論文 / 著書情報
Article / Book Information

題目(和文)	ビデオ投影とカラーカメラによる移動体平面上のマーカーレス対話型拡張現実
Title(English)	Markerless interactive augmented reality on moving planar surfaces with video projection and a color camera
著者(和文)	AudetSamuel Flaurent
Author(English)	SAMUEL AUDET
出典(和文)	学位:博士(工学), 学位授与機関:東京工業大学, 報告番号:甲第8695号, 授与年月日:2012年3月26日, 学位の種別:課程博士, 審査員:奥富 正敏
Citation(English)	Degree:Doctor (Engineering), Conferring organization: Tokyo Institute of Technology, Report number:甲第8695号, Conferred date:2012/3/26, Degree Type:Course doctor, Examiner:
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Markerless Interactive Augmented Reality on Moving Planar Surfaces with Video Projection and a Color Camera

Samuel Audet

Submitted in Partial Fulfillment of the
Requirements of the Degree of
Doctor of Engineering

Masatoshi OKUTOMI & Masayuki TANAKA Laboratory

Department of Mechanical and Control Engineering
Graduate School of Science and Engineering
Tokyo Institute of Technology
Toyko, Japan

February 2012

Abstract

Augmented reality applications based on video projection displaying content directly on surfaces should also perform tracking and make sure the display remains aligned even when the targets move, but the projection can severely alter their appearances to the point where traditional tracking methods based on computer vision algorithms fail. Current solutions consider the displayed content as interference and largely depend on channels orthogonal to visible light. They cannot directly align projector images with real-world surfaces, even though this may be the actual goal. We propose instead to model the light emitted by projectors and reflected into cameras, and to consider the displayed content as additional information useful for aligning and tracking directly both surfaces and objects such as hands for interactivity. Using a color camera, our current open-source software implementation successfully tracks with subpixel accuracy a planar surface of diffuse reflectance properties at an average of thirty frames per second on commodity hardware, providing a solid base for future enhancements and sufficiently usable in our opinion to start evaluating potential applications of this low cost solution, whose implementations for CPUs and GPUs (graphics processing unit) can be further accelerated and generalized.

Further, projector-camera systems drive applications in many other fields in the form of, for example, measurement and inspection systems, which could also benefit from our user-friendly calibration method. Although not always required, knowledge of the geometric and color parameters can often facilitate development of new systems. In particular, users often need to find their internal and external parameters via geometric and color calibration. For this process, both a printed pattern and a projector pattern are needed, but they can easily interfere with each other. Current methods compensate by decoupling their calibrations or by leveraging structured light and color channels, but the required manipulations are not user-friendly. Therefore, we cannot expect normal users to execute the procedure, which can also become a burden for researchers. To make the calibration process easier, we propose a method that uses fiducial markers, from which we can easily derive a *prewarp* that, once applied to the projector calibration pattern, prevents its interference. Using our open-source software, we confirmed that users can easily calibrate a projector-camera system in less than one minute, which we consider to be user-friendly, while still achieving typical subpixel geometric accuracy and less than one percent of color intensity error.

Acknowledgements

First and foremost, I am much indebted to my advisors Masatoshi OKUTOMI and Masayuki TANAKA for the research opportunities, the resources, the trust, and the guidance they have given me. I thank them for their patience and understanding during my stay as well as for the editorial help with my thesis. Furthermore, based on their recommendations, the Ministry of Education, Culture, Sports, Science and Technology (MEXT) of the Japanese Government awarded me a generously fully paid research scholarship, without which this work would not have been possible. I will also be forever thankful to Linda Cooper for teaching me how to write scientific material properly and clearly. Without her wonderful seminars, I fear that I would have never received the necessary knowledge to communicate effectively my findings to others, nor had the courage to defy rampant dogmas within the academic community. Finally, thank you to the members of my close family, in particular my wife Megumi KOBASHI, my mother Constance Boisvert, her husband Gilles Parent, my father Denis Audet, and my sister Élisabeth Audet, without whose moral support I am afraid would have lacked the motivation to complete successfully this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Traditional applications of augmented reality superimpose generated images onto the real world through goggles or monitors held between objects of interest and the user. The display must usually follow objects and developers often choose cameras to perform tracking, as computer vision methods are flexible and nonintrusive. Since the augmented objects remain in reality unchanged, we do not need to change the image processing algorithms either. To track surfaces as well as other objects, such as hands, for interaction with

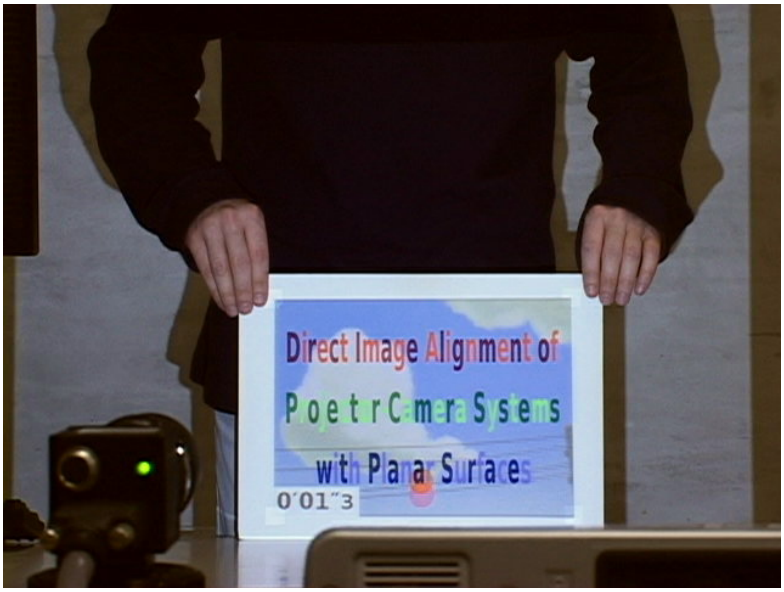


Figure 1.1: Snapshot of our demo video showing the patterns of Figure 6.7, where the system has aligned the projector displayed pattern with the printed one (alongside a chronometer, a video animation, and a red virtual ball)

traditional augmented reality or other purposes, one can directly exploit existing computer vision techniques [27, 34]. However, that no longer holds when using video projection to alter directly the appearance of surfaces and display computer graphics or data, as exemplified in Figure 1.1. Because the appearance of the objects may be severely affected, most vision-based methods fail under those conditions since they consider the light from the projector as unwanted interference.

1.1 Literature Review

For this reason, to avoid complications, current applications and existing systems either exploit information channels that do not overlap with the displayed content, relying on tracking methods that depend on something else than visible light or impeding it, or make assumptions that restrict their usefulness, placing limitations on the range of possible interactions.

Some count on fiducial markers that users must paste on objects, such as iLamps [42] and the volumetric data visualizer by Gupta and Jaynes [25], or a special tracking system not based on visible light, such as Dynamic Shader Lamps [9], but users need to paste specially prepared markers to objects they might want to interact with, and more recently with Foldable Interactive Displays [33], whose tracking system relies on near-infrared LEDs attached to objects. Even Wear Ur World [38], a recent and otherwise well-received interactive projector-camera system, relies on colored thimbles as markers. Other researchers have designed imperceptible structured light, as used in the Office of the Future [43]. However, this not only reduces the dynamic range of the projector, but requires a synchronized projector-camera system running at frequencies higher than 60 Hz to avoid flicker. More simply the Perceptive Workbench [36] has adopted computer vision to track using near-infrared imagery thus avoiding interferences from the projector. Although all the above options work, they cannot directly align the displayed content with real-world texture. They solely depend on accurate geometric calibration between the sensors, the projectors, and the real-world objects. Further, geometrically calibrating projectors and nonvisible-light cameras or sensors whose light spectra, their information channels, do not overlap can be problematic, albeit not impossible. Also, the system does not see the same thing as the user, possibly causing confusion when the algorithm fails. In contrast, Tele-Graffiti [50] features a paper tracking algorithm that detects the orientation of a piece of paper or a clipboard inside images captured from a normal camera. The authors designed the algorithm robustly enough to work in the one limited case where the edges of the rectangular object con-

trasts sharply with the background and for hands moving over plain texture surfaces only, but they still consider the light emitted from the projector as unwanted interference.

In a nutshell, the performance of markerless tracking leaves to be desired. If augmented reality based on video projection, also known as spatial augmented reality [11], is to become the basis for a new paradigm of user interaction, we need more general, robust, and easy-to-use methods.

1.2 Proposed Method

Contrarily to existing approaches, instead of considering the projected content as unwanted interference, we propose to harness its knowledge as additional information useful for alignment and tracking, information that can help the system align it with real-world objects and track hands over it, a method we initially presented at a conference [4], but have since enhanced both its speed and robustness [6] as well as proposed a way to add hand tracking for interactivity [5]. More specifically, we derived a direct image alignment algorithm that takes projector illumination into account using a generative model. It models how the light coming out of the projector reflects onto a real-world object and comes back in the camera. Not only can we easily and accurately calibrate such a system, as explained in Chapter 3, but since the projector is part of the feedback loop, the alignment algorithm does not rely solely on accurate geometric calibration.

In theory, the model can work with an uncalibrated system, but a calibrated one can more easily and robustly achieve successful alignment. Calibration amounts to finding the internal and external geometric and color parameters of both camera and projector. Although calibration methods exist, they are cumbersome, impractical, and could not realistically become part of an augmented reality system that anyone could use under a large range of casual conditions, such as the one shown in Figure 1.2. For this reason we also designed geometric and color calibration methods, which we briefly introduce next, that users of any background may easily perform.

1.3 User-Friendly Calibration

To calibrate geometrically projector-camera systems, many methods [1, 2, 24, 32, 41, 55] among others, operate in two phases: They first calibrate cameras, then they calibrate projectors. Calibrating cameras in one way and projectors in another automatically doubles the effort. To reduce the amount of work,

other researchers have proposed methods that integrate both calibrations together. They either exploit structured light or color channels, or both with one-shot structured light.

In the case of methods using structured light, [17, 22, 35, 37, 55] among others, during the calibration process, the machine may shut the projector off, capture an image, then project structured light, capture more images, and by decoding everything can geometrically relate images together. The ones captured with the projector off are used to calibrate the camera, and the rest, to calibrate the projector. Even though this procedure works, we need to secure physically the calibration target in place, because within a given set of images the machine assumes that the scene is static, often a requirement of structured light. We cannot simply hold a board in our hands in front of the projector-camera system.

To capture two images simultaneously, we can also exploit color channels, [1, 2, 22, 23, 55] among others, usually the red and blue ones. With properly designed complementary colors for the physical and projected patterns, we can recover them separately, *e.g.*: White and cyan squares show clearly in the red channel of a camera, but should hopefully not appear in the blue channel. While this obviously does not work for grayscale projectors, cameras, and printers, even with color ones, this approach presents two problems. First, depending on the properties of the color filters and inks, the blue channel of

Figure 1.2: Snapshot of our demo video showing the test user calibrating a casually installed projector-camera system

the projector might still interfere with the red channel of the camera, and vice versa. For example, results from Chen *et al.* [15] show a 4% overlap between the red channel of their camera and the blue channel of their projector, and about the same the other way around. Second, many cameras use only one sensor with an appropriate array of color filters, such that the resolution of the blue and red channel is half of the resolution of the whole sensor. Both of these issues may hurt accuracy, although current results are not conclusive, but more importantly requiring color complicates the execution of calibration.

As for color calibration, current methods [14, 15, 16, 30, 46] require either control over the shutter speed of the camera, more than one projector, or an elaborate 3D contraption, but we wanted a method that works without camera control and with only one projector and a planar surface.

For all the above reasons, we propose instead a user-friendly method to perform full geometric and color calibration of a projector-camera system. It is based on fiducial markers typically used for augmented reality applications. Fiala and Shu [19] proposed such a method for camera-only systems. We extend it to projector-camera systems plus color calibration, where half of the markers are physically printed, and half are projected using the projector. Each marker on its own carries information and can easily be identified. As we describe in Chapter 3, this allows the machine to prewarp markers that are projected, in a way that does not interfere with printed markers. Moreover, markers do not need color, and unlike structured light users can hold the calibration board in their hands. The user simply has to wave the calibration board, ideally at angles of approximately 45° with respect to the image plane as recommended by Zhang [56], and the machine automatically captures and saves about ten good images, with which it computes calibration parameters for both the camera and the projector. The machine also uses the markers to detect blank regions on the calibration board, which it takes as reference white to calibrate the colors. Color calibration is relatively simpler since we found that a strictly linear model to match colors between the camera and the projector works well.

Our results show that users could perform full geometric and color calibration in less than one minute, a use case that we consider user-friendly. Calibration results are also accurate, with less than one pixel of geometric error and, for color, a correlation coefficient R^2 of virtually one.

1.4 Direct Image Alignment

Once fully calibrated, image alignment becomes more robust. Nevertheless, even given calibration, it remains a challenging task. As a first step, we decided to make a few simplifying assumptions. Most importantly, the object must be planar and feature diffuse reflectance properties with no specular reflections. Still, thanks to the inherent robustness of direct alignment methods, we found that the algorithm can cope with large amounts of noise. To provide some results, we implemented in software our method for planar surfaces and ran experiments on both simulated and real images. At this time, although the program runs in real time at about thirty frames per second on commodity hardware, it still restricts the speed at which a user can move the object, but it achieves subpixel accuracy. Future research will revolve around optimization and generalization.

1.5 Interactive Augmentation

Once the system has aligned the images, we may wish to interact on the surface with the augmentation. Direct image alignment offers us more than simply the configuration under alignment, but also which parts of the image do not agree with the model, in other words, an occlusion map. The system can exploit these regions, and treat them as the result of objects users purposefully place in front of the surface, such as their hands. In this manner, we can offer them what basically amounts to interactive surfaces, ideally responding to gestures for interaction. As part of this work, although we have not implemented full gesture recognition yet, we show how one may accomplish hand tracking, locate its tip, and process the equivalent of mouse clicks, in real time.

In the following chapters, we first explain in more details the system model and its simplifying assumptions (Chapter 2), the required geometric and color calibration procedures (Chapter 3), including the detection of fiducial markers, the calibration patterns used, the prewarp of the projector pattern, and the real-time algorithm, and then finally the direct image alignment algorithm itself (Chapter 4), which minimizes an objective function that requires initialization when presented with a new planar surface. To simplify the explanations, we describe a system with only one camera and one projector, yet it can easily be extended to include any number of them. Following that, we present in Chapter 5 a way to integrate hand tracking within that framework and also give more details about how we optimized the bottleneck of the image alignment algorithm to achieve real-time execution speed,

an important consideration to make interaction with humans possible. Finally, in Appendix A, we present a guide to all the open-source software we developed and made freely available for download.

Chapter 2

System Model

A projector-camera system can be treated as a stereo pair, and multiple view geometry [26] still applies as is. However, unlike traditional stereo systems made from two or more similar camera devices, color information does not transfer as easily from a projector to a camera. We need a more sophisticated color model. In this section, before explaining the geometric and color models, to simplify them, we start by enumerating assumptions we found useful.

2.1 Simplifying Assumptions

We first assume that the projector and camera are fixed together, like a typical stereo camera, implying that the strong epipolar constraint pertains. The system may nonetheless move with respect to the scene or vice versa, the problem remains unchanged. Secondly, the color responses of both camera and projector must be linear with respect to the light intensity. We consider this to be reasonable for two reasons. On one hand, typical CCD and CMOS sensors respond linearly. On the other hand, most devices today support the sRGB color space standard (IEC [28]) that features a well defined color response curve approximating a transfer function with a gamma of 2.2, from which the intensities can be linearized. We expect any deviations to be engulfed in the inaccuracies introduced by the unknown light sources and reflectance properties of the surface material, which brings us to the third assumption. The scene consists of a planar object with a matte surface exhibiting uniform diffuse reflectance, where the radiance changes smoothly as the angle and distance vary. In particular, specularities are not modeled. Lastly, all light shining on the plane comes from point sources at infinity, such that no local shadows or highlights appear. Only global changes in illumination are observed, which we refer to as *ambient light*.

2.2 Geometric Model

To model geometrically both camera and projector devices, we use the familiar pinhole camera model [26]. Although originally designed for cameras, it also applies to projectors. It maps a (Cartesian) 3D point $\mathbf{x}_3$ of the surface to

$$\mathbf{x}_2 = \mathrm{K}_{3\times3} \left(\mathrm{R}_{3\times3}\mathbf{x}_3 + \mathbf{t}_3 \right), \tag{2.1}$$

a (homogeneous) 2D point on the image plane of the device, where K, the camera matrix, contains the internal projective parameters or intrinsics, and where R and $\mathbf{t}$, the external parameters or extrinsics, model the orientation and position in 3D space of the device. It follows that the projection onto the image planes of a camera placed at the origin and of a calibrated projector can be respectively written

$$\mathbf{x}_\mathrm{c} = \mathrm{K}_\mathrm{c} \left(\begin{array}{cc} \mathrm{I} & \mathbf{x}_3 + \mathbf{0} \end{array} \right), \tag{2.2}$$

$$\mathbf{x}_\mathrm{p} = \mathrm{K}_\mathrm{p} \left(\mathrm{R}_\mathrm{p}\mathbf{x}_3 + \mathbf{t}_\mathrm{p} \right). \tag{2.3}$$

To avoid confusion, we refer to the matrix K as the *camera matrix* even in the case of a projector. Further, even though equations dealing with homogeneous coordinates are only equal up to an arbitrary scaling factor, we use the equal sign for convenience and clarity.

2.3 Color Model

While the geometric model of a device stands independently on its own, for the color model, we decided instead for simplicity to model only the relationship between devices. Specifically, if a projector pixel shines the color $\mathbf{p}_\mathrm{p}$ on a surface point of *reflectance* $\mathbf{p}_\mathrm{s}$, we expect the corresponding camera pixel to observe the color

$$\hat{\mathbf{p}}_\mathrm{c} = \mathbf{p}_\mathrm{s} \times \left[\, g\mathrm{X}_{3\times3}\mathbf{p}_\mathrm{p} + \mathbf{a} \, \right] + \mathbf{b}, \tag{2.4}$$

where g is the *gain* of the projector light, which we assume varies smoothly and uniformly with the distance and angle to the surface; X is the *color mixing matrix* as defined by Chen *et al.* [15], also called the "projector-camera coupling matrix" by Caspi *et al.* [14]; $\mathbf{a}$, the ambient light; and $\mathbf{b}$, the noise bias of the camera. All vectors are three-vectors in the RGB color space, and their multiplication is done elementwise. We derived the model directly from the bidirectional reflectance distribution function (BRDF) of a flat matte surface, which Johnson and Fuchs [30] have shown to be valid in the case of projector-camera systems. Although the equation explicitly

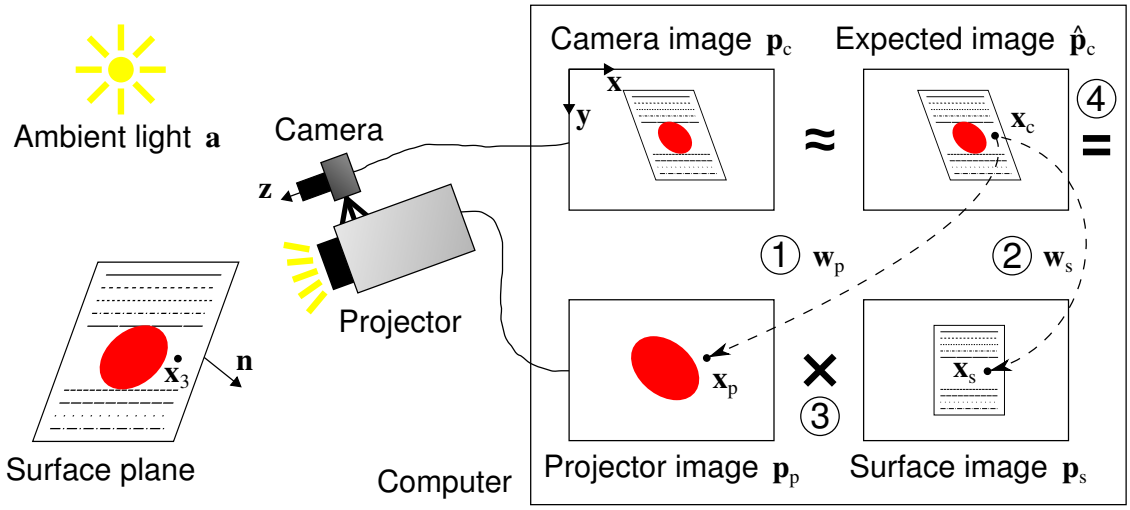


Figure 2.1: Sketch of the image formation model

models the relation between only one camera and one projector, as long as one identifies the reference device, the model can adapt to any number of them.

2.4 Image Formation Model

Using the geometric and color models, we can simulate how an image forms on the sensor of the camera. The explanation that follows is also summarized in Figure 2.1. Assuming the geometric parameters $\mathbf{n}$ of the plane are known and given a 3D point $\mathbf{x}_3$ on the surface plane such that $\mathbf{n}^T\mathbf{x}_3 + 1 = 0$, it follows that the 2D point $\mathbf{x}_p$ from Equation 2.3 can also be expressed as

$$\begin{aligned}
\mathbf{x}_p &= K_p \left(R_p\mathbf{x}_3 - \mathbf{t}_p\mathbf{n}^T\mathbf{x}_3 \right) \quad \text{since} \quad \mathbf{n}^T\mathbf{x}_3 = \text{-}1 \\
&= K_p \left(R_p - \mathbf{t}_p\mathbf{n}^T \right) \mathbf{x}_3 \\
&= K_p \left(R_p - \mathbf{t}_p\mathbf{n}^T \right) K_c^{-1}\, \mathbf{x}_c \qquad (2.5) \\
&= \qquad\quad H_{pc} \qquad\quad \mathbf{x}_c,
\end{aligned}$$

where the before last substitution comes from the fact that homogeneous vector $\mathbf{x}_c$ from Equation 2.2 can be arbitrarily scaled to fit in. The matrix H_{pc} embodies a homography, from which we define the *camera-to-projector warping function*, corresponding to step 1 in Figure 2.1:

$$\mathbf{w}_p(\mathbf{x}_c) \equiv H_{pc}\, \mathbf{x}_c. \qquad (2.6)$$

Similarly, one can map a camera image point $\mathbf{x}_c$ onto a point $\mathbf{x}_s$ of the image of the surface plane, assumed to have been initialized at a prior moment using the same camera. The camera motion R_s, $\mathbf{t}_s$ required to return to the prior orientation and position can be seen as a second camera (one can think of it as the "surface camera") with the same set of internal parameters, but with different external parameters, as follows:

$$\begin{aligned}
\mathbf{x}_s &= K_c \left(R_s \mathbf{x}_3 - \mathbf{t}_s \mathbf{n}^T \mathbf{x}_3 \right) \text{ since } \mathbf{n}^T \mathbf{x}_3 = \text{-1} \\
&= K_c \left(R_s - \mathbf{t}_s \mathbf{n}^T \right) \mathbf{x}_3 \\
&= K_c \left(R_s - \mathbf{t}_s \mathbf{n}^T \right) K_c^{-1} \mathbf{x}_c \\
&= \qquad H_{sc} \qquad \mathbf{x}_c,
\end{aligned} \tag{2.7}$$

which again becomes a homography H_{sc} from which we define the *camera-to-surface warping function*, corresponding to step 2 in Figure 2.1:

$$\mathbf{w}_s(\mathbf{x}_c) \equiv H_{sc}\ \mathbf{x}_c. \tag{2.8}$$

Finally, plugging these coordinates into the color model of Equation 2.4 by considering the pixel colors $\mathbf{p}_c$, $\mathbf{p}_s$, and $\mathbf{p}_p$ as functions over the images, we expect the pixel color at point $\mathbf{x}_c$ of the camera to be

$$\hat{\mathbf{p}}_c(\mathbf{x}_c) = \mathbf{p}_s(\mathbf{w}_s(\mathbf{x}_c)) \times \left[\ g \mathrm{X} \mathbf{p}_p(\mathbf{w}_p(\mathbf{x}_c)) + \mathbf{a}\ \right] + \mathbf{b}, \tag{2.9}$$

reflecting the final third and fourth steps in Figure 2.1.

Chapter 3

User-Friendly Calibration

Before we may use the models described in the previous chapter, a number of parameters are required for these models to function properly. To estimate these parameters, we proceed with calibration in a manner similar to current methods for both geometric parameters (K_c, K_p, R_p, and $\mathbf{t}_p$) [2, 56] and color parameters (X and $\mathbf{b}$) [14, 15], considering the black offset of the projector as part of the ambient light $\mathbf{a}$ and linearizing the color responses assuming they follow the sRGB standard (IEC [28]), which produces in our experience sufficiently linear responses, allowing us to calibrate fully a system from scratch in less than one minute. We first explain geometric calibration, since it can be carried out without color information, while the latter requires prior geometric knowledge.

3.1 Geometric Calibration

To calibrate geometrically the system, we use as is our previously proposed method [3]. It is based on fiducial markers typically used for augmented reality applications. Half of the markers are physically printed in black, and half are displayed in white using the projector, as shown in Figure 3.1. Each marker on its own carries information and can easily be identified. As described below, this allows the machine to prewarp projector markers in a way that does not interfere with printed markers. The user simply has to wave the calibration board in front of the projector and camera, and the machine automatically captures and saves about ten good images, with which it computes calibration parameters for both the camera and the projector via Zhang's method [56]. Also, markers do not need color, so one can calibrate the geometry before the colors.

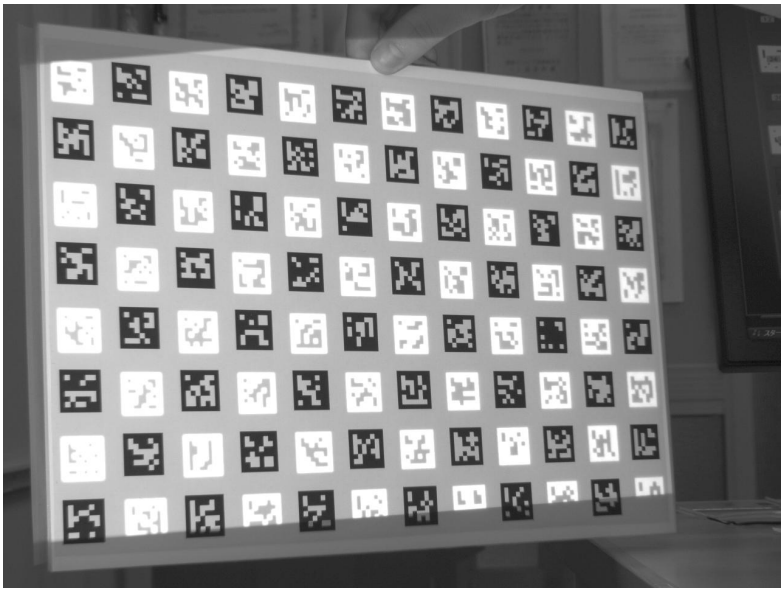
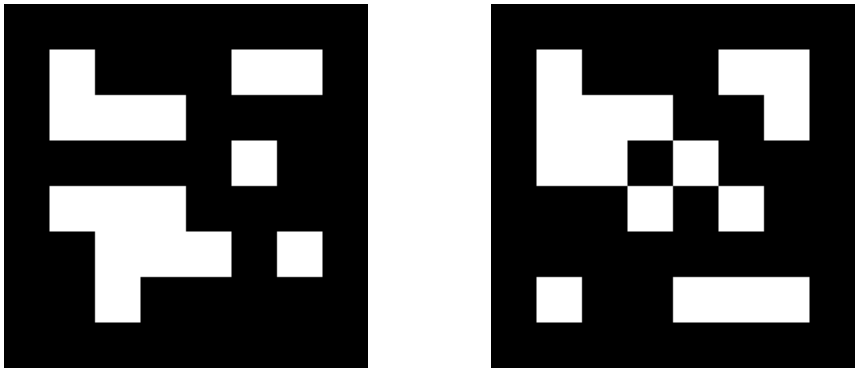


Figure 3.1: Sample camera image captured during geometric calibration, where the projector markers are in white, and the printed markers in black

3.1.1 Detection of Fiducial Markers

As fiducial markers, we opted for the BCH code provided by ARToolKitPlus [52]. Two examples with IDs of 0 and 1 are shown in Figure 3.2. There are 4096 such unique markers, arranged in squares of 8×8 black or white subsquares composing the black border and 36 interior bits, for only 12 bits of data, thus featuring strong error correction capabilities. Once printed or projected on a sheet of paper and imaged sufficiently large by a camera, ARToolKitPlus can detect these markers. Basically, the software first binarizes the image at an appropriate threshold, then analyzes contours to find shapes close to a quadrangle, estimates the four corners, and warps the shapes into squares, thus removing projective distortion from the camera. Lastly, by decoding the BCH code inside the squares, it recovers the IDs of the markers. In our case, even if a detected quadrangle is not a marker, since we use a small amount of markers and thanks to error correction, the decoding would most likely simply fail. Fiala [18] provides a more detailed analysis for markers used by ARTag, which are similar to the BCH code used by newer versions of ARToolKitPlus.

While ARToolKitPlus provides most of the desired functionality, we had to add two more features: adaptive thresholding and subpixel corner extraction. The former finds an optimal threshold for binarization even if the level of brightness varies across the image. The latter refines the location of the detected quadrangles, in the original grayscale image.



(a) ID = 0 (b) ID = 1

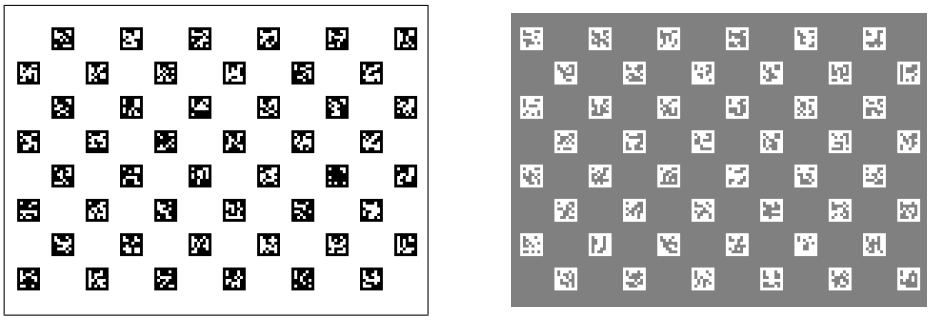
Figure 3.2: Two sample fiducial markers with IDs 0 and 1 made from the binary-coded BCH code of ARToolKitPlus

Adaptive thresholding adjusts the threshold depending on the local neighborhood of each pixel. We adopted Sauvola and Pietikäinen's method [44], which sets the threshold for each pixel (x, y) to

$$t(x,y) = m(x,y) \times \left[1 + k\left(\frac{s(x,y)}{128} - 1\right)\right] , \tag{3.1}$$

where k is a parameter close to zero, usually $[0.0, 0.5]$, and where $m(x, y)$ and $s(x, y)$ are the local mean and the local standard deviation, which are computed within a neighborhood of (x, y). In our case, we considered an adaptive neighborhood size, instead of a constant one. If the variance is very low, then the intuition is that the window of the neighborhood is too small. We need to make it larger to compute a good threshold. Inversely, if the variance is very high, then it is too large. We are no longer analyzing local statistics and the threshold would not be optimal. For each pixel, we thus need to adapt the window size dynamically. We chose to do a binary search to find the size where the standard deviation $s(x, y)$ drops below a certain threshold. Assuming a grayscale image with pixel values in the range $[0, 255]$, since the maximum standard deviation is a bit less than 128 and the minimum is 0, we found 64 to be an appropriate threshold. We implemented an efficient algorithm based on integral images that runs in $O(N^2 \log N)$ time for an $N \times N$ image.

For subpixel corner extraction, we were able to use directly the code provided in OpenCV [12]. The algorithm works by first considering a small window (*e.g.*, 11×11) around the pixel of interest in the original grayscale



(a) Printed pattern (b) Projector pattern

Figure 3.3: Sample calibration patterns composed of matrices of markers

image. It then assumes that the following holds for all subpixels $\mathbf{x}$ in the window:

$$(\mathbf{c} - \mathbf{x}) \cdot \nabla_{\mathbf{x}} = 0 \ , \tag{3.2}$$

where $\mathbf{c}$ is the exact subpixel corner location and $\nabla_{\mathbf{x}}$ is the gradient at $\mathbf{x}$. This is true if either $\nabla_{\mathbf{x}}$ equals $\mathbf{0}$ or is orthogonal to $(\mathbf{c} - \mathbf{x})$, which is valid for a corner (but also for a straight edge, so it cannot be used for corner detection). The algorithm finds the minimum of this function, which is not normally zero because of noise and nonlinear distortions.

In this manner, we obtain for each marker an ID and the subpixel coordinates of its four corners.

3.1.2 Calibration Patterns

Even though it is possible to calibrate using only four corners from only one marker, to obtain best calibration accuracy, we should in fact attempt to cover the largest area possible with a great amount of corners [49]. As calibration pattern, we decided to use a matrix of markers, similar to the one that Fiala and Shu [19] used, but where half of the markers are printed and the other half come from the projector, as shown in Figure 3.3. As explained in more details below, the projector markers are in inverted color, since it helps to detect them when projected on top of printed markers. Given the minimum focus distance of our projector, we found that 20×20 mm^2 markers spaced every 30 mm on a B4 size board works well. This gives a total of 12×8 markers, half of which are printed and the rest from the projector.

If one uses a board larger than the field of views, the system would obviously never be able to detect some markers, but as long as it can detect enough of them, there is no problem. In fact, since corners may in that case fully cover the image planes, it should improve accuracy.

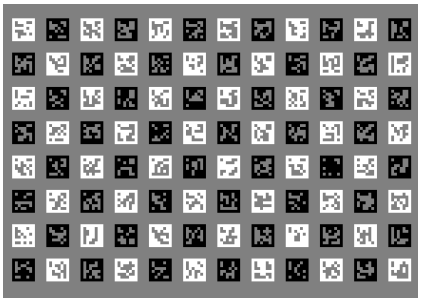


Figure 3.4: The ideal image that would be generated by projecting the pattern from Figure 3.3(b) over Figure 3.3(a) under perfect alignment

In any case, images captured by the camera contain a mix of both markers. For a given image to be considered appropriate for calibration purposes, the patterns need to be aligned well enough such that the markers from the projector do not interfere with the printed ones, and vice versa, as shown in Figure 3.4. In the following section, we explain how to accomplish this by prewarping the projector pattern.

3.1.3 Prewarp of the Projector Pattern

For camera-only systems, we can easily accommodate the number of corners shown in Figure 3.4, but this is challenging for projector-camera systems, because we need to image both physical corners and projector corners. Without proper considerations, the pattern from the projector would interfere with the printed one. In this section, we show that, to prevent interference, prewarping the projector markers does not change the calibration problem.

The prewarp transformation that we derive is a homography. Although the patterns contain many markers, we actually only need one to estimate a homography with maximum likelihood under the assumption of Gaussian noise [26]. Still, at least a few markers have to be decodable to compute a meaningful estimate of the error. In cases of large errors, the patterns are not well aligned, and we should not use the estimate for calibration purposes, but we can take it to update the projector prewarp and hope to get a better estimate from the next captured image. For this reason, we still need to find that first homography between the camera and the board.

When a few printed markers are decoded, that homography H_{sc} satisfies

$$\mathbf{x}_{\mathrm{s}} \approx \mathrm{H}_{\mathrm{sc}}\mathbf{x}_{\mathrm{c}} \ , \tag{3.3}$$

where $\mathbf{x}_{\mathrm{s}}$ is a point on the board surface and $\mathbf{x}_{\mathrm{c}}$, the corresponding point on the camera image plane. H_{sc} is also the initialization information required

to calibrate a camera using Zhang's method [56]. For the projector, we have the similar relationship

$$\mathbf{y}_\mathrm{s} \approx \mathrm{H}_\mathrm{sc}\mathrm{H}_\mathrm{cp}\mathbf{y}_\mathrm{p} \ , \tag{3.4}$$

where $\mathbf{y}_\mathrm{s}$ is a point on the surface; $\mathbf{y}_\mathrm{p}$, the corresponding point on the projector image plane; H_sc, the homography as defined above; and H_cp, the homography transforming points from the image plane of the projector to the one of the camera, which we find using detected projector markers in the camera image. With the combined homography $\mathrm{H}_\mathrm{sc}\mathrm{H}_\mathrm{cp}$, we can calibrate a projector in the same way as a camera. This is how all current projector calibration methods work. However, we can equivalently write

$$\mathbf{y}'_\mathrm{s} \approx \mathrm{H}_\mathrm{sc}\mathrm{H}_\mathrm{cp}\mathbf{y}'_\mathrm{p} \ , \tag{3.5}$$

where $\mathbf{y}'_\mathrm{p} = \mathrm{H}_\mathrm{p}\mathbf{y}_\mathrm{p}$ and H_p is the homography that prewarps our projector image. In other words, we can choose H_p, such that a transformed point $\mathbf{y}'_\mathrm{p}$ on the image plane of the projector appears where we want it on the board, $\mathbf{y}'_\mathrm{s}$, selected as to not interfere with the printed pattern. Because $\mathrm{H}_\mathrm{sc}\mathrm{H}_\mathrm{cp}$ remains untouched, we can say that the calibration method is equivalent.

3.1.4 Real-Time Algorithm and Issues

The sections above fully cover the theoretical aspects, but the machine executes in practice an interactive algorithm in real time. One of its iteration is depicted in Figure 3.5. First, it applies the prewarp of Section 3.1.3 to the projector pattern and sends it to the projector. We assume the user holds at all times the flat board with printed markers in front of the projector-camera system. In this case, projector markers appear alongside the printed ones. The system can then continue and attempt to detect all printed and projector markers, one by one as explained in Section 3.1.1. If it considers that the error on the detected corners is low enough (details below) it uses them to update the prewarp homography. Further, if the patterns are aligned well enough (details below) it saves their marker corners. Finally, it loops and expects the user to move the board, unless enough corners have been saved, at which point they are used for calibration.

Although a conceptually simple algorithm, a few thorny technical issues remain. First, if we want the results of the subpixel detector to mean anything, we must make sure the prewarped projector image contains as little aliasing as possible. Next, when projector and printed markers overlap, it is not usually possible to detect them. However, using inverted colors makes them more easily detectable, even when they interfere with each other. Third, even though we can derive a useful update for the prewarp from a rough

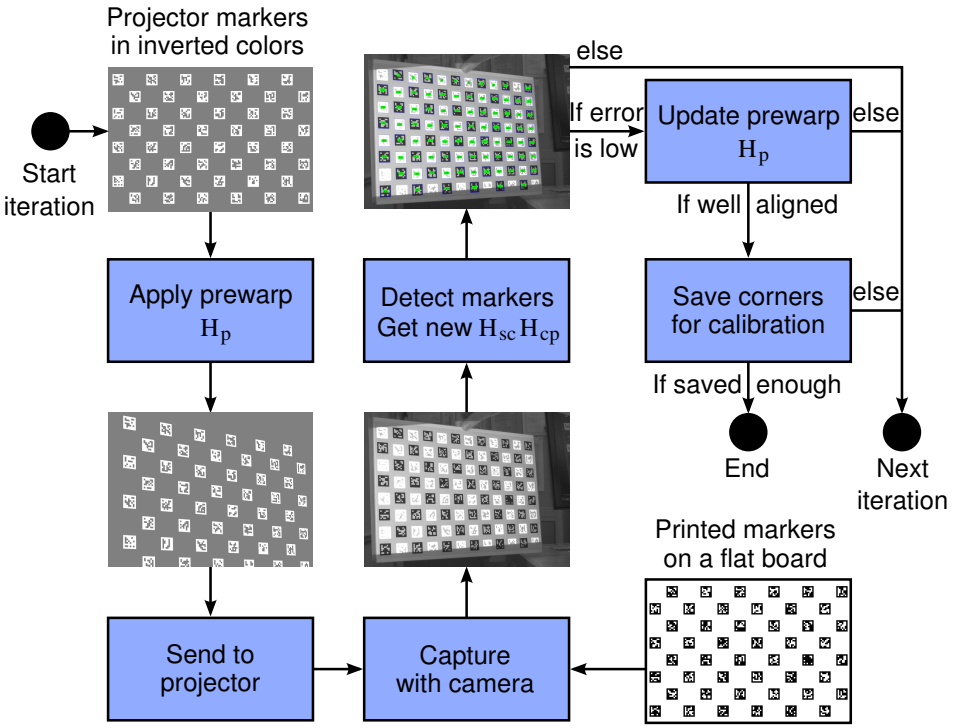


Figure 3.5: Flowchart illustrating one iteration of the interactive real-time algorithm for geometric calibration

homography estimation, outliers do not help. Finally, the machine must somehow judge when both patterns are aligned well enough and automatically select images for calibration. In the following subsections we explain in further details and provide practical solutions.

Antialiased Projector Images Typical projector calibration methods do not warp the projector image and do not have to worry about aliasing. In our case, arbitrary warps produce subpixel motion, and aliasing issues must be managed. As antialiasing measure, we opted for simple 4× supersampling [20], which gives good results and executes fast enough in software. Basically, we first create an image buffer that has four times the resolution of the projector. For example, a 4096×3072 image for a typical 1024×768 projector. Next, we draw the markers in the high-resolution buffer using well known drawing and filling algorithms [20] that work on a per-pixel basis. Since we make no attempt at drawing at the subpixel level, this high resolution image contains a lot of aliasing artifacts. However, when smoothed with simple averaging and decimated by a factor of four, the resulting image is

subpixel sharp and free from problematic aliasing artifacts. GPUs (graphics processing units) can also usually perform this operation quickly and efficiently via OpenGL or Direct3D, but the proprietary nature of the actual algorithms they use does not amend itself well to reproducible results. In either case, the system must then send the resulting image to the projector.

Detection of Overlapping Markers Afterward, if the user holds the calibration board in front of the projector and camera, printed and projector markers usually overlap up to some degree. Because of the limited dynamic range of a typical camera, if we use black markers over a white background only, both black regions from the projector and regions printed in black appear in the same shade of black. Any overlapping markers do not appear as quadrangles anymore, and detection fails. To remedy, we observed that using projector markers with inverted colors (*i.e.*, white markers over a black background) overlapping markers are more robustly detected. This can be explained by the fact that with markers designed this way, the camera automatically images as gray the empty regions of the board, while black printed regions appear black, and white projected regions appear white, naturally providing orthogonal intensity ranges for both types of markers. For this reason, we use inverted colors for projector markers. In addition, the machine can easily adjust the brightness of the markers to match the ambient light and fall within the dynamic range of the camera or alternatively, in dark environments, use the projector as an adjustable source of light, by setting the minimum intensity of pixels.

Rough Homography Estimation Even if overlapping markers are detected, the accuracy of their corners might obviously suffer, and consequently any homography we might estimate from them. In extreme conditions, markers may also be incorrectly detected. To compensate, we found it works well to reject homographies with large errors (*e.g.*, with an RMSE, root mean square error, greater than 10 pixels, or higher for strong nonlinear distortions). One marker, which has only four corners, always fits a homography perfectly, so this implies that the minimum number of detected markers must be two. While an RMSE of 10 pixels is still unacceptable for calibration, the hope is that updating the prewarp with this rough estimate moves the markers to a better position in the next captured image.

Automatic Image Selection Eventually, the machine should be able to detect a lot of the projector and printed markers (*e.g.*, over 50% for each). Also, if the corners of the calibration patterns have not moved much from

the last captured image (*e.g.*, less than 5 pixels on average, depending on the small expected motions from the user or oscillations when updating and applying the prewrap), then the system assumes that the patterns are aligned well enough, their motion blur negligible, and saves the detected corners for later calibration. Nevertheless, if the user does not move the board, it would keep saving corners, but similar images are not useful for calibration. For this reason, we added another criterion. The system only selects an image if, since the last saved one, the corners of the calibration patterns have moved a lot (*e.g.*, more than 50 pixels on average). When enough images have been selected (*e.g.*, 10 images), then all the saved corners go to the final stage: camera and projector calibration.

3.1.5 Calibration Algorithm

Once a sufficient amount of accurate corners have been accumulated, the calibration algorithm can process them directly. Using only the corners from printed markers $\mathbf{x}_s$ and their detected corners $\mathbf{x}_c$, OpenCV [12] can calibrate the camera with no difficulties using Zhang's method [56]. For the projector, on the other hand, we must first remove the linear (projective) and nonlinear (radial, etc.) distortions of the camera from the projector corners. Note that removing projective distortion is equivalent to applying the homography H_{sc} of Equation 3.5 to the imaged corners. Using such undistorted corner points $\mathbf{y}'_s$ on the board and corresponding $\mathbf{y}'_p$ on the image plane of the projector, we can apply the same method to calibrate the projector. Figure 3.6 illustrates the dataflow involved.

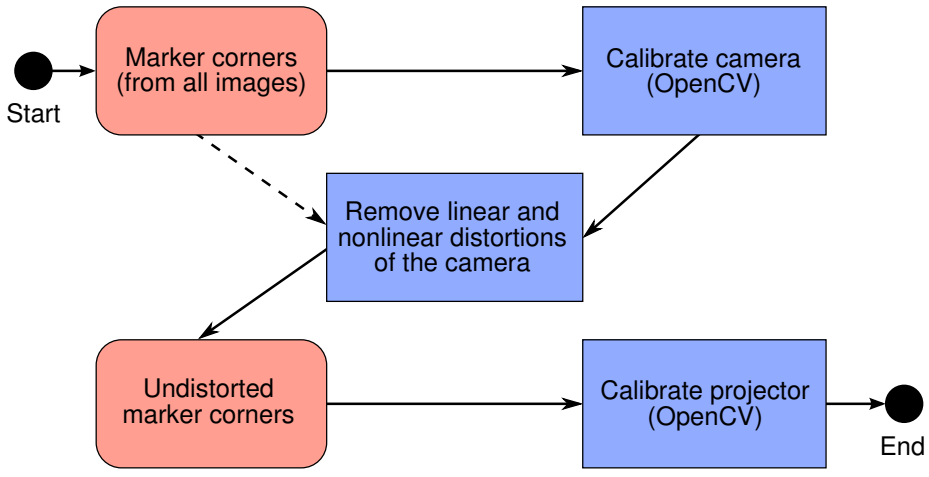


Figure 3.6: Dataflow diagram for calibrating the camera and the projector

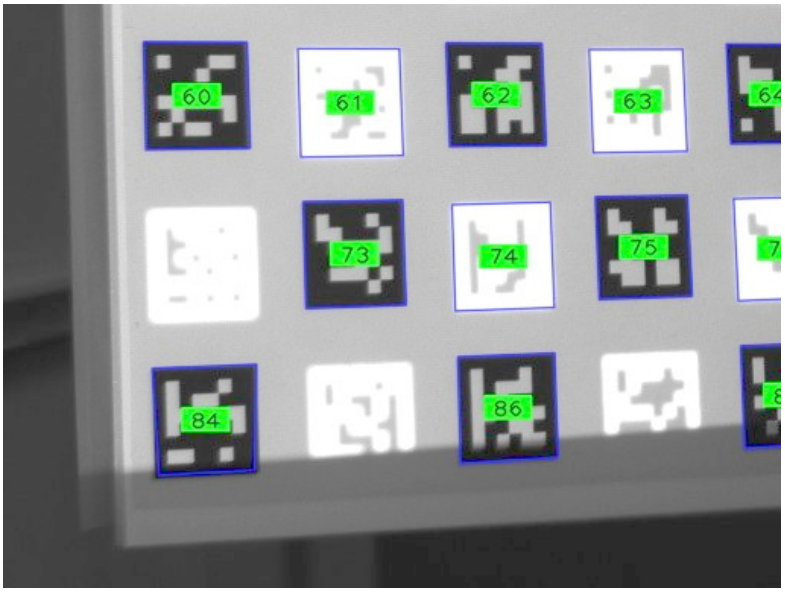


Figure 3.7: Lower left area of the detected markers from Figure 3.5: We can observe severe defocusing of projector markers

Although we have been writing about corners all along, we can also calibrate using the *center* of each marker. A center is defined as the physical point present in the middle, at the intersection of the diagonals. When nonlinear distortions are small, as is usually the case of projectors, Fiala and Shu [19] showed that we can obtain better results using centers than corners. This happens because corners that are not perfectly in focus are not accurately extracted. This is especially relevant in the case of projectors, since they usually have a narrow depth of field. To illustrate the problem, Figure 3.7 shows the lower left area of the detected markers in Figure 3.5: The edges of the defocused projector markers make them appear larger than they really are. Using what amounts to the average of the four corners lowers this bias.

3.2 Color Calibration

Once the geometric parameters of both the camera and projector have been estimated, the color calibration uses them to understand which portion of the calibration board can be covered by the projector. We then define that portion as a surface of white reference reflectance $\mathbf{p}_s = (1, 1, 1)$, information required to calibrate the colors. For simplicity, we reuse the same marked board (Figure 3.8). Further, the printed markers allow the machine to extract blank spots automatically, as shown in Figure 3.9. One may notice a

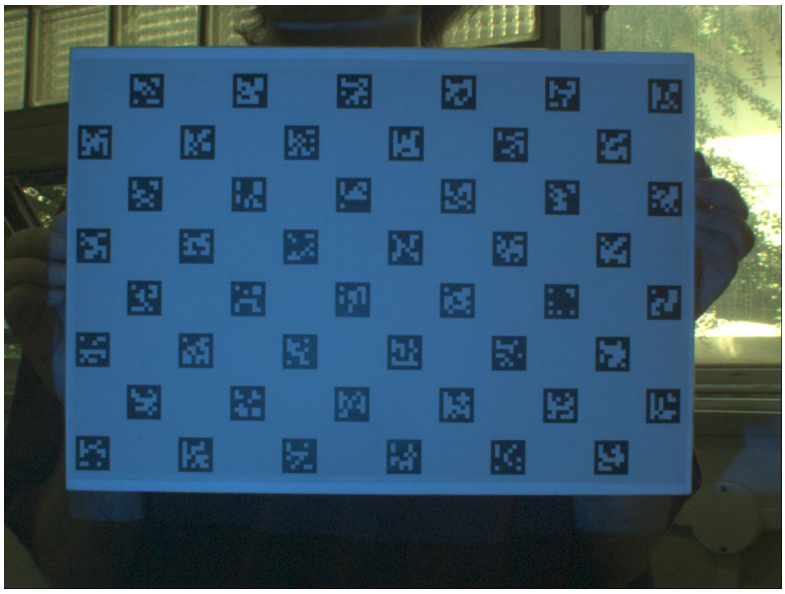


Figure 3.8: Sample camera image captured during color calibration when the projector displays the color blue at maximum intensity

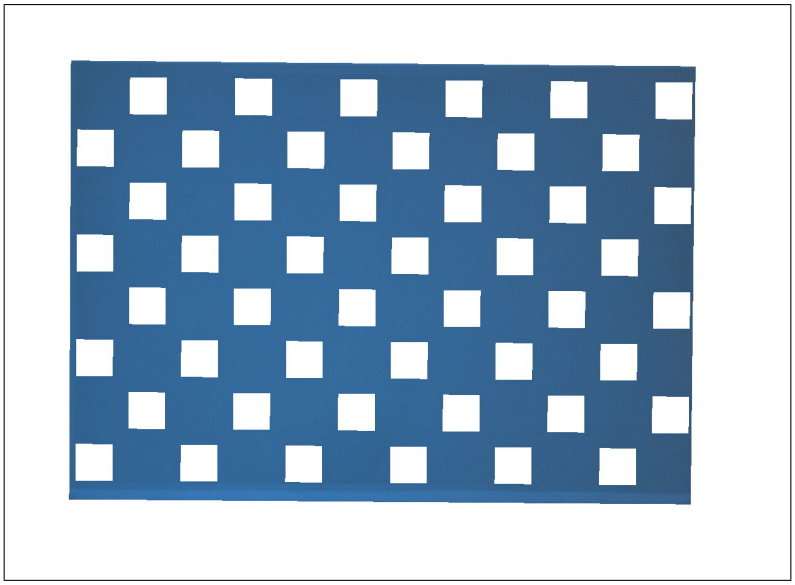


Figure 3.9: Blank area automatically extracted from Figure 3.8, whose average is used during color calibration

vignetting effect, since the points in the middle are in reality closest to the projector. During calibration, if the board remains more or less at the same position, this reduction of brightness can be ignored as the algorithm only cares about the average.

The procedure itself goes as follows. First, the user must disable all onboard processing of the camera (no automatic brightness, sharpness, white balance, hue, saturation, gamma, gain, etc.), and has to adjust manually the aperture and set the electronic shutter to a short enough exposure to avoid saturation at the highest projector intensity. Secondly, to find the camera noise bias $\mathbf{b}$ of Equation 2.4, one must blind the sensor and take the intensity values of the resulting image. We found it to be zero for all pixels of our camera, and in general we think it may safely be ignored as well. Next, by placing the calibration board in front of the projector and the camera, the system can detect its presence and start displaying and capturing an array of colors for calibration. For each color, it initially finds the pose of the calibration board via the detected markers, and assumes a gain $g = 1$. The pixel intensities are then averaged over the blank area of the board (Figure 3.9). Finally, the color mixing matrix X and the ambient light $\mathbf{a}$ remain the only two unknown parameters of Equation 2.4. They can be estimated by linearly regressing a sufficient number of samples. We found it enough to take a total of $4 \times 4 \times 4 = 64$ samples at constant intervals from the RGB color space of the projector as shown in Figure 3.10.

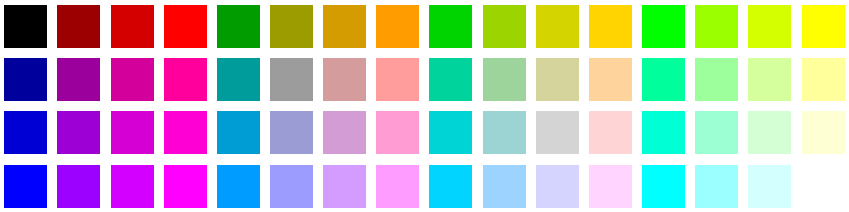


Figure 3.10: Samples from the RGB color space of the projector used during color calibration

Chapter 4

Direct Image Alignment

Based on the simplifying assumptions and the models for projector-camera systems listed and illustrated previously above in Chapter 2, we designed an iterative method to align directly arbitrary pairs of images, *i.e.*, not only specially crafted markers as used in Chapter 3 for calibration. In this chapter, we explain the objective function and the algorithm to minimize it, which can hopefully converge to the correct alignment under normal circumstances. Following this, we provide details as to how one may initialize the algorithm for a new surface plane. However, we assume at first that the reflectance $\mathbf{p}_s$ and the initial parameters $\mathbf{n}$ of the surface plane found during initialization are known.

4.1 Objective Function

To align best the images, we seek an optimal set of parameters. Intuitively, these parameters should minimally model the relative motion between the surface plane and the projector-camera system, since all other parameters are determined either during calibration or initialization. Looking back at Equations 2.6 and 2.8, these unknown parameters relating to motion are R_s and $\mathbf{t}_s$, a typical egomotion problem, which represents six degrees of freedom. Four others come from the gain g and the ambient light $\mathbf{a}$, which we assumed may change during motion, for a total of ten degrees of freedom and consequently ten parameters to optimize.

We can thus formulate our goal into an objective function to minimize, which basically consists of the norm of the residual between the real observed camera image $\mathbf{p}_\mathrm{c}$ and the expected one as defined by Equation 2.9, *i.e.*:

$$f(\boldsymbol{\theta}, g, \mathbf{a}) = \sum_{i=1}^{l} ||\mathbf{r}_i(\boldsymbol{\theta}, g, \mathbf{a})||^2, \text{ where} \tag{4.1}$$

$$\begin{aligned}\mathbf{r}_i(\boldsymbol{\theta}, g, \mathbf{a}) = &\mathbf{p}_\mathrm{c}(\mathbf{x}_{\mathrm{c},i}) - \mathbf{p}_\mathrm{s}(\mathbf{w}_\mathrm{s}(\mathbf{x}_{\mathrm{c},i}; \boldsymbol{\theta}))\times \\ &[\ g\mathrm{X}\mathbf{p}_\mathrm{p}(\mathbf{w}_\mathrm{p}(\mathbf{x}_{\mathrm{c},i}; \boldsymbol{\theta})) + \mathbf{a}\ ] - \mathbf{b},\end{aligned} \tag{4.2}$$

for the l pixels inside the ROI (region of interest) of the camera image with coordinates $\mathbf{x}_{\mathrm{c},i}$, using bilinear interpolation to resample, and where $\boldsymbol{\theta}$ denotes the geometric parameters R_s and $\mathbf{t}_\mathrm{s}$. Its minimization amounts to the maximum likelihood estimate (MLE) under the assumption of Gaussian noise. We chose the two-norm to simplify the implementation of the traditional Gauss-Newton optimization algorithm favored for image alignment problems [8], but more robust norms could also be used.

4.2 Analysis of Local Minima

To minimize the objective function of Equation 4.2, while one could optimize the 3D motion parameters directly, this parameterization lacks robustness. First, the inherent ambiguity between 3D rotation and translation may cause incorrect convergence [7]. Second, while we may argue that to solve this issue we could place the origin somewhere in the middle of the planar surface, numerous local minima remain around the global minimum of the objective function. As an example of this, consider the sketches of Figure 4.1. In this case, the halfway configuration associated with 3D parameters usually has an error larger than the initial one, rendering it a local minimum. This does not happen with the 2D parameterization, making it more robust under local minimization. As a consequence, we no longer recommend the hard constraint applied via a Lagrange multiplier proposed in one of our published papers [4].

For all these reasons, when minimizing the objective function, our new algorithm warps images using 2D objects exclusively. However, doing so, $\mathbf{w}_\mathrm{s}(\mathbf{x}_\mathrm{c})$ loses its 3D relation with $\mathbf{w}_\mathrm{p}(\mathbf{x}_\mathrm{c})$, and we need to model them separately. Naturally, we implemented the first warp as the homography H_sc and the second as a traditional stereo warp as expressed by Equation 2.5. Although the latter uses plane parameters, a 3D entity, the epipolar constraint limits motion to 1D lines, so it produces in reality well behaved 2D warps, as confirmed by Baker *et al.* [7]. Furthermore, to obtain even better convergence

properties based on their other findings, we parameterize the homography using four points, and the plane parameters using three disparities, all of which measured in pixel units. For the homography parameterization, we exploited directly the direct linear transformation (DLT) algorithm [26], which takes four pairs of points as input and outputs a homography matrix. To convert back and forth between three disparities and the plane parameters, we derived another algorithm that works in a similar fashion.

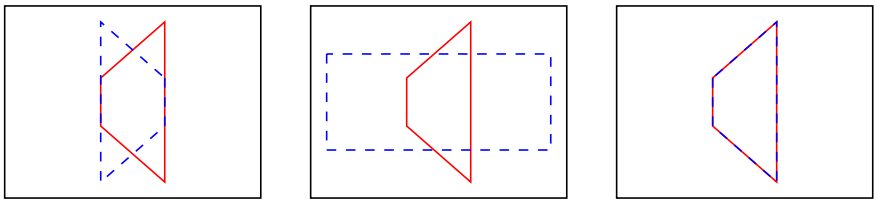
In particular, we are interested in recovering the plane parameters $\mathbf{n}$ from three pairs of corresponding points $\mathbf{x}_c$ and $\mathbf{x}_p$. Rearranging Equation 2.5, and using the cross product and skew-symmetric matrix notation to obtain as usual a true equality, we find that

$$\mathbf{x'}_p \times (\mathrm{R}_p - \mathbf{t}_p \mathbf{n}^T) \mathbf{x'}_c = \mathbf{0}, \tag{4.3}$$

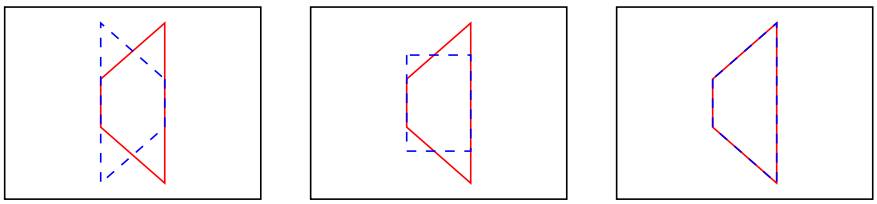
$$\mathbf{x'}_p \times \mathbf{t}_p \mathbf{n}^T \mathbf{x'}_c = \mathbf{x'}_p \times \mathrm{R}_p \mathbf{x'}_c, \tag{4.4}$$

$$[\mathbf{x'}_p]_\times \mathbf{t}_p \mathbf{x'}_c^T \mathbf{n} = [\mathbf{x'}_p]_\times \mathrm{R}_p \mathbf{x'}_c. \tag{4.5}$$

where $\mathbf{x'}_p = \mathrm{K}_p^{-1} \mathbf{x}_p$ and $\mathbf{x'}_c = \mathrm{K}_c^{-1} \mathbf{x}_c$. We arbitrarily choose the second row from Equation 4.5, which provides one constraint based on the x-coordinates. (Although this does not correspond to the exact definition of



(a) initial (b) halfway (3D) (c) target



(d) initial (e) halfway (2D) (f) target

Figure 4.1: Sample configurations (dashed blue line) an optimizer may encounter, with a plane rotated to the right as initial alignment and a plane rotated to the left as target alignment (continuous red line), using as parameterizations for the objective function either 3D rotation and translation (top row) or a 2D homography (bottom row)

the stereo disparity, one may easily project onto the x-axis a given disparity value.) Stacking three of these rows for three different pairs of points results in a linear system of the form $\mathrm{A}\mathbf{x} = \mathbf{b}$ with, in general, a single unique solution. Intuitively, this works because three points determine a unique plane. The three reference points in the source image $\mathbf{x}_\mathrm{c}$ provide the (x, y) coordinates, while their z-coordinates come from their depths, which vary directly proportionally to their disparities with $\mathbf{x}_\mathrm{p}$.

Even though the stereo warp works well in general, it may be stumped in cases such as the one illustrated in Figure 4.1, which cause large disparities, or when the projector image lacks in texture. To prevent those cases from causing trouble, after each iteration of the minimizer, our algorithm tries to reset the plane parameters to their initial values $\mathbf{n}$ transformed by the closest 3D rotation and translation to the homography H_sc as documented by Sturm [47].

4.3 The Subspace Error

Although 2D warps have less local minima, they are bound to produce physically impossible configurations such as the one of Figure 4.1(e), for which there are in reality only six degrees of freedom as 3D rotation and translation, so we devised a distance function to measure this *subspace error* and let the minimizer use the available 3D information as well for additional robustness. As explained in the previous section, these physical impossibilities also mean that a given 2D homography (eight parameters) may have no corresponding stereo warp (three parameters), so we need to optimize in fact a total of eleven 2D parameters. In the following equations, to differentiate the 2D geometric parameters from the 3D ones, we will denote them by the vectors $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$ respectively, the latter being equivalent to the $\boldsymbol{\theta}$ first defined under Equation 4.2. The former contains coordinates displaced from their reference points $\mathbf{x}^0_{\mathrm{s},1..m}$ for the surface ($m = 4$ in this case) and $\mathbf{x}^0_{\mathrm{p},1..n}$ for the projector image ($n = 3$ in this case). As reference points for the surface, the four corners of a quadrilateral ROI provide the perfect match, while for the projector ones, we decided to take the two top corners and the bottom middle point. For a 1024×768 projector image, those would be $(0, 0)$, $(1024, 0)$, and $(512, 768)$.

Generally, we want both sets of parameters to produce transformations close to one another, so we define a distance function such that $d(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3) = 0$ if and only if $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$ produce the same transformation, while a nonzero value indicates some proportional amount of subspace error. Since we would like this error to be as close as possible to zero, while still allowing our

algorithm to converge in situations such as the one of Figure 4.1, we simply append it to our objective function

$$f_d(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3, g, \mathbf{a}) = f(\boldsymbol{\theta}_2, g, \mathbf{a}) + k\, d(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3), \tag{4.6}$$

along with an adjustable coefficient k that we need to find experimentally. We chose to measure the error in pixel units by warping (inversely) the reference points, of the point-based parameterization described above, with parameters $\boldsymbol{\theta}_2$, then $\boldsymbol{\theta}_3$, and by comparing the resulting coordinates. Expanding this equation a bit more under this definition, we discover that it reduces to a sum of squared distances that fits well within the least squares framework:

$$\begin{aligned} f_d(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3, g, \mathbf{a}) &= \sum_{i=1}^{l} ||\mathbf{r}_i(\boldsymbol{\theta}_2, g, \mathbf{a})||^2 + \\ & k \left(\sum_{i=1}^{m} ||d_{\mathrm{s},i}\ (\boldsymbol{\theta}_2, \boldsymbol{\theta}_3)||^2 + \sum_{i=1}^{n} ||d_{\mathrm{p},i}(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3)||^2 \right), \end{aligned} \tag{4.7}$$

$$d_{\mathrm{s},i}\,(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3) = d(\mathbf{w}_\mathrm{s}^{-1}(\mathbf{x}_{\mathrm{s},i}^0; \boldsymbol{\theta}_2),\ \mathbf{w}_\mathrm{s}^{-1}(\mathbf{x}_{\mathrm{s},i}^0; \boldsymbol{\theta}_3)), \tag{4.8}$$

$$d_{\mathrm{p},i}(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3) = d(\mathbf{w}_\mathrm{p}^{-1}(\mathbf{x}_{\mathrm{p},i}^0; \boldsymbol{\theta}_2), \mathbf{w}_\mathrm{p}^{-1}(\mathbf{x}_{\mathrm{p},i}^0; \boldsymbol{\theta}_3)), \tag{4.9}$$

for the l pixels inside the ROI of the camera image.

Xiao *et al.* [54] proposed a similar method attempting to impose a hard constraint using a "suitably large value for k", but we realized that this may not be actually desirable. In fact, compared to a constant coefficient, we found that an adaptive one could work more efficiently. To balance the weight of the subspace error, it makes sense to base the coefficient on the residual error of the pixel values, normalized by the dimensionality of the subspace term $(m + n)$, *i.e.*:

$$k = \alpha^2\ \frac{f(\boldsymbol{\theta}_2, g, \mathbf{a})}{m + n}. \tag{4.10}$$

The rationale behind this equation derives from our previously explained observations of Figure 4.1. For configurations where the residual error is low, but where the subspace error may be high, such as illustrated in Figure 4.1(e), this coefficient effectively lessens its importance, as desired. On the other hand, in the case of high residual error, we would like our algorithm to try to make use of the additional 3D information found in the subspace term. To accommodate large displacements, such as the ones tested during our simulation experiments, values of $\alpha \approx 0.1$ work well, while values as high as 100 may be used to encourage faster convergence in cases when one does not anticipate large displacements of otherwise noisy data, such as with real images streaming from a video camera.

4.4 Minimization

Taking all the above into consideration, to minimize iteratively the augmented objective function of Equation 4.7 over its 21 parameters (17 geometric, parameterizing the rotation using the usual axis-angle three-vector representation, plus the gain and the ambient light), our Gauss-Newton update equation looks like this:

$$(\mathrm{J}_{\mathbf{r}}^{\mathrm{T}}\mathrm{J}_{\mathbf{r}} + k\mathrm{J}_{\mathbf{d}}^{\mathrm{T}}\mathrm{J}_{\mathbf{d}})\Delta_{\theta_2,\theta_3,g,\mathbf{a}} = -(\mathrm{J}_{\mathbf{r}}^{\mathrm{T}}\mathbf{r} + k\mathrm{J}_{\mathbf{d}}^{\mathrm{T}}\mathbf{d}). \tag{4.11}$$

where $\mathbf{r}$ is the vector of residual functions $\mathbf{r}_i$, and $\mathbf{d}$ the one for the subspace error functions $d_{\mathrm{s},i}$ and $d_{\mathrm{p},i}$, while $\mathrm{J}_{\mathbf{r}}$ and $\mathrm{J}_{\mathbf{d}}$ are their Jacobian matrices.

For simplicity and contrarily to Lucas and Kanade [8], we decided to estimate the Jacobian $\mathrm{J}_{\mathbf{r}}$ numerically using finite difference with a step $\delta = 0.1$. In other words, its i^{th} column

$$\mathbf{j}_{\mathbf{r},i}(\Theta) = \frac{\mathbf{r}(\Theta + \delta\mathbf{e}_i) - \mathbf{r}(\Theta)}{\delta}, \tag{4.12}$$

and similarly for $\mathrm{J}_{\mathbf{d}}$ [39], where $\Theta = (\theta_2, \theta_3, g, \mathbf{a})$ and $\mathbf{e}_i$ is a unit vector whose i^{th} element equals one. This avoids the need to differentiate analytically the objective function or to precompute image gradients with respect to the axes, which has to be done numerically in any case using an operator such as the Sobel filter. Rather, since we parameterize all our geometric warps in pixel units, we may skip these operations, and evaluating the Jacobian amounts to computing image gradients directly with respect to the parameters, in exactly the right way. Computationally, our approach requires warping eleven images per iteration versus only four gradient images for the more traditional approach, but ours does not need to precompute those and does not require the (cache) memory to hold them either. Additionally, we need to consider other important characteristics such as accuracy and numerical stability. Although we feel confident our approach provides good results, we plan to investigate the matter in the future to better understand the pros and cons.

For additional robustness and performance, we implemented the traditional coarse-to-fine multiresolution estimation, where each level higher in the resolution pyramid is first smoothed with a 5×5 Gaussian filter and then subsampled by a factor of two. We achieved best results using from five to seven levels, depending on size of the ROI, the largest anticipated displacement, and the quality of the textures. For an initial image size of 1024×768 pixels, this means resolutions down to as low as 16×12 pixels.

Even with a coarse-to-fine approach, computing the Jacobian takes at least an order of magnitude more time than the residual image. To help our

algorithm make progress faster, we included on top of everything a simple backtracking line search strategy, which splits iteratively the update length in half until the error of the objective function $f_d(\boldsymbol{\theta}_2, \boldsymbol{\theta}_3, g, \mathbf{a})$ diminishes or until it meets a stopping criterion. To not stop, the update has to be large enough, but not too large to start with either, *i.e.*:

$$\Delta_{\min} < ||\boldsymbol{\Delta}_{\boldsymbol{\theta}_2,\boldsymbol{\theta}_3,g,\mathbf{a}}|| < \Delta_{\max}, \tag{4.13}$$

where $\Delta_{\min} \approx 0.1$ for good accuracy such as used in the simulation experiments or, for faster convergence in the case of, for example, real-time applications, $\Delta_{\min} \approx 10$. We also define an upper bound $\Delta_{\max} \approx 300$, determined according to the image sizes mainly, to detect obvious instances of divergence.

4.5 Initialization

The above minimization algorithm works only if the reflectance map $\mathbf{p}_\mathrm{s}$ is known. The initial surface plane parameters $\mathbf{n}$ should also be acquired somehow for better robustness. Equation 2.4 shows that solving for the two unknowns $\mathbf{p}_\mathrm{s}$ and $\mathbf{a}$ requires capturing at least two images from the camera. Although there are undoubtedly many possible ways to perform initialization, we designed an approach that can cope with small movements, allowing users to hold the surface plane in their hands. This approach works in three phases, by first estimating the ambient light $\mathbf{a}$, then the reflectance map $\mathbf{p}_\mathrm{s}$ and finally the initial geometric plane parameters $\mathbf{n}$.

Concretely, the procedure goes as follows. It starts by the projection and the capture as fast as possible of three consecutive shots, to obtain images with the smallest interframe motion possible, from which the user may select a suitable ROI. For the first image, the system sets the projector color $\mathbf{p}_\mathrm{p}$ to zero, and for the second, to maximum intensity $\mathbf{p}_\mathrm{p}^{\max}$. (More details about the third image below.) Taking the first two images, it can then estimate the ambient light by defining the reference gain $g = 1$ and isolating $\mathbf{a}$ from Equation 2.4:

$$\mathbf{a} = \mathrm{X}\mathbf{p}_\mathrm{p}^{\max}\frac{\mathbf{p}_\mathrm{c}^1 - \mathbf{b}}{\mathbf{p}_\mathrm{c}^2 - \mathbf{p}_\mathrm{c}^1}, \tag{4.14}$$

where $\mathbf{p}_\mathrm{c}^1$ is the color from the first camera image, and $\mathbf{p}_\mathrm{c}^2$, from the second image. To reduce the undesirable effect of motion, we use at this phase severely smoothed versions of the first two images, by convolving with a 51×51 Gaussian kernel for example, which is reasonable since we assumed that ambient light varies only globally. This assumption also allows the

system to ignore points where $\mathbf{p}_\mathrm{c}^2 - \mathbf{p}_\mathrm{c}^1$ is too close to zero and to evaluate only the average.

The second phase consists of using the second image, this time the original crisp version, along with the estimated ambient light to recover a crisp reflectance map

$$\mathbf{p}_\mathrm{s} = \frac{\mathbf{p}_\mathrm{c}^2 - \mathbf{b}}{\mathrm{X}\mathbf{p}_\mathrm{p}^\mathrm{max} + \mathbf{a}}. \tag{4.15}$$

For the third phase, the idea is to actually run the minimization algorithm described in the previous subsection, optimizing for $\boldsymbol{\theta}_2$, g, and $\mathbf{a}$, with $\alpha = 0$, and extracting the initial $\mathbf{n}$ from $\boldsymbol{\theta}_2$. As initial values, one can reasonably assume $g = 1$, reuse the ambient light computed in the first phase, and compose $\boldsymbol{\theta}_2$ with the coordinates of the ROI plus the extreme left, right, and middle x-coordinates, in other words matching the projector reference points $\mathbf{x}_{\mathrm{p},1..n}^0$. For a 1280×960 camera image, those would be 0, 1280, and 640. For the algorithm to converge properly though, some sort of texture needs to be displayed on the surface plane. We chose a fractal image, the exact one shown in Figure 6.2(d), but orthogonally aligned to the epipolar lines. Because this texture contains the same pattern at all scales, running the minimization algorithm at multiple resolutions should allow for easy and accurate convergence over a wide range of displacements. More formally, assuming epipolar lines aligned with the x-axis, the intensities assigned to the projector image of coordinates $x \in [x_1, x_2]$ equal

$$\mathbf{p}_\mathrm{p}(x, y) = \begin{cases} \mathbf{1}\, p(x, x_1, x_m, 0, 0, \quad 1) & \text{if } \; x_1 \le x < x_m \\ \mathbf{1}\, p(x, x_m, x_2, 0, 0, -1) & \text{if } \; x_m \le x \le x_2 \end{cases}, \tag{4.16}$$

whose amplitude is scaled appropriately, and where

$$p(x, x_1, x_2, y_1, y_2, n) = \\ \begin{cases} p(x, x_1, x_m, y_1, y_m, \quad n') & \text{if } \; x_1 \le x < x_m \\ y_m & \text{if } \; x = x_m \\ p(x, x_m, x_2, y_m, y_2, -n') & \text{if } \; x_m < x \le x_2 \end{cases}, \tag{4.17}$$

$$x_m = \frac{x_1 + x_2}{2}, \quad y_m = \frac{y_1 + y_2}{2} + n, \quad n' = \frac{n}{\sqrt{2}}. \tag{4.18}$$

Although the barlike effect may give the impression of an image used for structured light, the approach is totally different as our pattern contains no code and tolerates well any surface texture.

Chapter 5

Interactive Augmentation

Taking as base the direct image alignment algorithm for projector-camera systems described in the previous chapter, we contrived a way to control the equivalent of a mouse pointer over the planar surface using hands and fingers (or other pointy objects). In the following sections, we first explain the algorithm as summarized in Figure 5.2 that we implemented to locate the tip of a hand, which requires outlier detection, hysteresis thresholding, and contour analysis of the binarized image delineating the region of the object, before providing more details about important implementation optimizations

Figure 5.1: Snapshot of our demo video showing the user operating with his hand a GUI projected onto a flat surface

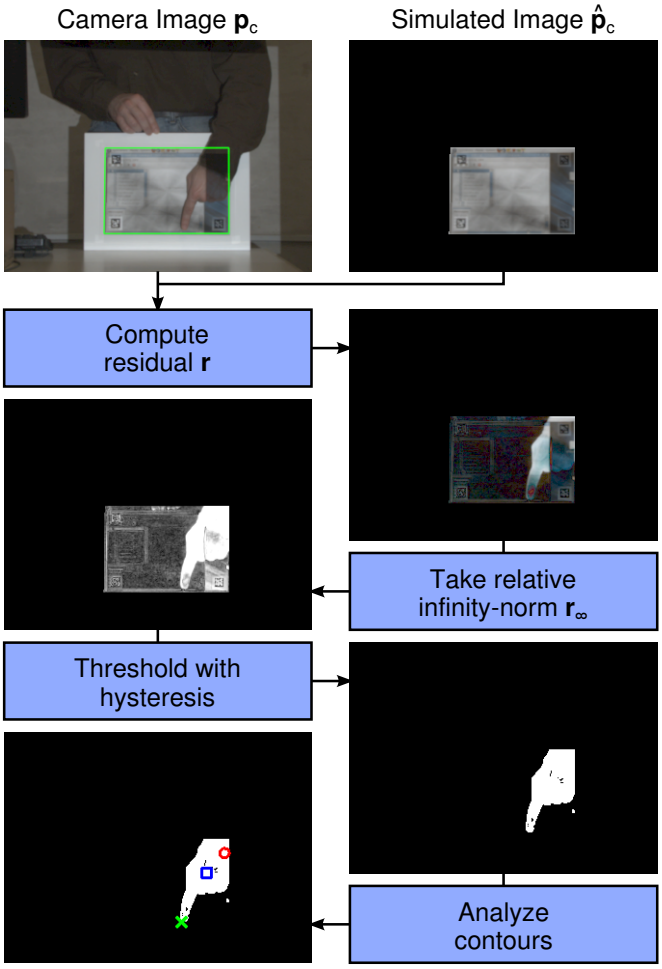
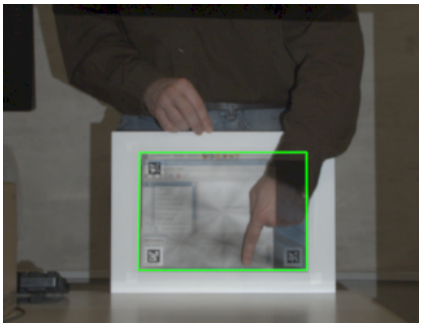


Figure 5.2: Flowchart of the overall algorithm, where the green rectangle indicates the aligned ROI and where the absolute value of the residual and relative infinity-norm are shown with their brightness increased for clarity

necessary to make real-time interaction possible, notably zero thresholds and adaptations to parallelize our algorithm for both CPUs and GPUs that capitalize on the locality of references, and before finally discussing potential applications, which include naturally and spatially varying augmentation, augmenting content on demand when requested by users as well as allowing them to interact with a completely normal GUI (graphical user interface), as the one in Figure 5.1. While many existing sophisticated algorithms can perform gesture recognition and what not from the binary image, our goal was to demonstrate the feasibility of the approach, and we based our contour analysis on the one from Tele-Graffiti [50] for simplicity.



(a) Camera image $\mathbf{p}_c$

(b) Expected image $\hat{\mathbf{p}}_c$

Figure 5.3: Sample camera image with a green rectangle indicating the alignment with the expected image

5.1 Hand Tracking

The alignment algorithm described in the previous chapter simulates or estimates, in a nutshell, the pixel values $\hat{\mathbf{p}}_c$ closest to the real ones $\mathbf{p}_c$ from the ROI (region of interest) of the camera image, examples of which are given in Figure 5.3. More precisely, it minimizes the *residual error*

$$\|\mathbf{r}(\boldsymbol{\theta})\|_2 = \|\mathbf{p}_c - \hat{\mathbf{p}}_c(\boldsymbol{\theta})\|_2, \tag{5.1}$$

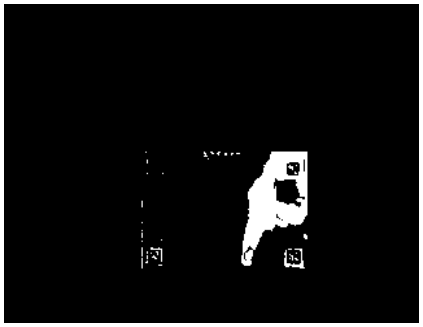
where the vector $\boldsymbol{\theta}$ contains all the geometric and color parameters of the model as detailed in Chapter 2. Under alignment, the residual vector usually contains pixel values close to zero, but when the user as in Figure 5.3(a) presents in front of the surface a hand, since the model cannot explain it, the errors of those pixels become large, as those inside Figure 5.4(a), constituting outlying values. We took advantage of this fact to produce an algorithm that can locate the position of the hand, or of another object, and its tip.

First, we had to modify the alignment algorithm to detect the outlying pixels and prevent their interference with alignment results. Second, although the detected outliers suffice in preventing alignment failures, we found that they were not accurate enough to extract a clean contour of the hand and had to include post-processing that exploits the relative infinity-norm and hysteresis thresholding during binarization. This then results in contours clean enough for contour analysis to locate the tip.

5.1.1 Outlier Detection

One popular and effective way to deal with outliers consists of the least trimmed squares method [29]. It does not directly detect outliers, but assumes

(a) Residual image $\mathbf{r}$



(b) Thresholded residual

Figure 5.4: Residual image from the images of Figure 5.3 (after taking the absolute value of the intensities and multiplying by two for better brightness), and a version thresholded optimally at 0.04

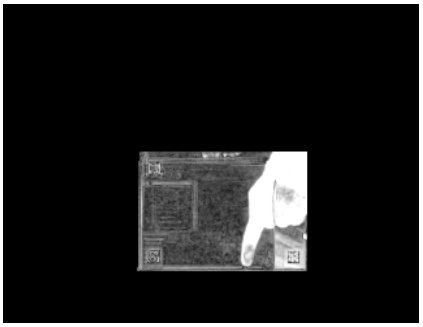
some maximum amount of them (e.g.: 50% of the data) and runs minimization only on the values with smallest errors. The data needs to be sorted, but good approximations can be calculated efficiently with histograms. However, this method does not actually help us detect the outliers themselves. It can only use a fixed amount of the data regardless of whether it contains no outliers or they amount to 90% of it, in which case it would usually fail. Instead, in our case, since correctly modeled values tend to produce errors close to zero, we found that we did not need such a method and could efficiently, with minimal overhead, detect outliers sufficiently accurately. By considering the vector $\mathbf{r}(\boldsymbol{\theta})$ as a function over pixel coordinates $\mathbf{x}$ returning RGB pixel values, we classify a given pixel to be outlying if

$$\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 > t_\mathrm{o}. \tag{5.2}$$

Given pixel values in the range $[0, 1]$, we found values of $t_\mathrm{o} \approx 0.1$ to be effective in preventing outliers from influencing the alignment results without affecting to a great extent the convergence properties of robustness and speed of the minimizer, as evidenced by our demo video. For reference, Figure 5.4(b) maps the outliers if t_o were 0.04.

5.1.2 Binarization

After correct convergence, simply thresholding the outliers detected by the method above does not produce a binarized image that shows the outlines of a hand clearly enough in challenging cases such as the one of Figure 5.4.



(a) Relative infinity-norm $\mathbf{r}_\infty$

(b) Thresholded version

Figure 5.5: Relative infinity-norm image from the images of Figure 5.3 (with intensities multiplied by two and saturated for better brightness) and a version thresholded optimally at 0.3

Computing instead the pixel values

$$\mathbf{r}_\infty(\mathbf{x}) = \left\| \frac{\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})}{\max(\mathbf{p}_\mathrm{c}(\mathbf{x}), \hat{\mathbf{p}}_\mathrm{c}(\mathbf{x}, \boldsymbol{\theta}))} \right\|_\infty , \tag{5.3}$$

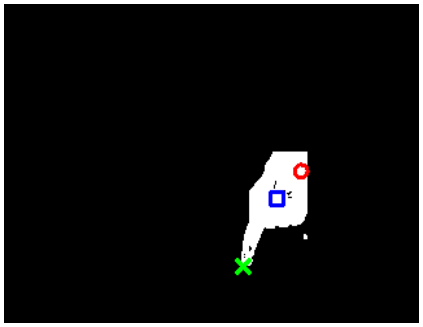
basically taking the relative infinity-norm of each RGB pixel value, produces an image of better quality, as shown in Figure 5.5(a). As the next step, even though we could use simple thresholding on that image, as shown in Figure 5.5(b), results can be improved further. With this intention, we implemented a variant of Canny's hysteresis thresholding [13], but instead of maneuvering along edges, it expands over a 4-connected neighborhood of pixels. Assuming pixel values in the range $[0, 1]$, the algorithm goes as follows:

for all $\mathbf{x} \in$ ROI **do** {forward pass}
  switch
    case $\mathbf{r}_\infty(\mathbf{x}) \geq t_\mathrm{h}$ **:** $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 1$
    case $\mathbf{r}_\infty(\mathbf{x}) < t_\mathrm{l}$ **:** $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 0$
    case $\exists\, \mathbf{x}_\mathrm{n}$ next to $\mathbf{x} \mid \mathbf{r}_\infty(\mathbf{x}_\mathrm{n}) = 1$ **:** $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 1$
    default: $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 0.5$

for all $\mathbf{x} \in$ ROI $\mid \mathbf{r}_\infty(\mathbf{x}) = 0.5$ **do** {backward pass}
  switch
    case $\exists\, \mathbf{x}_\mathrm{n}$ next to $\mathbf{x} \mid \mathbf{r}_\infty(\mathbf{x}_\mathrm{n}) = 1$ **:** $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 1$
    default: $\mathbf{r}_\infty(\mathbf{x}) \leftarrow 0$

where t_h and t_l are the higher and lower thresholds, respectively. We found that values of approximately 0.5 and 0.25 yield good results, as exemplified by the image of Figure 5.6(a).

(a) Hysteresis thresholded $\mathbf{r}_\infty$



(b) Contour analysis results

Figure 5.6: Figure 5.5(a) thresholded with hysteresis at $t_\mathrm{h} = 0.5$ and $t_\mathrm{l} = 0.25$, and results from contour analysis to locate the tip

This procedure obviously takes more processing power than the simpler outlier detection presented in the previous subsection, but it does not need to run more than once per frame.

5.1.3 Contour Analysis

Within the now thresholded and binarized image $\mathbf{r}_\infty$, after applying optional morphological opening and closing operations to clean up small pixel-size noise, we may then find the contours inside the image, as implemented by OpenCV [12]. To locate the tip of the object, we adapted an algorithm developed by Takao *et al.* [50], as follows:

1. Select the contour with the largest <area> × <the number of pixels on the edge of the ROI>,
2. Locate the centroid of the whole contour as well as the centroid of the pixels on the edge (marked in Figure 5.6(b) with a blue square and a red circle, respectively),
3. Draw a line passing through the two centroids, and
4. Find the pixel on the contour whose projection on the line is farthest away from the edge centroid, and designate it as the tip (marked in Figure 5.6(b) with a green cross).

The system can then use the location of the tip as mouse coordinates and can also produce mouse clicks when the user keeps the tip at the same location for a long enough period of time, something sensible like half a second for a click and one second for a double-click.

5.2 Real-Time Implementation

While tracking the hand requires some amount of processing time, it pales in comparison to what the alignment algorithm requires, so this section focuses on the latter. In particular, computing the expected pixels $\hat{\mathbf{p}}_c(\boldsymbol{\theta})$ takes most of the time, as one may judge from the results in Chapter 6, in part because the warping function requires expensive interpolation, but also because it needs to be called for each iteration at least as many times as the number of parameters, while convergence usually requires between five and ten iterations. Since it is crucial for user interface technology to run at interactive speed, we explain in this section important software optimization schemes we devised to achieve real-time execution performance.

5.2.1 Zero Thresholds

One trick we found to accelerate execution was skipping over pixels whose residual is below a certain threshold, a *zero threshold*. Intuitively, such residuals and associated differentials contain mostly noise, so we may as well save processing time. More precisely, we introduced this criterion into the differentiation to accelerate the evaluation of the Jacobian $\mathrm{J}_{\mathbf{r}}$ itself. Since we assume relatively small displacements, the residuals $\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})$ are often close to zero. The Jacobian intrinsically models small displacements, so the intuition is that, for a given pixel $\mathbf{x}$, if $\mathbf{r}(\mathbf{x}, \boldsymbol{\theta}) \approx \mathbf{0} \Rightarrow \mathrm{J}_{\mathbf{r}}(\mathbf{x}, \boldsymbol{\theta}) \approx \mathbf{0}$ for all parameters with high probability. Therefore, when the algorithm finds the magnitude of $\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})$ to be below a certain t_z, we let it skip the computation of the derivatives for that pixel and simply set them to zeros. However, the value of this threshold is influenced by the amount of noise, which in turn reduces at each level higher in the resolution pyramid. Fortunately, the residual error gives us for free a good indication of the noise level as the RMSE (root mean square error)

$$t_z = t\|\mathbf{r}(\boldsymbol{\theta})\|_2, \tag{5.4}$$

where values for t in the range $[0, 1]$ give usable results, with the largest one frequently skipping more than half of the pixels for an acceleration factor of about two. Moreover, since heavy processing occurs mostly in the lower pyramid levels, we found that setting bigger thresholds for those gives more optimal results. However, when we expect the data to contain outliers, as assumed in this chapter, we can no longer trust the RMSE. As with outlier detection, we found instead that a constant threshold performs adequately, and we may skip pixels for which

$$\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 < t_z, \tag{5.5}$$

where t_z is a zero threshold, usually in the range [0, 0.05], frequently skipping over half of the pixels. The optimized algorithm below uses this threshold, but we first discuss parallelization.

5.2.2 Parallelization

The heart of the alignment algorithm uses the Gauss-Newton method to minimize the residual of Equation 5.1 by iteratively solving the linear equation

$$\mathrm{J}_{\mathbf{r}}^{\mathrm{T}} \mathrm{J}_{\mathbf{r}} \, \Delta_{\boldsymbol{\theta}} = -\mathrm{J}_{\mathbf{r}}^{\mathrm{T}} \mathbf{r}, \tag{5.6}$$

where $\mathrm{J}_{\mathbf{r}}$ is the $\dim(\mathbf{r}) \times \dim(\boldsymbol{\theta})$ Jacobian matrix of the residual and $\dim(\boldsymbol{\theta}) = 15$ for one planar surface, excluding the subspace error term. The dimension of the residual vector depends on the size of the ROI (region of interest) as well as on the resolution of the input images. Images of lower resolutions can also cope better with large displacements, so typical algorithms adjust dynamically the resolution to enhance both performance and robustness. In any case, successful alignment requires a minimum of perhaps 1000 pixels and a lot more to achieve subpixel accuracy. Consequently, the machine spends most of its time computing the Jacobian matrix.

Fortunately, each row of the Jacobian and the residual represent data for exactly one pixel, which does not depend on any other pixel. This means we can easily parallelize the computation by splitting the ROI into as many regions as processing units available, taking into consideration data locality. For example, with a CPU of four processing units and an ROI containing 400 lines, the first unit would process the first 100 lines, the second, the next 100 lines, and so forth.

5.2.3 Optimized Algorithm for CPU

In addition to being well suited to parallel processing, inspection of Equation 5.6 reveals that we do not actually need the Jacobian matrix to solve it, but merely the $\dim(\boldsymbol{\theta}) \times \dim(\boldsymbol{\theta})$ Hessian matrix $\mathrm{J}_{\mathbf{r}}^{\mathrm{T}} \mathrm{J}_{\mathbf{r}}$ and the $\dim(\boldsymbol{\theta})$ gradient vector $\mathrm{J}_{\mathbf{r}}^{\mathrm{T}} \mathbf{r}$. The following procedure computes their values from the Jacobian evaluated using the finite difference method that needs a call to the warping function $\dim(\boldsymbol{\theta})$ times for each pixel. It follows well established guidelines to optimize software, namely it is easily parallelizable and capitalizes on the locality of references.

$g_i(j) \leftarrow 0 \quad \forall \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$
$h_i(j, k) \leftarrow 0 \quad \forall \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1), \quad k \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$

for all $\mathbf{x} \in \mathrm{ROI}_i$ **do**
  if $\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 < t_\mathrm{z}$ or $\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 > t_\mathrm{o}$ **then**
    continue
  for all $j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$ **do**
    $\mathbf{j}(\mathbf{x}, j) \leftarrow \hat{\mathbf{p}}_\mathrm{c}(\mathbf{x}, \boldsymbol{\theta}) - \hat{\mathbf{p}}_\mathrm{c}(\mathbf{x}, \boldsymbol{\theta} + \delta \mathbf{e}_j)$
    $g_i(j) \leftarrow g_i(j) + \mathbf{j}(\mathbf{x}, j) \cdot \mathbf{r}(\mathbf{x}, \boldsymbol{\theta})$
  for all $j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1), \quad k \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$ **do**
    $h_i(j, k) \leftarrow h_i(j, k) + \mathbf{j}(\mathbf{x}, j) \cdot \mathbf{j}(\mathbf{x}, k)$

$g(j) \leftarrow \sum g_i(j) \quad \forall \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$
$h(j, k) \leftarrow \sum h_i(j, k) \quad \forall \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1), \quad k \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$

where g, h, and $\mathbf{j}$ are functions over the gradient vector and the Hessian and Jacobian matrices respectively, and where ROI_i, g_i, and h_i indicate nonoverlapping regions of memory for processing unit i.

5.2.4 Optimized Algorithm for GPU

The number of threads executing on a CPU equal the number of processing units, and even today's high-end server machine do not have more than 64 of them, with vector instructions of length no larger than eight. Consequently, as long as the images we need to process have more than $64 \times 8 = 512$ pixels, the algorithm above can scale on CPUs with no problem. However, in contrast to CPUs, each processing unit of today's GPUs require many threads to obtain best performance, at least between 64 and 192 for the current generation. For this reason, to achieve an optimal number of threads in flight, we had to adapt the algorithm above to process in smaller chunks. The very small cache generally available on GPUs represents another reason we need to chunk down the processing. Typical CPUs now come with shared caches of about ten megs or more, while GPUs are still limited to an equivalent of 64 kB of cache known as *shared* or *local* memory. The following procedure in pseudocode describes an algorithm that efficiently fulfills these constraints, and which we can implement as a *device kernel* in the CUDA [40] or OpenCL [31] programming environment. Each thread receives a column x to process, looping and accumulating values over all rows y of the ROI. The local thread ID i, which is also related to the global column ID x, and thread ID j serve to accumulate in parallel and in registers the elements of the gradient and Hessian matrices, one thread for each element.

$w =$ width of the ROI
$h =$ height of the ROI
$x =$ the column of the ROI to process $\in (0, 1, ..., w-1)$
$i \;=$ the thread ID $\in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$ for the first dimension
$j \;=$ the thread ID $\in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$ for the second dimension

$g_x(j) \leftarrow 0$
$h_x(i, j) \leftarrow 0$
for all $y \in (0, 1, ..., h-1)$ **do**
  $\mathbf{x} \leftarrow (x, y)$
  if $\mathbf{x} \notin$ ROI **then**
    continue
  if $\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 < t_\mathrm{z}$ or $\|\mathbf{r}(\mathbf{x}, \boldsymbol{\theta})\|_2 > t_\mathrm{o}$ **then**
    continue
  $\mathbf{j}(\mathbf{x}, j) \leftarrow \hat{\mathbf{p}}_\mathrm{c}(\mathbf{x}, \boldsymbol{\theta}) - \hat{\mathbf{p}}_\mathrm{c}(\mathbf{x}, \boldsymbol{\theta} + \delta\mathbf{e}_j)$
  $g_x(j) \leftarrow g_x(j) + \mathbf{j}(\mathbf{x}, j) \cdot \mathbf{r}(\mathbf{x}, \boldsymbol{\theta})$
  synchronize all threads on $\mathbf{j}$
  for all $k \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$ **do**
    $\mathbf{x} \leftarrow (x - i + k, y)$
    $h_x(i, j) \leftarrow h_x(i, j) + \mathbf{j}(\mathbf{x}, i) \cdot \mathbf{j}(\mathbf{x}, j)$
  synchronize all threads on $\mathbf{j}$

$g(j) \leftarrow \sum g_x(j) \quad \forall \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$
$h(i, j) \leftarrow \sum h_x(i, j) \quad \forall \quad i \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1), \quad j \in (0, 1, ..., \dim(\boldsymbol{\theta}) - 1)$

where g, h, and $\mathbf{j}$ can again be considered as functions over the gradient vector and the Hessian and Jacobian matrices respectively, and where g_x, and h_x correspond to independent memory registers for column x, one for each thread.

We provide below an analysis of various performance parameters and a discussion of some other important implementation details:

- Our alignment algorithm performs minimization over 15 parameters, so the number of threads per processing unit $\dim(\boldsymbol{\theta}) \times \dim(\boldsymbol{\theta}) = 225$, greater than the recommended minimum of 64 and lower than the maximum of between 256 and 1024, according to the hardware model.

- The corresponding 225 elements $\mathbf{j}$ of the Jacobian matrices should be saved for each row of the image in *shared* or *local* memory. Given RGB

values in floating-point representation we need $225 \times 3 \times 4 = 2700$ bytes of this cache, for $2700/225 = 12$ bytes per thread, figures that are lower than the total maximum amount of 48 kB and also lower than the recommended maximum of 32 bytes per thread for full occupancy.

- The register usage count also fits within 32, achieving good occupancy.
- Although the algorithm above loops over all rows accumulating values in registers as it goes, one may find that in the case of small images limiting the amount of rows over which each thread loops may improve performance. Basically, we have to reach a balance with the overhead of the final reductions.
- Prior to entering the loop, the homography and color mixing matrices needed to transform the pixels into $\hat{\mathbf{p}}_c$ should also be copying into the cache, which has lower latency and greater bandwidth than both *global* and *constant* memory spaces.
- Reading pixels through the image sampler and its texture cache attains higher performance than accessing them directly from global memory. However, hardware linear interpolation of graphics card does not provide sufficient accuracy. We must therefore use the nearest-neighbor sampler and perform linear interpolation manually, as with a CPU.
- For best performance in practice, the final reductions of g_x and h_x should be undertaken in two steps, first locally on each processing unit, and then globally, across processing units, using a second kernel that has no other purpose than to reduce these values.

5.3 Potential Applications

While developing the algorithms described above, we had in mind three main classes of applications: Augmentation of the physical content that changes naturally with either the position or orientation of the surface, or in contrast that varies purposefully on user request only, and interaction with a GUI, as depicted in the images of Figures 5.1 and 6.8 taken from our demo video featuring these few possible interactions with this system. The full sequence can be obtained from our Web site at
http://www.ok.ctrl.titech.ac.jp/~saudet/procamtracker_2012-02-14.mp4 .

5.3.1 Spatially Varying Augmentation

The position and orientation of surfaces offer an affordance that the system can exploit to perform desirable actions seemingly invisibly. For example, users may want to have a translation displayed onto printed documents that are not in their native languages, matched by their locations at the table. We can also take into account previous position and orientation for dynamic augmentation. Our demo video features a virtual ball, as per Figures 6.8(b) and (c), constrained within the physical boundaries of the surface, but that is otherwise free to fall and bounce around, thanks to fast enough frame updates. Other interesting applications include an interface to visualize slices from volumetric data [25] or to offer privacy states [33], where some private content may only be displayed inside a limited range of orientations.

5.3.2 Augmenting Content On Demand

Another useful kind of application would provide augmentation for the content of physical surfaces on user demand. In our demo video, we present a scenario where a user wishing to read a page of an article first lays it down on the table. The projector in this initial state acts as a lamp, Figure 6.8(e). Nevertheless, the machine has in memory a demo video, which the user may choose to start watching at any moment by pointing a finger at the (in effect physical) hyperlink, and watching the content, Figure 6.8(f).

5.3.3 Interacting with a GUI

Going even further, users may operate a full-fledged GUI, such as the one of Figures 6.8(i) and (j), taken from the demo video where the user starts the playback of a video file, clicks on the seek bar to fast-forward and rewind around, and finally closes the window. Even though some mouse operations like double-clicking and right-clicking may not map directly to anything sensible, we have the advantage that recognition of hand gestures could easily match and even surpass the expressive power of the traditional mouse or of touchscreens.

Chapter 6

Experimental Results

The description of our methods is now complete, and we present in this chapter some results. We first developed applications in Java, integrating with OpenCV and OpenCL as appropriate, to implement the procedures and algorithms introduced in this thesis. We then performed various experiments to show the validity of the methods for accurate user-friendly calibration and robust alignment, as well as to evaluate its performance for offline alignment of real images and real-time tracking of surfaces and hands for interactive augmentation, obtaining the results described in the following sections.

6.1 Software Validation

To confirm that we could easily obtain accurate calibration needed by the alignment algorithm, we ran tests using the software on a Dell Vostro 400 computer with an Intel Core 2 Quad Q6600 2.4 GHz CPU. As calibration target, we used the one shown in Figure 3.8, consisting of fiducial markers printed on a B4 size sheet of paper and pasted on a (mostly) flat foam board. We present first results of the geometric calibration, then of the color calibration, and finally of the image alignment algorithm itself on simulated images, validating the correctness and robustness of the algorithm, given accurate calibration.

6.1.1 Geometric Calibration

For geometric calibration, our test hardware consisted of a NEC LT157 (1024×768 color LCD) projector and a PGR Grasshopper GRAS-14S5M-C (1280×960 grayscale CCD) camera attached to a Fujinon HF16SA-1 (16 mm) lens. As calibration target, we used the calibration patterns of Figure 3.3,

Table 6.1: Reprojection RMSE (root mean square errors) in pixels after calibrating using our method with either corners or centers

	Reprojection RMSE (pixels)	
	Camera	Projector
Corners	0.43	0.66
Centers	0.33	0.20

Table 6.2: Reprojection RMSE (root mean square errors) in pixels of other methods: For tidiness, we cite only the first author

Authors	Reprojection RMSE (pixels)	
	Camera	Projector
Ashdown [1]	0.25	0.47
Audet [2]	0.23	0.52
Drouin [17]	N/A	0.27
Gao [22]	0.63	0.43
Gao [23]	0.15	0.24
Griesser [24]	< 0.4	< 1.5
Kimura [32]	≈ 0.3	≈ 0.4
Legarda-Sáenz [35]	0.60	0.79
Li [37]	0.094	0.15
Zhang [55]	≈ 0.4	≈ 0.6
Average	0.34	0.54

laser printed the first one on a B4 size sheet of paper and pasted it on a (mostly) flat foam board.

Camera capture via software trigger took on average 158 ms, the display delay was about 64 ms, and the processing ran on only one processing unit in approximately 350 ms, for a total of 572 ms per iteration. The test user could fully calibrate both camera and projector within about 30 seconds of manipulation. Figure 1.2 shows him in action on the casually installed system. The full sequence can be downloaded from
http://www.ok.ctrl.titech.ac.jp/~saudet/procams2009.mp4 .

The reprojection errors, as listed in Table 6.1, also indicate that we were able to obtain an accurate subpixel calibration from this test session, which we could easily reproduce at will. Moreover, using a flatter wooden board, we could obtain better results of less than 0.15 pixels for both camera and projector, as shown in Figure A.2. The errors also show that, as predicted, because the projector has a narrower depth of field than the camera, it benefits more from the use of marker centers.

For quick comparison, we summarize in Table 6.2 other reprojection errors reported in the literature. For publications with more than one set of numbers, we took only the best.

6.1.2 Color Calibration

We could also obtain accurate results for color calibration. In this case, our test hardware consisted of a NEC LT157 (1024×768 color LCD) projector and a PGR Flea2 FL2G-13S2C-C (1280 × 960 Bayer color CCD) camera attached to a Cosmicar C814 (8 mm) lens, and we used the same calibration board as above. We placed in Table 6.3 a sample color mixing matrix obtained with our hardware. As shown in Figure 6.1, we typically obtain an RMSE (root mean square error) of approximately 0.5% of the total intensity range, and a correlation coefficient R^2 of almost one. This validates the assumptions of linear response of the camera sensor and of the sRGB response curve of the projector. One can observe that the green channel of the camera sensor is the most sensitive and the blue channel, the least, as expected from the specifications of the sensor.

6.1.3 Alignment of Simulated Images

To analyze the convergence behavior of the algorithm, we ran simulation experiments, using as parameters for the virtual camera and projector the values listed in Table 6.4 and as input images the ones shown in Figure 6.2. The orientation of the images in the first column matches with the initial

Table 6.3: Sample output of the color calibration, with bias $\mathbf{b}$ = zero

Color mixing matrix X			Ambient light $\mathbf{a}$
0.3949	0.04552	0.008233	5.947
0.05504	0.8004	0.08908	9.551
0.0009641	0.02049	0.2771	3.114

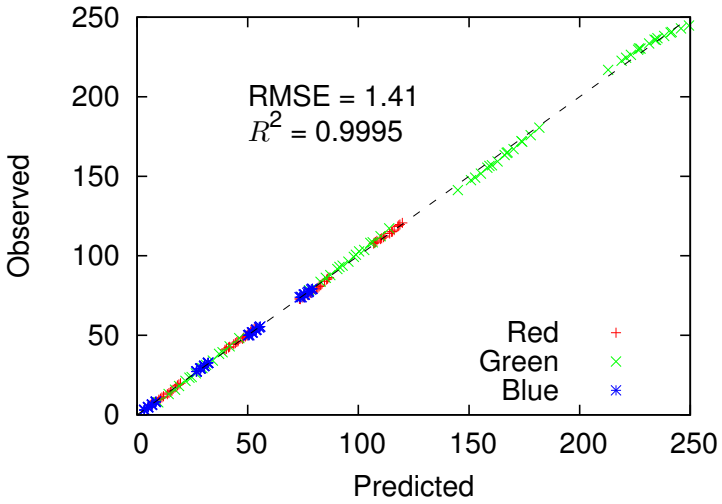


Figure 6.1: Predicted and observed intensities of the camera, where predicted values are computed as per Equation 2.4 using parameters of Table 6.3 and observed values come from data taken during the color calibration using the sampling pattern discussed in Section 3.2

Table 6.4: The camera and projector parameters used during our simulation experiments

	Camera	Projector
Image size	1280×960	1024×768
Camera matrix K	$\begin{bmatrix} 3200 & 0 & 640 \\ 0 & 3200 & 480 \\ 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 2560 & 0 & 512 \\ 0 & 2560 & 384 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation matrix R	I	Rodrigues$(0, \arctan \frac{0.1}{2}, 0)$
Translation vector $\mathbf{t}$	$\mathbf{0}$	R $[\ 0.1\,\mathrm{m}\ \ 0\ \ 0\ ]^{\mathrm{T}}$
Color mixing matrix X	I	I

images of the camera, while the corresponding ones from the second column indicate also at initialization the unmodulated projector images, which overlap the board almost exactly when $\mathbf{n} = [\ 0 \quad 0 \quad \frac{1}{2\,\mathrm{m}}\ ]$. We selected for processing the middle 640×480 rectangular ROI of the camera image.

Each simulation trial consisted of the following procedure that our test program followed to produce and align the transformed images:

1. Add noise taken from a normal distribution $\mathcal{N}(0, \sigma^2)$ to the four coordinates of the camera ROI.

2. Use this displacement to compute via DLT [26] a random homography H.

3. Decompose [51] H into rotation R, translation $\mathbf{t}$, and plane parameters $\mathbf{n}$, choosing the set associated with the $\mathbf{n}$ closest to the z-axis.

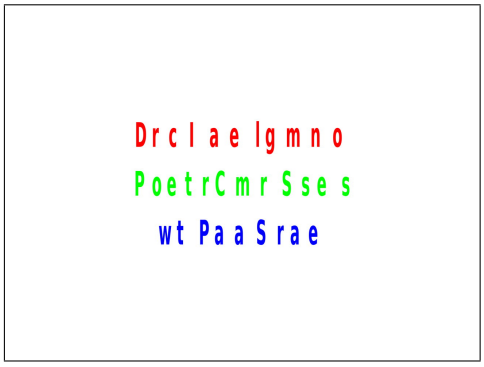
4. Scale $\mathbf{n}$ and $\mathbf{t}$ to place the board 2 meters in front of the camera.

5. Let the projector gain $g \sim \mathcal{N}(0.5, 0.1^2)$ and ambient light $\mathbf{a} \sim (\mathcal{A}, \mathcal{A}, \mathcal{A})$ where $\mathcal{A} = \mathcal{N}(0.1, (\frac{0.1}{3})^2)$.

6. Create a simulated camera image by first transforming the images using the above H, $\mathbf{n}' = \frac{\mathrm{R}\mathbf{n}}{1-\mathbf{t}^{\mathrm{T}}\mathbf{n}}$, g and $\mathbf{a}$ parameters, then adding to the pixel values an amount of Gaussian noise $\mathcal{N}(0, 0.01^2)$, and finally saturating (clamping) the intensity sums within their valid range $[0, 1]$.

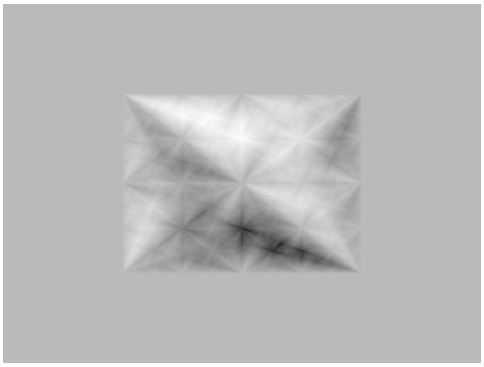
7. Execute the alignment algorithm using as settings seven pyramid levels, no zero thresholds, $\Delta_{\min} = 0.1$, and as initial parameters the four reference points of the camera ROI, $\mathbf{n}$ from step 4, $g = 0.5$, and $\mathbf{a} = (0.1, 0.1, 0.1)$.

We organized the results in Figure 6.3 for 10000 trials per data point, where we judged that our algorithm had successfully converged when the final RMSE with the four points of the warped ROI was below 0.1 pixels. We note that the subspace error term helps most the less well textured images (in order: demo, page, and fractal), as the additional information from the projector image helps better constrain the motion of the board. With the first set of images at $\sigma = 100$ pixels, the convergence frequency goes up by as much as 37%. The convergence speed however, measured in number of iterations, does not change significantly.

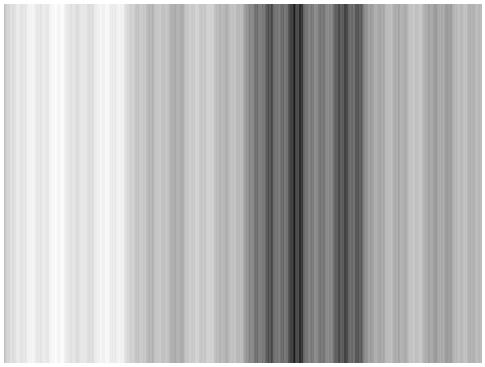


(a) Demo board image

(b) Demo projector image



(c) Fractal board image



(d) Fractal projector image

Direct Image Alignment of Projector-Camera Systems with Planar Surfaces

Samuel Audet, Masatoshi Okutomi, and Masayuki Tanaka

Tokyo Institute of Technology

2-12-1 Ookayama, Meguro-ku, Tokyo, Japan

Abstract

I. Introduction

(e) Page board image

(f) Page projector image

Figure 6.2: The source images used to generate our simulated images

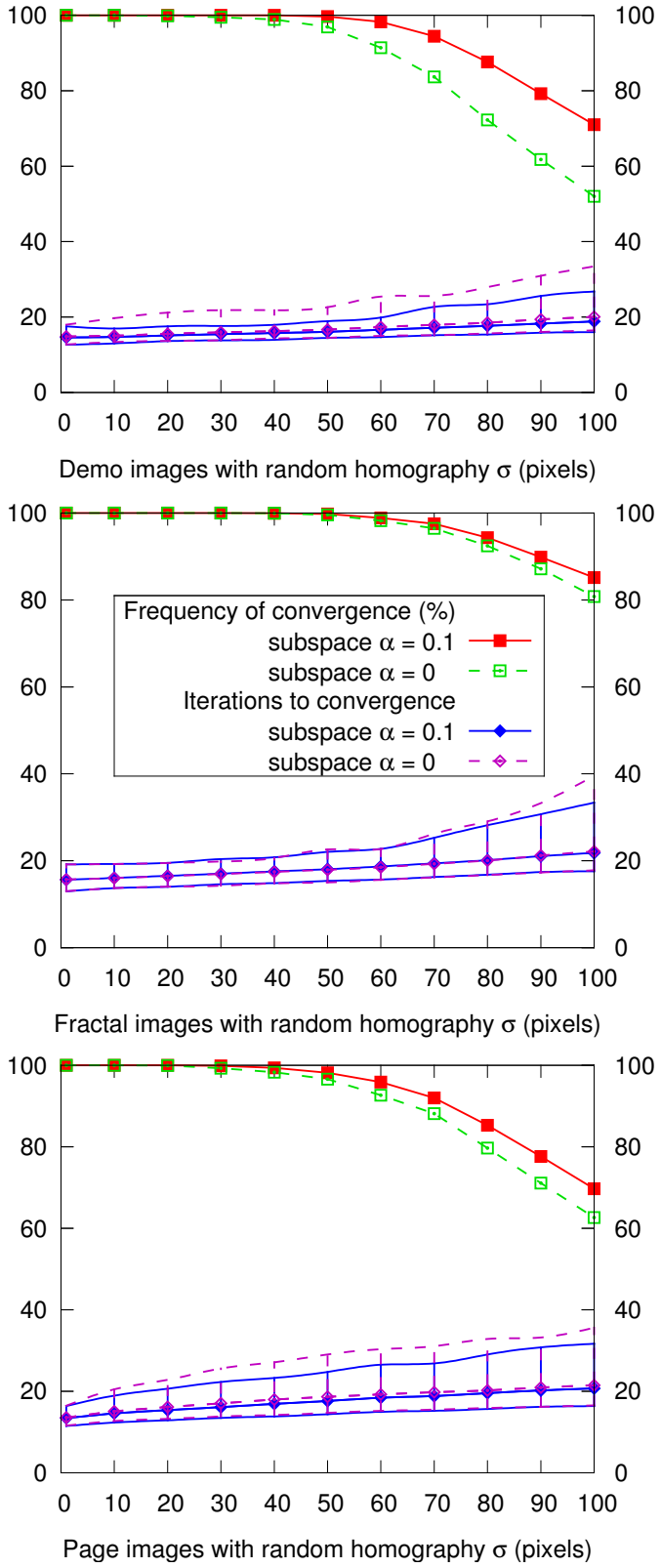


Figure 6.3: Results from the simulation experiments, where for a given σ associated with the random homography, the curves in the upper part of the graphs indicate the convergence frequency while the ones in the lower parts show how many iterations it took to reach convergence on average, and the error curves delineate the standard deviations (SD) below and above average

6.2 Performance Evaluation

To understand how well and how fast the alignment algorithm could actually work, we also challenged our algorithm with real images captured using libdc1394. Our test hardware consisted of a Dell XPS 8300 computer with an Intel Core i7-2600 8MB cache 3.4GHz CPU and an NVIDIA GeForce GTX 560 Ti GPU, a Casio XJ-S68 (1024×768 color DLP) projector, and a PGR Flea2 FL2G-13S2C-C (1280×960 Bayer color CCD) camera attached to a Pentax H1212B (12 mm) lens. For the surface planes, we inkjet printed the patterns described below on A4 size sheets of paper and pasted them on (mostly) flat foam boards. We conducted two sets of experiments, one to measure accuracy quantitatively by comparing offline alignment results with easy-to-detect markers, and the other to demonstrate real-time operation and support for arbitrary textures, including interaction from the test user.

6.2.1 Offline Alignment of Real Images

When aligning images in the real world, given that we seek an algorithm that runs as fast as possible and given that the displacements are usually less extreme than the ones tested with our simulations, we used only five pyramid levels with RMSE-based zero thresholds $t = 1.0$, 0.75, 0.5, 0.25, and 0 respectively, let the subspace $\alpha = 100$ and $\Delta_{\text{min}} = 10$, and reduced the line search to only two values: 1.0 and 0.25. Surprisingly perhaps, these diminished settings did not degrade accuracy at all.

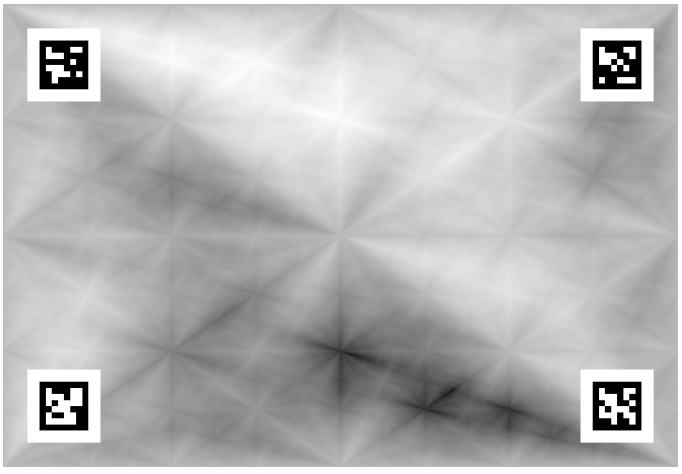
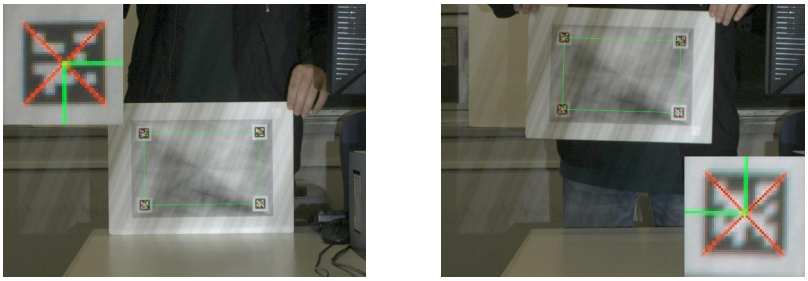
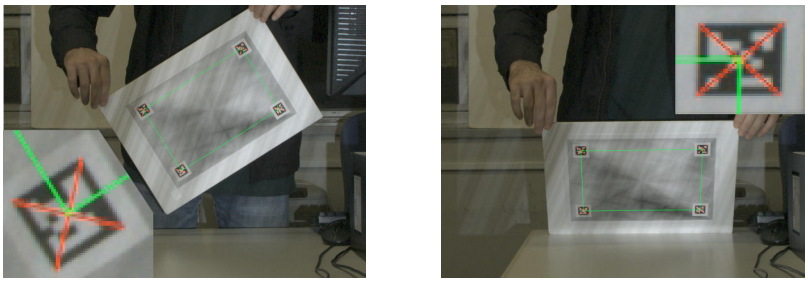


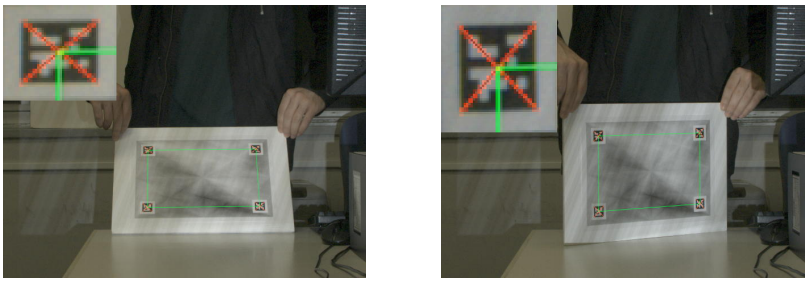
Figure 6.4: The printed triangular fractal pattern along with four markers that we used to obtain accuracy data



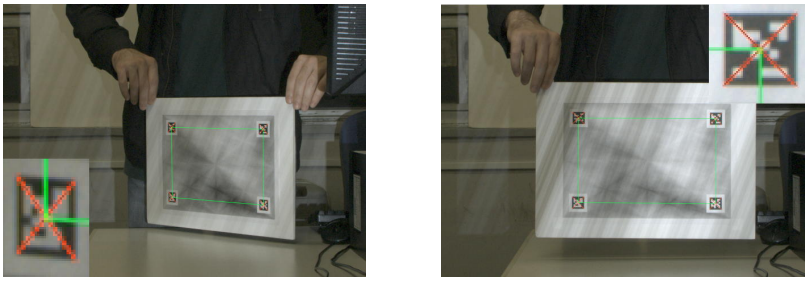
(a) frame 10 (b) frame 50



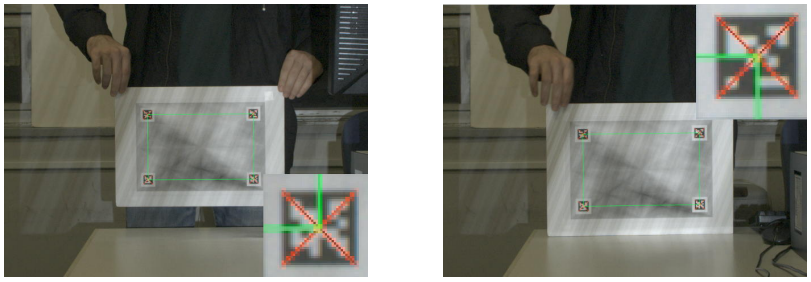
(c) frame 110 (d) frame 150



(e) frame 180 (f) frame 210



(g) frame 240 (h) frame 270



(i) frame 300 (j) frame 340

Figure 6.5: Frames from the markers test video, which features the patterns of Figure 6.2(d) and 6.4, with offline drawn red crosses representing detected markers and green rectangles denoting the corresponding regions aligned by our program: The rectangle corners match the crosses with subpixel accuracy as shown in the blowups

As representative of the current methods, we chose ARToolKitPlus [52] along with OpenCV [12] for subpixel detection, and compared its results with ours. We put the texture of Figure 6.4 on the planar surface. This fractal pattern is the 2D equivalent of Equation 4.16, but where triangles split the space at each recursion. Obviously, we also used Figure 6.2(d) for the projector pattern in an attempt to extract the best accuracy. On the initialized reflectance image, the markers delimited an ROI of 197255 pixels. For each subsequent frame, our algorithm for CPU took on average a total of 45 ± 5.9 (SD) ms to converge, but we noted that iterations at pyramid level 0 brought no additional accuracy, possibly due to the noise level, so the effective processing time equaled in fact 18 ± 2.7 (SD) ms. We placed shots of the frames as captured by the camera in Figure 6.5. The full sequence can be downloaded from http://www.ok.ctrl.titech.ac.jp/~saudet/markerstest_2012-01-30.mp4 .
In the case of the GPU, the corresponding figures are 27 ± 3.4 (SD) ms and 18 ± 2.0 (SD) ms. Tables 6.5 and 6.6 list the average number of iterations and the time taken for each at different levels of the resolution pyramid, in the case of the optimized algorithms for both CPU and GPU, where the warping function alone took most of the time, as can be seen from the processing time roughly halving at each level higher in the pyramid. From the execution on CPU, the difference in pixels between our results and the centers of the four markers over the whole test video is plotted in Figure 6.6. Although markers do not provide the ground truth either, we consider the difference as errors of our algorithm since the error of corner detection should be less than 0.10 pixels [12]. The total RMSE sums up to 0.24 pixels, which even includes images violating the assumed single gain g, especially true of slanted planes, and portions of motion blur (e.g., Figure 6.5(b)).

Table 6.5: Average number of iterations and time per iteration of the markers test $\pm$ their standard deviations (SD) with optimized algorithm for CPU

Pyramid level	Camera resolution	Projector resolution	Average iterations	Average time (ms)
0	1280×960	1024×768	1.00 ± 0.00	27 ± 4.1
1	640×480	512×384	1.00 ± 0.00	9.8 ± 1.3
2	320×240	256×192	1.00 ± 0.00	3.6 ± 0.7
3	160×120	128×96	1.59 ± 0.65	1.7 ± 0.5
4	80×60	64×48	1.54 ± 0.85	1.3 ± 0.5

Table 6.6: Average number of iterations and time per iteration of the markers test ± their standard deviations (SD) with optimized algorithm for GPU

Pyramid level	Camera resolution	Projector resolution	Average iterations	Average time (ms)
0	1280×960	1024×768	1.00 ± 0.00	11 ± 1.6
1	640×480	512×384	1.00 ± 0.00	5.5 ± 0.6
2	320×240	256×192	1.00 ± 0.00	3.4 ± 0.5
3	160×120	128×96	1.57 ± 0.64	2.2 ± 0.5
4	80×60	64×48	1.52 ± 0.84	1.7 ± 0.5

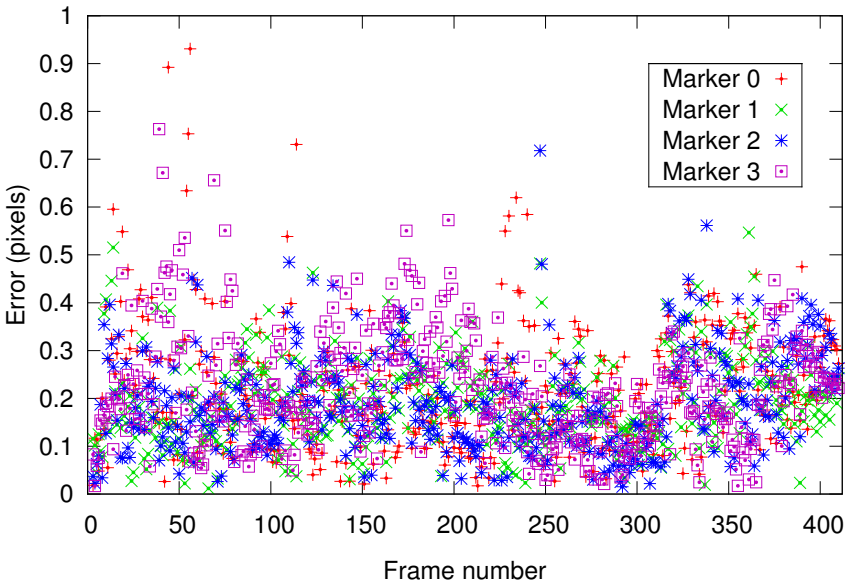
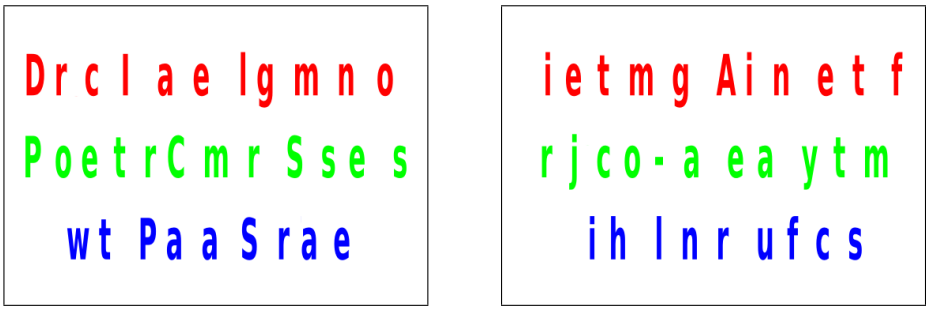


Figure 6.6: Difference in pixels between our results on CPU and the detected centers of the markers

6.2.2 Real-Time Interactive Augmentation

To show that our program works in real time even with poor and arbitrary textures that easily overlap on the surface, we ran our algorithm using the images shown in Figure 6.7, the first pasted on the surface plane and the second displayed by the projector, along with other graphical elements as displayed in Figure 1.1 showing the system in action. More precisely, after each frame, the system warped the projector image to achieve a geometric correction on the physical plane by applying the homography $H_{ps} = H_{pc}H_{sc}^{-1}$, which transforms points from the surface to the projector. (Although we could also correct the projector colors to match the printed ones, we intentionally left them as is for simplicity and ease of visual inspection.) We also tested our hand tracking algorithm to show that the system has the potential to fulfill the needs of the applications described in Section 5.3.

Using the GPU algorithm, we limited the amount of time our program could spend iterating to 15 ms, the sweet spot for our test hardware. During our test session, the machine spent in reality on average per frame 17 ± 1.5 (SD) ms iterating, 1.9 ± 0.6 (SD) ms confirming that new alignment parameters are in fact better, 3.3 ± 0.6 (SD) ms tracking the hand, and 3.7 ± 2.9 (SD) ms displaying, grabbing, updating, (un)distorting, and building resolution pyramids of the projector and camera images in parallel on both CPU and GPU for a total of 26 ± 3.3 (SD) ms, running at a full 30 frames per second, limited only by the camera. The relatively large variance occurs mainly due to the jitter caused by the camera and the projector, for which we need to reserve at this moment about 5 ms on average to compensate. According to data obtained from profiling the software, with some more effort optimizing the code, further increase in speed appears achievable on similar commodity hardware. Still, as can be seen from the shots of the demo video we placed in Figure 6.8, in addition to Figures 1.1 and 5.1, the system could



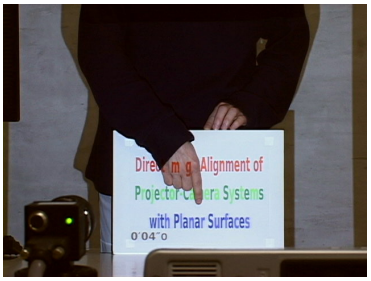
(a) Printed on the board (b) Projector displayed

Figure 6.7: The images used for our demo video

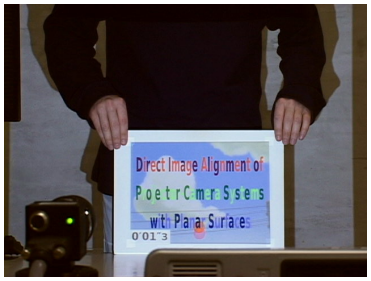
perform the interactions we wanted in real time. The full sequence can be downloaded from our Web site at
http://www.ok.ctrl.titech.ac.jp/~saudet/procamtracker_2012-02-14.mp4 .

In this real-time case as well, the algorithm successfully converged given any displacements reasonable for a direct alignment method [8]. We also found that the single gain g used for the entire image was sufficient when all points of the surface plane are not far from each other compared to their average distance from the projector, otherwise vignettinglike effects obviously occur, a possibly impractical limitation for some applications.

As for hand tracking, although calibration and surface tracking are subpixel accurate, as shown above in the previous section, potentially offering the same level of accuracy to hand tracking, the simple analysis described in the section 5.1.3 is only as accurate as the size of the tip. Furthermore, the binarization, not unlike background subtraction, needs occluded regions to contrast visibly, or else tracking failures may occur. We can usually avoid those by insuring sufficient illumination given the dynamic range of the camera and by using a surface of plain texture inside interactive regions, which users usually prefer anyway when operating a complex virtual interface. In fact, we may easily adapt the algorithm to work with no texture at all, simply rectifying the display with the plane. Mobile devices could exploit this to provide a larger screen to users, as long as they can find a flat surface, such as a wall or a (physical) desktop, where to project the GUI.



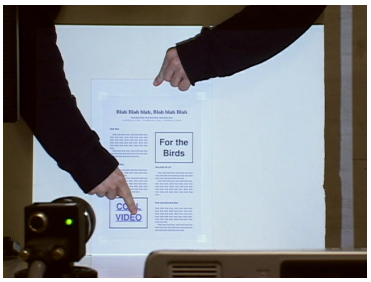
(a) frame 464



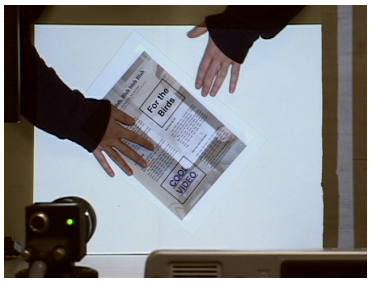
(b) frame 511

(c) frame 663

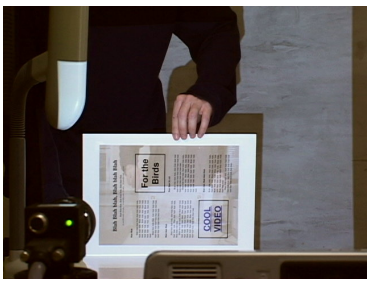
(d) frame 1210



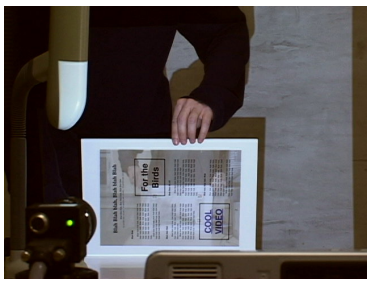
(e) frame 2052



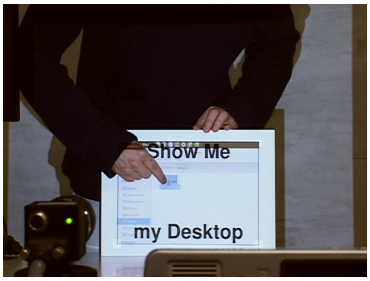
(f) frame 2196



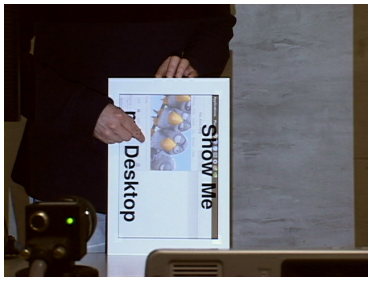
(g) frame 3072



(h) frame 3085



(i) frame 3756



(j) frame 4353

Figure 6.8: Frames from our demo video, where the first four depict a red virtual ball bouncing off the edges of the surface, alongside the patterns of Figure 6.7 plus a chronometer and a video animation, the next one features the user "clicking" on a physical hyperlink, followed by occlusion and changes in the ambient light, and the last two, interaction with a GUI, all being aligned robustly in real time

Chapter 7

Discussion and Conclusion

With regards to our calibration method, judging from our results and comparing them to data available from previous methods, little of which include information about usability, we conclude that our method is the easiest to use. A calibration session that typically requires only a light board, a printed pattern, and less than one minute justifies in our eyes this designation. On the other hand, the subpixel accuracy and the less than one percent of error achieved for color prediction compare favorably with all previous methods. Moreover, since we based our geometric method solely on existing theory of camera calibration, assuming the noise characteristics of projectors are similar to cameras, published results for those [49, 56], with regards to the actual physical 3D accuracy, should translate similarly. However, this remains to be proven. For the current work, we fulfilled our main goal of a user-friendly calibration method.

We hope that being able to calibrate easily projector-camera systems will help research in two ways. First, from the point of view of a user, since the calibration procedure is simple and does not require any special hardware, it could realistically become part of standard installation instructions targeted at end users. Second, researchers may save time when (re)calibrating their systems, instead spending hopefully more time actually using the calibration information.

Nevertheless, the calibration method does have a few quirks. It can fail when all markers completely overlap, but we confirmed that a human user can easily detect such conditions and slightly move the board to help the machine. Furthermore, to obtain the most accurate calibration, the casual user may not be aware that the calibration board should be placed at angles of approximately 45° with regards to image planes of the projector and the camera, and that it should also cover the largest area possible. A future implementation might be able to solve this by properly decomposing the

homographies [53] and by providing appropriate feedback to users, *i.e.*: Point them in the right direction for the next ideal sweet spot. Despite all this, prewarps that use homographies alone might still fail for projectors that exhibit strong nonlinear distortions, such as omnidirectional projectors.

As for the image alignment algorithm itself, using mathematical approximations and algorithmic tricks detailed in this thesis, even though we managed to boost the performance of our algorithm to decent levels, while retaining robustness and accuracy, there surely exists other ways to accelerate it still further. One interesting idea we had was to formulate an approximate model that could profit from schemes such as inverse composition (IC) [8, 10, 29], but they cannot unfortunately align simultaneously, and thus robustly, projected and surface textures. The intuition was that since we physically place the camera and projector close together, the displacement of the projector image on the physical surface could be assumed negligible. We could let the optimizer deal with any remaining misalignment as noise. Sadly, the noise generated proved to be too great, and both robustness and accuracy dropped unacceptably. Nevertheless, other avenues for optimization undoubtedly exist and we should be able to enhance the execution performance further with even better implementations for CPUs or GPUs.

Once sufficiently accelerated, our method could also be generalized to more complex reflectance properties and 3D surfaces. We could for example extend the mathematical model to piecewise planar surfaces to support effectively specular nonplanar surfaces, and align multiple planes simultaneously, a widely adopted approach, as implemented by Sugimoto and Okutomi [48], among others. Such a model could also compensate for our limited reflectance model. By dividing the surface in pieces that way, each may support different gain and ambient light, as demonstrated by Silveira and Malis [45] for camera-only systems, an approach which even allows for specularities.

Other possibly interesting directions of future work could center on dealing with occlusions and changes in surface reflectance. For example, the system could detect and exploit occlusions occurring as the result of gestures by the users in front of the projector as user interface input for a desktop system at the office, not unlike the scenarios introduced in Chapter 5, or to provide larger interactive screens to mobile devices. Further, they may want to add physically new information on the surface. In that case, the system would need a way to gradually update the reflectance properties.

In any case, using only a video projector and a color camera, our flexible software method offers a low cost solution to augment interactively everyday surfaces, which we have shown can also be used for tracking hands and objects. Multiple instances of the algorithm can also track multiple surfaces in parallel and can detect multiple hands as more than one mouse, offering

users the possibility to cooperate locally or even remotely [50]. However, our method does not yet support changes in texture, so at this moment users may not write or draw onto the surface. Moreover, to offer a richer 3D experience, compatibility with more sophisticated hand tracking and gesture recognition algorithms [27, 34] should be investigated.

As secondary contributions, some of the insights conceived within the course of our research, most notably zero thresholds and the subspace error term, can be adapted to provide solutions to other related computer vision problems. In particular, piecewise planar models should benefit most effectively from the subspace error term. We also contributed to all of the community better tools to do research and development on the Java platform, the most popular software development platform, in the form of JavaCV, with about 1000 downloads per month, and JavaCPP, with about 100 downloads per month, as of February 2012.

Although much work remains, we have demonstrated, at the very least, the feasibility of the approach. We have shown that our software method can respond fast enough for interactive applications, sufficient in our opinion to start conducting usability experiments of novel user interfaces. With some optimizations and further enhancements, we consider that direct image alignment could even become the basis of future applications using projector-based augmented reality. Most importantly, to encourage research in this direction, we are making the whole open-source software system available for free under the names of ProCamCalib and ProCamTracker, which are based on JavaCV and JavaCPP, as introduced next in Appendix A.

Appendix A

Software Guide

We implemented in software all of the methods and algorithms described in the previous chapters, packaged them as ProCamCalib and ProCamTracker, and released them as open-source software under the GPLv2 license [21]. ProCamCalib is a user-friendly tool to perform full geometric and color calibration of projector-camera systems as described in Chapter 3. It supports configurations of multiple cameras and projectors, but does not (yet) do global optimization of parameters. All devices are stereo calibrated with the first camera, which is placed at the origin. Also, the calibration board needs to stay visible in all cameras. Hopefully, these restrictions will be relaxed in the future. Additionally, given that camera-only systems are a subset of projector-camera systems, the application can happily calibrate a bunch of cameras with zero projectors, effectively implementing Fiala's method [19]. ProCamTracker, on the other hand, is a computer application equipped with an easy-to-use GUI to turn a perfectly normal pair of video projector and color camera into a system that can track without markers a real-world object (currently limited to matte planes), while simultaneously projecting on its surface geometrically corrected video images using the direct image alignment algorithm described in Chapter 4 as well as the interactive features of Chapter 5. One may obtain these applications from our Web site at these addresses:

- ProCamCalib: http://www.ok.ctrl.titech.ac.jp/~saudet/procamcalib/
- ProCamTracker: http://www.ok.ctrl.titech.ac.jp/~saudet/procamtracker/

In the following sections, before explaining how to use these tools, we first provide a list of additional software required to launch them.

A.1 Requirements

We wrote the applications themselves in Java and their binaries should run on any platform where an implementation of Java SE 6 or 7 exists. The binary distribution also contains natively compiled code for Linux, Mac OS X, and Windows, needed by JavaCV. Still, additional software is required. (For answers to problems frequently encountered with OpenCV on the Windows platform, please refer to "Common issues with OpenCV under Windows 7": http://code.google.com/p/javacv/wiki/Windows7AndOpenCV .)

In short, before running the applications, the following needs to be installed:

- An implementation of Java SE 6 or 7
  - OpenJDK http://openjdk.java.net/install/ or
  - Sun JDK http://www.oracle.com/technetwork/java/javase/ or
  - IBM JDK http://www.ibm.com/developerworks/java/jdk/ or
  - Java SE for Mac OS X http://developer.apple.com/java/ etc.
- OpenCV 2.3.1 http://sourceforge.net/projects/opencvlibrary/files/
- JOCL and JOGL from JogAmp http://jogamp.org/

The installed versions of Java and OpenCV must have the same bitness: 32-bit and 64-bit modules do not mix under any circumstances. Further, the applications run a lot faster under the "server" JVM than the "client" JVM, but because of its bigger size, not all distributions of Java come with the server one.

Additionally, for IIDC/DCAM cameras, Microsoft's Kinect stereo camera, or other cameras supported via FFmpeg:

- libdc1394 2.1.x (Linux and Mac OS X)
  http://sourceforge.net/projects/libdc1394/files/
- PGR FlyCapture 1.7∼2.2 (Windows only)
  http://www.ptgrey.com/products/pgrflycapture/
- OpenKinect http://openkinect.org/
- FFmpeg 0.6.x or 0.7.x http://ffmpeg.org/download.html
  - Precompiled for Windows http://ffmpeg.zeranoe.com/builds/
    - Known compatible builds: ffmpeg-0.7.1-win32-shared.7z, ffmpeg-0.7.1-win64-shared.7z

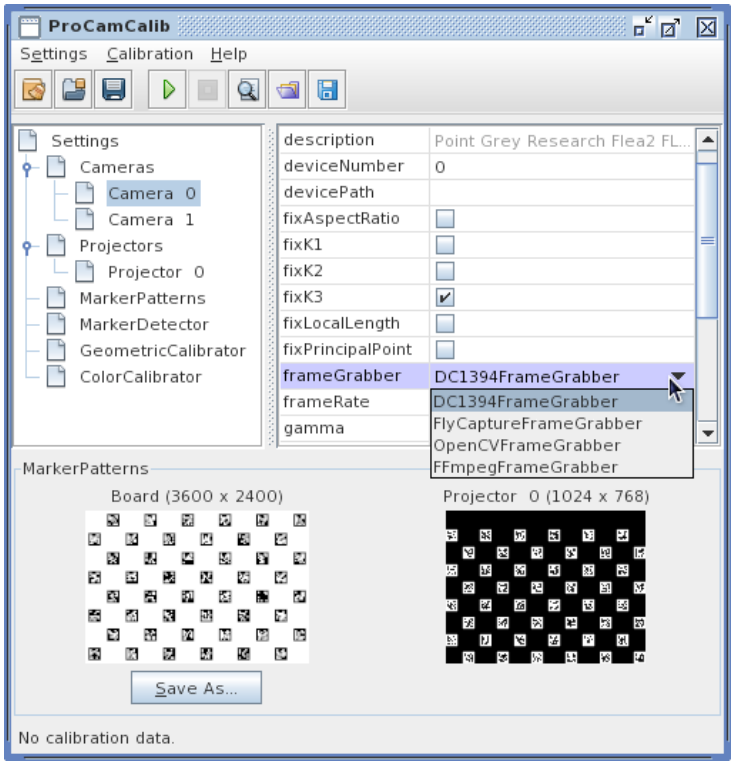


Figure A.1: Screenshot of the main window of ProCamCalib, where all the settings can be adjusted: They are currently configured for calibrating two cameras and one projector

A.2 ProCamCalib

To launch ProCamCalib under Linux, Mac OS X, and other Unix variants, execute either `procamcalib-nativelook` or `procamcalib-oceanlook`, according to the theme that works best on a given system. ("Ocean" being Java's original look and feel.) The equivalent files under Windows are `procamcalib-nativelook.cmd` and `procamcalib-oceanlook.cmd`.

After launch, the user interface that appears allows the user to change the number of cameras and projectors to calibrate, as shown in Figure A.1. There are also a lot of settings, although the defaults should be good enough for the usual cases. We do not provide a detailed description, but most of them should be clear to readers familiar with our work as described in Chapter 3 based on previous work by Fiala and Shu [19], Zhang [56], and many others as part of OpenCV [12].

Once all the settings have been modified to the desired values, since the application may crash during the operations described below, we recommend saving them in an XML file via the "Settings" menu.

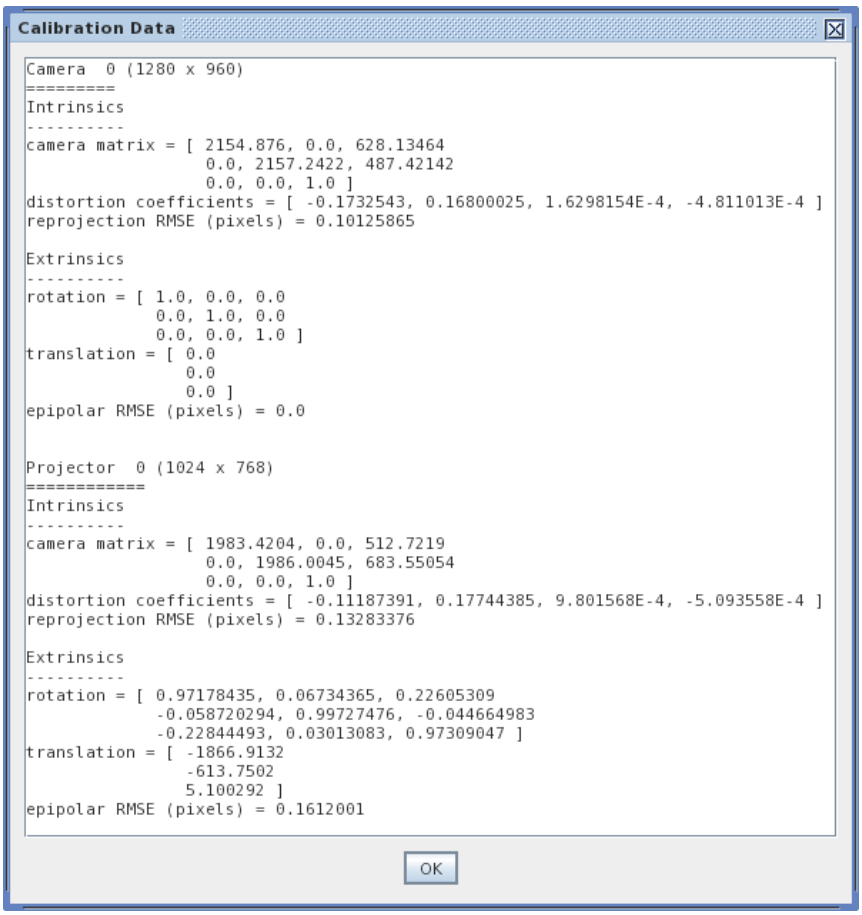


Figure A.2: Screenshot of the Calibration Data window of ProCamCalib displaying sample geometric calibration results obtained for one camera and one projector using a flat wooden calibration board

Before going any further, one needs to print out the board pattern. We may export the image file by clicking on the "Save As..." button at the bottom of the main window, and print out the resulting file (written in PNG, BMP, PGM, or any other format supported by OpenCV, depending on the extension provided to the filename).

After pasting the pattern on a flat calibration board, and making sure that the "screenNumber" settings correspond to the desired projectors, the user may start the calibration process via the "Calibration" menu. However, before starting, we recommend, if possible, to set the cameras in a mode with more than 8 bits per pixel (*e.g.*: 10 or 16 bits). The added dynamic range may make the calibration process easier and more accurate. The algorithm calibrates all cameras simultaneously, while calibrating projectors only one at a time, for obvious reasons. When the user wishes ProCamCalib to take an image for calibration, the board should stay as steady as possible for a few

seconds until the camera image "flashes". ProCamCalib may however fail to detect markers properly if they are not clear or big enough. One may have to adjust properly the focus, exposure, resolution, etc. of the projectors and cameras. We refer the interested reader to Chapter 3 and its demo video to understand further the algorithm and hopefully how to perform calibration better.

As for color calibration, it is enabled by default, and starts automatically after geometric calibration has completed. During this process, the application displays the array of colors of Figure 3.10 during a period of a few seconds. Users who do not need color calibration should make sure to disable it in the settings before starting calibration.

After a successful calibration session, the application holds in memory the "Calibration Data", as shown in Figure A.2. The user may examine and save this data via the "Calibration" menu. The program uses OpenCV to output a file in the YAML or XML format. This information can easily be parsed back in any program by using the `CvFileStorage` facilities of OpenCV.

A.3 ProCamTracker

To launch ProCamTracker under Linux, Mac OS X, and other Unix variants, execute either `procamtracker-nativelook` or `procamtracker-oceanlook`, according to the theme that works best on a given system. ("Ocean" being Java's original look and feel.) The equivalent files under Windows are `procamtracker-nativelook.cmd` and `procamtracker-oceanlook.cmd`.

After launch, the user interface that appears allows the user to change settings for the camera, the projector, and the various modules of the tracking algorithm, as shown in Figure A.3. We describe below a typical usage scenario and will not explain all the settings in details. The default values of the ones we do not mention should be good enough for most cases, but the keen user should feel free to experiment.

1. Camera Settings
   Select a suitable `FrameGrabber` for the hardware, and fill in either `deviceFile`, `deviceNumber` or `devicePath`, and `format` as appropriate. Place the path to the `calibration.yaml` file created by ProCamCalib in the `parametersFile` field, while the `name` field must correspond to an entry in that file.

2. Projector Settings
   As with the camera settings, fill in the `parametersFile` field, but also confirm that the `screenNumber` corresponds to the one of the projector.

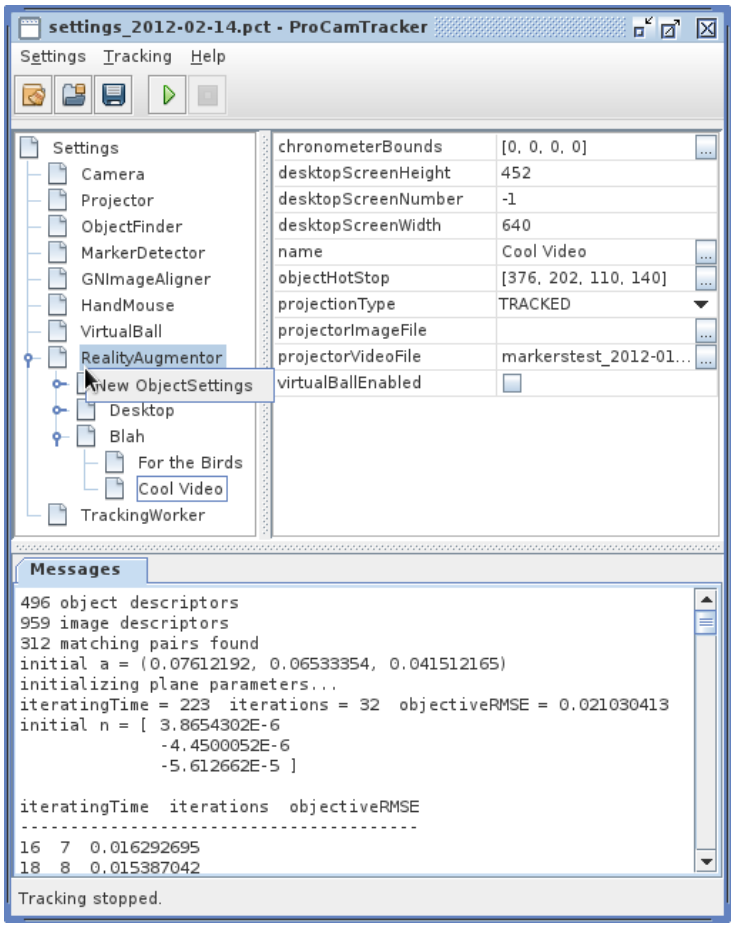


Figure A.3: Screenshot of the main window of ProCamTracker, where all the settings can be adjusted, taken after a tracking session, which displayed a few output messages related to the algorithm

3. RealityAugmentor/ObjectSettings/VirtualSettings
   Locate an image file (PNG, JPEG, etc.) or video file (AVI, MPEG-4, etc.) to be projected on the planar surface and specify its path in the `projectorImageFile` or `projectorVideoFile` field respectively.

Once all the settings have been modified to the desired values, since the application may crash during the operations described below, we recommend saving them in an XML file via the "Settings" menu.

At this point, the application is ready to start the tracking algorithm. From the "Tracking" menu, after clicking on the "Start" item and if all goes well, a window entitled "Initial Alignment" should appear containing an image captured from the camera. In this window, the user needs to click four different points that delimit the ROI (region of interest) on the planar surface that should map to the four corners of the projector image, in this order: upper left, upper right, lower right, and lower left. After the last click, tracking should start within a few seconds. The user may then begin to move the planar surface and see if the system can successfully track it or not.

A.4 Source Code

Although we are distributing all the source code from our site given above, we are also making it available on the following more developer-oriented site:

- JavaCV http://code.google.com/p/javacv/

In fact, we implemented as part of JavaCV most of the code not related to the GUI. It is an open-source computer vision library that we developed to facilitate software development, based mainly on JavaCPP and OpenCV. One will also need the following to modify and build the applications:

- A C/C++ compiler (Microsoft C/C++ Compiler, GNU C/C++ Compiler, etc.)
- JavaCPP http://code.google.com/p/javacpp/
- NetBeans 6.9 http://netbeans.org/downloads/
- ARToolKitPlus 2.1.1t http://code.google.com/p/javacv/downloads/list

(We copied the icons from the source code repository of NetBeans, which is also licensed under the GPLv2 [21].)

Since we are making all the source code of our work available, we welcome any modifications, enhancements, or fixes to the code that readers and users may wish to contribute. Please keep us informed of any updates so that we may integrate them into our own version.

References

[1] Mark Ashdown and Yoichi Sato. Steerable Projector Calibration. In *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005) - Workshops (Procams 2005)*, volume 3, page 98. IEEE Computer Society, 2005.

[2] Samuel Audet and Jeremy R. Cooperstock. Shadow Removal in Front Projection Environments Using Object Tracking. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007) - Workshops (Procams 2007)*. IEEE Computer Society, June 2007.

[3] Samuel Audet and Masatoshi Okutomi. A User-Friendly Method to Geometrically Calibrate Projector-Camera Systems. In *2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2009) - Workshops (Procams 2009)*, pages 47–54. IEEE Computer Society, June 2009.

[4] Samuel Audet, Masatoshi Okutomi, and Masayuki Tanaka. Direct Image Alignment of Projector-Camera Systems with Planar Surfaces. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2010)*, pages 303–310. IEEE Computer Society, June 2010.

[5] Samuel Audet, Masatoshi Okutomi, and Masayuki Tanaka. Augmenting Moving Planar Surfaces Interactively with Video Projection and a Color Camera. In *IEEE Virtual Reality 2012 (VR 2012)*. IEEE Computer Society, March 2012.

[6] Samuel Audet, Masatoshi Okutomi, and Masayuki Tanaka. Augmenting Moving Planar Surfaces Robustly with Video Projection and Direct Image Alignment. *Special Issue: Models for Mixed and Augmented Reality, Virtual Reality*, 2012. In print.

[7] Simon Baker, Ankur Datta, and Takeo Kanade. Parameterizing Homographies. Technical Report CMU-RI-TR-06-11, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, March 2006.

[8] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *International Journal of Computer Vision*, 56(1):221–255, March 2004.

[9] Deepak Bandyopadhyay, Ramesh Raskar, and Henry Fuchs. Dynamic Shader Lamps: Painting on Movable Objects. In *2001 IEEE and ACM International Symposium on Augmented Reality (ISAR 2001)*, page 207. IEEE Computer Society, 2001.

[10] Adrien Bartoli. Groupwise Geometric and Photometric Direct Image Registration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(12):2098–2108, December 2008.

[11] Oliver Bimber and Ramesh Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A. K. Peters, Ltd., Natick, MA, USA, 2005.

[12] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[13] John F. Canny. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[14] Dalit Caspi, Nahum Kiryati, and Joseph Shamir. Range Imaging With Adaptive Color Structured Light. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):470–480, 1998.

[15] Xinli Chen, Xubo Yang, Shuangjiu Xiao, and Meng Li. Color Mixing Property of a Projector-Camera System. In *Fifth International Workshop on Projector-Camera Systems (Procams 2008)*, pages 1–6. ACM, 2008.

[16] Paul E. Debevec and Jitendra Malik. Recovering High Dynamic Range Radiance Maps from Photographs. In *24th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 97)*, pages 369–378. ACM Press/Addison-Wesley Publishing Co., 1997.

[17] Marc-Antoine Drouin, Guy Godin, and Sébastien Roy. An Energy Formulation for Establishing the Correspondence Used in Projector Calibration. In *Fourth International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*, June 18–20, 2008.

[18] Mark Fiala. Comparing ARTag and ARToolkit Plus Fiducial Marker Systems. In *IEEE International Workshop on Haptic Audio Visual Environments and their Applications (HAVE 2005)*, page 6. IEEE Computer Society, October 1–2, 2005.

[19] Mark Fiala and Chang Shu. Self-identifying patterns for plane-based camera calibration. *Machine Vision and Applications*, 19(4):209–216, July 2008.

[20] James Foley, Andries van Dam, Steven Feiner, and John Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Professional, Second edition, 1990.

[21] Free Software Foundation. GNU General Public License, version 2 - GPLv2, 1991. http://www.gnu.org/licenses/.

[22] Wei Gao, Liang Wang, and Zhan-Yi Hu. Flexible Calibration of a Portable Structured Light System through Surface Plane. *Acta Automatica Sinica*, 34(11):1358–1362, November 2008.

[23] Wei Gao, Liang Wang, and Zhan-Yi Hu. Flexible Method for Structured Light System Calibration. *Optical Engineering*, 47(8):083602, August 2008.

[24] Andreas Griesser and Luc Van Gool. Automatic Interactive Calibration of Multi-Projector-Camera Systems. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshop (CVPRW 2006), Procams 2006*, page 8. IEEE Computer Society, 2006.

[25] Shilpi Gupta and Christopher Jaynes. Active Pursuit Tracking in a Projector-Camera System with Application to Augmented Reality. In *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005) - Workshops (Procams 2005)*, volume 3, page 111. IEEE Computer Society, 2005.

[26] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, Second edition, 2004.

[27] Steve Henderson and Steve Feiner. Opportunistic Tangible User Interfaces for Augmented Reality. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 16(1):4–16, January/February 2010.

[28] International Electrotechnical Commission. IEC 61966-2-1 (1999-10-18): Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB, 1999.

[29] Takahiro Ishikawa, Iain Matthews, and Simon Baker. Efficient Image Alignment with Outlier Rejection. Technical Report CMU-RI-TR-02-27, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, October 2002.

[30] Tyler Johnson and Henry Fuchs. Real-Time Projector Tracking on Complex Geometry Using Ordinary Imagery. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007) - Workshops (Procams 2007)*, pages 1–8. IEEE Computer Society, June 2007.

[31] Khronos OpenCL Working Group. The OpenCL Specification - Version 1.2 - Document Revision: 15, November 2011. http://www.khronos.org/registry/cl/.

[32] Makoto Kimura, Masaaki Mochimaru, and Takeo Kanade. Projector Calibration using Arbitrary Planes and Calibrated Camera. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007) - Workshops (Procams 2007)*. IEEE Computer Society, June 2007.

[33] Johnny Chung Lee, Scott E. Hudson, and Edward Tse. Foldable Interactive Displays. In *21st Symposium on User Interface Software and Technology (UIST2008)*, pages 287–290. ACM, 2008.

[34] Taehee Lee and Tobias Höllerer. Hybrid Feature Tracking and User Interaction for Markerless Augmented Reality. In *IEEE Virtual Reality 2008 (VR 2008)*, pages 145–152. IEEE Computer Society, March 2008.

[35] Ricardo Legarda-Sáenz, Thorsten Bothe, and Werner P. Jüptner. Accurate procedure for the calibration of a structured light system. *Optical Engineering*, 43(2):464, March 3, 2004.

[36] Bastian Leibe, Thad Starner, William Ribarsky, Zachary Wartell, David Krum, Brad Singletary, and Larry Hodges. The Perceptive Workbench: Towards Spontaneous and Natural Interaction in Semi-Immersive Virtual Environments. In *2000 IEEE Virtual Reality Conference (VR 2000)*, pages 13–20. IEEE Computer Society, March 2000.

[37] Zhongwei Li, Yusheng Shi, Congjun Wang, and Yuanyuan Wang. Accurate calibration method for a structured light system. *Optical Engineering*, 47(5):053604, May 29, 2008.

[38] Pranav Mistry, Pattie Maes, and Liyan Chang. WUW - Wear Ur World - A Wearable Gestural Interface. In *27th International Conference on Human Factors in Computing Systems (CHI 2009) - Extended Abstracts*, pages 4111–4116. ACM, 2009.

[39] Takayuki Moritani, Shinsaku Hiura, and Kosuke Sato. Real-Time Object Tracking without Feature Extraction. In *2006 IEEE Conference on Pattern Recognition (ICPR 2006)*, volume 1, pages 747–750, Los Alamitos, CA, USA, June 2006. IEEE Computer Society.

[40] NVIDIA. CUDA C Programming Guide Version 4.1, November 2011. http://developer.nvidia.com/cuda-downloads/.

[41] Ramesh Raskar and Paul Beardsley. A Self-Correcting Projector. In *2001 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2001)*, volume 2, pages 504–508. IEEE Computer Society, 2001.

[42] Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. iLamps: Geometrically Aware and Self-Configuring Projectors. In *ACM Transactions on Graphics (SIGGRAPH 2003)*, pages 809–818. ACM, 2003.

[43] Ramesh Raskar, Greg Welch, Matt Cutts, Adam Lake, Lev Stesin, and Henry Fuchs. The Office of the Future: A Unified Approach to Image-Based Modeling and Spatially Immersive Displays. In *25th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 98)*, pages 179–188. ACM Press, 1998.

[44] Jaakko Sauvola and Matti Pietikäinen. Adaptive document image binarization. *Pattern Recognition*, 33(2):225–236, February 2000.

[45] Geraldo Silveira and Ezio Malis. Real-time Visual Tracking under Arbitrary Illumination Changes. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007)*. IEEE Computer Society, June 2007.

[46] Peng Song and Tat-Jen Cham. A Theory for Photometric Self-Calibration of Multiple Overlapping Projectors and Cameras. In *2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005) - Workshops (Procams 2005)*, volume 3, page 97. IEEE Computer Society, 2005.

[47] Peter Sturm. Algorithms for Plane-Based Pose Estimation. In *2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2000)*, pages 1706–1711, Hilton Head Island, South Carolina, USA, June 2000. IEEE Computer Society.

[48] Shigeki Sugimoto and Masatoshi Okutomi. A Direct and Efficient Method for Piecewise-Planar Surface Reconstruction from Stereo Images. In *2007 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2007)*. IEEE Computer Society, June 2007.

[49] Wei Sun and Jeremy R. Cooperstock. An empirical evaluation of factors influencing camera calibration accuracy using three publicly available techniques. *Machine Vision Application*, 17(1):51–67, 2006.

[50] Naoya Takao, Jianbo Shi, and Simon Baker. Tele-Graffiti: A Camera-Projector Based Remote Sketching System with Hand-Based User Interface and Automatic Session Summarization. *International Journal of Computer Vision*, 53(2):115–133, July 2003.

[51] Bill Triggs. Autocalibration from Planar Scenes. In *5th European Conference on Computer Vision (ECCV '98)*, volume I, pages 89–105. Springer-Verlag, 1998.

[52] Daniel Wagner and Dieter Schmalstieg. ARToolKitPlus for Pose Tracking on Mobile Devices. In *12th Computer Vision Winter Workshop (CVWW'07)*. Graz University of Technology, St. Lambrecht, Austria, February 6–8, 2007.

[53] TzuYen Wong, Peter Kovesi, and Amitava Datta. Projective Transformations for Image Transition Animations. In *14th International Conference on Image Analysis and Processing (ICIAP '07)*, pages 493–500. IEEE Computer Society, 2007.

[54] Jing Xiao, Simon Baker, Iain Matthews, and Takeo Kanade. Real-Time Combined 2D+3D Active Appearance Models. In *2004 IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 2, pages 535–542, Washinton, DC, USA, June 2004. IEEE Computer Society.

[55] Song Zhang and Peisen S. Huang. Novel method for structured light system calibration. *Optical Engineering*, 45(8):083601, August 2006.

[56] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.