an external camera and the projectors if this solution is preferred instead of an integral camera to each projector. It should be connected to the media management server over a high speed LAN connection, and through it have access to the RAID array with pre-recorded high quality material. All general control of the video projection system in Hems lab should be done through the controller computer.

4.5.3 The embedded computers

The embedded computers should take care of each projector's video stream, by decoding and warping the video stream. By using SceneProfiles, the video stream for a given projector can be routed directly to its embedded computer, without the need to send the entire high resolution video. This reduces the bandwidth needed to each projector, limited by the capabilities in resolution, refresh rate and color depth. The market for projectors with higher specifications than full HD, needing more than the bandwidth available through HDMI or DisplayPort, is mainly for very specific markets, such as high-end digital cinemas. This means that one can assume that a realization of Hems lab would use an array of full HD projectors, where each can have its own high speed data connection.

By using compressed video streams over a network connection instead of transferring the uncompressed image as would have been done by a standard video interface, one would save considerable amounts of bandwidth without losing image quality. This then gives the possibility to use either HDMI, DisplayPort, dual DVI, InfiniBand or 10Gb Ethernet as transmission mediums, depending on how one chooses to implement the network architecture. The interfaces can be split in three groups, the pure video transmission cables, the serial communication cables and the general network interfaces, each with their own pros and cons. If compressed video is to be used, the general network interfaces have the advantage because of their ease-of-use and how the AppTraNet routing scheme fits in the 7 layer ISO OSI model together with Ethernet standards. On the other hand, if uncompressed or partially processed video data is sent, the video and serial interfaces have less overhead and better-working fault finding and correction.

The use of PCIe/104 as the basis for the embedded computers open up a large number of possibilities, as one may design the computers with many different configurations, by including an FPGA card in the stack, it is possible to design hardware specialized for a given task, for instance decoding and warping video, making this very fast operations and neglecting the need for new, specialized video interfaces.

4.5.4 Other system parts

Having a small camera integral to the projector would make it possible for the projector to calibrate itself by calculating and applying a homography between the screen and the projector. Cameras in mobile phones on the market today have cameras with resolution in excess of 8 megapixels[22], with 12-megapixel cameras[21] on the way, the sensors used in these phones could also be used in a projector to provide images of high enough resolution to be able to accurately find the edges of the screen and calculate the homography between the projector and the screen. By using a compact, high-resolution camera and good screen detection algorithms, no user input would even be necessary for calibrating the projector correctly. If, instead of embedded cameras, one or more external cameras are wanted, these should be of very high quality to ensure good calibration results. Advantages of using stand-alone cameras are the reduced cost of having to use less cameras and the possibility to have better sensors and optics in a single large camera than small, integrated cameras, but the disadvantages include the extra work with setting up the cameras and possibly needing to move them before one can use the collaboration lab.

The media management server and RAID array acts as a hub in the system, getting in the media streams either from local storage or from another source over the Internet, and relaying this to the projectors. It needs to handle very large data rates and may need to decode and process some of this information in order to route individual video streams to the correct projector. By using SceneProfiles and DMP's ability to split a scene into sub-objects with different requirements for bandwidth, it is possible to save bandwidth and processing power.

The thought of using small, movable IR LEDs to tell the calibrating projectors and controller computer where the screen is, could help the automation of the calibration, as they can be used as guide point for the cameras. By placing the LEDs at predefined spots in the room, each projector could find out where its projected output was in relation to where its desired output area was, and calibrate itself accordingly. One could also imagine negotiation between the projectors on coverage area, so that any area that can be covered with two or more projectors are covered by the projector that has least distortion, in form of rotation and keystone, and hence the highest pixel density for that area.

5 Discussion

In this section, I will look into the possibilities and limitations, and reflect over how this solution will stand up against other similar systems.

5.1 Applications

The use of the projector calibration demonstrators have shown that quickly setting up an array of accurately calibrated projectors is easy, and may be a good addition to the Hems lab in the future.

All surfaces the projector system is to be used on should be planar in order to achieve correct warping and light intensity. At current, the programs have no support for curving edges or surfaces. In order to achieve this, detailed light intensity maps and curve functions have to be included[53], OpenCV does not at current support advanced warping functions to perform these tasks.

As projectors may be placed at angles that will reduce the final resolution, in order to not lose pixel density, more projectors have to be added with increasing angles between the projectors and screen in the setup. Because of the high modularity in using a distributed system, tiling several images into one should not be a problem. This then means that having a high-resolution video wall can be built using several cheap, simple projectors, and the projectors may be placed almost anywhere. Many will then place the projectors in areas where they are least noticeable, both visually and audibly. As the outputs can be calibrated to fit with each other, using the corner pin method to input corners is as fast for tiling several images as using it to fit only a single screen.

An idea for a future system would include automatic identification of the surface area through a camera. This camera could be either external or internal to the projector, connected either to a PCIe/104-based computer located inside the projector, or at a central location. These computers would be connected to a central controlling computer as shown in figure 22. In this manner, it is possible to automatically calibrate the system as a whole, while still have detailed control over each projector if needed.

5.2 Software alternatives

For calibrating projectors quickly, especially projectors which are placed in awkward spots, this system is very good. For most users, the alternative is using a manual calibration, which is much more tedious and may not prove to be as accurate. The calibration patterns used in the calibration programs have proven to be very good for finding correct settings for correcting images, as the images are created to detect and help correcting the individual parameters needed to get a perfect perspective transformation. There exists some other software for keystoning and corner pinning images and video. Graphics card maker nVidia has for some time had corner pinning in their drivers, called nvKeystone[16]. This software works quite well, and because it is written specifically for certain graphics processors, the warping of the images is done virtually instant. On the downside is it often hard to adjust to correct settings, as the only input is mouse drag-and-drop, and this often moves the wrong corner. It also requires that the user has a GPU made by nVidia. It also has the problems with the loss of actual resolution when warping and keystoning an image.

Another option is the powerful Video Projection Tools[29], which warps, blends and masks video. Both nvKeystone and Video Projection Tools are available for download for free.

5.3 Hardware alternatives

There are already some systems for large video arrays and for automatic calibration of projectors. One of the most famous arrays is the Princeton Scalable Display Wall, with a resolution of 6000 x 3000 pixels, made by 24 individual projectors[39]. It has systems for automatic calibration and warping of output images. This video wall is made with all projectors placed in front of the display, taking up a lot of space, and it is not flexible when it comes to rearranging the surface or placement of the projectors.

Johnny C. Lee describes a system that uses embedded light sensors in the display for automatic calibration[46]. By projecting binary Gray-coded patterns, he finds that it is possible to uniquely identify the placement of the display screen. This gives quick and accurate placement and warping of the image, however the system relies on the sensors in the screen. If the system was possible to purchase for the Hems lab, it would seemingly give fast calibration and good usability

results, however as it is not developed commercially, one cannot test or use this system without developing and recreating it from scratch.

The Virtual Reality lab at NTNU's Department of Petroleum Engineering and Applied Geophysics[13] is an example of how a room similar to Hems lab would work, with stereoscopic views on four walls. However, this is aimed at researching geological issues and training students in the use of visualization technology. The constraints in Hems lab and DMP with regards to QoS, spatial and temporal resolution and with surround sound are quite different from what exist in this lab today, as users would have higher demands for a near-natural interaction with humans than they do when looking at visualized geological data.

By using homographies to find the relation between the projectors and the screen and then finding the overlapping areas, the video array demonstrated by Mitsubishi[55] uses edge blending to create a seamless image from several projectors, with no visible edges in the overlapping areas. This system relies on the use of GPUs to calculate the warping and blending of the images.

The multi-projector array put up in [37] uses a design philosophy not far away from that presented in this thesis by using projectors with an integral computer and camera. They envision the use of several of these units to create mobile and flexible display walls, with distributed calibration and easy deployment.

The self calibrating projector in [50] uses, in addition to a camera, a two-way tilt sensor to detect the projector's tilt. It is intended primarily for single projector use, but give good results by being able to warp an image and project it with correct rotation with no need for physical markings on a screen to give inputs on how the image should be oriented.

5.4 Advantages compared to other solutions

As the system has inherent support for stereoscopic multi-projector setups, these may be used in Hems lab or in other projects completely with no modification. This also means that setting up a 3D demonstration anywhere is faster and takes less time than before, as time previously used to carefully position and calibrate projectors now is redundant. It reduces the calibration time from many minutes for manual calibration down to mere seconds, and in many cases even more accurate.

By making a system based on a distributed set of self-adjusting projectors, only synchronization and simple control information needs to be sent from a controller computer. This fits in with the DMP philosophy of having individual SceneProfiles and having the possibility to route video stream packets based on the physical location in the output image, where video images can be sent with custom bandwidth to a specific projector.

Any number of projectors can be used together, as the warping is performed distributed and each projector is connected to its own computer. This means that the system may be used to control arrays of sizes previously unheard of, and the time used to calibrate will remain linear as long as the shape of each screen is rectangular. This is advantageous compared to other large arrays, especially when the use of cameras will enable self-calibrating projectors. The calibration time for these may be reduced to give close to the same speed for a single projector as an array of a hundred units.

6 Conclusion

By developing and testing a software demonstrator using OpenCV, I have found that making a system for distributed calibration of multi-projector arrays not only is possible, it also provides many benefits by saving time, increasing accuracy and giving more flexible placement of projectors. The software developed during this work is intended to run on an embedded computer either in or in close proximity to the projector, so that each projector has its own computer, and it relies on the use of OpenCV's core functions for most image manipulation. It warps the video output for the projector to fit the projection surface, it can rotate, scale, move and keystone images, giving freedom in all six axes, as well as having the ability to use one image as a mask for another to create new effects. By giving two calibration options, either corner pinning the image, stretching it out to fit the surface of the screen exactly, or by perspective correction with precise adjustment possibilities, users and developers may choose the approach that fits best. By also having integral support for the dual-projector output needed to provide stereoscopic images, the system fits in with the goals of Hems lab, and a realization of the DMP architecture. In the Hems lab, using it can help properly calibrate and set up stereoscopic projections on five walls with much less work than previously thought, and by being able to reconfigure the projector set up often, much more research can be done with regards to projector types and technologies. By being able to rapidly put up, calibrate and test a multi projector array, it also becomes viable to put up a large display wall or use the projectors for stage lighting effects in a large scale.

The use of cameras or other means of further automating the calibration process has not been implemented, but the system has many expansion possibilities to later add this into the system. The most flexible solution seems to be the use of IR LEDs to mark the screen and, at least in the beginning, rely on external cameras to find the placement of these as this is simpler and require less resources than integrating cameras into the projectors. This could make the calibration system automatically recognize the screen area and create a homography between the projector and the screen, giving fully automatic calibration.

The ease of use and flexibility that the demonstrator displays, has never been showed before, neither in hardware or software, the controls are simple and intuitive, an still give great control over the result. By using an open software library and a programming language that encourages code readability, the code

can easily be either ported to other programming languages or to hardware implementation, reducing the issues with low frame rates. By using current standard transmission interfaces, the possibility of actually realizing an entire system becomes more probable, as I have shown, these currently have the bandwidth needed to provide each computer and projector. The speed and accuracy, combined with the vast array of possible usage areas may prove that projector technology is far superior to other screen technologies such as flat panel displays in settings where flexibility or screen size is important, while giving the same levels of image quality.

7 Future work

The demonstrator developed and used in this project has weaknesses in that it does not take advantage of the parallel nature of the calculations being performed. By using either a pure graphics library or by realizing the program on an FPGA, higher frame rates will be possible. This is currently the biggest problem, as the use of high quality video today gives too low frame rate for any use. Also, this could enable the use of other video encoding schemes, allowing compressed video to be used. Appendix D outlines some possible alterations that may increase the overall speed of the program that may be included in further development.

The perspective transformation does not at current work as intended. Because it does not maintain the ratio of the original image and is much slower than its corner pin counterpart, it can be seen as no more than a programming warm-up exercise. If a way to easily find the homography between the screen and projector is found, it will be of more value as the output image will always be true to the original source, in contrast to the corner pin output.

With more development time, camera-based calibration using homographies would be possible. This could automate the whole process of calibration, thereby giving true "plug'n'play" projectors that require no user input to achieve fast and accurate image calibration.

8 Reference list

References

[1] An Introduction to RTLinux. http://www.linuxdevices.com/articles/AT3694406595.html. Last visited 22.05.2009.

[2] ctypes-opencv - Google Code. http://code.google.com/p/ctypes-opencv/. Last visited 22.05.2009.

[3] "CVCell" - Module developed by Fixstars that accelerates OpenCV Library for the Cell/B.E. processor. http://www.fixstars.com/en/company/press/20071128.html. Last visited 22.05.2009.

[4] DARPA Grand Challenge 2005. http://www.darpa.mil/grandchallenge05/gcorg/index.html. Last visited 22.05.2009.

[5] Elliot Scientific - Product - Elliot Gold Series Six-Axis Positioner fitted with High Precision Manual Adjusters. http://www.elliotscientific.com/product.asp?product=178. Last visited 22.05.2009.

[6] How Important is Keystone Correction? http://www.aboutprojectors.com/news/2006/06/28/how-important-is-keystone-correction/. Last visited 25.05.2009.

[7] IEEE 802.3ae 10Gb/s Ethernet Task Force. http://www.ieee802.org/3/ae/index.html. Last visited 22.05.2009.

[8] IEEE 802.3af Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications Amendment: Data Terminal Equipment (DTE) Power via Media Dependent Interface (MDI). http://standards.ieee.org/getieee802/download/802.3af-2003.pdf.

[9] IEEE P802.3at DTE Power Enhancements Task Force. http://www.ieee802.org/3/at/objectives.html. Last visited 22.05.2009.

[10] Image of a DMX512 cable. http://www.jpleisure.co.uk/3PinXLRtoFemaleDMX.jpg. Last visited 22.05.2009.

[11] Image of a MIDI cable. http://musikality.net/wp-content/uploads/2009/01/midi_cable.jpg. Last visited 22.05.2009.

[12] InfiniBand Roadmap: IBTA - InfiniBand Trade Association. http://www.infinibandta.org/content/pages.php?pg=technology_overview. Last visited 30.05.2009.

[13] IPT | VR lab, virtual reality - Institutt for petroleumsteknologi og anvendt geofysikk, NTNU. http://www.ntnu.no/ipt/lab/vrlab. Last visited 22.05.2009.

[14] ITU-T G.114: SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS. http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E&type=items. Last visited 22.05.2009.

[15] Laboratory for Embedded Collaborative Systems (LECS) – Testbed General Overview. http://lecs.cs.ucla.edu/Resources/testbed/testbed-overview.html. Last visited 22.05.2009.

[16] NvKeystone. http://www.nvidia.com/object/feature_nvkeystone.html. Last visited 22.05.2009.

[17] OpenCV - Wiki for RobotCub and Friends. http://eris.liralab.it/wiki/OpenCV. Last visited 22.05.2009.

[18] PC/104 Consortium - PC/104 Specifications. http://www.pc104.org/pc104_specs.php. Last visited: 22.05.2009.

[19] PEP 20 – The Zen of Python. http://www.python.org/dev/peps/pep-0020/. Last visited 22.05.2009.

[20] Problems with opencv and ffmpeg. http://evolving-life.vox.com/library/post/problems-with-opencv-and-ffmpeg.html. Last visited 03.06.2009.

[21] Sony Ericsson - Oversikt - Satio. http://www.sonyericsson.com/cws/products/mobilephones/overview/satio?lc=no&cc=no. Last visited 03.06.2009.

[22] Sony Ericsson - Oversikt - W995. http://www.sonyericsson.com/cws/products/mobilephones/overview/w995?lc=no&cc=no. Last visited 22.05.2009.

[23] Stage Lighting Tech Pages: Combating Keystone. http://freespace.virgin.net/tom.baldwin/keyst_deriv.html. Last visited 25.05.2009.

[24] Stage Lighting Tech Pages: Combating Keystoning. http://freespace.virgin.net/tom.baldwin/keystoning.html. Last visited 25.05.2009.

[25] Stanford Racing :: Home. http://cs.stanford.edu/group/roadrunner//old/index.html. Last visited 22.05.2009.

[26] Storage and Data Rates for Uncompressed Video. http://www.blackmagic-design.com/support/detail.asp?techID=30. Last visited 22.05.2009.

[27] The Caruso Lab. http://www.item.ntnu.no/ leifarne/Caruso/TheLast visited 22.05.2009.

[28] Using Video On The Stage by Boyd Ostroff. http://www.dvinfo.net/articles/production/videostage.php. Last visited 22.05.2009.

[29] Video Projection Tools v3.1. http://hcgilje.wordpress.com/resources/video-projection-tools/. Last visited 22.05.2009.

[30] Welcome - OpenCV Wiki. http://opencv.willowgarage.com/wiki/. Last visited 22.05.2009.

[31] What is PCI/104-express? http://www.pc104.org/pdfs/What_is_PCI104_Express.pdf.

[32] YouTube - Automatic Projector Calibration with Embedded Light Sensors. http://www.youtube.com/watch?v=XgrGjJUBF_I&feature=channel. Last visited 22.05.2009.

[33] THE I 2C-BUS SPECIFICATION. http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf, 2000. Last visited 22.05.2009.

[34] A. Ferone, M. Miralto, A. P. A real-time streaming server in the RTLinux environment using VideoLanClient. In *Department of Applied Science, University of Naples* (2007).

[35] AMD. ATI RADEON E2400 MXM-II Module Product Brief. http://www.amd.com/us-en/assets/content_type/DownloadableAssets/ADVCORK72799_EDG_Launch_E2400_F.pdf.

[36] Berge, H. The Hems Lab - Perceptual test of scene objects with variable temporal resolution. *Semester assignment at Department of Telematics* (2008).

[37] Bhasker, E. S., Juang, R., and Majumder, A. Advances towards next-generation flexible multi-projector display walls. In *EDT '07: Proceedings of the 2007 workshop on Emerging displays technologies* (New York, NY, USA, 2007), ACM, p. 11.

[38] Bradski, G., and Kaehler, A. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, Cambridge, MA, 2008.

[39] Chen, H., Sukhthankar, R., Wallace, G., and jen Cham, T. Calibrating Scalable Multi-Projector Displays Using Camera Homography Trees. In *In Computer Vision and Pattern Recognition* (2001), pp. 9–14.

[40] Digital-Logic. MSM200X/XU/XP Datasheet. http://www.digitallogic.com/fileadmin/dlag/files/ Produkte/Datasheets/EMBEDDED/802370DS_E.pdf.

[41] Digital-Logic. MSMEC104EX Datasheet. http://www.digitallogic.com/fileadmin/dlag/files/ Produkte/Datasheets/EMBEDDED/801756DS_E.pdf.

[42] Digital-Logic. MSMMX104EX Datasheet. http://www.digitallogic.com/fileadmin/dlag/files/ Produkte/Datasheets/EMBEDDED/801725DS_E.pdf.

[43] Dodgson, N. Autostereoscopic 3D Displays. *Computer 38*, 8 (Aug. 2005), 31–36.

[44] Heikkila, J., and Silven, O. A four-step camera calibration procedure with implicit image correction. pp. 1106–1112.

[45] Hendrik Dahlkamp, Adrian Kaehler, D. S. S. T., and Bradski, G. Self-supervised Monocular Road Detection in Desert Terrain. In *Stanford University* (2006).

[46] Lee, J. C., Dietz, P. H., Maynes-Aminzade, D., Raskar, R., and Hudson, S. E. Automatic projector calibration with embedded light sensors. In *UIST '04: Proceedings of the 17th annual ACM symposium on*

User interface software and technology (New York, NY, USA, 2004), ACM, pp. 123–126.

[47] Lie, A., and Ronningen, L. Distributed multimedia plays with QoS guarantees over IP. pp. 12–15.

[48] Optoma. EX525ST-M DLP Projector Product Manual. http://asia.optoma.com/UploadFiles/DownloadFiles/Brochure/Brochure_080814024841.pdf.

[49] Raij, A., and Pollefeys, M. Auto-calibration of multi-projector display walls. vol. 1, pp. 14–17 Vol.1.

[50] Raskar, R., and Beardsley, P. A self-correcting projector. vol. 2, pp. II–504–II–508 vol.2.

[51] Rønningen, L. A. Part 1: Introduction to DMP The DMP system and physical architecture. http://www.item.ntnu.no/ leifarne/TheLast visited 25.05.2009.

[52] Rønningen, L. A. Part 9: SceneProfiles The DMP system and physical architecture. http://www.item.ntnu.no/ leifarne/TheLast visited 25.05.2009.

[53] Ruigang Yang, A. M., and Brown, M. S. Camera-Based Calibration Techniques for Seamless Multiprojector Displays. *IEEE Transactions on Visualization and Computer Graphics 11*, 2 (2005), 193–206. Member-Brown, Michael and Member-Majumder, Aditi and Member-Yang, Ruigang.

[54] Tilera Corporation. TILE64 Processor Overview. http://www.tilera.com/pdf/ProductBrief_Tile64_Web_v3.pdf.

[55] van Baar, J., Raskar, R., Raskar, R., Baar, J., Chai, J. X., and Chai, J. X. A Low-Cost Projector Mosaic with Fast Registration. In *Asian Conference on Computer Vision (ACCV)* (2002).

Appendix

A Source files

An archive of the demonstrators are located at folk.ntnu.no/nordbryh/Diplom/Demonstrator.zip or in electronic versions attached to this thesis. In order to run, OpenCV in version 1.0 or above must be installed and properly set up with its .dll files added to the path.

Unzip the archive and launch with one or two video files as parameters (samples are located in the /video/ folder), or no parameters to use a web cam. The two included video files are sample video files from Microsoft Windows, reencoded to uncompressed AVI.

B Usage examples

This is a step-by-step example of how the software works. For clarity, I started out with a warped output on the display screen and corrected it, taking screen shots for every step. It should be possible to follow and replicate this process closely.

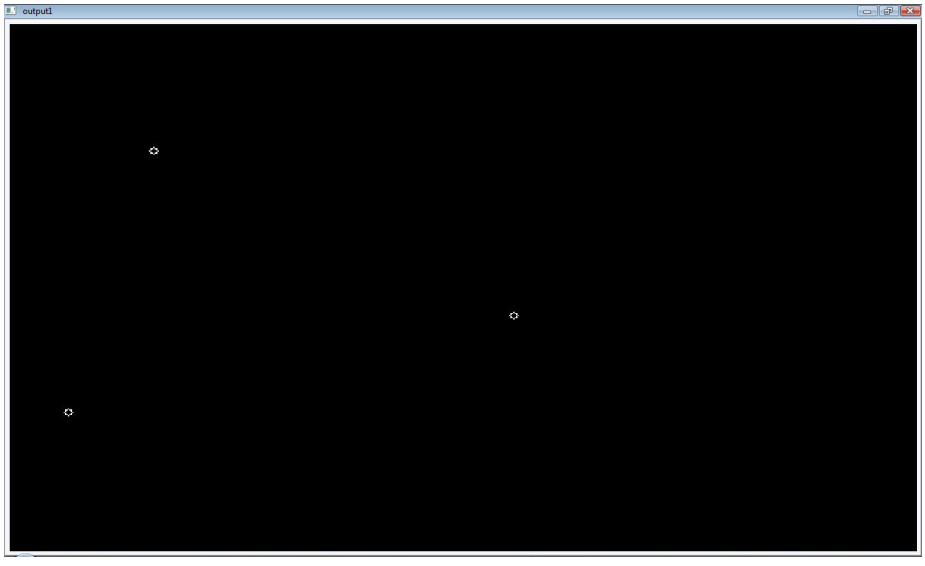

Figure 24: Corner pin inputs

First, four corners are given, in this case they are placed so that the image needs calibration to fit the screen. The corners are given in the sequence bottom left, bottom right, top left, top right, however, on some computers, this needs to be reversed to get correct orientation of the image. As far as I have tracked it, the problem lies in either the operating system or graphics drivers, as all input parameters down to the warp matrix, as well as program versions, can be identical between computers, but still produce different orientations of the image.

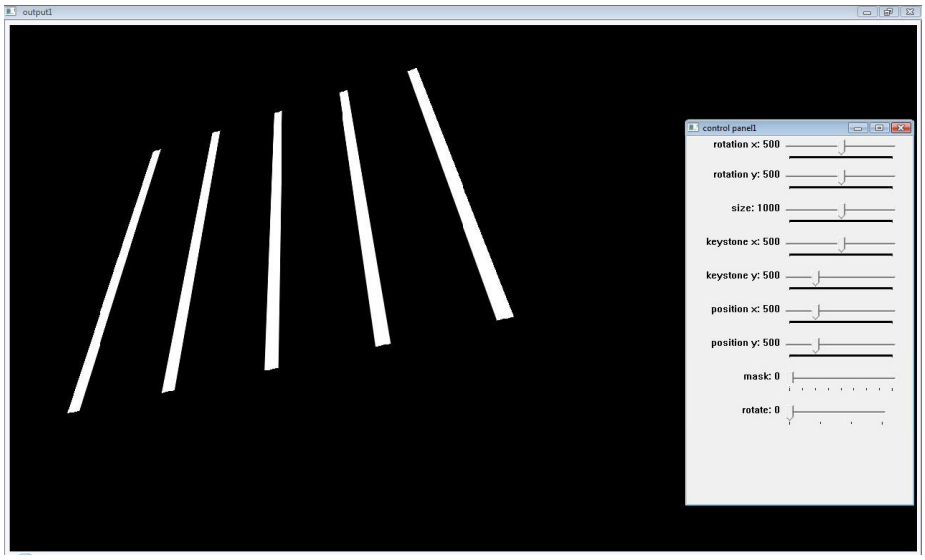

Figure 25: Vertical lines before correction

In the first calibration step, the vertical bars should be aligned by using keystoning to make them parallel. Using rotation, size adjustment and placement will make it easy to find when the lines are correctly aligned.

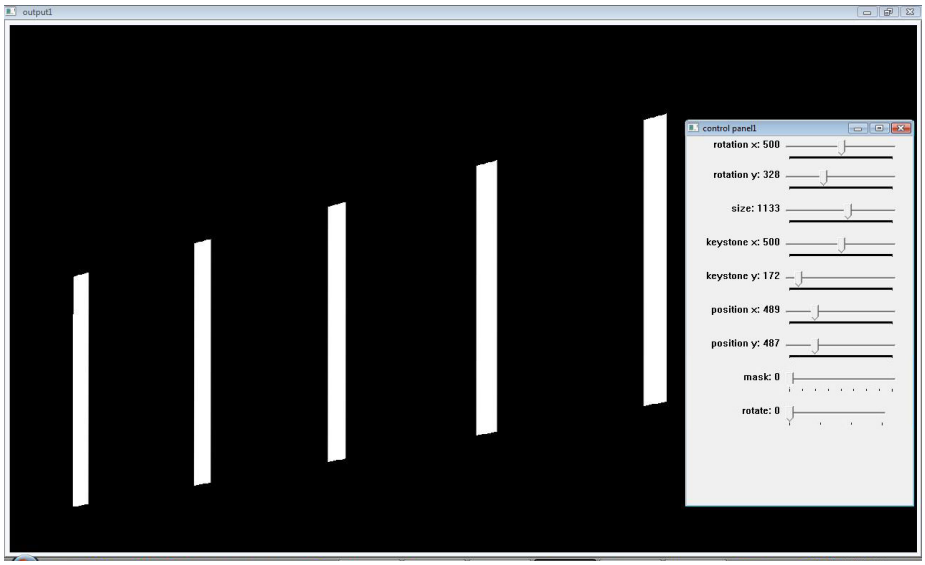

Figure 26: Vertical lines after correction

When the lines are parallel, press Spacebar to jump to next step.

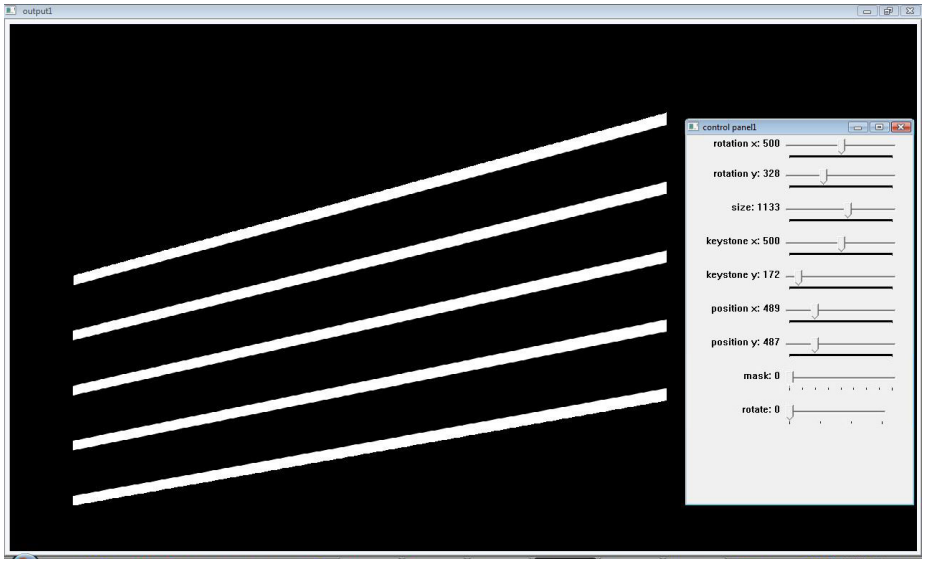

Figure 27: Horizontal lines before correction

Again a set of lines, these are horizontal. Align these like those in the previous

step with keystoning and the use of rotation, size and placement for help.

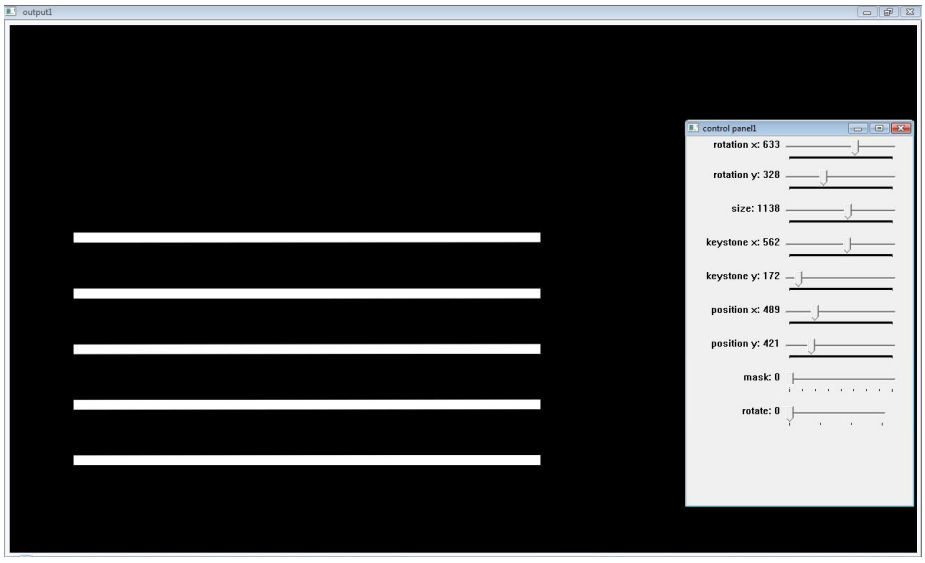

Figure 28: Horizontal lines after correction

Now the output is starting to look better. When satisfied with how parallel these lines are, press Spacebar.

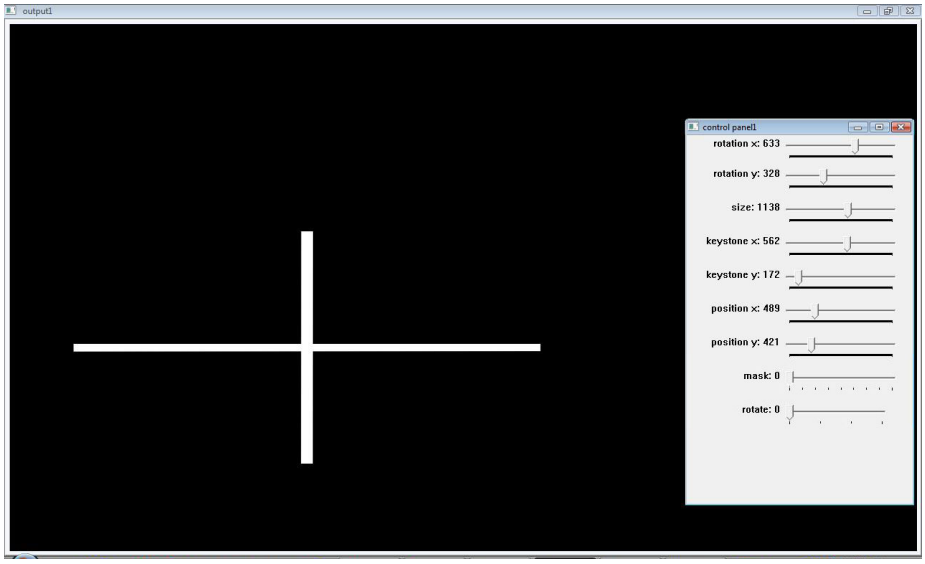

Figure 29: Rotation adjustment before correction

Correcting the rotation is in this example a bit redundant, as the previous two steps already has given us close to perfect rotation. However, if this is not the case, this step uses a cross that lets the user verify and correct the rotation angle.

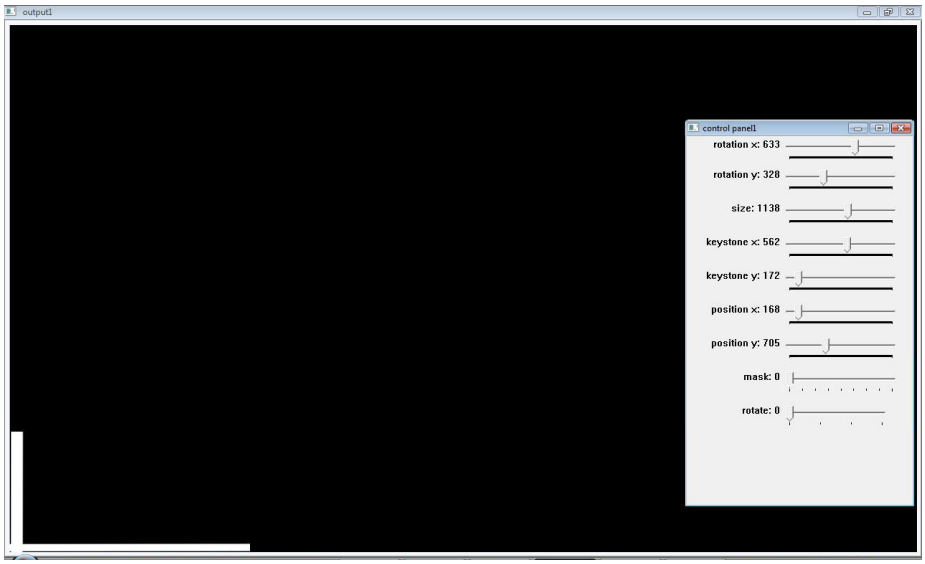

Figure 30: Rotation adjustment after correction

By placing the cross in the lower left corner, it is clear that the rotation has been corrected. Spacebar to continue.

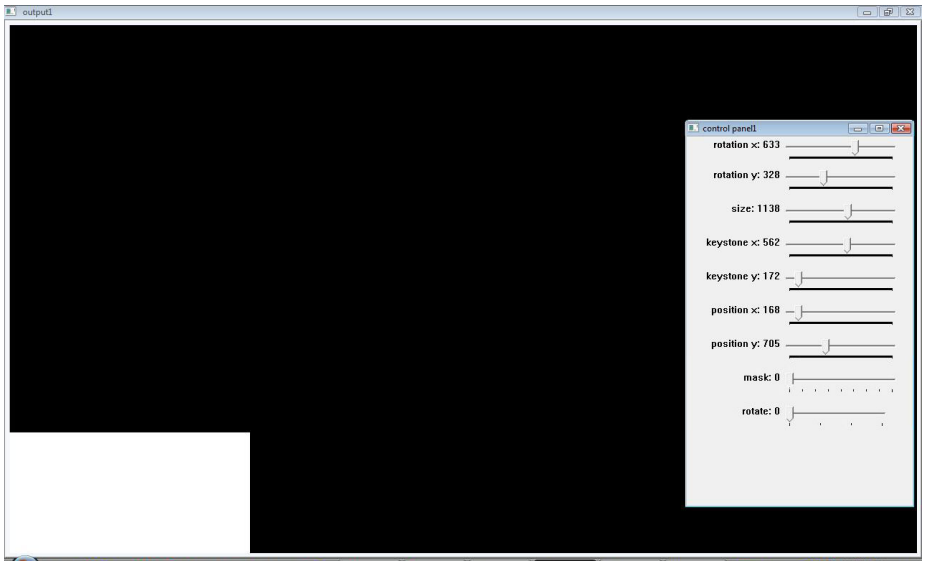

Figure 31: Size adjustment before correction

This step lets the user size the image to correct size, by using an all-white image that is scaled to fit the output screen area.

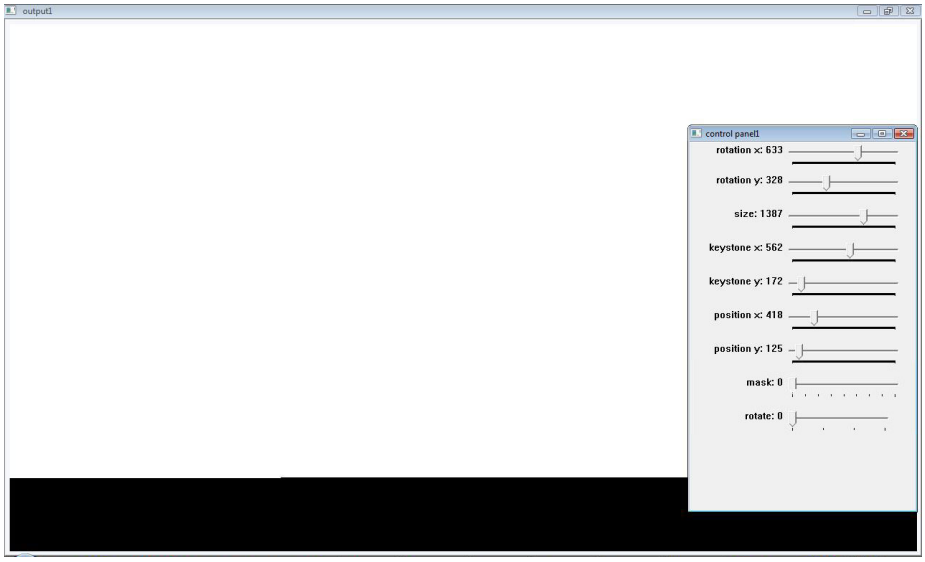

Figure 32: Size adjustment after correction

After rescaling it to fit the largest possible area, it is possible to jump to the last calibration step.

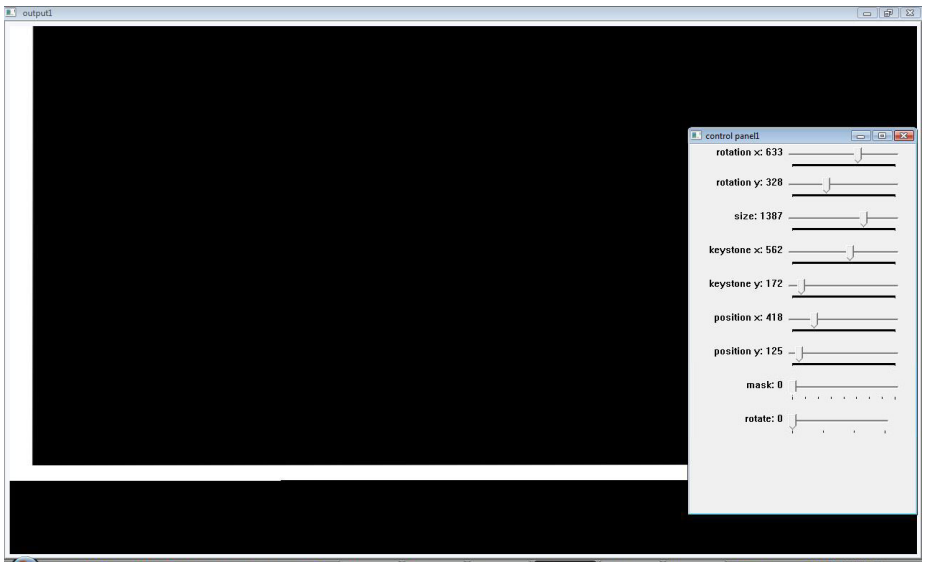

Figure 33: Corner placement before correction

This final step lets the user place the corner of the screen, by using this and the previous step correctly, the correct fit and placement of the output screen is ensured.

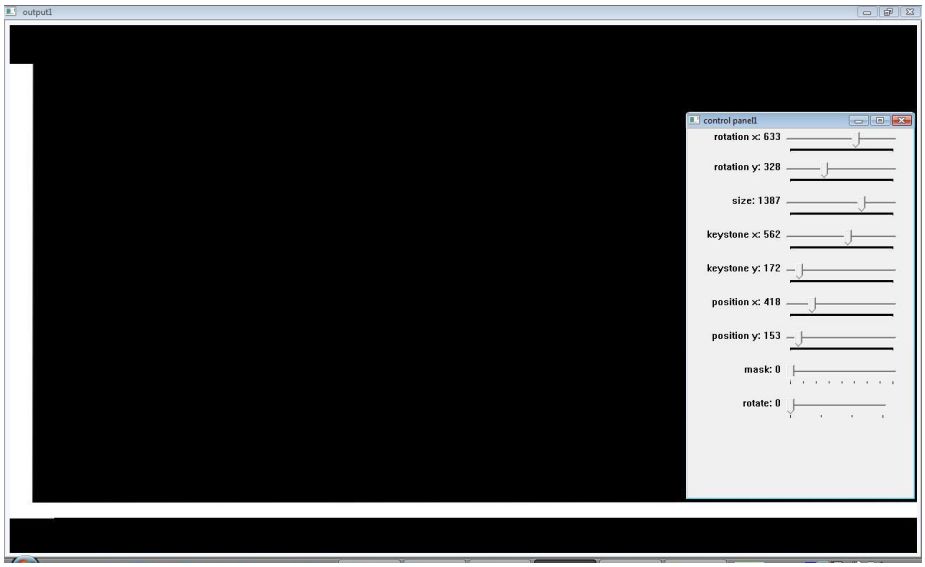

Figure 34: Corner placement after correction

The image is now calibrated properly, and placed at the wanted spot. By pressing Spacebar one last time, the program starts and the input video is warped to fit the calibration result.

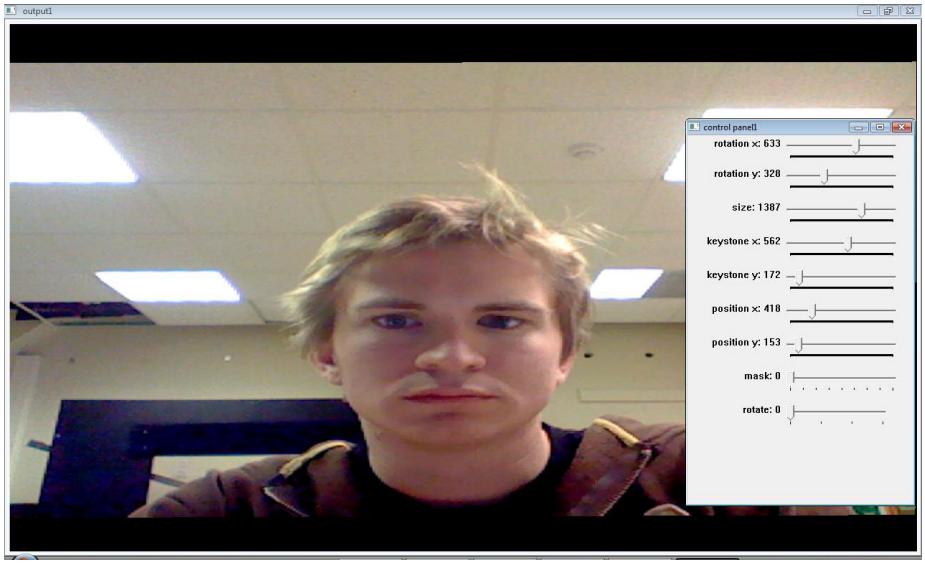

Figure 35: Warped video after calibration

Yours sincerely, on a web cam that feeds video to the program, warped to fit. After the initial calibration, it is still possible to alter all settings for the image, as well as rotating and adding masks.

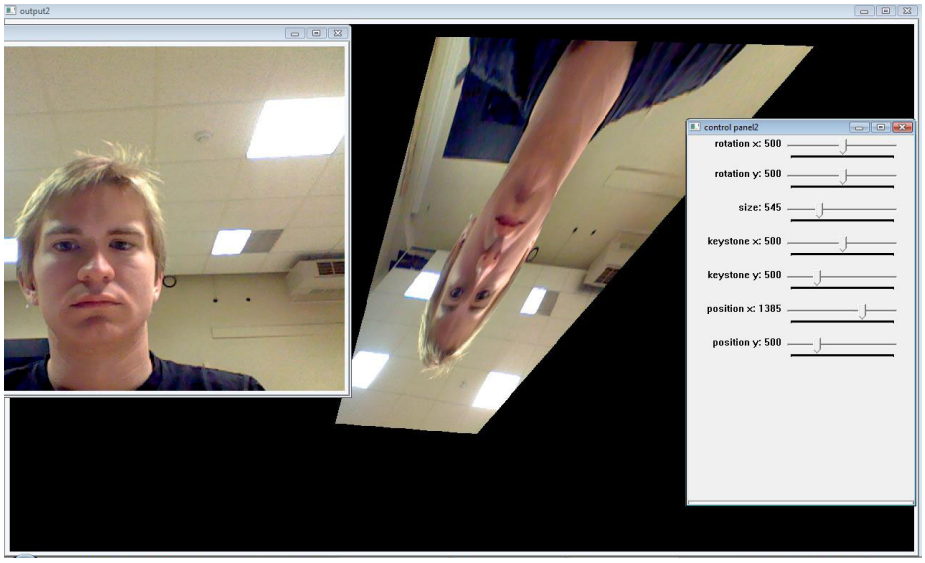

Figure 36: Warped, inverted and flipped output

The image can be stretched to any trapezoid shape using corner pinning and keystoning.

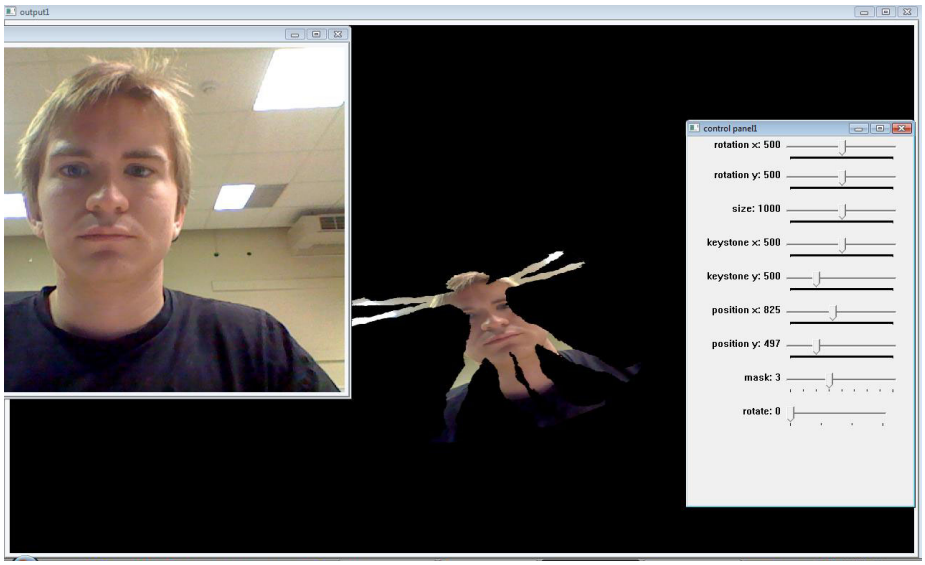

Figure 37: Warped and masked output

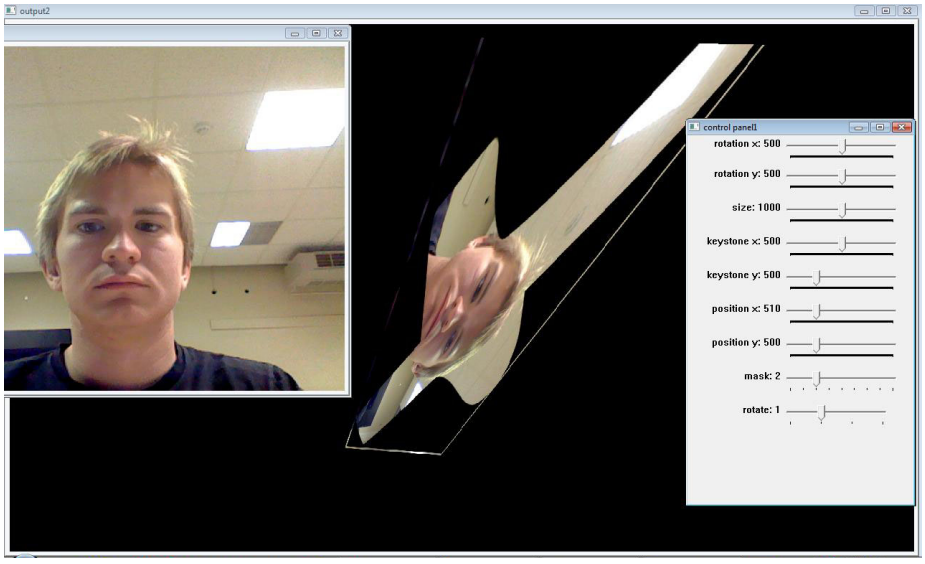

Figure 38: Warped, rotated and masked output

Mask image files put in the folder named '/masks', preferably in jpg format will automatically be added to the program. The images are added to the source image with inverted colors, making black areas in the source image become

transparent, white areas are masked out and all colors are inverted. This was done because it was the best way to both have transparent and black areas.

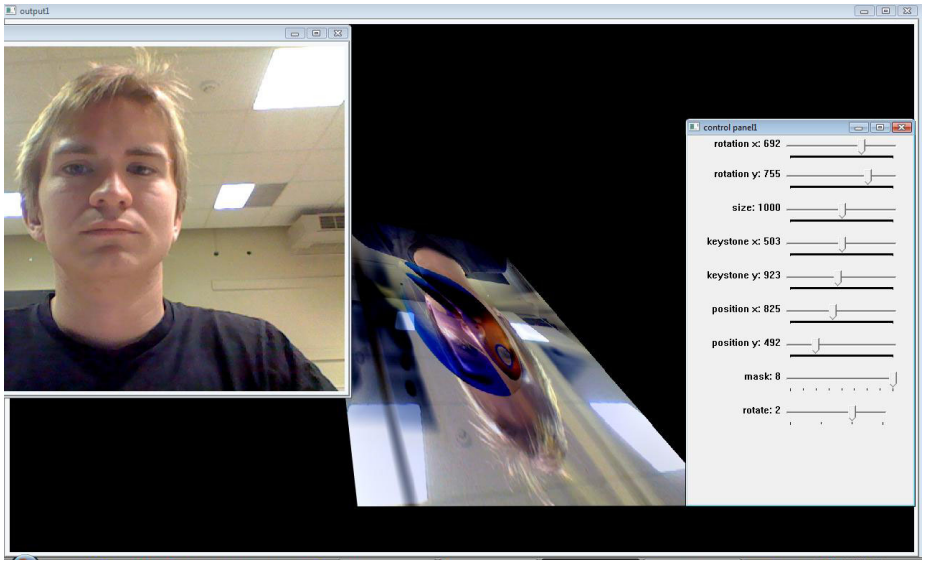

Figure 39: Warped, flipped and masked output

Images used as masking will have an inverted color space, so to get correct colors, one needs to invert the colors before the image is put in the mask folder. The use of color images as masks is a nice effect, and may be used for many purposes.

C Code example

Listing 2: Source code for StereoProjectCornerpin.py

```
from opencv import *
from math import sin, cos, tan, pi
import os
import time
import datetime
import platform

###Creating global parameters
alpha = 1
beta = -1
activeFrame=1
mask =cvCreateImage(cvSize(1,1), 8, 3)
hiFPS=0
maxFPS=60
capture1 = cvCreateFileCapture( None )
capture2 = cvCreateFileCapture( None )
rotation=cvCreateMat(2,3,CV_32FC1)
matrix1=cvCreateMat(3,3,CV_32FC1)
matrix2=cvCreateMat(3,3,CV_32FC1)
basisMatrix1=cvCreateMat(3,3,CV_32FC1)
basisMatrix2=cvCreateMat(3,3,CV_32FC1)

#####Updating image 1 from Control Panel 1 input

def update1(pos):
        global matrix1
        global basisMatrix1
        matrise=cvCreateMat(3,3,CV_32FC1)
        matrise=matrix1

        ####Getting data from CP and performing calculations

        rotx=cvGetTrackbarPos("rotation x", "control panel1")
        roty=cvGetTrackbarPos("rotation y", "control panel1")
        size=cvGetTrackbarPos("size", "control panel1")
        posx=cvGetTrackbarPos("position x", "control panel1")
        posy=cvGetTrackbarPos("position y", "control panel1")
        keysx=cvGetTrackbarPos("keystone x", "control panel1")
        keysy=cvGetTrackbarPos("keystone y", "control panel1")
        rotx=((rotx-500)*pi)/1000
        roty=((roty-500)*pi)/1000
        size=((size-1000)*pi)/4000

        ####Inserting new data in warp matrix

        matrise[1,0] =basisMatrix1[1,0]+sin(rotx)
        matrise[0,1] =basisMatrix1[0,1]+sin(roty)
        matrise[2,2]=(1-(2*sin(size)))
        matrise[0,2]=basisMatrix1[0,2]+posx-500
        matrise[1,2]=basisMatrix1[1,2]+posy-500
        matrise[2,0]=basisMatrix1[2,0]+(float(keysx-500)/250000)
        matrise[2,1]=basisMatrix1[2,1]+(float(keysy-500)/250000)
        matrix1=matrise

####Updating image 2 from Control Panel 2 input
def update2(pos):
        global matrix2
        global basisMatrix2
        matrise=cvCreateMat(3,3,CV_32FC1)
        matrise=matrix2

        ####Getting data from CP and performing calculations

        rotx=cvGetTrackbarPos("rotation x", "control panel2")
        roty=cvGetTrackbarPos("rotation y", "control panel2")
        size=cvGetTrackbarPos("size", "control panel2")
        posx=cvGetTrackbarPos("position x", "control panel2")
```

```
68          posy=cvGetTrackbarPos("position y", "control panel2")
69          keysx=cvGetTrackbarPos("keystone x", "control panel2")
70          keysy=cvGetTrackbarPos("keystone y", "control panel2")
71          rotx=((rotx-500)*pi)/1000
72          roty=((roty-500)*pi)/1000
73          size=((size-1000)*pi)/4000
74
75          ####Inserting new data in warp matrix
76          matrise[1,0] =basisMatrix2[1,0]+sin(rotx)
77      matrise[0,1] =basisMatrix2[0,1]+sin(roty)
78      matrise[2,2]=(1-(2*sin(size)))
79      matrise[0,2]= basisMatrix2[0,2]+posx-500
80      matrise[1,2]= basisMatrix2[1,2]+posy-500
81      matrise[2,0]= basisMatrix2[2,0]+(float(keysx-500)/250000)
82          matrise[2,1]= basisMatrix2[2,1]+(float(keysy-500)/250000)
83          matrix2=matrise
84
85 ###Updating rotation and masking of both images
86
87 def maskAndRotate(pos):
88          global mask
89          global alpha
90          global beta
91          global rotation
92          global capture1
93          global capture2
94
95          ###Creating mask
96
97          masks=os.listdir('./masks')
98          mask_src = cvLoadImage("./masks/"+masks[cvGetTrackbarPos("mask", "control
               panel1")])
99          cvConvertImage( mask_src, mask_src, 1)
100         cvResize(mask_src, mask, CV_INTER_AREA )
101
102         ###Creating rotation matrix
103
104         rotate=cvGetTrackbarPos("rotate", "control panel1")
105         cv2DRotationMatrix(cvPoint2D32f(mask.width/2, mask.height/2), 90*rotate,
               1, rotation)
106
107 ###Backing up parameters from both Control Panels
108
109 def backup(man):
110         handle = open("calibStereoLog2.txt","a")
111         logline = []
112
113         ###Adding "Manual save" to differentiate between autosaves and manual
               backup
114
115         if man:
116                 logline.append("Manual save:\n")
117
118         ###writing all parameters to log file
119
120         logline.append(str(-1))
121
122         logline.append(str(cvGetTrackbarPos("rotation x", "control panel1")))
123         logline.append(str(cvGetTrackbarPos("rotation y", "control panel1")))
124         logline.append(str(cvGetTrackbarPos("size", "control panel1")))
125         logline.append(str(cvGetTrackbarPos("keystone x", "control panel1")))
126         logline.append(str(cvGetTrackbarPos("keystone y", "control panel1")))
127         logline.append(str(cvGetTrackbarPos("position x", "control panel1")))
128         logline.append(str(cvGetTrackbarPos("position y", "control panel1")))
129         logline.append(str(cvGetTrackbarPos("mask", "control panel1")))
130         logline.append(str(cvGetTrackbarPos("rotate", "control panel1")))
131
132         logline.append(str(cvGetTrackbarPos("rotation x", "control panel2")))
133         logline.append(str(cvGetTrackbarPos("rotation y", "control panel2")))
134         logline.append(str(cvGetTrackbarPos("size", "control panel2")))
135         logline.append(str(cvGetTrackbarPos("keystone x", "control panel2")))
136         logline.append(str(cvGetTrackbarPos("keystone y", "control panel2")))
137         logline.append(str(cvGetTrackbarPos("position x", "control panel2")))
```

```
138         logline.append(str(cvGetTrackbarPos("position y", "control panel2")))
139
140         handle.write((" ".join(logline))+"\n")
141         handle.close()
142         print "saved settings!"
143
144 ###Loading either autosaves (1) or manual saves(0)
145
146 def load(autosaves):
147         log = open("calibStereoLog2.txt","r")
148         txt= "1 500 500 1000 500 500 500 500"
149         saved="Manual save:\n"
150         if autosaves:
151                 for line in log:
152                         txt=line
153         else:
154                 next=0
155                 for line in log:
156                     if next:
157                         txt=line
158                         next=0
159                     if line == saved:
160                         next=1
161         log.close
162         return txt
163
164 ###Method used to replace a single value in a string, used to ease things in the
    calibration method
165 ###Splits string into its individual pieces, replacing the value at the place
    indicated with the value given
166
167 def calibStringReplace(calibString, value, place):
168         i=0
169         ##finding numbers of values in the input string
170         try:
171             while (1):
172                         i+=1
173                         str(calibString.split()[i])
174                         if i>100:
175
176                             ###used to detect and stop any infinite loops
177                                 print "overflow in stringReplace. Too long input
                                        string"
178                                 sys.exit(-1)
179         except IndexError:
180                 i-=1
181         if int(place)>i:
182                 print "Attempted replace position does not exist!"
183                 print place
184                 print i
185                 return None
186         endResult=""
187         ii=0
188
189         ###putting values back in place, up to the place that is to be replaced
190         while ii<int(place):
191                 endResult=endResult+(calibString.split()[ii])+" "
192                 ii+=1
193
194         ###inserting wanted value at wanted position
195         endResult=endResult+(value)+" "
196         ii+=1
197
198         ###Putting back rest of values
199         while ii<=i:
200                 endResult=endResult+(calibString.split()[ii])+" "
201                 ii+=1
202
203         return endResult
204
205 def calibrate(frame, frameName):
206     global spots
207     global basisMatrix1
```

```
208     global basisMatrix2
209     global matrix1
210         global matrix2
211 
212         #####Stereoscopic add-on test
213         matrix=cvCreateMat(3,3,CV_32FC1)
214     basisMatrix=cvCreateMat(3,3,CV_32FC1)
215         stereo=0
216         if frameName=="output1":
217                 stereo=1
218                 cp="control panel1"
219                 ncp="control panel2"
220                 matrix=matrix1
221         elif frameName=="output2":
222                 stereo=2
223                 cp="control panel2"
224                 ncp="control panel1"
225                 matrix=matrix2
226 
227         ###loading calibration images
228 
229         calibCorner_src = cvLoadImage( "calibration/Calib-corner.bmp" );
230         calibCorner=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
231         cvResize(calibCorner_src, calibCorner, CV_INTER_AREA );
232 
233         calibCross_src = cvLoadImage( "calibration/Calib-cross.bmp" );
234         calibCross=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
235         cvResize(calibCross_src, calibCross, CV_INTER_AREA );
236 
237         calibFull_src = cvLoadImage( "calibration/Calib-full.bmp" );
238         calibFull=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
239         cvResize(calibFull_src, calibFull, CV_INTER_AREA );
240 
241         calibHorisont_src = cvLoadImage( "calibration/Calib-horizontal.bmp" );
242         calibHorisont=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
243         cvResize(calibHorisont_src, calibHorisont, CV_INTER_AREA );
244 
245         calibVertical_src = cvLoadImage( "calibration/Calib-vertical.bmp" );
246         calibVertical=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
247         cvResize(calibVertical_src, calibVertical, CV_INTER_AREA );
248 
249         calibEnd_src = cvLoadImage( "calibration/Calib-end.bmp" );
250     calibEnd=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
251         cvResize(calibEnd_src, calibEnd, CV_INTER_AREA );
252 
253         dst = cvCreateImage (cvSize (frame.width*2, frame.height*2), 8, 3)
254 
255     calibBlank_src = cvLoadImage( "calibration/Calib-blank.bmp" );
256     calibBlank=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
257         cvResize(calibBlank_src, calibBlank, CV_INTER_AREA );
258 
259     ###Actual calibration
260     ###Only the values that are relevant to a given test is possible to modify
261     ###These are saved before next step
262 
263         print "calibration will now commence:"
264         i=-1
265 
266         spots=[]
267     params=load(1)
268 
269     while (i<0):
270 
271         ###Getting corners from mouse input
272             if i==-1:
273                 src=calibBlank
274                 dst=calibBlank
275                 cvSetMouseCallback(frameName, mouse, param=None)
276                 print "Click bottom left corner"
277                 ii=0
278 
279                 i+=.5
280             if i==-.5:
```

```
281                 if len(spots)==4:
282                     i+=.5
283                     pt_array = CvPoint2D32f * 4
284                 c1 =pt_array(*(cvPoint2D32f(0,0), cvPoint2D32f(src.width,0),
                        cvPoint2D32f(0,src.height), cvPoint2D32f(src.width,src.height))
                        )
285                 c2 =pt_array(*(cvPoint2D32f(max(0,spots[0].x*2),max(0,spots[0].y
                        *2)), cvPoint2D32f(max(0,spots[1].x*2),max(0,spots[1].y*2)),
                        cvPoint2D32f(max(0,spots[2].x*2),max(0,spots[2].y*2)),
                        cvPoint2D32f(max(0,spots[3].x*2),max(0,spots[3].y*2))))
286
287                 basisMatrix = cvGetPerspectiveTransform(c1, c2, basisMatrix)
288
289                 ##Check for invalid corner inputs, if found, reset the warp matrix
290                 if(basisMatrix[0,1]<-1 or basisMatrix[1,0]<-1):
291                     print "Invalid data, resetting image transformation matrix to
                            default"
292                     basisMatrix[0,0]=1
293                     basisMatrix[0,1]=0
294                     basisMatrix[0,2]=0
295                     basisMatrix[1,0]=0
296                     basisMatrix[1,1]=1
297                     basisMatrix[1,2]=0
298                     basisMatrix[2,0]=0
299                     basisMatrix[2,1]=0
300                             basisMatrix[2,2]=1
301
302                 cvSetTrackbarPos("rotation x", cp, 500)
303                 cvSetTrackbarPos("rotation y", cp, 500)
304                 cvSetTrackbarPos("size", cp, 1000)
305                 cvSetTrackbarPos("position x", cp, 500)
306                 cvSetTrackbarPos("position y", cp, 500)
307                 cvSetTrackbarPos("keystone x", cp, 500)
308                 cvSetTrackbarPos("keystone y", cp, 500)
309                 matrix[0,0]=basisMatrix[0,0]
310                 matrix[0,1]=0+basisMatrix[0,1]
311                 matrix[0,2]=0+basisMatrix[0,2]
312                 matrix[1,0]=0+basisMatrix[1,0]
313                 matrix[1,1]=basisMatrix[1,1]
314                 matrix[1,2]=0+basisMatrix[1,2]
315                 matrix[2,0]=0+basisMatrix[2,0]
316                 matrix[2,1]=0+basisMatrix[2,1]
317                         matrix[2,2]=basisMatrix[2,2]
318                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                            rotation x", cp)), 1+(9*(stereo-1)))
319                 params=calibStringReplace(params, str(cvGetTrackbarPos("rotation y
                        ", cp)), 2+(9*(stereo-1)))
320                 params=calibStringReplace(params, str(cvGetTrackbarPos("size", cp)
                        ), 3+(9*(stereo-1)))
321                 params=calibStringReplace(params, str(cvGetTrackbarPos("keystone x
                        ", cp)), 4+(9*(stereo-1)))
322                 params=calibStringReplace(params, str(cvGetTrackbarPos("keystone y
                        ", cp)), 5+(9*(stereo-1)))
323                 params=calibStringReplace(params, str(cvGetTrackbarPos("position x
                        ", cp)), 6+(9*(stereo-1)))
324                             params=calibStringReplace(params, str(cvGetTrackbarPos
                                ("position y", cp)), 7+(9*(stereo-1)))
325
326                 else:
327                     while ii < len(spots):
328                         cvCircle( src, spots[ii], 3, CV_RGB(255,255,255), 1, 8, 0
                                )
329                         dst=src
330                         ii+=1
331
332             if i<-1:
333                 print "already at beginning"
334                         i=-1
335
336             cvShowImage(frameName, dst )
337
338         keyPressed=cvWaitKey(10)
339                 if keyPressed==27:
```

```
340                     i=0
341                     basisMatrix[0,0]=1
342             basisMatrix[0,1]=0
343             basisMatrix[0,2]=0
344             basisMatrix[1,0]=0
345             basisMatrix[1,1]=1
346             basisMatrix[1,2]=0
347             basisMatrix[2,0]=0
348             basisMatrix[2,1]=0
349                     basisMatrix[2,2]=1
350                     cvSetTrackbarPos("rotation x", cp, 500)
351             cvSetTrackbarPos("rotation y", cp, 500)
352             cvSetTrackbarPos("size", cp, 1000)
353             cvSetTrackbarPos("position x", cp, 500)
354             cvSetTrackbarPos("position y", cp, 500)
355             cvSetTrackbarPos("keystone x", cp, 500)
356             cvSetTrackbarPos("keystone y", cp, 500)
357             matrix[0,0]=basisMatrix[0,0]
358             matrix[0,1]=0+basisMatrix[0,1]
359             matrix[0,2]=0+basisMatrix[0,2]
360             matrix[1,0]=0+basisMatrix[1,0]
361             matrix[1,1]=basisMatrix[1,1]
362             matrix[1,2]=0+basisMatrix[1,2]
363             matrix[2,0]=0+basisMatrix[2,0]
364             matrix[2,1]=0+basisMatrix[2,1]
365             matrix[2,2]=basisMatrix[2,2]
366             params=calibStringReplace(params, str(cvGetTrackbarPos("rotation x",
                    cp)), 1+(9*(stereo-1)))
367             params=calibStringReplace(params, str(cvGetTrackbarPos("rotation y",
                    cp)), 2+(9*(stereo-1)))
368             params=calibStringReplace(params, str(cvGetTrackbarPos("size", cp)),
                    3+(9*(stereo-1)))
369             params=calibStringReplace(params, str(cvGetTrackbarPos("keystone x",
                    cp)), 4+(9*(stereo-1)))
370             params=calibStringReplace(params, str(cvGetTrackbarPos("keystone y",
                    cp)), 5+(9*(stereo-1)))
371             params=calibStringReplace(params, str(cvGetTrackbarPos("position x",
                    cp)), 6+(9*(stereo-1)))
372                     params=calibStringReplace(params, str(cvGetTrackbarPos("
                            position y", cp)), 7+(9*(stereo-1)))
373
374         if(stereo==1):
375             basisMatrix1=basisMatrix
376             matrix1=matrix
377         elif stereo==2:
378             basisMatrix2=basisMatrix
379             matrix2=matrix
380         while(i<5):
381                 if i<0:
382             calibrate(frame, frameName)
383                 break
384                 if i==0:
385                         print "adjust vertical keystone until lines are paralell"
386                         src= calibVertical
387                         i+=0.5
388
389                 if i==0.5:
390                         cvSetTrackbarPos("rotation x", cp, int(params.split()
                                [1+(9*(stereo-1))]))
391                         cvSetTrackbarPos("keystone x", cp, int(params.split()
                                [4+(9*(stereo-1))]))
392                         cvSetTrackbarPos("mask", cp, int(params.split()[8]))
393                         cvSetTrackbarPos("rotate", cp, int(params.split()[9]))
394                         cvSetTrackbarPos("rotation x", ncp, int(params.split()
                                [10-(9*(stereo-1))]))
395                         cvSetTrackbarPos("rotation y", ncp, int(params.split()
                                [11-(9*(stereo-1))]))
396                         cvSetTrackbarPos("size", ncp, int(params.split()[12-(9*(
                                stereo-1))]))
397                         cvSetTrackbarPos("keystone x", ncp, int(params.split()
                                [13-(9*(stereo-1))]))
398                         cvSetTrackbarPos("keystone y", ncp, int(params.split()
                                [14-(9*(stereo-1))]))
```

```
399                         cvSetTrackbarPos("position x", ncp, int(params.split()
                               [15-(9*(stereo-1))]))
400                         cvSetTrackbarPos("position y", ncp, int(params.split()
                               [16-(9*(stereo-1))]))
401                 if i==1:
402                         src= calibHorisont
403                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation y", cp)), 2+(9*(stereo-1)))
404                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               size", cp)), 3+(9*(stereo-1)))
405                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               keystone y", cp)), 5+(9*(stereo-1)))
406                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position x", cp)), 6+(9*(stereo-1)))
407                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position y", cp)), 7+(9*(stereo-1)))
408                         print "adjust horizontal keystone until lines are paralell
                               "
409                         i+=0.5
410                 if i==1.5:
411                         cvSetTrackbarPos("rotation y", "control panel"+str(stereo
                               ), int(params.split()[2+(9*(stereo-1))]))
412                         cvSetTrackbarPos("keystone y", "control panel"+str(stereo
                               ), int(params.split()[5+(9*(stereo-1))]))
413                         cvSetTrackbarPos("mask", "control panel"+str(stereo), int
                               (params.split()[8]))
414                         cvSetTrackbarPos("rotate", "control panel"+str(stereo),
                               int(params.split()[9]))
415                         cvSetTrackbarPos("rotation x", ncp, int(params.split()
                               [10-(9*(stereo-1))]))
416                         cvSetTrackbarPos("rotation y", ncp, int(params.split()
                               [11-(9*(stereo-1))]))
417                         cvSetTrackbarPos("size", ncp, int(params.split()[12-(9*(
                               stereo-1))]))
418                         cvSetTrackbarPos("keystone x", ncp, int(params.split()
                               [13-(9*(stereo-1))]))
419                         cvSetTrackbarPos("keystone y", ncp, int(params.split()
                               [14-(9*(stereo-1))]))
420                         cvSetTrackbarPos("position x", ncp, int(params.split()
                               [15-(9*(stereo-1))]))
421                         cvSetTrackbarPos("position y", ncp, int(params.split()
                               [16-(9*(stereo-1))]))
422                 if i==2:
423                         src= calibCross
424                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation x", cp)), 1+(9*(stereo-1)))
425                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               size", cp)), 3+(9*(stereo-1)))
426                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               keystone x", cp)), 4+(9*(stereo-1)))
427                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position x", cp)), 6+(9*(stereo-1)))
428                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position y", cp)), 7+(9*(stereo-1)))
429                         print "rotate until cross is aligned with plane"
430                         i+=0.5
431                 if i==2.5:
432                         cvSetTrackbarPos("size", "control panel"+str(stereo), int
                               (params.split()[3+(9*(stereo-1))]))
433                         cvSetTrackbarPos("keystone x", "control panel"+str(stereo
                               ), int(params.split()[4+(9*(stereo-1))]))
434                         cvSetTrackbarPos("keystone y", "control panel"+str(stereo
                               ), int(params.split()[5+(9*(stereo-1))]))
435                         cvSetTrackbarPos("mask", "control panel"+str(stereo), int
                               (params.split()[8]))
436                         cvSetTrackbarPos("rotate", "control panel"+str(stereo),
                               int(params.split()[9]))
437                         cvSetTrackbarPos("rotation x", ncp, int(params.split()
                               [10-(9*(stereo-1))]))
438                         cvSetTrackbarPos("rotation y", ncp, int(params.split()
                               [11-(9*(stereo-1))]))
439                         cvSetTrackbarPos("size", ncp, int(params.split()[12-(9*(
                               stereo-1))]))
```

```
440                         cvSetTrackbarPos("keystone x", ncp, int(params.split()
                               [13-(9*(stereo-1))]))
441                         cvSetTrackbarPos("keystone y", ncp, int(params.split()
                               [14-(9*(stereo-1))]))
442                         cvSetTrackbarPos("position x", ncp, int(params.split()
                               [15-(9*(stereo-1))]))
443                         cvSetTrackbarPos("position y", ncp, int(params.split()
                               [16-(9*(stereo-1))]))
444                 if i==3:
445                         src= calibFull
446                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation x", cp)), 1+(9*(stereo-1)))
447                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation y", cp)), 2+(9*(stereo-1)))
448                         print "adjust size until proper fit"
449                         i+=0.5
450                 if i==3.5:
451                         cvSetTrackbarPos("rotation x", "control panel"+str(stereo
                               ), int(params.split()[1+(9*(stereo-1))]))
452                         cvSetTrackbarPos("rotation y", "control panel"+str(stereo
                               ), int(params.split()[2+(9*(stereo-1))]))
453                         cvSetTrackbarPos("keystone x", "control panel"+str(stereo
                               ), int(params.split()[4+(9*(stereo-1))]))
454                         cvSetTrackbarPos("keystone y", "control panel"+str(stereo
                               ), int(params.split()[5+(9*(stereo-1))]))
455                         cvSetTrackbarPos("mask", "control panel"+str(stereo), int
                               (params.split()[8]))
456                         cvSetTrackbarPos("rotate", "control panel"+str(stereo),
                               int(params.split()[9]))
457                         cvSetTrackbarPos("rotation x", ncp, int(params.split()
                               [10-(9*(stereo-1))]))
458                         cvSetTrackbarPos("rotation y", ncp, int(params.split()
                               [11-(9*(stereo-1))]))
459                         cvSetTrackbarPos("size", ncp, int(params.split()[12-(9*(
                               stereo-1))]))
460                         cvSetTrackbarPos("keystone x", ncp, int(params.split()
                               [13-(9*(stereo-1))]))
461                         cvSetTrackbarPos("keystone y", ncp, int(params.split()
                               [14-(9*(stereo-1))]))
462                         cvSetTrackbarPos("position x", ncp, int(params.split()
                               [15-(9*(stereo-1))]))
463                         cvSetTrackbarPos("position y", ncp, int(params.split()
                               [16-(9*(stereo-1))]))
464                 if i==4:
465                         src= calibCorner
466                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation x", cp)), 1+(9*(stereo-1)))
467                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               rotation y", cp)), 2+(9*(stereo-1)))
468                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               size", cp)), 3+(9*(stereo-1)))
469                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position x", cp)), 6+(9*(stereo-1)))
470                         params=calibStringReplace(params, str(cvGetTrackbarPos("
                               position y", cp)), 7+(9*(stereo-1)))
471                         print "place in upper left corner"
472                         i+=0.5
473                 if i==4.5:
474                         cvSetTrackbarPos("rotation x", "control panel"+str(stereo
                               ), int(params.split()[1+(9*(stereo-1))]))
475                         cvSetTrackbarPos("rotation y", "control panel"+str(stereo
                               ), int(params.split()[2+(9*(stereo-1))]))
476                         cvSetTrackbarPos("size", "control panel"+str(stereo), int
                               (params.split()[3+(9*(stereo-1))]))
477                         cvSetTrackbarPos("keystone x", "control panel"+str(stereo
                               ), int(params.split()[4+(9*(stereo-1))]))
478                         cvSetTrackbarPos("keystone y", "control panel"+str(stereo
                               ), int(params.split()[5+(9*(stereo-1))]))
479                         cvSetTrackbarPos("mask", "control panel"+str(stereo), int
                               (params.split()[8]))
480                         cvSetTrackbarPos("rotate", "control panel"+str(stereo),
                               int(params.split()[9]))
481                         cvSetTrackbarPos("rotation x", ncp, int(params.split()
```

```
                                [10-(9*(stereo-1))]))
482                             cvSetTrackbarPos("rotation y", ncp, int(params.split()
                                [11-(9*(stereo-1))]))
483                             cvSetTrackbarPos("size", ncp, int(params.split()[12-(9*(
                                stereo-1))]))
484                             cvSetTrackbarPos("keystone x", ncp, int(params.split()
                                [13-(9*(stereo-1))]))
485                             cvSetTrackbarPos("keystone y", ncp, int(params.split()
                                [14-(9*(stereo-1))]))
486                             cvSetTrackbarPos("position x", ncp, int(params.split()
                                [15-(9*(stereo-1))]))
487                             cvSetTrackbarPos("position y", ncp, int(params.split()
                                [16-(9*(stereo-1))]))
488
489                     dst = cv.cvCreateImage (cv.cvSize (frame.width*2, frame.height*2),
                            8, 3)
490                     if stereo==1:
491                             cvWarpPerspective( src, dst, matrix1, (CV_INTER_LINEAR+
                                CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
492                     if stereo==2:
493                             cvWarpPerspective( src, dst, matrix2, (CV_INTER_LINEAR+
                                CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
494                     cvShowImage(frameName, dst );
495
496                     ###Finding actions to perform for key input
497
498                     keyPressed=cvWaitKey(10)
499                     if keyPressed!=-1:
500                         ###Movement of image
501             if keyPressed==100:
502                                     posx=cvGetTrackbarPos("position x", cp)
503                                     posx+=1
504                                     cvSetTrackbarPos("position x", cp, posx)
505                             elif keyPressed==97:
506                                     posx=cvGetTrackbarPos("position x", cp)
507                                     posx-=1
508                                     cvSetTrackbarPos("position x", cp, posx)
509                             elif keyPressed==119:
510                                     posy=cvGetTrackbarPos("position y", cp)
511                                     posy-=1
512                                     cvSetTrackbarPos("position y", cp, posy)
513                             elif keyPressed==115:
514                                     posy=cvGetTrackbarPos("position y", cp)
515                                     posy+=1
516                                     cvSetTrackbarPos("position y", cp, posy)
517                             elif keyPressed==43:
518                                     size=cvGetTrackbarPos("size", cp)
519                                     size+=1
520                                     cvSetTrackbarPos("size", cp, size)
521                             elif keyPressed==45:
522                                     size=cvGetTrackbarPos("size", cp)
523                                     size-=1
524                                     cvSetTrackbarPos("size", cp, size)
525                             elif keyPressed==65:
526                                     roty=cvGetTrackbarPos("rotation y", cp)
527                                     roty-=1
528                                     cvSetTrackbarPos("rotation y", cp, roty)
529                             elif keyPressed==68:
530                                     roty=cvGetTrackbarPos("rotation y", cp)
531                                     roty+=1
532                                     cvSetTrackbarPos("rotation y", cp, roty)
533                             elif keyPressed==83:
534                                     rotx=cvGetTrackbarPos("rotation x", cp)
535                                     rotx+=1
536                                     cvSetTrackbarPos("rotation x", cp, rotx)
537                             elif keyPressed==87:
538                                     rotx=cvGetTrackbarPos("rotation x", cp)
539                                     rotx-=1
540                                     cvSetTrackbarPos("rotation x", cp, rotx)
541                             elif keyPressed==101:
542                                     keysx=cvGetTrackbarPos("keystone x", cp)
543                                     keysx-=1
544                                     cvSetTrackbarPos("keystone x", cp, keysx)
```

```
545                         elif keyPressed==113:
546                                 keysx=cvGetTrackbarPos("keystone x", cp)
547                                 keysx+=1
548                                 cvSetTrackbarPos("keystone x", cp, keysx)
549                         elif keyPressed==122:
550                                 keysy=cvGetTrackbarPos("keystone y", cp)
551                                 keysy+=1
552                                 cvSetTrackbarPos("keystone y", cp, keysy)
553                         elif keyPressed==99:
554                                 keysy=cvGetTrackbarPos("keystone y", cp)
555                                 keysy-=1
556                                 cvSetTrackbarPos("keystone y", cp, keysy)
557 
558                         ###Backup current settings
559                         elif keyPressed==19:
560                                 backup(1)
561 
562                         ###Printing current tranformation matrix
563                         elif keyPressed==112:
564                     if stereo==1:
565                     it1=0
566                     it2=0
567                     print "Transformation matrix for output"+str(stereo)
568                     for it1 in range (0,3):
569                         for it2 in range (0,3):
570                             print "["+str(it1)+", "+str(it2)+"] "+str(matrix1[it1,
                                it2] )
571                             print ""
572                 else:
573                     it1=0
574                     it2=0
575                     print "Transformation matrix for output"+str(stereo)
576 
577                     for it1 in range (0,3):
578                         for it2 in range (0,3):
579                             print "["+str(it1)+", "+str(it2)+"] "+str(matrix2[it1,
                                it2] )
580                             print ""
581 
582                         ###Next calibration step
583                         elif keyPressed==32:
584                                 i+=0.5
585 
586                         ###Jump back one calibration step
587                         elif keyPressed==2:
588                                 i-=1.5
589                                 params=calibStringReplace(params, str(
                                    cvGetTrackbarPos("rotation x", cp)), 1+(9*(
                                    stereo-1)))
590                 params=calibStringReplace(params, str(cvGetTrackbarPos("rotation y
                    ", cp)), 2+(9*(stereo-1)))
591                 params=calibStringReplace(params, str(cvGetTrackbarPos("size", cp)
                    ), 3+(9*(stereo-1)))
592                 params=calibStringReplace(params, str(cvGetTrackbarPos("keystone x
                    ", cp)), 4+(9*(stereo-1)))
593                 params=calibStringReplace(params, str(cvGetTrackbarPos("keystone y
                    ", cp)), 5+(9*(stereo-1)))
594                 params=calibStringReplace(params, str(cvGetTrackbarPos("position x
                    ", cp)), 6+(9*(stereo-1)))
595                                 params=calibStringReplace(params, str(cvGetTrackbarPos
                                    ("position y", cp)), 7+(9*(stereo-1)))
596 
597                         ###Escaping rest of calibration
598                         elif keyPressed==27:
599                                 print "skipping rest of calibration"
600                                 i=5
601 
602                         ###Loading a previously manual saved setting
603                         elif keyPressed==12:
604                                 params=load(0)
605                                 cvSetTrackbarPos("rotation x", "control panel1",
                                    int(params.split()[1]))
606                                 cvSetTrackbarPos("rotation y", "control panel1",
```

```
                                        int(params.split()[2]))
607                                     cvSetTrackbarPos("size", "control panel1", int(
                                        params.split()[3]))
608                                     cvSetTrackbarPos("keystone x", "control panel1",
                                        int(params.split()[4]))
609                                     cvSetTrackbarPos("keystone y", "control panel1",
                                        int(params.split()[5]))
610                                     cvSetTrackbarPos("position x", "control panel1",
                                        int(params.split()[6]))
611                                     cvSetTrackbarPos("position y", "control panel1",
                                        int(params.split()[7]))
612                                     cvSetTrackbarPos("mask", "control panel1", int(
                                        params.split()[8]))
613                                     cvSetTrackbarPos("rotate", "control panel1", int(
                                        params.split()[9]))
614                                     cvSetTrackbarPos("rotation x", "control panel2",
                                        int(params.split()[10]))
615                                     cvSetTrackbarPos("rotation y", "control panel2",
                                        int(params.split()[11]))
616                                     cvSetTrackbarPos("size", "control panel2", int(
                                        params.split()[12]))
617                                     cvSetTrackbarPos("keystone x", "control panel2",
                                        int(params.split()[13]))
618                                     cvSetTrackbarPos("keystone y", "control panel2",
                                        int(params.split()[14]))
619                                     cvSetTrackbarPos("position x", "control panel2",
                                        int(params.split()[15]))
620                                     cvSetTrackbarPos("position y", "control panel2",
                                        int(params.split()[16]))
621
622                             ###Loading last saved setting (autosave)
623                             elif keyPressed==76:
624                                     params=load(1)
625                                     cvSetTrackbarPos("rotation x", "control panel1",
                                        int(params.split()[1]))
626                                     cvSetTrackbarPos("rotation y", "control panel1",
                                        int(params.split()[2]))
627                                     cvSetTrackbarPos("size", "control panel1", int(
                                        params.split()[3]))
628                                     cvSetTrackbarPos("keystone x", "control panel1",
                                        int(params.split()[4]))
629                                     cvSetTrackbarPos("keystone y", "control panel1",
                                        int(params.split()[5]))
630                                     cvSetTrackbarPos("position x", "control panel1",
                                        int(params.split()[6]))
631                                     cvSetTrackbarPos("position y", "control panel1",
                                        int(params.split()[7]))
632                                     cvSetTrackbarPos("mask", "control panel1", int(
                                        params.split()[8]))
633                                     cvSetTrackbarPos("rotate", "control panel1", int(
                                        params.split()[9]))
634                                     cvSetTrackbarPos("rotation x", "control panel2",
                                        int(params.split()[10]))
635                                     cvSetTrackbarPos("rotation y", "control panel2",
                                        int(params.split()[11]))
636                                     cvSetTrackbarPos("size", "control panel2", int(
                                        params.split()[12]))
637                                     cvSetTrackbarPos("keystone x", "control panel2",
                                        int(params.split()[13]))
638                                     cvSetTrackbarPos("keystone y", "control panel2",
                                        int(params.split()[14]))
639                                     cvSetTrackbarPos("position x", "control panel2",
                                        int(params.split()[15]))
640                                     cvSetTrackbarPos("position y", "control panel2",
                                        int(params.split()[16]))
641
642                             ###for debug purposes
643                             else:
644                                     print keyPressed
645             print "Calibration complete \n"
646             backup(0)
647
648             ###using a reference image on the screen that is not used
```

```
649     if stereo==1:
650         src=calibEnd
651         dst = cv.cvCreateImage (cv.cvSize (frame.width*2, frame.height*2), 8, 3)
652         cvWarpPerspective( src,  dst, matrix1, (CV_INTER_LINEAR+
              CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
653     if stereo==2:
654             src=calibEnd
655         dst = cv.cvCreateImage (cv.cvSize (frame.width*2, frame.height*2), 8, 3)
656             cvWarpPerspective( src,  dst, matrix2, (CV_INTER_LINEAR+
                CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
657         cvShowImage(frameName, dst )
658
659 ###Determining action for a key input
660
661 def keys(keyPressed):
662         global activeFrame
663
664         ###Movement of image
665         if keyPressed==100:
666                 posx=cvGetTrackbarPos("position x", "control panel"+str(
                    activeFrame))
667                 posx+=1
668                 cvSetTrackbarPos("position x", "control panel"+str(activeFrame),
                    posx)
669         elif keyPressed==97:
670                 posx=cvGetTrackbarPos("position x", "control panel"+str(
                    activeFrame))
671                 posx-=1
672                 cvSetTrackbarPos("position x", "control panel"+str(activeFrame),
                    posx)
673         elif keyPressed==119:
674                 posy=cvGetTrackbarPos("position y", "control panel"+str(
                    activeFrame))
675                 posy-=1
676                 cvSetTrackbarPos("position y", "control panel"+str(activeFrame),
                    posy)
677         elif keyPressed==115:
678                 posy=cvGetTrackbarPos("position y", "control panel"+str(
                    activeFrame))
679                 posy+=1
680                 cvSetTrackbarPos("position y", "control panel"+str(activeFrame),
                    posy)
681         elif keyPressed==43:
682                 size=cvGetTrackbarPos("size", "control panel"+str(activeFrame))
683                 size+=1
684                 cvSetTrackbarPos("size", "control panel"+str(activeFrame), size)
685         elif keyPressed==45:
686                 size=cvGetTrackbarPos("size", "control panel"+str(activeFrame))
687                 size-=1
688                 cvSetTrackbarPos("size", "control panel"+str(activeFrame), size)
689         elif keyPressed==65:
690             roty=cvGetTrackbarPos("rotation y", "control panel"+str(activeFrame))
691             roty-=1
692             cvSetTrackbarPos("rotation y", "control panel"+str(activeFrame), roty)
693     elif keyPressed==68:
694             roty=cvGetTrackbarPos("rotation y", "control panel"+str(activeFrame))
695             roty+=1
696             cvSetTrackbarPos("rotation y", "control panel"+str(activeFrame), roty)
697     elif keyPressed==83:
698             rotx=cvGetTrackbarPos("rotation x", "control panel"+str(activeFrame))
699             rotx+=1
700             cvSetTrackbarPos("rotation x", "control panel"+str(activeFrame), rotx)
701     elif keyPressed==87:
702             rotx=cvGetTrackbarPos("rotation x", "control panel"+str(activeFrame))
703             rotx-=1
704             cvSetTrackbarPos("rotation x", "control panel"+str(activeFrame), rotx)
705         elif keyPressed==101:
706             keysx=cvGetTrackbarPos("keystone x", "control panel"+str(activeFrame))
707                 keysx-=1
708                 cvSetTrackbarPos("keystone x", "control panel"+str(activeFrame),
                    keysx)
709         elif keyPressed==113:
710                 keysx=cvGetTrackbarPos("keystone x", "control panel"+str(
```

```
                    activeFrame))
711                 keysx+=1
712                 cvSetTrackbarPos("keystone x", "control panel"+str(activeFrame),
                        keysx)
713             elif keyPressed==122:
714                 keysy=cvGetTrackbarPos("keystone y", "control panel"+str(
                        activeFrame))
715                 keysy+=1
716                 cvSetTrackbarPos("keystone y", "control panel"+str(activeFrame),
                        keysy)
717             elif keyPressed==99:
718                 keysy=cvGetTrackbarPos("keystone y", "control panel"+str(
                        activeFrame))
719                 keysy-=1
720                 cvSetTrackbarPos("keystone y", "control panel"+str(activeFrame),
                        keysy)
721
722             ###Resetting image to default
723             elif keyPressed==114:
724                 global matrix1
725                 global matrix2
726                     backup(0)
727                     print "resetting"
728
729                     cvSetTrackbarPos("rotation x", "control panel1", 500)
730                     cvSetTrackbarPos("rotation y", "control panel1", 500)
731                     cvSetTrackbarPos("size", "control panel1", 1000)
732                     cvSetTrackbarPos("keystone x", "control panel1", 500)
733                     cvSetTrackbarPos("keystone y", "control panel1", 500)
734                     cvSetTrackbarPos("position x", "control panel1", 500)
735                     cvSetTrackbarPos("position y", "control panel1", 500)
736                     cvSetTrackbarPos("mask", "control panel1", 0)
737                     cvSetTrackbarPos("rotate", "control panel1", 0)
738                     cvSetTrackbarPos("rotation x", "control panel2", 500)
739                     cvSetTrackbarPos("rotation y", "control panel2", 500)
740                     cvSetTrackbarPos("size", "control panel2", 1000)
741                     cvSetTrackbarPos("keystone x", "control panel2", 500)
742                     cvSetTrackbarPos("keystone y", "control panel2", 500)
743                     cvSetTrackbarPos("position x", "control panel2", 500)
744                     cvSetTrackbarPos("position y", "control panel2", 500)
745
746             matrix1[0,0]=1
747             matrix1[0,1]=0
748             matrix1[0,2]=0
749             matrix1[1,0]=0
750             matrix1[1,1]=1
751             matrix1[1,2]=0
752             matrix1[2,0]=0
753             matrix1[2,1]=0
754             matrix1[2,2]=1
755             matrix2[0,0]=1
756             matrix2[0,1]=0
757             matrix2[0,2]=0
758             matrix2[1,0]=0
759             matrix2[1,1]=1
760             matrix2[1,2]=0
761             matrix2[2,0]=0
762             matrix2[2,1]=0
763                     matrix2[2,2]=1
764
765             ###Saving image parameters
766             elif keyPressed==19:
767                     backup(1)
768
769             ###Exiting and autosaving
770             elif keyPressed==32:
771                     backup(0)
772                     return -1
773             ###Exit without autosaving (escape)
774             elif keyPressed==27:
775                     return -1
776
777             ###Recalibrating active frame
```

```
778         elif keyPressed==67:
779             backup(0)
780                 print "Recalibrating active frame"
781                 frame = cvQueryFrame( capture1 )
782                 dumpframe=cvQueryFrame(capture2)###to maintain sync between videos
783         calibEnd_src = cvLoadImage( "calibration/Calib-end.bmp" )
784         calibEnd=cvCreateImage(cvSize(frame.width,frame.height), 8, 3)
785             cvResize(calibEnd_src, calibEnd, CV_INTER_AREA )
786 
787                 if activeFrame==1:
788                     matrix2
789             src=calibEnd
790             dst = cv.cvCreateImage (cv.cvSize (frame.width*2, frame.height*2), 8,
                  3)
791             cvWarpPerspective( src,  dst, matrix2, (CV_INTER_LINEAR+
                  CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
792                     cvShowImage("output2", dst )
793                     calibrate(frame, "output1")
794 
795                 else:
796                     matrix1
797             src=calibEnd
798             dst = cv.cvCreateImage (cv.cvSize (frame.width*2, frame.height*2), 8,
                  3)
799             cvWarpPerspective( src,  dst, matrix1, (CV_INTER_LINEAR+
                  CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
800                     cvShowImage("output1", dst )
801                     calibrate(frame, "output2")
802 
803         ###Switch active frame
804         elif keyPressed==9:
805                 if activeFrame==1:
806                         activeFrame=2
807                 else:
808                         activeFrame=1
809 
810         ###print transformation matrix
811         elif keyPressed==112:
812             if activeFrame==1:
813             it1=0
814             it2=0
815             for it1 in range (0,3):
816                 for it2 in range (0,3):
817                     print "["+str(it1)+", "+str(it2)+"] "+str(matrix1[it1, it2] )
818                     print ""
819         else:
820             it1=0
821             it2=0
822             for it1 in range (0,3):
823                 for it2 in range (0,3):
824                     print "["+str(it1)+", "+str(it2)+"] "+str(matrix2[it1, it2] )
825                     print ""
826         ###Loading a previously manual saved setting
827         elif keyPressed==12:
828                 params=load(0)
829                 cvSetTrackbarPos("rotation x", "control panel1", int(params.split
                      ()[1]))
830                 cvSetTrackbarPos("rotation y", "control panel1", int(params.split
                      ()[2]))
831                 cvSetTrackbarPos("size", "control panel1", int(params.split()[3])
                      )
832                 cvSetTrackbarPos("keystone x", "control panel1", int(params.split
                      ()[4]))
833                 cvSetTrackbarPos("keystone y", "control panel1", int(params.split
                      ()[5]))
834                 cvSetTrackbarPos("position x", "control panel1", int(params.split
                      ()[6]))
835                 cvSetTrackbarPos("position y", "control panel1", int(params.split
                      ()[7]))
836                 cvSetTrackbarPos("mask", "control panel1", int(params.split()[8])
                      )
837                 cvSetTrackbarPos("rotate", "control panel1", int(params.split()
                      [9]))
```

```
838                 cvSetTrackbarPos("rotation x", "control panel2", int(params.split
                        ()[10]))
839                 cvSetTrackbarPos("rotation y", "control panel2", int(params.split
                        ()[11]))
840                 cvSetTrackbarPos("size", "control panel2", int(params.split()
                        [12]))
841                 cvSetTrackbarPos("keystone x", "control panel2", int(params.split
                        ()[13]))
842                 cvSetTrackbarPos("keystone y", "control panel2", int(params.split
                        ()[14]))
843                 cvSetTrackbarPos("position x", "control panel2", int(params.split
                        ()[15]))
844                 cvSetTrackbarPos("position y", "control panel2", int(params.split
                        ()[16]))
845
846         ###Loading last saved setting (autosave)
847         elif keyPressed==76:
848                 params=load(1)
849                 cvSetTrackbarPos("rotation x", "control panel1", int(params.split
                        ()[1]))
850                 cvSetTrackbarPos("rotation y", "control panel1", int(params.split
                        ()[2]))
851                 cvSetTrackbarPos("size", "control panel1", int(params.split()[3])
                        )
852                 cvSetTrackbarPos("keystone x", "control panel1", int(params.split
                        ()[4]))
853                 cvSetTrackbarPos("keystone y", "control panel1", int(params.split
                        ()[5]))
854                 cvSetTrackbarPos("position x", "control panel1", int(params.split
                        ()[6]))
855                 cvSetTrackbarPos("position y", "control panel1", int(params.split
                        ()[7]))
856                 cvSetTrackbarPos("mask", "control panel1", int(params.split()[8])
                        )
857                 cvSetTrackbarPos("rotate", "control panel1", int(params.split()
                        [9]))
858                 cvSetTrackbarPos("rotation x", "control panel2", int(params.split
                        ()[10]))
859                 cvSetTrackbarPos("rotation y", "control panel2", int(params.split
                        ()[11]))
860                 cvSetTrackbarPos("size", "control panel2", int(params.split()
                        [12]))
861                 cvSetTrackbarPos("keystone x", "control panel2", int(params.split
                        ()[13]))
862                 cvSetTrackbarPos("keystone y", "control panel2", int(params.split
                        ()[14]))
863                 cvSetTrackbarPos("position x", "control panel2", int(params.split
                        ()[15]))
864                 cvSetTrackbarPos("position y", "control panel2", int(params.split
                        ()[16]))
865
866         ###Pausing the video
867         elif keyPressed==16:
868             print "pause, press any key to continue"
869             capture1
870             capture2
871             masking=cvGetTrackbarPos("mask", "control panel1")
872                 rotate=cvGetTrackbarPos("rotate", "control panel1")
873
874                 frame1 = cvQueryFrame( capture1 )
875                 frame2 = cvQueryFrame( capture2 )
876
877                 maskedFrame1=cvCreateImage(cvSize(frame1.width,frame1.height),
                        8,3)
878         rotframe1=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
879         maskedFrame2=cvCreateImage(cvSize(frame1.width,frame1.height), 8,3)
880         rotframe2=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
881         dst1 = cv.cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2), 8,
                3)
882             dst2 = cv.cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2),
                    8, 3)
883
884             cvAddWeighted(frame1, alpha, mask, beta, 0.0, maskedFrame1)
```

```
885           cvAddWeighted(frame2, alpha, mask, beta, 0.0, maskedFrame2)
886
887           cvWarpAffine( maskedFrame1, rotframe1, rotation,CV_INTER_LINEAR+
                CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
888           cvWarpAffine( maskedFrame2, rotframe2, rotation,CV_INTER_LINEAR+
                CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
889
890           cvWarpPerspective( rotframe1, dst1, matrix1, (CV_INTER_LINEAR+
                CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
891           cvWarpPerspective( rotframe2, dst2, matrix2, (CV_INTER_LINEAR+
                CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
892
893           cvShowImage("output1", dst1 )
894           cvShowImage("output2", dst2 )
895
896           keyPressed=cvWaitKey(10)
897           while(keyPressed==-1):
898                keyPressed=cvWaitKey(10)
899
900           ###For debug operations
901           else:
902                     print keyPressed
903
904 ###Mouse event listener, used as first step in the calibration
905
906 def mouse(event, x, y, flags, param):
907      global spots
908
909      if event ==1:
910           spots.append(cvPoint(x, y))
911           if len(spots)==1:
912                print "Click bottom right corner"
913           elif len(spots)==2:
914                print "Click top left corner"
915           elif len(spots)== 3:
916                     print "Click top right corner"
917           #print len(spots)
918           #print str(x)+", "+str(y)
919
920
921 ###Video loop for low framerate performance.
922
923 def videoLoop(framecount, framesincelast, lastcount, FPS, frame1):
924
925      ###Loading global parameters
926      matrix1
927      matrix2
928      alpha
929      beta
930      mask
931      activeFrame
932      global hiFPS
933      global maxFPS
934      global capture1
935      global capture2
936
937      ###creating empty images with size according to source
938      ###Used for masks, rotations and destination images
939      maskedFrame1=cvCreateImage(cvSize(frame1.width,frame1.height), 8,3)
940      rotframe1=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
941      maskedFrame2=cvCreateImage(cvSize(frame1.width,frame1.height), 8,3)
942      rotframe2=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
943      dst1 = cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2), 8, 3)
944      dst2 = cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2), 8, 3)
945
946      ###The loop itself
947      while True:
948           framecount+=1
949
950           ###Grabbing frames from source video
951           frame1 = cvQueryFrame( capture1 )
952           frame2= cvQueryFrame( capture2 )
953           if not frame1 or not frame2:
```

```
954
955                 ###Reloading source if no frames are available
956                 capture1 = cvCreateFileCapture( captureFile1 )
957                 capture2 = cvCreateFileCapture( captureFile2 )
958                 frame1 = cvQueryFrame( capture1 )
959                 frame2 = cvQueryFrame( capture2 )
960
961                 if not frame1 or not frame2:
962
963                     print "Error when looping video, exiting. . ."
964                 break
965
966             ###showing image before transformations
967             cvShowImage("source material", frame1)
968
969             ###Getting mask and rotation parameters
970             masking=cvGetTrackbarPos("mask", "control panel1")
971             rotate=cvGetTrackbarPos("rotate", "control panel1")
972
973             ###Choosing correct steps, for performance only needed operations are
                    performed
974             ###Warping of source image will of course always be done.
975
976             ###If both masking and rotation operations are needed
977             if masking!=0 and rotate!=0:
978                 cvAddWeighted(frame1, alpha, mask, beta, 0.0, maskedFrame1)
979                 cvAddWeighted(frame2, alpha, mask, beta, 0.0, maskedFrame2)
980
981                 cvWarpAffine( maskedFrame1, rotframe1, rotation,CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
982             cvWarpAffine( maskedFrame2, rotframe2, rotation,CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
983
984                 cvWarpPerspective( rotframe1, dst1, matrix1, (CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
985             cvWarpPerspective( rotframe2, dst2, matrix2, (CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
986
987             ###If only rotation is needed
988             elif masking!=0 and rotate==0:
989                 cvAddWeighted(frame1, alpha, mask, beta, 0.0, maskedFrame1)
990                 cvAddWeighted(frame2, alpha, mask, beta, 0.0, maskedFrame2)
991
992                 cvWarpPerspective( maskedFrame1, dst1, matrix1, (CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
993             cvWarpPerspective( maskedFrame2, dst2, matrix2, (CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
994
995             ###If only masking is needed
996             elif masking==0 and rotate!=0:
997                 cvWarpAffine( frame1, rotframe1, rotation,CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
998             cvWarpAffine( frame2, rotframe2, rotation,CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS,cvScalarAll(0) )
999
1000                cvWarpPerspective( rotframe1, dst1, matrix1, (CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1001            cvWarpPerspective( rotframe2, dst2, matrix2, (CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1002
1003            ###If neither operations are needed
1004            elif masking==0 and rotate==0:
1005                cvWarpPerspective( frame1, dst1, matrix1, (CV_INTER_LINEAR+
                        CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1006            cvWarpPerspective( frame2, dst2, matrix2, (CV_INTER_LINEAR+
                    CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1007
1008        ###The loop cannot, under any normal cirumstance end here.
1009        else:
1010            dst1=frame1
1011            dst2=frame2
1012
1013            ###Detecting if keys are pressed
```

```
1014            keyPressed=cvWaitKey(100)
1015            if keyPressed!=-1:
1016                    if keys(keyPressed)==-1:
1017                            sys.exit(0)
1018
1019        ###Crude FPS counter and statistics management
1020        thisframetime=time.ctime(time.time())
1021        if thisframetime==lastcount:
1022            framesincelast+=1
1023        else:
1024            if framesincelast !=0:
1025                FPS.append(framesincelast)
1026            try:
1027                print str(framesincelast)+" current "+str(sum(FPS, 0.0)/len(FPS)) + "
                       average fps, " + str(sum(FPS, 0))+ " total frames."
1028
1029                ###Jump to high speed loop if framerate is above threshold
1030                if framesincelast>maxFPS:
1031                    print "FPS too high, entering monitor mode"
1032                    hiFPS=1
1033                    break
1034            except ZeroDivisionError:
1035                print "no FPS"
1036            framesincelast=0
1037            lastcount=time.ctime(time.time())
1038
1039            ###Displaying the warped images
1040            cvShowImage("output1", dst1 )
1041            cvShowImage("output2", dst2 )
1042
1043
1044    ###Video loop with FPS controls for when performance is higher than needed
1045
1046    def videoFPSLoop(framecount, framesincelast, lastcount, FPS, frame1):
1047
1048        ###Loading global parameters
1049        matrix1
1050        matrix2
1051        alpha
1052        beta
1053        mask
1054            activeFrame
1055            global hiFPS
1056            global maxFPS
1057            global capture1
1058            global capture2
1059
1060            ###Saving time for FPS control
1061        lastFrame = datetime.datetime.now()
1062
1063            ###creating empty images with size according to source
1064            ###Used for masks, rotations and destination images
1065            maskedFrame1=cvCreateImage(cvSize(frame1.width,frame1.height), 8,3)
1066            rotframe1=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
1067            maskedFrame2=cvCreateImage(cvSize(frame1.width,frame1.height), 8,3)
1068            rotframe2=cvCreateImage(cvSize(frame1.width*2,frame1.height*2), 8,3)
1069            dst1 = cv.cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2), 8,
                   3)
1070            dst2 = cv.cvCreateImage (cv.cvSize (frame1.width*2, frame1.height*2), 8,
                   3)
1071
1072            ###The loop itself
1073            while True:
1074                framecount+=1
1075
1076                    ###Grabbing frames from source video
1077                    frame1 = cvQueryFrame( capture1 )
1078                    frame2= cvQueryFrame( capture2 )
1079                    if not frame1 or not frame2:
1080
1081                        ###Reloading source if no frames are available
1082                        capture1 = cvCreateFileCapture( captureFile1 )
1083                        capture2 = cvCreateFileCapture( captureFile2 )
```

```
1084                    frame1 = cvQueryFrame( capture1 )
1085                    frame2 = cvQueryFrame( capture2 )
1086
1087                if not frame1 or not frame2:
1088
1089                    print "Error when looping video, exiting. . ."
1090                    break
1091
1092                ###showing image before transformations
1093                cvShowImage("source material", frame1)
1094
1095                ###Getting mask and rotation parameters
1096                masking=cvGetTrackbarPos("mask", "control panel1")
1097                rotate=cvGetTrackbarPos("rotate", "control panel1")
1098
1099                ###Choosing correct steps, for performance only needed operations
                       are performed
1100                ###Warping of source image will of course always be done.
1101
1102                ###If both masking and rotation operations are needed
1103                if masking!=0 and rotate!=0:
1104                    cvAddWeighted(frame1, alpha, mask, beta, 0.0, maskedFrame1)
1105                cvAddWeighted(frame2, alpha, mask, beta, 0.0, maskedFrame2)
1106
1107                cvWarpAffine( maskedFrame1, rotframe1, rotation ,CV_INTER_LINEAR+
                       CV_WARP_FILL_OUTLIERS, cvScalarAll(0) )
1108            cvWarpAffine( maskedFrame2, rotframe2, rotation ,CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS, cvScalarAll(0) )
1109
1110                    cvWarpPerspective( rotframe1, dst1, matrix1, (CV_INTER_LINEAR
                           +CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1111            cvWarpPerspective( rotframe2, dst2, matrix2, (CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1112
1113                ###If only rotation is needed
1114                elif masking!=0 and rotate==0:
1115                    cvAddWeighted(frame1, alpha, mask, beta, 0.0, maskedFrame1)
1116                    cvAddWeighted(frame2, alpha, mask, beta, 0.0, maskedFrame2)
1117
1118                cvWarpPerspective( maskedFrame1, dst1, matrix1, (CV_INTER_LINEAR+
                       CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1119            cvWarpPerspective( maskedFrame2, dst2, matrix2, (CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1120
1121                ###If only masking is needed
1122                elif masking==0 and rotate!=0:
1123                cvWarpAffine( frame1, rotframe1, rotation ,CV_INTER_LINEAR+
                       CV_WARP_FILL_OUTLIERS, cvScalarAll(0) )
1124            cvWarpAffine( frame2, rotframe2, rotation ,CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS, cvScalarAll(0) )
1125
1126                cvWarpPerspective( rotframe1, dst1, matrix1, (CV_INTER_LINEAR+
                       CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1127            cvWarpPerspective( rotframe2, dst2, matrix2, (CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1128
1129                ###If neither operations are needed
1130                elif masking==0 and rotate==0:
1131                cvWarpPerspective( frame1, dst1, matrix1, (CV_INTER_LINEAR+
                       CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1132            cvWarpPerspective( frame2, dst2, matrix2, (CV_INTER_LINEAR+
                   CV_WARP_FILL_OUTLIERS), cvScalarAll(0) )
1133
1134        ###The loop cannot, under any normal cirumstance end here.
1135            else:
1136                dst1=frame1
1137            dst2=frame2
1138
1139                ###Detecting if keys are pressed
1140            keyPressed=cvWaitKey(10)
1141            if keyPressed!=-1:
1142                if keys(keyPressed)==-1:
1143                        sys.exit(0)
```

```
1144
1145                ####Crude FPS counter, management and statistics
1146                thisframetime=time.ctime(time.time())
1147                if thisframetime==lastcount:
1148                    framesincelast+=1
1149                else:
1150                    if framesincelast !=0:
1151                    FPS.append(framesincelast)
1152                try:
1153                    print str(framesincelast)+" current "+str(sum(FPS, 0.0)/len(FPS))
                        + " average fps, " + str(sum(FPS, 0.0))+ " total frames."
1154
1155                    ###Jump to low speed loop if framerate is too low
1156                    if  (framesincelast < (maxFPS/2) and framesincelast != 0):
1157                        print "FPS below threshhold, entering non-monitor-mode"
1158                        hiFPS=0
1159                        break
1160                except ZeroDivisionError:
1161                    print "no FPS"
1162                framesincelast=0
1163                lastcount=time.ctime(time.time())
1164
1165          ###Sleeping until it is time to display the frame
1166          time.sleep (max(float(((1./maxFPS)*1000000)-(datetime.datetime.now()-
               lastFrame).microseconds)/1000000, 0))
1167          lastFrame = datetime.datetime.now()
1168
1169              ###Displaying the warped frames
1170              cvShowImage("output1", dst1 )
1171              cvShowImage("output2", dst2 )
1172
1173 ###Main program start
1174 if __name__ == "__main__":
1175     print str(platform.system())+" "+str(platform.release())+ " "+ str(sys.
           getwindowsversion())+" . . . "
1176
1177     ###Getting global parameters
1178     matrix1
1179     matrix2
1180     alpha
1181     beta
1182     mask
1183     activeFrame
1184     hiFPS
1185     capture1
1186     capture2
1187
1188     ###initializing capture
1189     capture = None
1190     if len(sys.argv)==1:
1191         print "Capturing from webcam, debug only . . . "
1192         capture1 = cvCreateCameraCapture( 0 )
1193         capture2 = capture1
1194
1195     ###Getting video source from input parameters
1196     elif len(sys.argv)==3:
1197         captureFile1=sys.argv[1]
1198         captureFile2=sys.argv[2]
1199
1200         print "Capturing from file, source path: "+captureFile1+ " and " +
               captureFile2+" . . ."
1201         capture1 = cvCreateFileCapture( captureFile1 )
1202         capture2 = cvCreateFileCapture( captureFile2 )
1203     else:
1204         print "You must specify the path of two video files when launching this
               script"
1205         sys.exit(-1)
1206
1207     ###If no capture media can be found
1208     if not capture1:
1209         if not capture2:
1210                 print "Could not initialize capturing. . ."
1211                 sys.exit(-1)
```

```
1212
1213        ###Grabbing frame for size measurement and check
1214        frame1 = cvQueryFrame( capture1 )
1215        frame2 = cvQueryFrame( capture2 )
1216        if not (frame1.width==frame2.width and frame1.height==frame2.height):
1217            print "sources aren't of identical size. They must have identical height&
                width"
1218            print str(frame1.width) +" "+ str(frame2.width)+" " + str(frame1.height)+"
                " + str(frame2.height)
1219            print frame1.width==frame2.width
1220            print frame1.height==frame2.height
1221            sys.exit(-1)
1222
1223        ###counting number of masks in folder and resizing mask master to fit source
1224        num_masks=0
1225        mask2=cvCreateImage(cvSize(frame1.width, frame1.height), 8, 3)
1226        cvResize(mask, mask2, CV_INTER_AREA );
1227        mask=mask2
1228        masks=os.listdir('./masks')
1229        for file in os.listdir('./masks'):
1230            num_masks+=1
1231
1232        ###Creating Control Panel windows
1233        cvNamedWindow ("control panel1")
1234        cvResizeWindow("control panel1",300,500)
1235        cvNamedWindow ("control panel2")
1236        cvResizeWindow("control panel2",300,500)
1237
1238        ###Loading last used parameters
1239        params=load(1)
1240
1241        ####Adding trackbars to Control Panels
1242        cvCreateTrackbar("rotation x", "control panel1", int(params.split()[1]),
            1000, update1)
1243        cvCreateTrackbar("rotation y", "control panel1", int(params.split()[2]),
            1000, update1)
1244        cvCreateTrackbar("size", "control panel1", int(params.split()[3]), 2000,
            update1)
1245        cvCreateTrackbar("keystone x", "control panel1", int(params.split()[4]),
            1000, update1)
1246        cvCreateTrackbar("keystone y", "control panel1", int(params.split()[5]),
            2000, update1 )
1247        cvCreateTrackbar("position x", "control panel1", int(params.split()[6]),
            2000, update1 )
1248        cvCreateTrackbar("position y", "control panel1", int(params.split()[7]),
            2000, update1)
1249        cvCreateTrackbar("mask", "control panel1", int(params.split()[8]), num_masks
            -1, maskAndRotate)
1250        cvCreateTrackbar("rotate", "control panel1", int(params.split()[9]), 3,
            maskAndRotate)
1251
1252        cvCreateTrackbar("rotation x", "control panel2", int(params.split()[10]),
            1000, update2)
1253        cvCreateTrackbar("rotation y", "control panel2", int(params.split()[11]),
            1000, update2)
1254        cvCreateTrackbar("size", "control panel2", int(params.split()[12]), 2000,
            update2)
1255        cvCreateTrackbar("keystone x", "control panel2", int(params.split()[13]),
            1000, update2)
1256        cvCreateTrackbar("keystone y", "control panel2", int(params.split()[14]),
            2000, update2 )
1257        cvCreateTrackbar("position x", "control panel2", int(params.split()[15]),
            2000, update2 )
1258        cvCreateTrackbar("position y", "control panel2", int(params.split()[16]),
            2000, update2)
1259
1260        ###Updating all parameters to properly display image
1261        update1(0)
1262        update2(0)
1263        maskAndRotate(0)
1264
1265        ###Creating output images
1266        cvNamedWindow( "output1", 0)
```

```
1267        calibrate(frame1, "output1")
1268
1269        cvNamedWindow( "output2", 0)
1270        calibrate(frame2, "output2")
1271
1272        cvNamedWindow ("source material")
1273
1274        ###Initializing framerate counters and statistical material
1275        framecount=0
1276        framesincelast=0
1277        lastcount=0
1278        FPS=[]
1279        hiFPS=0
1280
1281        ###Loop for jumping between high and low framerate loops
1282        while (1):
1283            if hiFPS==1:
1284                print "entering high FPS mode"
1285                videoFPSLoop(framecount, framesincelast, lastcount, FPS, frame1)
1286            if hiFPS==0:
1287                print "entering low FPS mode"
1288                videoLoop(framecount, framesincelast, lastcount, FPS, frame1)
1289
1290        ###Destroying all windows and cleaning up for closing of the program
1291        cvDestroyWindow("output1")
1292        cvDestroyWindow("output2")
1293        cvDestroyWindow("source material")
1294        cvDestroyWindow("control panel1")
1295        cvDestroyWindow("control panel2")
1296
1297        cvWaitKey()
```

D Speedup Enhancements

The last days before submission, during the cleanup of the code, some improvements on how to speed up the code was discovered by chance. The two major improvements were in the keyboard event listener and the upscaling of images. The suggested changes have been included in an experimental version, called testBed.py. This is a version of SingleProjectCornerpin.py, with the changes outlined here. It is not thoroughly tested and known errors include problems with masks when the image is rotated.

D.1 Keyboard Listener

It seems as though the keyboard listener in openCV runs in the same thread as the rest of the program, and the polling delay that is given when calling it actually halts the running of the program for that time. This means that having a 100 ms polling interval gives a 90 ms added delay per frame versus having a 10 ms polling interval. Obviously, setting this as low as possible is the best solution to increase frame rate. Setting the delay to 0, lets the program wait indefinitely, and because the delay must be an integer, 1 ms is the lowest possible, using this gives a significant increase in frame rate. This will also increase the maximum bandwidth usage for the program to 5 kb/s. Even with several embedded computers using this at the same time, both MIDI and DMX512 still will have no problems handling the traffic loads.

By replacing

```
1 keyPressed=cvWaitKey(10)
```

with

```
1 keyPressed=cvWaitKey(1)
```

the program video runs smoother, in most cases.

D.2 Image rescaling

One of the problems encountered during initial programming was that the displayed image would not retain its initial shape if it is rotated, and masks added to the image would not fit, because the height and width of the image was

flipped. This was solved by upscaling the image to double size while doing most of the work on it, this also had the effect that warping the image produced less visible jaggy edges. By not upscaling the image, the warping is much faster, as there is less data that needs processing, but the outer edges of an image might be clipped.

By removing all references to the doubling of the sizes for the images, for instance

```
1 dst = cvCreateImage (cvSize (frame.width*2, frame.height*2), 8, 3)
```

and instead using

```
1 dst = cvCreateImage (cvSize (frame.width, frame.height), 8, 3)
```

one can get good speed increases. Tests with uncompressed AVI in 720p shows that one can get frame rates in the area of 30 frames per second while adding effects on the image, using a current high-end computer.

In addition to these two speedups, the testBed.py has removed the two different video loops, replacing them with only one. This was done because of the lack of improvement in using the high-speed loop, the time used for closer monitoring of the frame rate was not significant compared to other parts of the code. I have also experimented with combining the key listener and wait interval for the frame rate lock, by combining the two lines

```
1 keyPressed=cvWaitKey(10)
```

and

```
1 time.sleep (max(float(((1./maxFPS)*1000000)-(datetime.datetime.now()-
    lastFrame).microseconds)/1000000, 0))
```

to a single operation, using the key wait interval to slow down operations if the frame rate is too high, but this has not worked very well because the key listener seems to be less accurate with regards to timing than the built-in timers, producing irregular frame rates that often were either too slow or too fast.

The speedups I have found are probably not the only ones that are possible, and they are not tested and debugged enough to be put into the final code. With some more work, these probably could have been refined to work properly.