Diplomarbeit

User Interfaces based on a hand-held projection screen

Kristian Bergmann
220693

Gutachter: Prof. Dr.-Ing. Marc Alexa
Gutachter: Prof. Dr.-Ing. Andrew Nealen

Technische Universität Berlin
Fakultät IV - Elektrotechnik und Informatik
Institut für Technisch-Naturwissenschaftliche Anwendungen
Fachgebiet Computer Graphics

For my parents.

Eidesstattliche Versicherung

Die selbständige und eigenhändige Anfertigung versichere ich an Eides Statt.

_____

Berlin, den 1. Dezember 2008

Zusammenfassung

In dieser Diplomarbeit werden Hardware und ein Softwaresystem vorgestellt, mit denen es dem Nutzer möglich ist Objekte im virtuellen dreidimensionalen Raum auf eine intuitive Art und Weise zu erkunden. Die von uns entwickelte Hardware besteht aus einer Projektionsfläche, die der Benutzer in einer Hand halten kann und einem Stift für die andere Hand. Die virtuelle Kamera, durch die das 3D-Objekt betrachtet wird, kann durch Veränderung der Pose (Position und Orientierung) der Projektionsfläche gesteuert werden. Der zur Projektionsfläche gehörige Stift ersetzt die Maus und kann genau wie ein normaler Stift benutzt werden um direkt auf der Projektionsfläche den Cursor zu bewegen und darauf zu zeichnen.

Zum Ermitteln der Projektionsflächenpose nimmt eine Kamera die Infrarot-Leuchtdioden (IR-LEDs) auf, die am Rahmen der Projektionsfläche angebracht sind. Zuerst werden die Positionen der LEDs im Kamerabild bestimmt und anschließend den bekannten Positionen der LEDs auf dem Projektionsflächenrahmen zugeordnet. Anhand dieser Zuordnung kann die Pose des Rahmens und damit der gesamten Projektionsfläche bestimmt werden. Ein Beamer wird genutzt, um Bilder des betrachteten Objekts auf die Projektionsfläche zu werfen. Da diese allerdings beweglich ist, muss entsprechend der Pose des Projektionsrahmens das erzeugte Bild skaliert und verzerrt werden. Auf diese Weise füllt es dann genau die Projektionsfläche aus, anstatt den gesamten durch den Beamer abgedeckten Bereich zu füllen. Die dafür nötigen Berechnungen finden mit dem vorgestellten System alle in Echtzeit statt.

Außerdem werden verschiedene Wege diskutiert wie man die virtuelle Kamera, durch die das 3D-Objekt betrachtet wird, anhand der Pose der Projektionsfläche kontrollieren kann. Letztendlich wird gezeigt, wie das System verwendet werden kann um mit der Projektionsfläche und dem Stift FiberMesh, ein skizzenbasiertes Programm zum Erstellen frei definierbarer Flächen, zu bedienen.

Abstract

This thesis presents a software system and hardware that allows the user to explore virtual 3D objects in an intuitive way. The main hardware component is a hand-held projection screen with a pen. The user can control the virtual camera that observes a 3D object by altering the projection screen's pose (position and orientation). The pen is a replacement for the computer mouse and can be used to point and draw directly on the projection screen.

A camera is used to observe the infrared light emitting diodes (LEDs) that are mounted on the frame of the projection screen. After extracting the observed LEDs' positions from the camera image and matching them to their known positions on the screen's frame, the pose of the projection screen can be estimated. The virtual 3D object's image is projected onto the screen by a stationary projector. As the screen is hand-held and therefore moving, the image of the object has to be scaled and warped to appear exactly on the projection screen instead of filling the whole projector beam. All this can be done in real-time, using the system we present in the thesis.

Furthermore, different ways of utilizing the presented system to control a virtual camera are discussed. Finally, a way of using the projection screen as input and output device for FiberMesh, a sketch based interface for designing free-form surfaces, is presented.

Acknowledgments

First of all I want to thank Andrew Nealen for his continuous support and for broadening my perspective from time to time. Without him and Marc Alexa this thesis would never have been written. Therefore, I want to thank Marc Alexa and the members of the CG research team for providing feedback and support.

I am deeply grateful to Erika and Hannes who supported me in improving my written English and to Patricia, Frank, Stephan and Christoph for providing valuable feedback. Although not involved with my thesis, Thomas and Karo never failed to motivate me or provide distraction.

Most of all I thank Kirsten for being herself, motivating me and making me happy.

Contents

Chapter 1

Introduction

A hand-held device capable of delivering pose (position + orientation) data has been evaluated for manipulating objects in 3D scenes by Ware et al. as early as 1988 [WJ88]. Two years later they presented a study [WO90] about different virtual camera control metaphors. In 1993, Fitzmaurice et al. suggested combining a hand-held computer with a pose tracking device and implemented a prototype of this idea [FZC93]. Several researchers have investigated different methods of tracking the pose of mainly hand-held display devices. Of course the interaction with and by such devices has been analyzed as well. The devices and methods presented in [SG97, TFK+02, BvBRF05, WZC06] are just a few examples of the work that has been done in this area.

This thesis focuses on navigating through virtual 3D space, utilizing the pose of a hand-held display device. In particular we investigate the application of this idea to 3D modeling applications. The main interaction we are targeting is to control the virtual camera that views the 3D object, in order to increase the usability of 3D modeling software. Our vision is to give novice users of such software an easier access, as well as increasing the ease of use for advanced users.

Therefore, the device constituting the input and display peripheral should satisfy the following requirements: On the one hand, the device needs to be capable of showing images large enough for modeling which excludes current Smartphones or PDAs. On the other hand we want the user to be able to hold the device and move around with it as freely as possible in order to explore new ways of controlling the virtual camera. This excludes approaches like the Boom Chameleon [TFK+02]. The Personal Interaction Panel [SG97] has many desirable qualities for our purpose, but we do not want to use head mounted displays, magnetic field pose trackers or other equipment which is expensive or complicated to install. Nevertheless, the Personal Interaction Panel has one advantage we incorporated in our design: The panel and pen themselves do not contain an active display or other potentially heavy components. They are just enhanced with small, magnetic, 6 degrees of freedom (DOF) tracking units, which makes them lightweight enough to prevent the user from unnecessary fatigue when holding them for an extended

Figure 1.1: Modeling session in FiberMesh, using the projection screen

period of time.

Inspired by the research work of Lee et al. [LHSD05, LHD07] as well as the Wiimote projects of Johnny Chung Lee [Lee08], we decided to use a hand-held projection screen as display device, while its pose(often referred to as screen pose in this thesis) is optically tracked. A picture of a user drawing in FiberMesh, using the projection screen can be seen in figure 1.1.

1.1 Contributions

Many studies deal with the various problems we encountered throughout the process of implementing our prototype. Yet there is, as far as we are aware, no system for exploring virtual 3D space that uses a hand-held projection screen. Therefore our contributions are not mainly algorithmic, instead, they are exploratory and include:

- **Hardware for a hand-held projection screen**. We present and discuss some hardware-configurations which have been tested throughout the process of creating the current prototype.
- **Software for tracking the screen pose**. Several subsystems are necessary to estimate the projection screen's pose. The algorithms of those subsystems are described and discussed. For some of the subsystems alternative algorithms are given as well.
- **An informal evaluation of different camera control metaphors**. Different ways of controlling the virtual camera that observes the modeled object in FiberMesh have been implemented. In addition to describing those approaches, we present results of an informal user study concerning the usability of the camera controls for 3D-modeling.

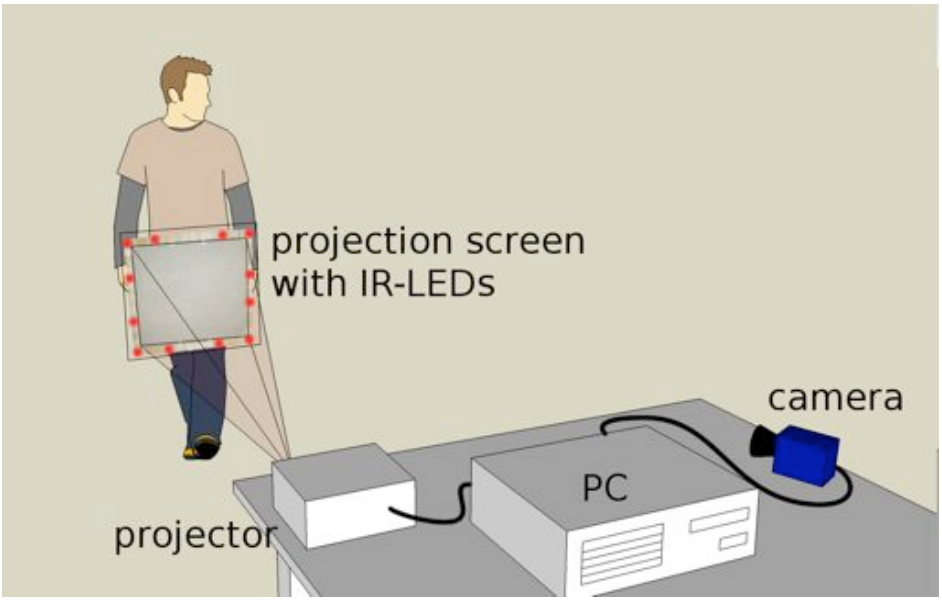


Figure 1.2: Schematic setup of the system's hardware

1.2 System Setup and General Procedure

A schematic overview of the involved hardware can be seen in figure 1.2. The projection screen's pose is tracked using IR-LEDs as visible features. For image acquisition, a camera with a reasonably high image capturing rate ($\geq$ 60Hz) and resolution ($\geq 640 * 480$), equipped with an IR-pass filter are utilized. The image is generated by an off-the-shelf projector and pre-warped in software in order to fit the projection screen. The projection surface of our hand-held projection screen consists of translucent paper, which yields an acceptable image quality for a prototype. Additionally, we designed a pen that allows the user to click, point and draw directly on the projection screen. Altogether the input components of the system have 8 DOF: The projection screen pose can be measured in 6 dimensions (3 translation, 3 orientation) and 2 additional dimensions are added with the pen input.

The first step in order to estimate the projection screen's pose is analyzing the camera's monochrome intensity image (see fig. 1.3, left), in which the positions of the projection screen's IR-LEDs are visible as bright spots. The **blob tracking subsystem** determines the barycenter for each of the bright spots in the image (green crosses in fig. 1.3, middle), by identifying connected regions of bright pixels.

In order to obtain the projection screen pose we need a **model of the projection screen**. This model consists of the IR-LEDs' positions on the screen frame (black crosses in fig. 1.3, right). The next step towards a pose estimate is estab-

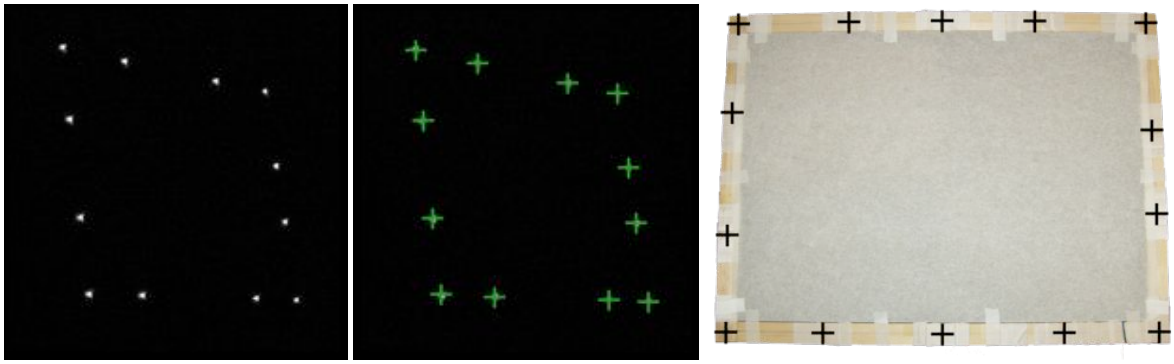


Figure 1.3: Left: Original camera image, showing IR-LEDs, Middle: Extracted image space coordinates of the bright blobs, Right: Projection screen model with known LED positions

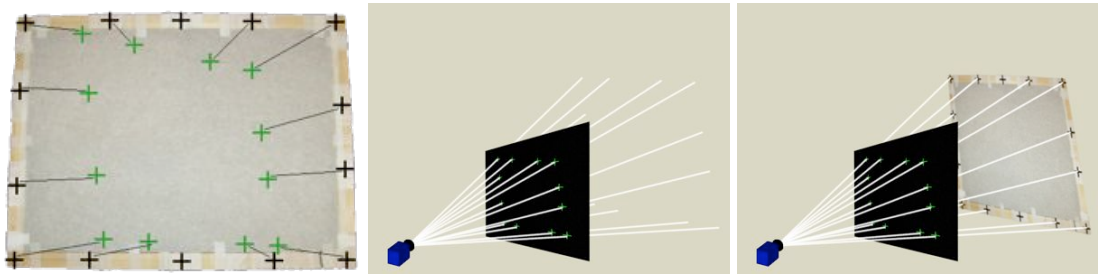


Figure 1.4: Left: Observed points are matched with the known LED positions, Middle: 3D rays obtained from the camera image, Right: Pose estimation from rays and model knowledge

lishing a point matching between the model points and the observed points. The **point matching subsystem** uses the circumstance that the LEDs on our projection screen are aligned in lines and some properties of the lines on the projection screen are still recognizable in image space. Therefore, lines can be utilized to establish the point matching. Figure 1.4, left depicts the projection screen model points (black crosses at the known positions of LEDs), the observed image points (green crosses) and the established matching between both sets of points (black lines).

As the projection screen is located in actual 3D space we need to relate the image space coordinates of seen points to positions in actual 3D space. This relation is given by the camera intrinsic parameters, which are amongst others containing measures to convert pixel distances to metric distances on the camera's image plane. Using the intrinsic camera parameters, we obtain a ray in 3D space (white lines in fig. 1.4, middle) for each point position in the image. The IR-LED from which the light blob in the image is originating lies on this ray. Combining the information contained in the projection screen model and the rays, we can estimate the pose of the projection screen in 3D space (see fig. 1.4, right). The **pose esti-**

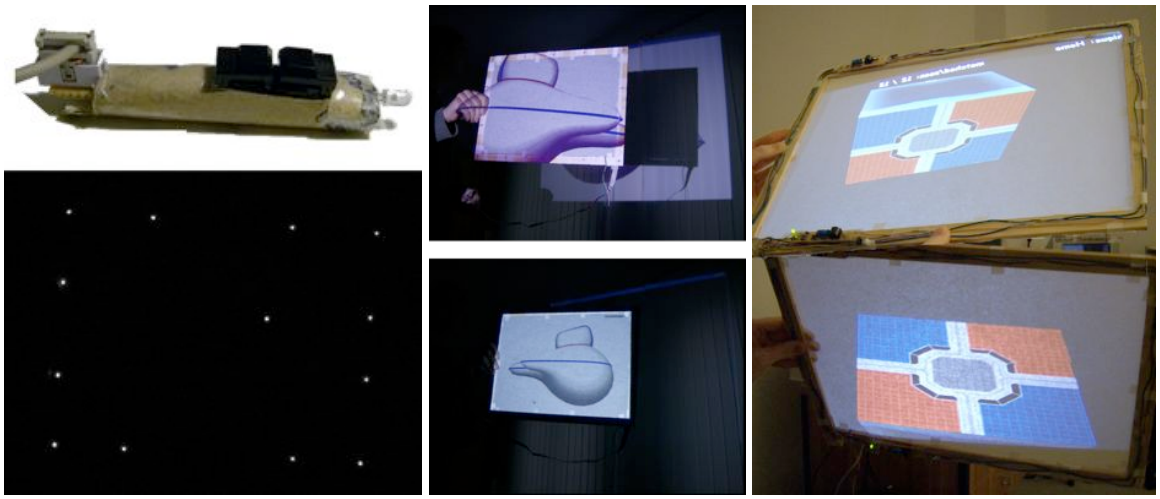


Figure 1.5: Left: Light pen and IR-image of the projection screen with pen, Middle: Projected image without and with pre-warping to fit the projection screen, Right: Controlling the virtual camera's pose by screen pose

mation module computes the intrinsic camera parameters as well as delivering a continuous pose estimation for the projection screen

The estimated pose of the projection screen is not the only input users can give to our system. The tip of the projection screen's pen is an IR-LED and the light emitted onto the screen surface from this LED can be seen in the captured image. See figure 1.5, left for an image of the pen and a camera image showing the projection screen with the pen pointing on its surface. Additionally the push buttons on the pen can be used to turn on the IR-LEDs at the centers of the horizontal projection screen frame. Therefore, the pen can replace the mouse when using the projection screen.

The visual output of the program that is controlled using the projection screen gets projected directly onto the screen. Therefore the projected image has to be pre-warped. This can only be done, if the spatial relationship between the projector and the camera is known. Figure 1.5, middle shows an object modeled in FiberMesh as it is normally projected (top) and in a pre-warped version that exactly fits the projection screen (bottom).

In 3D applications the pose of the projection screen can be used to control the (virtual) camera through that the user views the 3D space of the application. In figure 1.5, right two different views on the same object can be seen. Metaphors for the camera control method depicted there are "eyeball in hand" [WO90] or "camcorder for the virtual environment".

1.3 Outline

In chapter 2 we discuss the subsystems leading to the pose estimation of the projection screen: Section 2.1 presents the blob tracking subsystem that extracts point positions from the image. In section 2.2, the algorithms for matching those extracted points with the known model points on the projection screen frame are discussed. The actual pose estimation process is presented and analyzed in section 2.3. Although the cursor position does not belong to the projection screen's actual pose, it will be discussed in section 2.4, as it is part of the input obtained by analyzing the camera image.

Our method of pre-warping the projected image, which is necessary to project the image right onto the projection screen is presented in chapter 3. Algorithms for determining the projectors pose relative to the camera and for pre-warping the image are given.

Chapter 4 deals with implementation issues and the technical details of the system's hardware. The implementation issues are at the one hand arising from runtime checks performed by C# when accessing arrays and on the other hand related to interoperation with software written in C and C++. After laying out technical details of the system's hardware, parameters (e.g. the effective resolution on the projection screen) that are intrinsic to the utilized hardware are analyzed.

Interaction with by our system is discussed in chapter 5. On the one hand, usability issues arising from technical properties of the system are discussed. The first of those issues is communicating the boundaries of the system's working space to the user. Those boundaries are given by the viewing pyramid of the camera and the beam of the projector. The second issue is filtering noise that is present in the screen pose estimate. On the other hand, methods for controlling the virtual camera's pose through the projection screen's pose are discussed and ways to use FiberMesh without keyboard input are presented. Finally, we analyze the results of an informal user study.

A short summary of the thesis is given in chapter 6. Additionally we examine problems that are still open as well as providing a perspective on future work.

1.4 Nomenclature

Before dealing with our implementation, we need to define a nomenclature to discuss them conveniently. Preceding the nomenclature, we want to point out that we will (not completely correct) handle vectors and points interchangeably throughout this thesis as this oftentimes simplifies expressing and therefore comprehending facts involving them.

Assume the camera observed n points. The positions of those points in the camera image are **the image coordinates** $CI = \{i_k = (i_x^k, i_y^k)^T \in \mathbb{R}^2 | k \in O\}$ of the observed points (green crosses in fig. 1.6), where O is the **set of observed point**

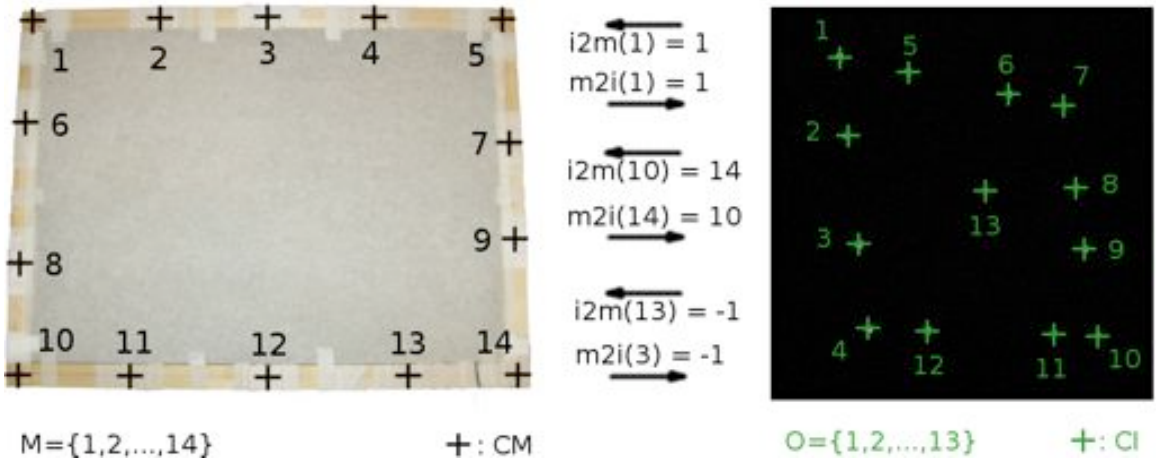


Figure 1.6: Left: Model point IDs M and model point coordinates CM, Middle: Some values of the matching functions $i2m$ and $m2i$, Right: Image point IDs O and image point coordinates CI

IDs: $O = \{0,..,n-1\} \subset \mathbb{N}$ (green numbers in fig. 1.6). Note that we will use the index k to relate image points to their ID. For each image every observed point has a unique ID, by which it is identified. The observed points will as well be called image points throughout the thesis. Image coordinates of the seen points are obtained by the blob extraction module.

In addition to the image coordinates, we have m model points. Each of those model points is uniquely identified by one of the **model point IDs** contained in $M = \{0,..,m-1\} \subset \mathbb{N}$ (black numbers in fig. 1.6). The model points are marking the **coordinates of the IR-LEDs** $CM = \{(m_x^l, m_y^l, m_z^l)^T \in \mathbb{R}^3 | l \in M\}$ on the untransformed projection screen model (black crosses in fig. 1.6). Note that we will use the index l to relate model points to their ID. The locations of those points will oftentimes be called **model coordinates** as they represent our computational model of the projection screen. They are obtained by manual measurement of the projection screen.

The **point matching** is depicted as lines, connecting the observed points (green) and the model points (black) in figure 1.4 on page 4, left. It consists of 2 functions: One **function relating observed points to model points** $i2m : O \to M \cup \{-1\}$ and one **relating model points to observed points** $m2i : M \to O \cup \{-1\}$. $i2m$ returns the model point ID that is associated to the given image point ID and $m2i$ returns the associated image point ID for a given model point ID. Both functions may return -1, if there is no matching point in the other set or the matching point is unknown. Whenever one of those two functions is set, the other one has to be set as well: $i2m(k) \geq 0 \vee m2i(l) \geq 0 \Rightarrow i2m(k) \geq 0 \wedge m2i(l) \geq 0$. Additionally $i2m$ and $m2i$ are inverse to each other, as they represent a matching: $i2m(k) = l \neq -1 \Leftrightarrow m2i(l) = k \neq -1$. Some values of the point matching functions are denoted in figure 1.6, middle. The point matching is built and maintained by the point

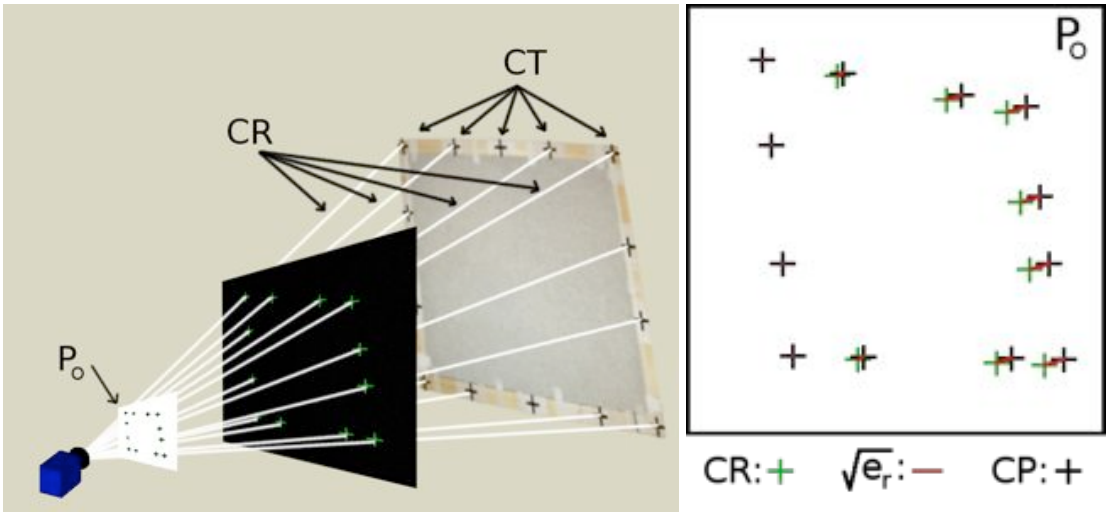


Figure 1.7: Left: The metric image plane P_O, transformed model points PT and rays implied by CR, Right: Points CR that define image rays, re-projections CP of model points and re-projection errors e_r on P_O

matching subsystem.

Knowing the point matching, we can define the **set of matched model IDs** $MM = \{m \in M \,|\, m2i(m) \geq 0\}$ containing only those model point IDs for which a corresponding observed point is known. Similarly, we define the set of **matched observed IDs** $MO = \{o \in O \,|\, i2m(o) \geq 0\}$.

If we have information about the intrinsic camera parameters A, we can relate image points to points on the **metric image plane** $P_O = \{(p_x, p_y, 1) \in \mathbb{R}^3\}$ (the plane is depicted in fig. 1.7, left)**.** Doing so yields homogeneous coordinates $CR = \{(r_x^k, r_y^k, 1)^T \in P_O | k \in O\}$ (green crosses in fig. 1.7, right) **that define rays** from the camera for each image point (see fig. 1.7, left). Originating from the optical center of the camera (being the center of our world coordinate system for convenience) those rays point to $(r_x^k, r_y^k, 1)^T$ in 3D space. Note the fact that naturally the observed points will not lie behind the camera, which can serve as a constraint for pose estimation. The intrinsic camera parameters can be described by a matrix, containing horizontal and vertical scale factors α and β, the skew of the camera image c and the pixel coordinates of the focal point $(u_0, v_0)^T$. A point $m = (u, v)^T$ in pixel coordinates can be related to the according ray $(x, y, 1)$ by A, as follows:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = A \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \quad \text{where } A = \begin{bmatrix} \alpha & c & u_o \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

One output of the pose estimation process are the estimated coordinates of **transformed model points** $CT = \{(t_x^l, t_y^l, t_z^l)^T \in \mathbb{R}^3 | l \in M\}$ (marked in fig. 1.7, left). This is our estimate of the real world position of the observed IR-LEDs. The

transformed model points in CT can be related to image space coordinates through projecting them on the metric image plane P_o. This yields the **re-projections of transformed model points** $CP = \{\mathbf{p_l} \in \mathbb{R}^3 \,|\, \mathbf{p_l} = (p_x^l, p_y^l, 1)^T = (t_x^l/t_z^l, t_y^l/t_z^l, 1)^T\}$ (black crosses in fig. 1.7, right).

If we have a point matching we can additionally define subsets of anything related to model-points. All those subsets only contain those model-points that have matching observed points. We will indicate those sets by an over-lining hat: For example we have $\hat{CM} = \{m^l \in CM \,|\, l \in MM\}$**, the coordinates of matched model points.** The **transformed matched model points** $\hat{CT}$ and the **re-projections of matched model points** $\hat{CP}$ are defined similarly. In the same manner we can define the **image coordinates of matched observed points** $\hat{CI} = \{i^k \in CI \,|\, k \in MO\}$**,** as well as the **rays** $\hat{CR} = \{r^k \in CR \,|\, k \in MO\}$ **of the matched observed points**.

A very important concept for assessing the quality of a screen pose estimate, is the **re-projection error** $e_r \in \mathbb{R}$. Assume we have a re-projection $(p_x^l, p_y^l, 1)^T \in \hat{CP}$ of a transformed point $c \in \hat{CT}$ and the metric plane location $(r_x^k, r_y^k, 1)^T \in \hat{CR}$ of the related ($k = m2i(l)$) image point. The re-projection error is calculated as the squared distance between those two points on the metric image plane: $e_r^l = (r_x^k - p_x^l)^2 + (r_y^k - p_y^l)^2$ (red lines in fig. 1.7, right).

Chapter 2

Continuous Visual Pose Estimation

This chapter details the algorithms we use for pose tracking, including the preprocessing steps in image space and estimation of the pen's position on the projection screen as an additional source of input.

2.1 Blob Tracking

This section deals with our approach to locate the IR-LEDs' positions $CI \in P(\mathbb{R}^2)$ in the camera image. The positions of the IR-LEDs in images can be easily identified via their brightness (see 2.1, left). As an IR-pass filter is installed in front of the camera lens, almost no environmental light is captured by the camera, making the IR-LEDs easily distinguishable from the background in the obtained images.

2.1.1 Algorithm

A simpleapproach for locating bright spots in an image is the blob extraction algorithm which is described by Horn in [Hor86]. We adapted this approach to meet the needs of our environment. The adapted algorithm is divided into 2 phases: Trying to relocate known blobs and searching for new blobs.

Assume we have an image with intensities $i_{x,y}, x \in [0,..,639], y \in [0,..,479], i_{x,y} \in [0,..,255]$. These values are the ones used in our current approach. Of course image resolution and intensity depth can be changed. Furthermore, we have an intensity threshold $t \in [0,..,255]$. Points with an intensity of t or above will be considered active, i.e. belonging to a blob. Our last parameters are the detection radius r_d and the re-detection radius r_r. They will be used to stop searching the vicinity of an interesting pixel for a blob in order to limit the execution time of the algorithm.

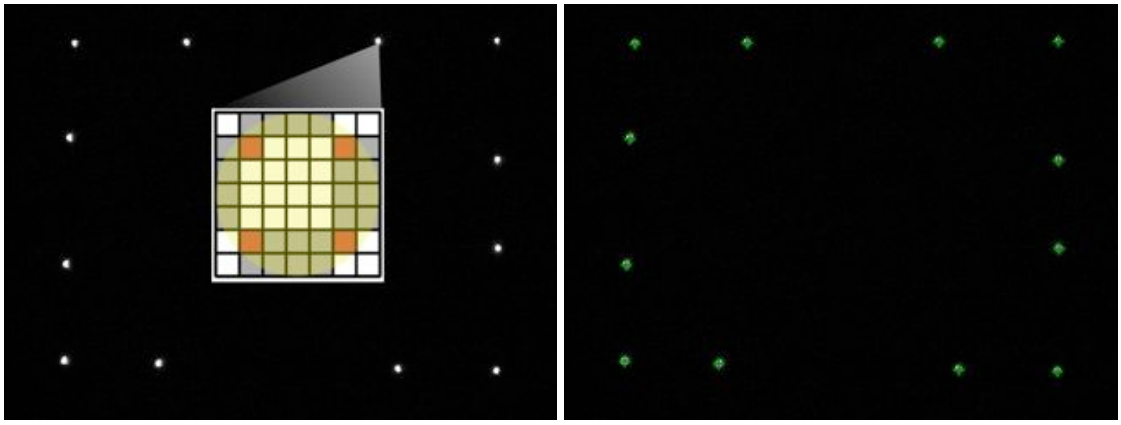


Figure 2.1: Left: Original gray-scale image of the projection screen with schema of the scanning pattern ($s_c = s_r = 4$) showing scanned pixels (red), skipped pixels (white/gray) and a light blob (yellow), Right: Image with extracted blob positions

The main idea of this algorithm is not to scan each pixel for being active. This can be justified using the assumption, that blobs have a certain size and are therefore detectable within an area and not just on one pixel (see fig. 2.1, left). The parameters s_c and s_r determine how many columns or rows will be skipped after each scanned pixel. Obviously the amount of scanned pixels decreases quadratically with uniform increments in s_c and s_r. If we assume that the diameter of a blob is at least 7 pixels, we can ensure to detect all blobs although not scanning 15 out of 16 pixels.

As we do not exhaustively scan the image, we need a separate method to include all bright pixels in a blob. Therefore we need two helper functions: The first one, `search_blob`, gets a pixel $(x,y)^T \in \mathbb{Z}^2$ as input, and returns all active pixels of the blob that is closest to $(x,y)^T$. This algorithm has two stages: In the first stage it exhaustively searches the vicinity of the input pixel for an active pixel $(i_{x,y} \geq t)$. This is done by growing a quadratic region around the starting position. In each growing step, the new edges of the square are scanned for active pixels. If after r search steps no active pixel is found, the region around the input point contains no active pixels. If an active pixel $s = (x_s, y_s)^T \in \mathbb{Z}^2$ is found in one of the growing steps, the algorithm proceeds to its second phase. Note that the algorithm will entirely skip this phase if the input pixel is active.

The second phase of the blob search algorithm grows a region around the starting pixel $s = (x_s, y_s)^T$, as long as it can find new active pixels. The approach is similar to the one in phase one, but it allows the grown region to be rectangular in order to better adapt to the shape of a blob. Each edge of the rectangle is individually tested for active pixels and if it contains an active pixel, the edge is moved one step away from s. Within each iteration all 4 edges of the rectangle are tested for active pixels. The iterations are stopped if none of the 4 edges contains an active pixel and thus the rectangle can not be grown anymore. Note that in contrast to blobs extracted with the algorithm described in [Hor86], our blobs may contain

pixels that have no neighboring active pixels and thus are not directly connected to the blob. The whole procedure is specified in algorithm 1.

Algorithm 1 search_blob

Input: a starting position $(x,y)^T \in \mathbb{Z}^2$ and a search radius $r \in \mathbb{Z}$.
Output: A set $A \in P(\mathbb{Z}^2)$, containing the set of active connected pixels that is closest to $(x,y)^T$.

```
//search for initial active pixel
FOR d = 0 TO r − 1 DO //grow a quadratic region
  IF the set of possible starting pixels is not empty
    ( S = {(x_p, y_p)^T ∈ ℕ^2 | p_{x_p,y_p} ≥ t ∧
      ( (x_p = x ± r ∧ y_p ≤ y + r ∧ y_p ≥ y − r) ∨
        (y_p = y ± r ∧ x_p ≤ x + r ∧ x_p ≥ x − r)) } )
  THEN
    Choose s := (x_s, y_s) ∈ S
IF no active starting point found (S = ∅) THEN
  return ∅
Initialize the active pixels of the blob:  A := {s}.
Initialize search distances for left, right, top and bottom
border:
  d_l := d_r := d_t := d_b := 1.
Define active pixel sets to detect the borders of our blob:
  B_l := {(x_l, y_l) | p_{x_l,y_l} ≥ t ∧ x_l = x_s − d_l ∧ y_l ≤ y_s + d_b ∧ y_l ≥ y_s − d_t},
  B_r := {(x_r, y_r) | p_{x_r,y_r} ≥ t ∧ x_r = x_s + d_r ∧ y_r ≤ y_s + d_b ∧ y_r ≥ y_s − d_t},
  B_t := {(x_t, y_t) | p_{x_t,y_t} ≥ t ∧ y_t = y_t − d_t ∧ x_t ≤ x_s + d_r ∧ x_t ≥ x_s − d_l} and
  B_b := {(x_b, y_b) | p_{x_b,y_b} ≥ t ∧ y_b = y_b + d_b ∧ x_b ≤ x_s + d_r ∧ x_b ≥ x_s − d_l}
WHILE B_l ≠ ∅ ∨ B_r ≠ ∅ ∨ B_t ≠ ∅ ∨ B_b ≠ ∅ DO
  FOR EACH border B ∈ {B_l, B_r, B_t, B_b} DO
    IF B ≠ ∅ THEN
      Add B to the set of blob pixels:  A := A ∪ B
      Increase the associated distance by 1
        (d_l for B_l, d_t for B_t etc.)
      Recompute B with the new distance
Return the active pixels A of the blob
```

The second helper routine (alg. 2) is called `center` and simply computes a weighted barycenter of a set of pixels. The weight for each pixel $(x,y)^T \in \mathbb{Z}^2$ is the pixel's intensity $i_{x,y}$.

The main algorithm of our blob extraction builds a set $B \in P(P(\mathbb{Z}^2))$ of blobs, each described by the set of its active pixels. This set is initially empty. The algorithm returns the centers $C \in P(\mathbb{R}^2)$ of those blobs. We will need a row offset o_r and a column offset o_c that are both initially 0 and will be remembered for subsequent calls of the algorithm.

Algorithm 2 center

Input: A nonempty set $A \in P(\mathbb{Z}^2)$ of active pixels
Output: The center $(c_x, c_y)^T \in \mathbb{R}^2$ of A, weighted by the intensities of pixels

`Iterate over` A `to obtain` $(c_x, c_y)^T := \frac{\sum_{(a_x,a_y)\in A} (p_{a_x,a_y} \cdot (a_x,a_y))}{\sum_{(a_x,a_y)\in A} p_{a_x,a_y}}$.

`Return` $(c_x, c_y)^T$.

The first step of this algorithm uses the positions of known blobs from the last image to relocate them. It simply calls `search_blob` with the last position of the blob as center and with the re-detection radius. This ensures that a blob that is once detected will not be lost again, as long as it stays within the re-detection radius. Without this step small dots would frequently fall through the raster implied by s_c and s_r and would therefore not be reliably detected in the image.

Secondly, the blob detection algorithm searches for new blobs. Therefore, in every s_r-th row every s_c-th pixel is examined for being active. If an active pixel is found, the blob around it is determined by calling `search_blob`. If the new-found blob does not overlap with a known blob, it is added to the set of known blobs.

The last step is closely related to searching new blobs. It moves the pattern of scanned points over the image, by changing the offsets o_c and o_r that are added to the scanned column and row indices. The movement has its effect when scanning the next image. Without it, small blobs that remain in a fixed position might never be recognized by the blob extraction algorithm. The current approach is simply to move the offset so that it covers each row from left to right and jumps to the next row, once s_c is reached.

For easier understanding, algorithm 3 describes the check for overlapping blobs as a pixel by pixel check. In practice, this can be done by building and saving rectangular boundaries in `search_blob` and checking those rectangles for overlap. This is computationally much cheaper than testing against each active pixel of the other blobs.

Algorithm 3 gives a possible implementation of this ideas.

2.1.2 Discussion

An alternative to the presented blob extraction algorithm would be using the Wii Remote, mostly referred to as Wiimote. This device is able to track up to 4 IR-blobs simultaneously at a rate of 100Hz. The high update rate is a clear advantage, but the restricted amount of trackable points completely outweighs this advantage, as analyzed in the discussion of pose estimation algorithms (see sec. 2.3.4).

Other existing approaches are based on the framework that is called scale-space

Algorithm 3 blob_extraction

```
//Relocate known blobs
Save the last images blobs:  B_o := B.
  Clear current images blobs:  B := ∅
  FOR EACH b_o ∈ B_o
    Find according blob in new image:  b_n = search_blob(center(b_o), r_r)
    IF blob found (b_n ≠ ∅) AND
      no pixel of b_n is included in another new blob
      ({(x,y) ∈ ℤ² | (x,y) ∈ p_n ∧ ∃b ∈ B : (x,y) ∈ b} = ∅)
    THEN
      Add b_n to the blobs:  B := B ∪ {b_n}.
//Find new blobs
FOR r = o_r TO 480 − 1 INCREMENT s_y DO
  FOR c = o_c TO 640 − 1 INCREMENT s_x DO
    IF pixel intensity exceeds threshold (p_{c,r} > t) THEN
      Find blob:  b_n = search_blob((c,r)^T, r_d)
      IF no pixel of b_n is included in an already known blob
        ({(x,y) ∈ ℤ² | (x,y) ∈ b_n ∧ ∃b ∈ B : (x,y) ∈ p} = ∅)
      THEN
        Add b_n to the blobs:  B := B ∪ {b_n}
//move search pattern
  Increment column offset:  o_c = o_c + 1
  IF column offset reaches column step-width (o_c ≥ s_c) THEN
    Reset column offset:  o_c = 0
    Increase row offset, respecting step-width:
      o_r = (o_r + 1) mod (s_r)
```

(see [Lin94] for an overview), which is the basis for some powerful image analysis tools. Examples for this include blob detection [Lin93] or feature detection and matching across multiple images [Low04]. Due to the simple nature of our intensity image (nearly black except for the positions of the IR-LEDs) and to limit the execution time, we decided against using approaches based on scale-space transformation of the image, as this transformation would need a full scan of all pixels.

We chose to adapt the blob extraction algorithm presented by Horn in [Hor86] for its simplicity and, with that, low time complexity. Additionally, taking into account that the light blobs of our IR-LEDs have a certain diameter, we can save computation time by not scanning every single pixel. For very small blobs, the algorithm may need some iterations until an active pixel of the blob is found, but this does not cause any problems for our pose tracking algorithms. The time consumption of `search_blob` is quadratic in the size of blobs. If the image contains many big blobs, this would, at some point, undermine the idea of not scanning every pixel. But as the amount of blobs and their size is relatively small in our scenario, this does not apply.

2.2 Point Matching

This section deals with the problem of matching seen point IDs $s \in O$ to model point IDs $m \in M$ (see sec. 1.4 for definitions). The output of this algorithm are the two functions matching model points and observed points: $m2i : M \to O \cup \{-1\}$ and $i2m : O \to M \cup \{-1\}$. There are two different sub tasks that need to be solved in order to build those matchings: The first is to initialize the matching functions for an image without prior knowledge besides the model of our projection screen. The second task is easier, as additional information is available: Once an initial matching has been obtained, knowledge about the last image's point matching can be used to establish a frame to frame correspondence in the matching. The following sections present the different algorithms used for establishing the point matching functions.

2.2.1 Brute Force Initialization

This algorithm first determines the set $U_o = \{o \in O \,|\, i2m(o) \leq 0\}$ of unmatched seen points and the set $U_m = \{m \in M \,|\, m2i(m) \leq 0\}$ of unmatched model points. Afterwards, all $v = \frac{|U_m|!}{(|U_m|-|U_o|)!}$ variations of assigning unmatched model points to unmatched seen points are built and checked for the existence of a valid pose. The matchings with a valid pose are kept for checking against the next available image. For each "surviving" matching, a frame to frame point matching is done and the process is repeated until only one candidate remains. As we have to maintain the matching for subsequent camera images, a strong frame to frame similarity in the subsequent images used for the initialization is necessary. In order to keep this similarity high enough, we only examine a limited number of potential matches every frame. Doing so keeps the required computation time between two frames short enough to maintain the matchings.

The frame to frame matching for all the potential matchings can be done symbolically with one of the matchings and can then be applied to all other potential matchings. This reduces the time needed for maintaining frame to frame matchings, but still includes the number of variations $O(v)$ as the symbolic matching has to be applied to each remaining matching.

2.2.2 Line Section Ratio Initialization

If we disregard radial distortion of the image caused by the camera optics as well as the effects caused by perspective projection, the transformations that are involved in forming the image of our projection screen (translation and rotation) are affine transformations. Under affine transformation, collinear LEDs on the projection screen will yield collinear points in the image. Additionally, the ratios of distances along a line will be preserved.

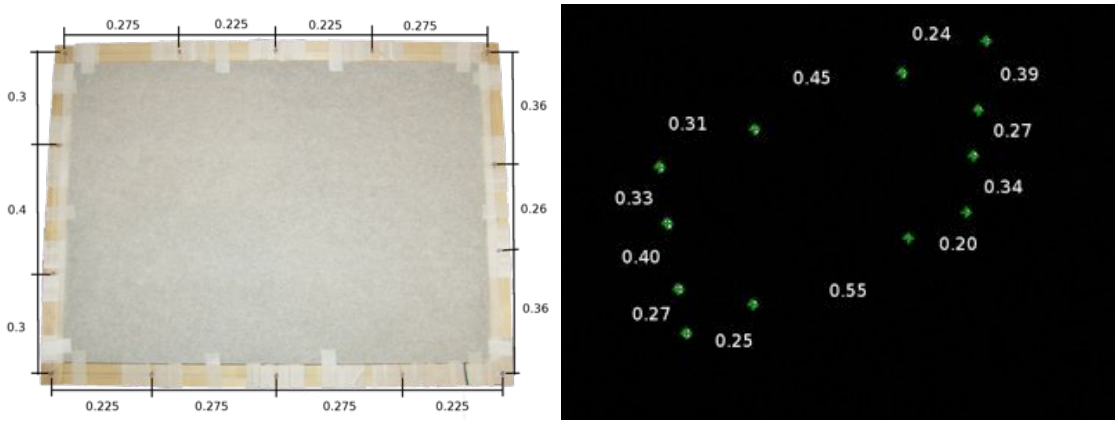


Figure 2.2: Left: Line ratios of the model, Right: Line ratios in the image

The effects of radial distortion of the camera image are in most cases small enough to be neglected. Perspective projection ,of course, can not be neglected that easily in general. However, we found that the properties of our problem favor the neglection of the effects of perspective projection. As our projection screen is a rigid object with fixed size, there is a maximal difference in z-components of the different points on its frame. Additionally, the frame needs to have a certain distance from the camera in order to be seen completely. Hence, the ratio between difference in two point's z-coordinates and the distance of those points to the camera favors the neglection of perspective's effects in the image. The neglection is further supported by the fact that many researchers use weak perspective, an affine approximation of real perspective, for their analysis of 3D object's images [BWR93].

The positioning of the IR-LEDs on the projection screen frame is chosen in order to support the initialization process: We placed 5 LEDs along the horizontal borders of the frame and 4 LEDs along the shorter vertical borders. This allows the initialization algorithm to distinguish horizontal from vertical borders if all LEDs are visible. To distinguish the top from the bottom border and the left one from the right one, we altered the positions of 2 LEDs on each line. The LEDs at the top and right border of the screen are positioned closer to each other and the ones on the other two borders are a bit farther apart. With the different LED counts and ratios of the line section lengths we can now uniquely identify each border by the line of LEDs on it (see fig. 2.2, left for the distance ratios on the projection screen).

The steps for building a point matching with this approach are as follows: First, we extract collinear points from the set of model points and from the set of seen points. From those lines, we extract sequences of line section ratios, which are then compared to identify pairs of matching lines containing one image line and one model line. Finally the matching lines need to be analyzed in order to obtain a matching between observed and model points.

The dynamic programming algorithm (alg. 4) for extracting lines from a set of points first builds all lines of length 2 by creating a line from each pair of distinct points. Lines are in this case represented by a set of points. Next, the algorithm iterates over all lines generated in the last step and tries to add each point that is not yet in the line. A point $\mathbf{p}$ will be considered as lying on a line L (described by two of its points $\mathbf{s} \in L$ and $\mathbf{e} \in L$), if the distance d (see appendix A for distance calculation) of $\mathbf{p}$ from L (point $\mathbf{s}$, direction $\mathbf{e}-\mathbf{s}$) is small enough ($d < \varepsilon$, where ε is a small threshold). If $\mathbf{p}$ is on L, then a new line consisting of $L \cup \{\mathbf{p}\}$ is generated. Generating a new line is necessary, as there may be different points that can be added to L. All newly generated lines will be tested for the possible inclusion of additional points. Therefore the procedure is repeated until no new lines can be found.

Algorithm 4 lines from points

Input: A set of points $P \in P(\mathbb{R}^2)$
Output: A set of lines $R \in P(P(\mathbb{R}^2))$

```
Initialize the empty set of lines:  R := ∅
Find all lines with 2 points:
  currentLines = P × P \ {(x,y)^T ∈ P × P | x = y}
WHILE new lines found in the last step (currentLines ≠ ∅) DO
  Initialize set of new found lines:  newFound := ∅
  FOR EACH line with current length (L ∈ currentLines) DO
    FOR EACH point that is not yet in L (p ∈ P \ L) DO
      Let d be the distance of p from the line L
      IF p is on L (d < ε) THEN
        Add the new line to newFound:
          newFound := newFound ∪ {L ∪ {p}}
  Add all new lines to the set of found lines:
    R := R ∪ newFound
  Prepare for the next iteration:  currentLines := newFound
Return the set of found lines R.
```

Algorithm 4detects all lines and sub lines with at least 3 points. If the detection of sub lines is not desirable, the addition of *newFound* can be delayed by one iteration. All lines expanded to contain an additional point can be removed (from *newFound*) this way before adding them to L. When implementing the algorithm, special care has to be taken in order to assure *newFound* implements the behaviour of a set. Otherwise, *newFound* would grow exponentially as lines are found more than once in each iteration. ($\{\mathbf{a},\mathbf{b},\mathbf{c}\}$ is found 3 times with $\{\mathbf{a},\mathbf{b}\}$, $\{\mathbf{a},\mathbf{c}\}$ and $\{\mathbf{b},\mathbf{c}\}$ as predecessors). If *newFound* can ensure (in $O(1)$), that no set of points can be added more than once, the algorithm's time complexity is $O(|P|^3)$, where P is the set of input points. Without ensuring that *newFound* is a set, the time complexity of the algorithm would be $O(|P|^{|P|})$.

After running algorithm 4 with removal of sub lines on the model points and on the seen points, we have two sets of lines. The lines contained in those sets are described by a set of points. For matching lines of the two sets through line ratios, the lines have to be preprocessed. This is done by first ordering the points by their position on the line and then extracting length ratios of the line sections. After doing so, we have data that is comparable due to the assumed affinity of the transformation. In figure 2.2, right the set of line ratios are shown for a real image. Obviously the observed line ratios do not match the model line ratios exactly due to the effects of perspective projection. Nonetheless they are sufficiently similar to be matched against each other.

Now lines in the image have to be matched with lines in the model. This is done by repeatedly iterating over both sets of lines and choosing the best match (by sum of squared differences in the line length ratios) for each complete iteration over both sets. When comparing an observed line and a model line, both possible directions for one of the lines have to be regarded as the order of the line sections might be reverse to each other. After extracting the line matches, we use the intersections of our lines to identify the correct order of points. Finally, we extract initial point matches from this information. This algorithm's time complexity can be estimated by $O(\max(|L_m|, |L_O|)^3 \cdot |M|)$, where L_m and L_O are the amounts of lines in the model and in the image. As the amount of lines is limited by the amount of points in this case, we can write this complexity as $O(\max(|O|, |M|)^4)$.

One problem of this approach occuring frequently are turned off or obstructed LEDs. Therefore, we need to consider sub lines of the model as well. Our implementation tries to match against all sub lines of model lines with exactly one point removed. This is solely done if there is an unmatched line in the image with an appropriate point count. In figure 2.2 the test for sub lines needs to be done for the horizontal borders of the projection screen, as the middle LEDs on them are not visible in the image.

Another issue of this algorithm that needs to be addressed is the detection of false matches. There are two helper systems in place to make wrong matchings less likely: The first one can only be applied if we already have a matching. This is the case if we run the initialization to add matchings for a few unmatched points. Those points can be unmatched because their match was lost in frame to frame matching or they were discovered recently. In this case we can check the new matching against the old one and reject it if one of the previously matched points is re-matched differently. The second approach to detect wrong matchings needs the pose estimation to be run after initializing the matching: If the point matching is wrong, the pose estimation will yield wrong results. This will lead to a high re-projection error (see sec. 1.4). Thus, a new initialization of the matching is started if the re-projection error is to high.

2.2.3 Frame to Frame Matching

This mechanism is used to carry an existing point matching forward from the last image to the current image. The underlying algorithm is a simple similarity check between the last image's point positions and the current ones. Assume we have the set of image coordinates $\overline{CI} \subset \mathbb{R}^2$ of the seen points in the last image. Additionally, we have a function $p2m : \overline{CI} \rightarrow M$ that relates the seen points of the last image to model point IDs. Furthermore, we have the current seen points CI and a lower bound d_{max}. This lower bound determines the the maximum difference between the last image's position of a point and the current image's one that can yield a match.

Algorithm 5 describes the procedure of maintaining the point matching from image to image. It simply iterates over all pairs $(\mathbf{po}, \mathbf{pc_k}) \in \overline{CI} \times CI$ of old point positions and new image point IDs that are both not matched $(m2i(p2m(\mathbf{po})) = -1 \wedge i2m(k) = -1)$ in order to determine the pair with the lowest distance $d_{min} = |\mathbf{po} - \mathbf{pc_k}|$. If the best distance is small enough $d_{min} \leq d_{max}$, the pair is matched. This procedure is iterated $|\overline{CI}|$ times in order to find all potential matches.

The upper bound d_{max} is computed as a fraction of the minimal distance between any two points in the current image. Its purpose is to prevent the algorithm from matching two points that are in no relation to each other. If d_{max} was not in place, the last few iterations of the matching algorithm would frequently match points that are not even close to each other.

Our implementation of algorithm 5 additionally stores the position of previously matched points which could not be found in the current image, making re-detection of blinking or obstructed LEDs possible.

2.2.4 Re-Projection Matching

This algorithm needs an existing matching in order to work. Its purpose is to match those points which could not be matched using frame to frame similarities. This can either be points that have been invisible before and therefore have no previous matching or points that moved too fast to be re-matched. The approach to match those points is simple: The projection screen pose is estimated, using the points that could be matched by other methods. Afterwards, the points of the estimated pose are re-projected onto the metric image plane (as described in sec. 1.4), yielding a point $(p_x^l, p_y^l, 1)^T \in CP$. If the metric plane point $(r_x^k, r_y^k, 1) \in CR$ of an unmatched $(i2m(k) < 0)$ observed point is close enough to an unmatched $(m2i(l) < 0)$ reprojected point, the algorithm matches those two points $(i2m(k) := l,\ m2i(l) := k)$. The implementation is similar to the frame to frame matching and iterates over all pairs $(\mathbf{p_l}, \mathbf{r_k}) \in CP \times CR$ of reprojected points and rays of observed points. For each full iteration the best pair of points is matched if the distance between $\mathbf{p_l}$ and $\mathbf{r_k}$ is small enough. If no new match is found, iterating is stopped.

Algorithm 5 frame to frame matching

```
Clear the matching functions:
  ∀m ∈ M : m2i(m) := −1
  ∀i ∈ O : i2m(i) = −1 .
//Try to find a match for each seen point
FOR i := 0 TO |CI| − 1 DO
  Initialize threshold:  d_best := d_max
  Initialize best point in last image  l_best := ∅
  Initialize best point in current image:  c_best := ∅
  FOR EACH po ∈ OI DO
    FOR EACH pc_k = (i_x^k, i_y^k) ∈ CI DO
      IF po is not yet matched (m2i(p2m(po)) ≤ 0) AND
        current point is not yet matched (i2m(k) ≤ 0) AND
        distance is best (d = |p_l − p_c^k| < d_best)
      THEN
        Store new best last image point  l_best := {po}
        Store new best current image point  c_best := {pc_k}
        Store new best distance  d_best := d
  IF a match has been found (l_best ≠ ∅) THEN
    Update model to image matching:  m2i(p2m(po)) := k
    Update image to model matching:  i2m(k) := p2m(po)
```

2.2.5 Discussion

At first we review the brute force matching initialization: Assume we can examine 10 possible matches per frame (which needs computing 10 pose estimates) without loosing any point matches and keeping up a frame rate of 60Hz. Therefore, we could examine all possibilities for 5 unmatched points ($|U_o| = |U_m| = 5$, where $|U_o|$ and $|U_m|$ are the number of unmatched observed points and unmatched model points) within 0.2 seconds. For $|U_o| = |U_m| = 6$ this would already need 1s and for $|U_o| = |U_m| = 7$ even 7s. Remember that these are just the times for the first iteration over all possible matchings. We would need to re-examine all candidates that yielded a valid pose within the first step and repeat the procedure until we have found a single survivor of this process. As the initialization process has to be executed for initially matching model and observed points only, it is acceptable to consume some time. It may even be acceptable if this time is long enough to be noticed by the user of the system. But times exceeding a second (yielded by $|U_o| = |U_m| > 6$) are not acceptable. In addition to the long time for a single iteration over the initial candidates, the rate of success for the full initialization process is not 100% as sometimes no candidate survives the process. As the overall time complexity of this algorithm even exceeds the number of possible matches ($\frac{|U_m|!}{(|U_m|-|U_o|)!}$), this algorithm is not suitable for the usage in our scenario, as we want to use as many feature points as possible in order to obtain a good pose

estimation.

The line section ratio approach to the initialization of the point matching most notably scales better than the brute force approach. Directly using knowledge about the projection screen model, in terms of lines of LEDs, this algorithm can achieve a time complexity of $O(\max(|O|,|M|)^4)$. One constraint given by this algorithm is that the points need to be placed along lines. A second constraint is that the weak perspective assumption will not work for scenarios where the distance between feature points is close to the distance from the camera. As our projection screen naturally meets those constraints, this algorithm for initializing the point matching is the one we are using.

The frame to frame point matching is a simple method of maintaining a point matching over subsequent frames. The only constraint it imposes is that the stream of images on which the matching should be carried forward needs strong frame to frame similarities. The algorithm's time complexity is $O(\max(|O|,|M|)^3)$ in the presented form and it works reliably. Although the line ratio matching algorithm could be used for all occurrences of the point matching problem, the frame to frame matching is a faster and more stable algorithm. Therefore, it is preferred as long as a previous matching is known.

The frame to frame point matching algorithm can obviously only match those observed points, which have been matched to model points in the previous image. This means that if an additional point becomes visible or a point's matching to the model is lost, this point can not be matched to a model point again using the frame to frame matching. Therefore, we supplement it with the re-projection matching which can match previously unmatched points when a pose estimate and with it re-projections of the model points are given. The time complexity for this is again $O(\max(|O|,|M|)^3)$.

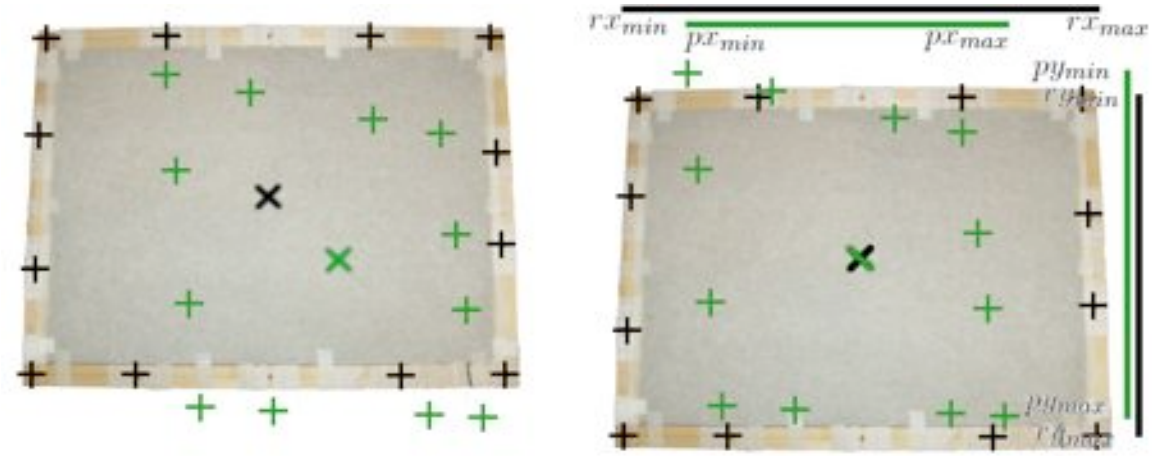


Figure 2.3: Detected image points (green) and re-projection of the pose estimate (black). Left: Initial pose estimate, Right: Pose estimate's barycenter aligned with seen points' barycenter. Dimensions of *CR* (green bars) and *CP* (black bars) are shown

2.3 Pose Estimation

Many publications deal with the problem of reconstructing objects and their pose in 3D space from one ore more 2D images (e.g. [Rob63, HCLL89, LH81]). Most newer publications related to computing the pose of objects in 3D space deal with camera calibration ([Tsa87, Har94, Zha99], just to name a few). The problem of camera calibration includes estimating the camera pose which is described by the extrinsic camera parameters in this context. The second result of camera calibration are the intrinsic camera parameters *A* (see sec. 1.4).

2.3.1 Steepest Descent Pose Estimation

This algorithm is not part of the pose estimation system anymore. Therefore, the implementation details can be found in appendix C.1. The algorithm is based on a series of steepest descent [Hei06] searches designed to optimize an initial pose estimate to match the observed points as good as possible. The coding of a solution consists of 6 values: One 3D vector for the estimated position of the projection screen and one 3D vector for its orientation. The solution of each steepest descent instance is used as input for the next instance. All of those instances are based on minimizing different measures in image space. Therefore we operate on the rays *CR* (see sec. 1.4 on page 6) which are obtained from the observed image points and on the re-projections of the estimated pose's points *CP* (see fig. 2.3, left for situation with an initial solution).

The first steepest descent run, using the initial pose estimate as input, aligns the

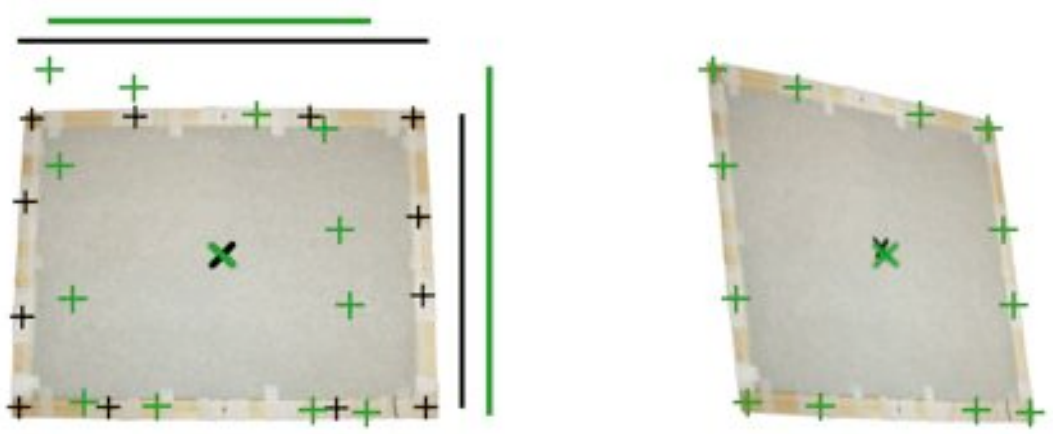


Figure 2.4: Image points (green) and re-projection of pose estimate (black). Left: Dimensional error minimized by translation of pose estimate, Right: Re-projection error minimized by rotation and translation

barycenter of the re-projected model points with the barycenter of the observed points by translating the pose estimate. This preprocessing step roughly aligns the re-projection of the estimated pose to the seen image points (see fig. 2.3, right for result of this run).

The second steepest descent instance is also limited to translation and minimizes the difference in the dimensions of the observed points and the re-projected points in the image. If the initial pose's orientation is close to the real orientation of the projection screen, this step yields a solution that approximates the distance between camera and projection screen (see fig. 2.4, left for result).

For the next two steps, rotation of the current estimate is allowed as well as translation and the sum of squared re-projection errors is minimized. Thereby, the pose estimation is finished in all 6 DOF. This process is split to two runs in order to reduce computation time. The first step uses a smaller neighborhood and produces a first approximation of the projection screen pose which is refined by the final steepest descent instance (see fig. 2.4, right for final pose estimate).

2.3.2 Single Parameter Pose Estimation

Just like the steepest descent approach to pose estimation, this algorithm is not used by our most recent code. Therefore, the details are presented in appendix 11. The idea of this algorithm is to estimate a pose based on only 3 observed points. Therefore, we assume that our model does only have 3 model points $\mathbf{m_a}$, $\mathbf{m_b}$ and $\mathbf{m_c} \in CM$ (see sec. 1.4) which are matched to 3 rays A,B and C defined by metric image plane points in CR. Let $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c} \in CT$ be the according points of a pose estimate (see fig. 2.5).

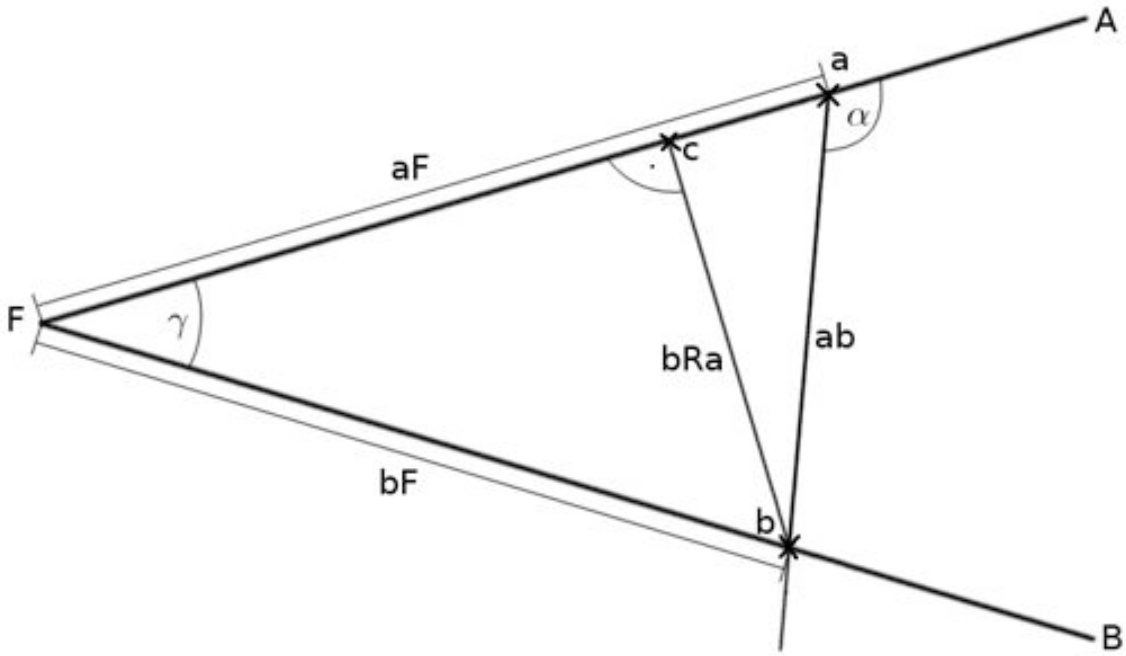


Figure 2.5: Sketch of 2 rays A and B originating at the camera focal point F. The points $\mathbf{a}$ and $\mathbf{b}$ can be determined if α and $|\mathbf{b}-\mathbf{a}|$ are known.

Knowing the model points $\mathbf{m_a}$ and $\mathbf{m_b}$, we know the distance $|\mathbf{ab}| = |\mathbf{b}-\mathbf{a}|$ between those points. Therefore, in each valid pose estimate, the according points $\mathbf{a}$ and $\mathbf{b}$ have distance $|\mathbf{ab}|$ as well. By fixing the angle α between the ray A and $\mathbf{ab}$, the positions $\mathbf{a}$ and $\mathbf{b}$ can be determined on their respective rays.

Assume we fixed $\mathbf{a}$ and $\mathbf{b}$ by choosing an angle α. The third point's movement of the estimated pose is then restricted to a rotation about the axis $\mathbf{ab}$ (see fig. 2.6 and fig. 2.7). As for each valid pose estimate the third point $\mathbf{c}$ has to lie on the according ray C, we can determine whether an estimate is correct by computing the distance d between the rotation circle R of $\mathbf{c}$ and the ray C. If this distance is 0, the angle α obviously yields a correct pose and we can determine the position $\mathbf{c}$, which is the intersection of R and C (see fig. 2.7).

As figure 2.7 shows, the correct pose can not be uniquely determined from 3 observed points. We can define a distance function $a2d$ that maps from angles α to the according distance between C and R. Scanning $a2d$ for roots, up to 4 angles α yielding a zero distance can be found for a given configuration of rays A, B and C. From each those roots we extract a matrix which transforms the model points $\mathbf{m_a}$, $\mathbf{m_b}$ and $\mathbf{m_c}$ into the points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$, yielding a valid pose estimate. Using a fourth point which is seen and matched or the image to image similarities between poses, the "correct" candidate of those 4 pose candidates can be found.

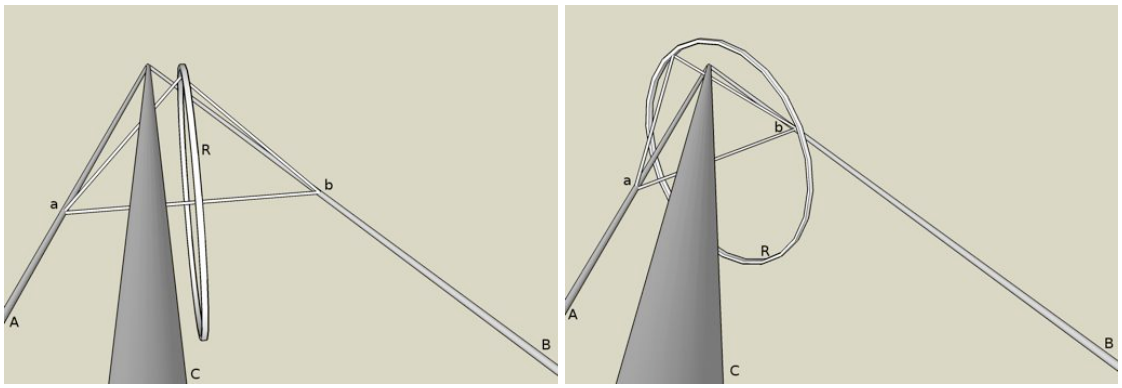


Figure 2.6: Pose estimate candidates with wrong angle for positions of third point

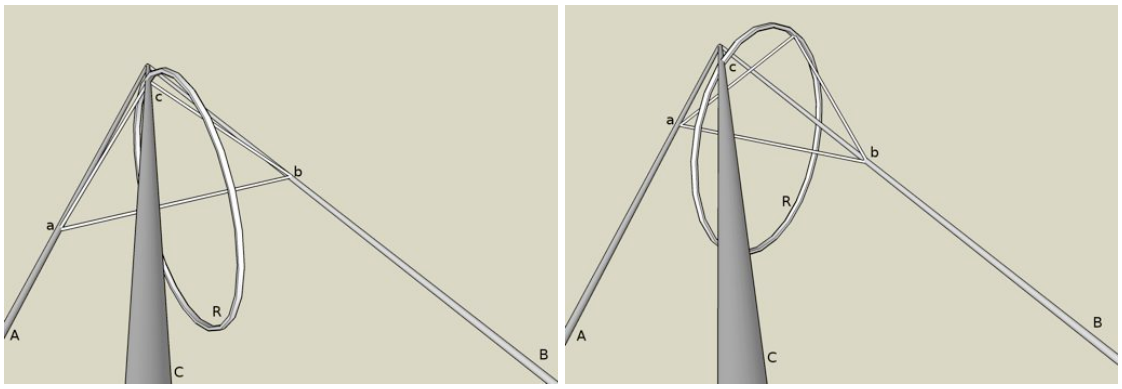


Figure 2.7: Pose estimate candidates with correct angle for positions of third point

2.3.3 Camera Calibration Pose Estimation

This approach for pose estimation is based on the camera calibration method presented by Zhang [Zha99]. In contrast to their scenario we can not view a printed calibration pattern on a planar surface, as we have an IR-pass filter installed on our camera. But as the presented algorithms do not necessarily depend on any properties of the pattern other than being planar we can apply their calibration algorithm to our case: We use the projection plane's LEDs as feature points of the observed plane.

Our algorithm can be subdivided into two phases: Within the first phase we collect a set of images that we can use for a full camera calibration. Having collected enough images to do this, we can extract the camera intrinsic parameters A. Once we know the camera intrinsic parameters we can quickly compute the camera extrinsic parameters for each new image.

Below, we describe our algorithm based on the notation and procedures presented in [Zha99]. The basic notation and observations are directly taken from their paper for. Instead of mentioning that we describe their results all through the text, we will explicitly mention modifications we made to the formulas or algorithms described in their paper. 2D points are denoted by $m = (u, v)^T \in \mathbb{R}^2$ and 3D points by $M = (x, y, z)^T \in \mathbb{R}^3$. $\tilde{M}$ and $\tilde{m}$ are the according homogeneous coordinates and the camera is modeled by a pinhole.Therefore, the image of a 3D point can be described by applying the camera rotation R, the camera translation t and the camera intrinsic parameters A (see sec. 1.4) to the observed 3D point as follows:

$$s\tilde{m} = A\,[R\,t]\,\tilde{M}\ ,\ \text{where } s \text{ is an arbitrary scale factor} \tag{2.1}$$

Exactly like the points of Zhang's calibration pattern, all our LEDs are coplanar. Therefore the positions of our LEDs on the projection screen can be described by $M = (x, y, 0)^T$. This assumes that our model plane is located at $z = 0$. Exploiting this assumption and denoting the i-th column vectors of R as r_i, the relation between observed points $m = (u, v)^T$ and LED's positions can be written as:

$$s\tilde{m} = s\,(u, v, 1)^T = A\,[R\,t]\,\tilde{M} = A\,[r_1\,r_2\,r_3\,t]\,(x, y, 0, 1)^T = A\,[r_1\,r_2\,t]\,(x, y, 1)^T \tag{2.2}$$

Computing an image's homography: By denoting the points $M = (x, y)^T$ of the projection screen plane as vectors with 2 components, they can be related to image points by a homography H: $s\tilde{m} = H\tilde{M}$. From this and equation 2.2 we can conclude $H = A\,[r_1\,r_2\,t]$. As this equation is the link between H and the camera parameters, we want to calculate the homography for each picture. Denoting the entry of H in the i-th column and the j-th row by h_{ij}, we can apply a standard method: For each model point $M = (x, y)^T$ with the according image point $m = (u, v)^T$, $s\tilde{m} = H\tilde{M}$ can be rewritten as:

$$\begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -ux & -uy & -u \\ 0 & 0 & 0 & x & y & 1 & -vx & -vy & -v \end{bmatrix} x = 0,$$

$$\text{where } x = (h_{11}, h_{21}, h_{31}, h_{12}, h_{22}, h_{32}, h_{13}, h_{23}, h_{33})^T$$

A matrix, build by stacking those constraints for at least 4 points, determines x up to scale, if the 4 points are not degenerate. Let C be a $2n \times 9$ matrix containing n ($n \geq 4$) of the above constraint pairs. Then we have $Cx = 0$ and can compute $C = U\Sigma V^*$, the singular value decomposition (SVD) of C. The column vector of V^* that is associated with the smallest singular value in Σ is a solution for x and, therefore, we have determined H up to a scalar factor.

Estimating the intrinsic parameters: Now that we know the homography H of a picture, we can exploit r_1 and r_2 being orthonormal for expressing 2 constraints that can be obtained from one image (see [Zha99] for the derivation of this formula):

$$\begin{bmatrix} v_{12}^T \\ (v_{11} - v_{22})^T \end{bmatrix} b = 0$$

where $v_{ij} = (h_{i1}h_{j1},\ h_{i1}h_{j2} + h_{i2}h_{j1},\ h_{i2}h_{j2},\ h_{i3}h_{j1} + h_{i1}h_{j3},\ h_{i3}h_{j2} + h_{i2}h_{j3},\ h_{i3}h_{j3})^T$
and $b = (\frac{1}{\alpha^2}, -\frac{c}{\alpha^2\beta}, \frac{c}{\alpha^2\beta} + \frac{1}{\beta^2}, \frac{f}{\alpha^2\beta}, -\frac{cf}{\alpha^2\beta^2} - \frac{v_0}{\beta^2}, \frac{f^2}{\alpha^2\beta^2} + \frac{v_0^2}{\beta^2} + 1)^T$, with $f = c\,v_0 - u_0\,\beta$

By stacking those constraints for n images we obtain a $2n \times 6$ matrix C, for which $Cb = 0$. b codifies the camera intrinsic parameters, contained in A and using the SVD $U\Sigma V^*$ of C, a solution for b can be obtained up to a scalar factor. As in the computation of the homography, the solution for b is, up to scale, determined by the column vector of V^* that is associated with the smallest singular value in Σ.

By denoting the i-th component of b as b_i, we can compute the camera intrinsic parameters as follows:

$$\begin{aligned}
v_0 &= (b_2b_4 - b_1b_5)/(b_1b_3 - b_2^2) \\
\lambda &= b_6 - \frac{b_4^2 + v_0(b_2b_4 - b_1b_5)}{b_1} \\
\alpha &= \sqrt{\lambda/b_1} \\
\beta &= \sqrt{\lambda b_1/(b_1b_3 - b_2^2)} \\
c &= -b_2\alpha^2\beta/\lambda \\
u_0 &= c\,v_0/\beta - b_4\alpha^2/\lambda
\end{aligned}$$

The results of the above procedure are rough estimates of the camera intrinsic parameters. But as they are not based on a physically meaningful measure, they often do not have sufficient quality. Therefore they are refined by minimization of the re-projection error. The target function for n images with m model points can be expressed as:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} ||m_{ij} - \hat{m}(A, R_i, t_i, M_j))||^2,$$

where $\hat{m}$ is the re-projection of a model point. This minimization is done, using the Levenberg-Marquardt algorithm, as implemented in ALGLIB [BB]. The rotation R_i of the camera for the i-th picture is coded in a 3-vector that has the direction of the rotation axis as direction and a length equal to the angle of the rotation.

Extracting the camera pose for an image: Knowing the homography H for an image and the camera intrinsic parameters A, we can simply compute the camera extrinsic parameters from the following formulas:

$$r_1 = \lambda A^{-1} h_1 \ , \ r_2 = \lambda A^{-1} h_2 \ , \ r_3 = r_1 \times r_2 \ , \ t = \lambda A^{-1} h_3,$$
$$\text{where } \lambda = 1/||A^{-1} h_1|| = 1/||A^{-2} h_2||$$

Because of noise in the blob positions, the rotation matrix $R = [r_1, r_2, r_3]$ does not necessarily fulfill the requirements of a rotation matrix. Therefore we set a rotation matrix R' by using $R' = UV^*$, where $U\Sigma V^*$ is the SVD of R (see the technical report accompanying [Zha99] for the deduction of this).

Complete approach: Knowing the components of Zhang's camera calibration approach that are needed for our purpose, we can describe the complete algorithm for estimating the camera pose for each image:

For each image, in which we have at least 4 matched observed points, we estimate the homography between the image plane and our projection screen model plane. This homography estimate for single images is prone to noise in the extracted image points as in contrast to the original work of Zhang, we only have up to 14 feature points (our LEDs) for estimating the homography and not 128 as on Zhang's calibration plane. Therefore, we want to use more than the proposed 3-5 images of our projection screen for acquisition of the camera internal parameters.

As our images are obtained from a continuous camera stream, another problem arises, which is special to our use-case: two subsequent images of the camera stream will most likely show the projection screen from almost the same position and orientation. As pointed out by Zhang, images with very similar orientations of the plane will yield very low amounts of additional information for the internal parameters. Instead of using every picture the camera delivers, we only examine every n-th picture.

This is still not sufficient for guaranteeing different orientations and thus we have added a second mechanism to sort out images with similar rotations: Whenever an amount of m pictures has been obtained with the according homographies $H_i, i \in \{1, ..., m\}$, we estimate temporary internal parameters A' from this set of m images. Then we extract the camera pose (extrinsic parameters) for each of those images, using the image's homography H_i and the temporary internal parameters A'. The images are compared pairwise and from each pair, where the orientation is too similar, one of the images is rejected.

If a sufficient amount of images (in our current implementation 17) have not been rejected by this procedure, those are used to compute a final estimate for the camera intrinsic parameters A. Knowing A, we now estimate a homography H for each camera image and a camera pose relative to the projection can be obtained from A and H. Pseudo code for the update procedure provided one new picture, is given in algorithm 6.

Algorithm 6 pose estimation based on Zhang's camera calibration

Input: image coordinates of matched points $\hat{C}I$,
 model point coordinates M,
 point matching $i2m : O \to M \cup \{-1\}$
Variables: intrinsic camera parameters $A := null$,
 set of homographies for initial calibration $C_H := \emptyset$
Output: pose estimate of the camera P

```
Approximate the homography H from
  image points m_k ∈ ĈI and according model points M_{i2m(k)} ∈ ĈM.
IF intrinsic parameters not yet estimated (A =null) THEN
  Add H to the homographies:  C_H := C_H ∪ {H}
  IF number of homographies is sufficient (|C_H| > m) THEN
    Estimate temporary intrinsic parameters A' from C_H
    Reject homographies from C_H, using A'
      so that no two of them have similar rotations
  IF number of homographies is still sufficient (|C_H| > x)
  THEN
    Estimate camera intrinsic parameters A from C_H
IF intrinsic parameters estimated (A ≠ null) THEN
  Extract pose P from H
  return P
ELSE
  return null
```

2.3.4 Discussion

We have presented three approaches to pose estimation. Now we will discuss their features and to what extent they can be used for estimating our projection screen's pose.

The first algorithm we discussed is based on subsequent steepest descent searches. The underlying local search can converge to local minima, which are not the correct pose. Therefore, it is necessary to initialize the first steepest descent run with a pose that is already close to the real pose of the projection screen. When a pose is known from the previous image, this last pose can be

used as the initial argument for the new image's steepest descent runs. If we do not have any information about the current pose, the only way to initialize the search is guessing a pose. Although the pose estimate of the algorithm can be confirmed or rejected based on the re-projection error, we would need to try many different initial poses in order to find the correct pose. This is unacceptable due to computation time reasons, as we cannot give an upper bound for the number of necessary trials in order to initialize this pose estimation algorithm.

The second method for pose estimation is based on estimating a single value: The angle α between the line connecting two of the feature points and the ray going through one of these feature points. This implies an advantage over the steepest descent algorithm: The single parameter approach does not need a prior pose to initialize the search for a pose estimate. Besides this the most notable feature of this algorithm is that it works on as few as 3 feature points. On the one hand this is advantageous, as it allows to track an object's pose even when only 3 or 4 feature points are matched to model points. This allows us to use the Nintendo Wii Remote for tracking the IR-LEDs on the frame, which tracks up to 4 IR-blobs at an update-rate of 100Hz without requiring blob extraction in software. On the other hand in figure 2.7 we can observe a severe problem of the algorithm: With only 3 known points, there are different correct angles α and with them different poses for the same constellation of observed points. This ambiguity can only be resolved through post processing the correct pose estimates. Methods to do this can either be based on the re-projection error or on the pose estimation of previous images. Another drawback connected to only using 3 points for pose estimation is the strong sensitivity of the pose estimation to noise in the observed image points' positions. One final drawback of this approach is the run time and precision of finding roots of $a2d$, which strongly depends on the pose of the projection screen. Altogether this algorithm is not suited for obtaining a reliable and stable pose estimation.

The last approach we have described is based on the camera calibration method of Zhang[Zha99]. A constraint given by this approach is that all feature points of our projection screen need to be aligned coplanar. One advantage, at least over the single parameter pose estimation, is that this approach is able to handle many feature points. Another advantage over both of the other methods for pose estimation is that this one allows us to obtain the internal parameters of the camera using the feature points of our projection screen. If we were using other approaches to calibrate the camera, we would have to take the IR-pass filter from the camera for calibrating it and reinstall the filter, thereafter. Reinstalling the filter bears the risk of changing the intrinsic parameters of the camera. Another advantage of this approach is that once the intrinsic camera parameters are known, the estimation of the projection screen pose can be achieved by a singular value decomposition of a $2|O| \times 9$ matrix and a small amount of one-step computations. As this algorithm uses information from more than 3 points for pose estimation, it is not as sensitive to noise in the projection screen points' positions as the single parameter approach. So the only downside to this algorithm is the necessity for the observed

object to be planar. As our projection screen is naturally planar, this constraint does not impose a problem. Therefore, the camera calibration algorithm proposed by Zhang is our favorite for estimating the projection screen pose.

One problem that remains although using 12 feature points on our projection screen is noise in the camera image and the resulting noise in the screen pose estimation. For future use, re-evaluation and usage of algorithms that are purely dedicated to pose estimation (as for example those proposed in [HCLL89, LHM00, Fio01, AD03]) could improve the performance and precision of the obtained pose. One big achievement of a new algorithm would be taking the last pose estimates into account in order to produce a less noisy pose estimate over time. This improved algorithm could still rely on the intrinsic camera parameters obtained by using our adaptation of the camera calibration algorithm of Zhang. One possible method could be using Zhang's method for initialization of a gradient based method as described in section C.1.

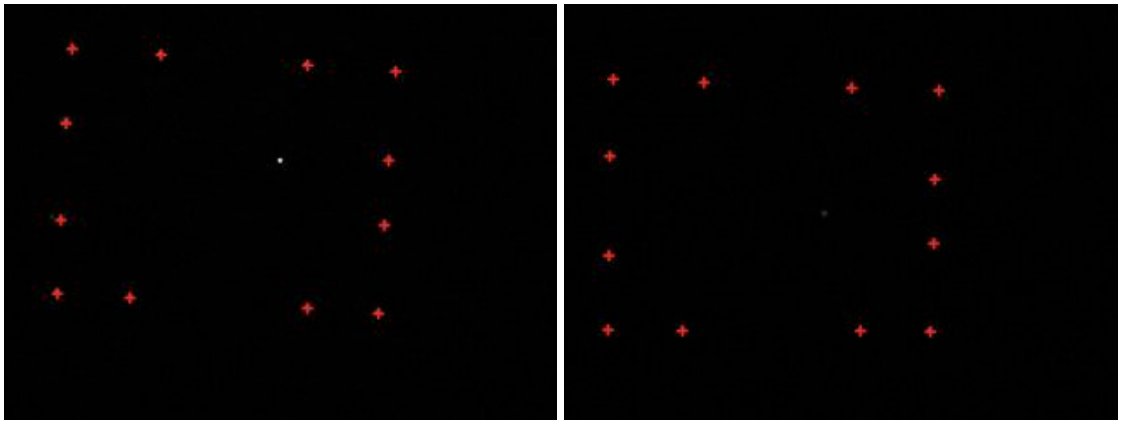


Figure 2.8: Camera image of projection screen with pen's light point, red points are matched. Left: Pen on screen surface, Right: Pen 1cm from screen surface

2.4 Cursor Handling

Up to now we detailed all steps necessary to determine the real world pose of the projection screen. This section presents a method of providing a 2 DOF input device that can be used to control the cursor on the screen. Our approach is based on a pen with a narrow opening angle IR-LED (TSHA520) as the pen's tip(see fig. 2.9 for an image of the pen). When this LED points towards the projection screen from small distances, a strong IR-light point is visible on the screen (see fig. 2.8, left). Moving the pen away from the screen more than two centimeters will make the light dot on the screen unrecognizable due to diffusion (see fig. 2.8, right). This could be used to issue mouse events, but would have the drawback of the user not seeing the cursor before the "pen on surface-event" is triggered.

Additionally the pen features two push buttons for issuing left- and right-clicks. As long as one of the buttons on the pen is held down, the associated IR-LED on the projection frame will change it's state which can be observed by the camera. This section describes how the cursor position is computed and clicks are detected.

2.4.1 Cursor Detection

The first problem we need to solve when computing the cursor position is to identify the observed point $id_c \in O$ (see sec. 1.4 for definitions) that has to be interpreted as the cursor's position in the image. As the pen does not have a fixed position on the projection screen, it can not be identified by using line ratios or the re-projection of points into the image (see sec. 2.2). In most cases, however, the identification of the cursor can be done by exclusion, as all other blobs are matched with model points. We assume that this is the case, if the number of seen points exactly exceeds the number of known model points by one and all but one of the seen points are matched ($|CI| = |M| + 1 \wedge |\hat{CI}| = |M|$). In this scenario

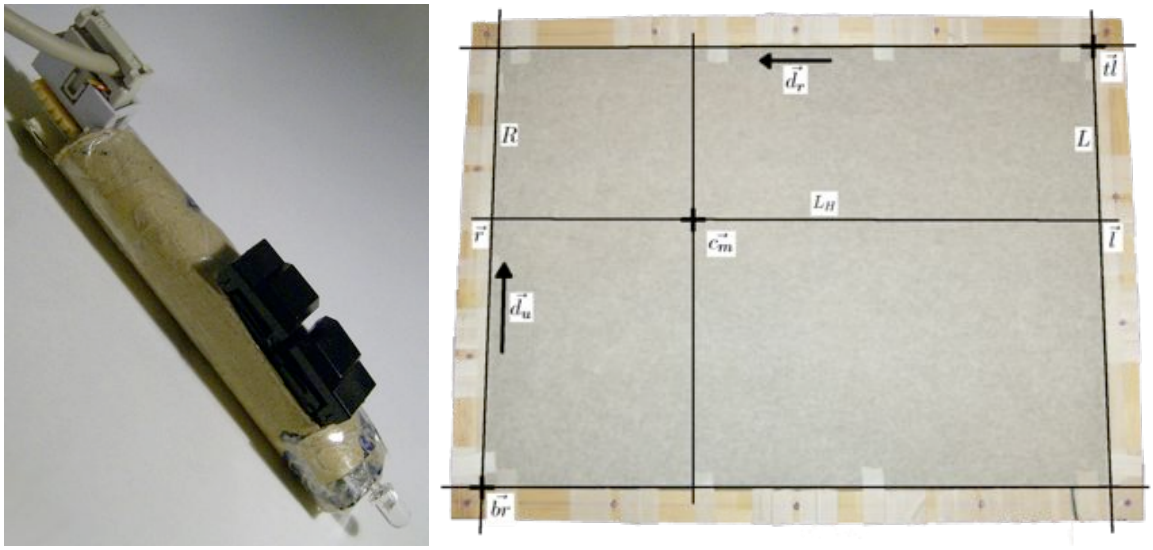


Figure 2.9: Left: The screen's pen, Right: Geometry of cursor position estimation

the unmatched seen point is assumed to belong to the pen's LED. Figure 2.8, left shows an image of the projection screen, in which the pen's image point can be identified by this method. Once we have identified the cursor by this method, the frame to frame point matching (see sec. 2.2.3) keeps track of the identified cursor point as if it was matched to a model point.

Assuming we know the cursor's position $\mathbf{p} = (p_x, p_y, 1)^T \in CI$ in image space, we can get its position $\mathbf{c_m} = (c_x, c_y, 0)^T$ in the projection screen's model plane. This is achieved by multiplying $\mathbf{p}$ it with the inverse H^{-1} (see appendix A for the pseudo-inverse operation) of the homography H that maps from the camera image onto the plane of the projection screen model. This homography is calculated for each image by the pose estimation method adapted from Zhang's camera calibration method (see section 2.3.3).

To compute the relative cursor position $(h, v)^T \in \{(x, y)^T \in \mathbb{R}^2 \,|\, 0 \leq x, y \leq 1\}$ on the projection screen, we need some additional information about the screen (see fig. 2.9 for the geometry laid out here): We need to know the projected image's top left point **tl** and bottom right point **br** on the screen surface model. Additionally we need either up and right directions $\mathbf{d_u}$ and $\mathbf{d_r}$ or equivalently the positions of the other 2 corners of the screen. As from the user's perspective the image is back-projected, left and right are reversed from the projector's vantage point.

Now we determine the relative horizontal position h of the cursor. Let L_H denote the horizontal line that contains the cursor position $\mathbf{c_m}$ and has the direction $\mathbf{d_r}$. Additionally the line L (R) is the left (right) boundary of the area of the screen on which the image should be projected. L (R) can be described by the point **tl** (**br**) and the direction $\mathbf{d_u}$. Next we compute the intersection points **l** and **r** of the horizontal line L_H with left boundary L and the right boundary R of the projection screen. Knowing these intersection points, we can compute the horizontal por-

tion $h = |\mathbf{c_m} - \mathbf{l}|/|\mathbf{r} - \mathbf{l}|$ of the full screen-width at which the cursor resides. The procedure to get v, the relative vertical position, is analogous.

Knowing the relative cursor position, the pen can be used as absolute input device on the screen. The input is absolute, as the cursor position is directly bound to the pen's tip in contrast to a mouse that will in most cases give relative input. For some applications we want to use the combination of the projection screen pose and the cursor position to mark a point $\mathbf{p} = (p_x, p_y, p_z) \in \mathbb{R}^3$. The marked point is of course be the point at which our pen's tip touches the projection screen. To compute this point we just need to transform the cursor position from the screen model's coordinate system into the world coordinate system: If the projection screen pose is described by the matrix M, then the homogeneous 3D position of the cursor is $(p_x, p_y, p_z, 1)^T = M \times (c_x, c_y, 0, 1)^T$.

2.4.2 Click Detection

The next step for making our light-pen usable as an input device is to notice clicks on the pen's push buttons (see fig. 2.9, left). As mentioned earlier, clicking one of the two buttons causes one of the IR-LEDs on the projection frame to perform a state-change that is recognizable by the camera.

"LED on" on button press This method of signalling a button press changes the state of the LED from off to on as long as the associated button is pressed. Thus, the IR-LED in question is completely dedicated to signaling the button's state. Those LEDs are exactly like the pose estimation LEDs identified by a model point ID $l \in M$ and have a known location $m_l \in CM$ on the projection screen model. Therefore the button state can be determined by querying if the l is is matched to an observed point ($m2i(l) \geq 0$).

"LED blinks" on button press This approach to signal a button press lets the associated LED blink on button press. As long as the button is not pressed, the LED stays on. To be sure that each on- and off-state of the LED can be seen in at least one complete image, it has to stay in each state for the duration of at least 2 frames. For an image acquisition frequency of 60Hz this means that the LED may change its state 30 times a second, implying it may have up to 15 on and off-cycles.

If images are coming constantly and are not skipped, blinking blobs can be recognized by a simple procedure: In the matching algorithm the positions of lost observed points are stored together with the last frame in which they were seen. If a new blob is close enough to a recently lost one, it is considered to be the same blob. We consider an observed blob as blinking if it has vanished within the last x ($x = 10$ in our implementation) frames. The variable x is extremely important here: If set it too high, users will experience the delay in between releasing the

button and the system's reaction. Setting x too low will result in interruptions of the button-down state, as soon as some frames indicating state changes are missed.

Our implementation of a blinking LED circuit (see sec. 4.1) achieves 16.4 on-off-cycles per second. Although this does not ensure that no state of the LED is missed, the 16.4 cycles per second are recognized as long as the camera does not skip any images. In order to handle images that are not showing a clear state of the LED, we added a mechanism to the blob extraction algorithm. In such images, the LED appears in a strongly dimmed state, indicating it has been off in a portion of the image's recording time.

As our blink detection is based on the off-state of the LED we want to consider such images as not showing the LED. Let $B \in P(\mathbb{Z}^2)$ be the set of pixels of a blob and I the overall intensity of this blob. The average $\bar{I}$ of the overall intensities $I_{i \in n}$ over the last n frames will be maintained. If, in the current image, the intensity is far below this average (we use factor 0.5), we consider the LED to be off.

$$I = \sum_{(p_x,p_y) \in B} i_{p_x,p_y}, \text{where } i_{x,y} \text{ is the intensity of the pixel at } (x,y)^T$$

2.4.3 Discussion

As holding the screen in one hand and using the mouse with the other one is not an option, we decided for the pen based input approach. The main reason for the exclusion of the mouse as an option is that the user needed to remain close to the mouse and a desk and thus could not move with the projection screen. As virtually all people are familiar with pens and other researchers have been using the interaction metaphor of a pen successfully in similar scenarios [MPL+02, TFK+02, Yee03], we decided to mimic a pen . Additionally the projection screen serves as a good canvas-like surface and the metaphor of drawing fits our goal of modeling in FiberMesh very well.

As stated before, the pen only produces a focused and strong spot on the screen, if the tip is held very close to the screen surface. This could be used to issue mouse events, but as the strength of the observed IR-point produced by the pen is not constant for a fixed distance from the screen it would be very complicated to find a state between "point is invisible" and "pen is on screen surface". Therefore, the system would trigger the "pen on surface"-event in a way that is incomprehensible for users. Thus we decided to include push buttons on the pen to enable the user to issue the pendant to mouse clicks.

The approach to signal button presses by switching one of the LEDs on has one drawback: LEDs associated to a button can not be used for pose estimation constantly, as its off state signals that the button is not pressed. With our most recent blob tracking approach this is only a question of mounting additional IR-LEDs on the projection screen, hence, this drawback can be neglected.

A theoretical alternative to turning the LED on when the button is pressed would be to turn it off instead. This had the advantage that most of the time the LED can

be used for pose estimation. In practice this would most probably not work: As soon as the point matching looses matching for the LED, a mouse click would be issued. Reasons for loosing a point include obstruction of the LED or too quick movements of the projection screen.

The approach of using a blinking LED for click detection addresses the problem that a button-LED can not be tracked while the button is not pressed. As the LED stays on constantly as long as the button is not pressed, it can be tracked. But the approach has several drawbacks: While the button is pressed, users experienced occasional point matching problems due to the blinking LEDs, which in most cases produced a defect screen pose estimation for some frames. Another problem of the blinking LED approach is a slightly varying frame pose estimate produced by switching forth and back between the estimate with and without the blinking LED. Exhibiting such behavior, the approach is unacceptable.

An early approach of using a blinking LED for cursor detection was using the pen's LED for this purpose. In addition to the problems described above, the temporal resolution of the cursor position signal is decreased by the images in which the LED does not appear. The problem of loosing the point matching for the cursor occurs more frequently then with frame mounted blinking LEDs. The reason for this are quick movements of the pen, moving the pen's tip out of the point matching's re-detection radius while the LED is turned off. An advantage of this approach is that it does not require a connection between the pen and the projection screen, as the LED and the blinking circuit can easily be powered by a battery. Nonetheless it is not suited for our purposes as it is not robust.

Chapter 3

Pre-Warping the Projected Image

3.1 Projector Calibration

In this chapter we discuss how to project the image that would normally fill the whole projection beam onto the image's portion in which the projection screen resides. Therefore, the projected image needs to be pre-warped in order to appear exactly on the projection screen. Several researchers have investigated projecting images on surfaces that are not optimally aligned with respect to the projector ([RB01, Pin01, RvBB+05, LHSD05], just to name a few). In order to fit the projected image onto the projection surface, the original image points have to be remapped on the projectors image plane. See figure 3.1 for a comparison of an original image along with a pre-warped version and figure 1.5 on page 5, middle for the effects of not pre-warping the image for the projection.

As shown by in [FL88] a 3×3 homography H, which can be built from 4 point correspondences, can be used to relate the images of two cameras viewing the same planar object. This in combination with the insight that (from a geometrical point of view) projectors are similar to cameras, (see [Pin01, RB01], for example) leads to the idea of using such a homography for relating the camera- and the projector image. In this case our projection screen would be the planar object that is "observed" (the projector's way of observing is projecting an image) by the projector and the camera. The homography H, however, looses its validity if the observed plane moves, which is constantly the case for the hand-held projection screen. Therefore, we need to use an approach that is based on measures which are invariant under movement of the observed object: The intrinsic parameters and relative pose of the camera and the projector, are such measures. This chapter will show how to calibrate use them for pre-warping the projected image.

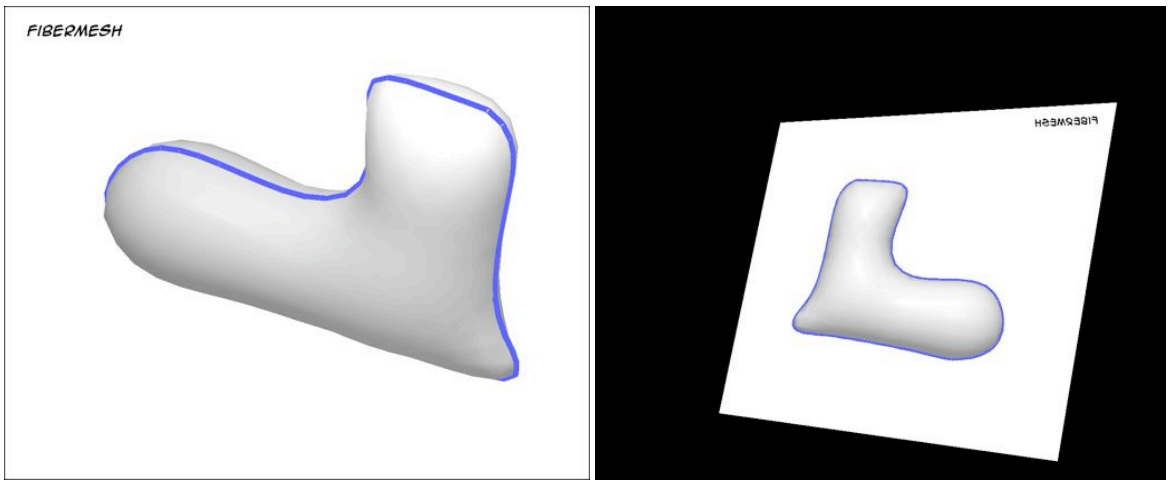


Figure 3.1: Left: A scene in FiberMesh, Right: Pre-warped image, fitting the original scene onto the projection screen

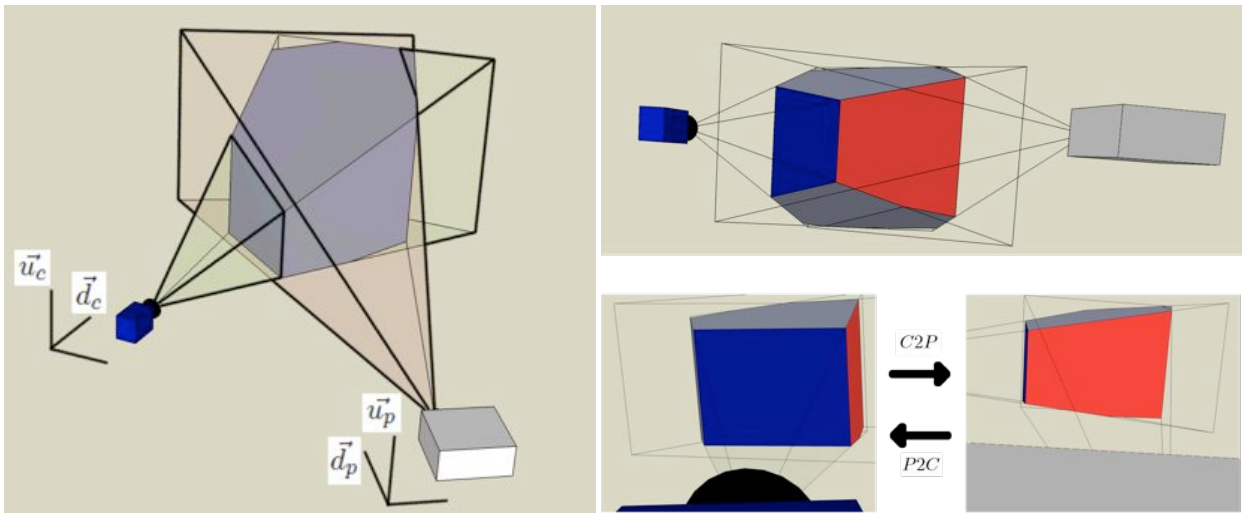


Figure 3.2: Left: The intersection of the camera viewing area and the projection area is available as working space, Right: Coordinate system transformations between camera and projector

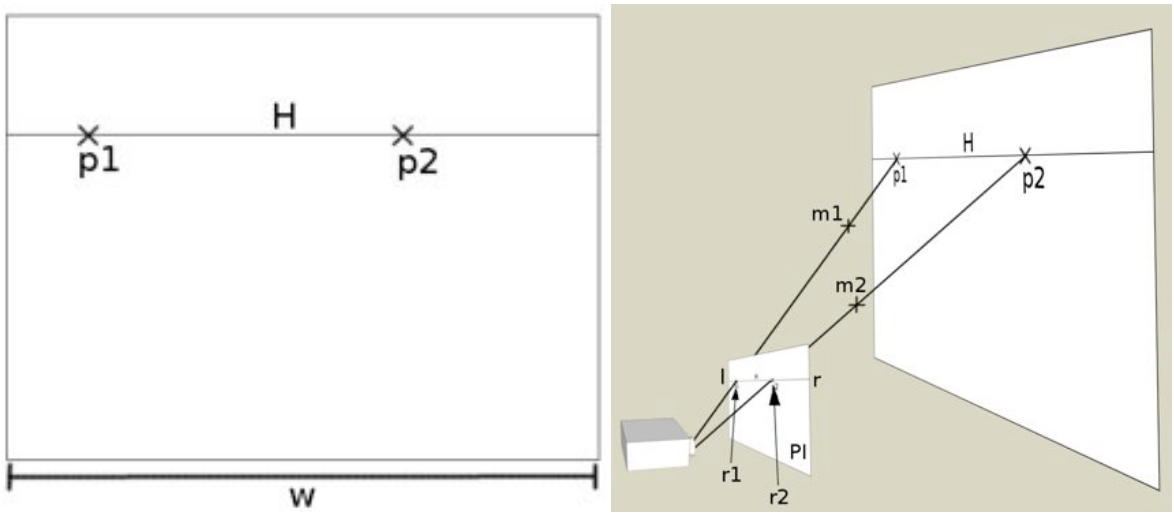


Figure 3.3: Geometry of projector's intrinsic parameter's estimation. Left: The projected image, Right: Relation between projected image, the projector's metric image *PI* and 3D space

Calibrating the camera-projector system The first step in order to relate the 3D projector frame and the 3D camera frame to each other is to find coordinate transformations that transform camera centric coordinates to projector centric coordinates and vice versa. Assume we know the position $\mathbf{p_p}$, the forward direction $\mathbf{d_p}$ and the up direction $\mathbf{u_p}$ of the projector (see fig. 3.2). Furthermore, we assume that our world coordinate system is centered at the camera, the camera's viewing direction is $\mathbf{d_c} = (0,0,-1)^T$ and the camera's up direction is $\mathbf{u_c} = (0,1,0)$.

Let R_{p2c} be the matrix that rotates in a way that $R_{p2c} \times (0,0,-1)^T = \mathbf{d_p}$ and $R_{p2c} \times (0,1,0)^T = \mathbf{u_p}$ (see appendix A for getting the rotation between 2 orientations). R_{c2p} is the rotation inverse to R_{p2c} that transforms the projector orientation vectors into the camera orientation vectors. Additionally let T_{p2c} be a matrix translating by $\mathbf{p_p}$ and T_{c2p} a matrix translating by $-\mathbf{p_p}$. A vector given in the camera centered (world-)coordinate system can be transformed to reference the same point within the projector centric coordinate system by the matrix $C2P = R_{c2p} \times T_{c2p}$. $P2C = T_{p2c} \times R_{p2c}$ transforms projector centric vectors into the camera centric coordinate system. Figure 3.2, right visualizes this (note that the perspective projection producing the image shown there is not yet known). Nonetheless, we can now transform the camera centric coordinates of the projection screen into the projector's coordinate system , although, the link between the projector centric 3D coordinates and the projected image is still missing as the projector's intrinsic parameters are not yet known.

In order to relate projector centric 3D coordinates to the projector's image, we need first assume to know projector's resolution $(w,h)^T \in \mathbb{Z}^2$. In order to compute the projector's intrinsic parameters, we first choose a horizontal line H in the image space of the projector (see fig. 3.3, left). Then we choose two pixels

$p_a, a \in \{1,2\}$ on this line, with pixel coordinates $(u_a, v_a)^T \in \mathbb{Z}^2$. Without loss of generality we assume $u_2 > u_1$. For each of those points we assume to know the projector centric coordinates $\mathbf{m_a} = (x_a, y_a, z_a)^T \in \mathbb{R}^3$ of a point lying on the ray associated with the pixel (see fig. 3.3, right).

First the 3D points are projected onto a plane *PI* parallel to the image plane of the projector at distance 1 from the coordinate origin (see fig. 3.3, right), yielding the points $\mathbf{r_a}$. Assuming that the pixels of our projector have uniform size, we can compute the width wp of a pixel on *PI* using the re-projected points $\mathbf{r_a}$. Finally, we can compute points $\mathbf{l}$ and $\mathbf{r}$ on the left and right vertical edges of the image projected on *PI*.

$$\begin{aligned}
\mathbf{r_a} &= (px_a, py_a, 1)^T = (\frac{x_a}{z_a}, \frac{y_a}{z_a}, 1)^T \\
wp &= \frac{u_2 - u_1}{px_2 - px_1} \\
\mathbf{l} &= (l_x, l_y, 1)^T = (px_1 - wp \cdot u_1\,, \frac{py_1 + py_2}{2}\,, 1)^T \\
\mathbf{r} &= (r_x, r_y, 1)^T = (px_2 + wp \cdot (w - u_2)\,, \frac{py_1 + py_2}{2}\,, 1)^T
\end{aligned}$$

Note, that both vertical components of the points $\mathbf{r_a}, a \in \{1,2\}$ in image space should in theory be the same. Averaging over them in $\mathbf{l}$ and $\mathbf{r}$ is a tribute we pay to the usage of real data. This real data are the points $\mathbf{m_a}$ which are obtained by the pose estimation system and then transformed into the projector's coordinate system by *C2P*. Analogous to obtaining the points $\mathbf{l}$ and $\mathbf{r}$ on the left and right border of the projected image on *PI*, we can get points on the top border $\mathbf{t} = (t_x, t_y, 1)^T$ and on the bottom border $\mathbf{b} = (b_x, b_y, 1)^T$ on *PI*. In the next step a line is formed along each of the borders, using the directions $(0,1,0)^T$ and $(1,0,0)^T$. By calculating the intersections of those lines we can now determine the corners of the intersection between *PI* and the projection pyramid.

Let $(ul_x, ul_y, 1)^T$ be the upper left corner on *PI*. A point $(r_x, r_y, 1)^T \in \mathbb{R}^3$ on the image plane *PI* of the projector can be related to the matching pixel coordinates $(u, v)^T \in \mathbb{R}^2$ by the following formulas, where hp is the height of a pixel on *PI*:

$$\begin{aligned}
u &= \frac{r_x - ul_x}{wp} \\
v &= \frac{r_y - ul_y}{hp} \\
r_x &= u \cdot wp + ul_x \\
r_y &= v \cdot hp + ul_y
\end{aligned}$$

Obtaining data for the projector calibration Now that we have the mathematical framework to calibrate the projector, we just have to obtain the data we have taken for granted so far. The methodology for doing so is as follows. The projector shows a target texture, indicating a pixel the user should mark. Moving the

projection screen in place, the user can clicks on the target texture with the pen. The pose estimation system then delivers the obtained 3D position of the cursor to the projector calibration routine. The 3D points $m1$ and $m2$ in figure 3.3 depict two such points marked by the user, where $p1$ and $p2$ are the marks on the image, indicating the rays in 3D space on which the user has to mark a point. A discussion of this technique under user interaction aspects and pictures of a real calibration process are given in subsection 5.1.

For the coordinate transformations $C2P$ and $P2C$ we need the position $\mathbf{p_p}$, the forward direction $\mathbf{d_p}$ and the up direction $\mathbf{u_p}$ of the projector. For computation of the projector's intrinsic parameters we need two 3D points corresponding to known pixel locations on a horizontal line in the projector's image and two of them on a vertical line.

The first two points requested from the user are two different points on the ray C, which is cast by the projection of the center pixel (image coordinates: $(\frac{w}{2},\frac{h}{2})^T$). Additionally we request two points on the ray U that is cast by the projection of a pixel above the center pixel (pixel coordinates $(\frac{w}{2},y)^T, y < \frac{h}{2}$)). The two points per ray can be used to describe the line along which the ray is cast.

Calculating the intersection point of the two lines will yield the projector position $\mathbf{p_p}$ (in camera centric coordinates, as the camera delivers the marked point positions in its own coordinate system). This is done by finding the closest point to the other line (see appendix A), as typically the lines obtained will not have an intersection point due to noise in the camera image and imprecise marking of the points. Knowing $\mathbf{p_p}$ and the fact that all marked points are in front of the projector, we can extract the forward direction $\mathbf{d_p} = \mathbf{c_1} - \mathbf{p_p}$ of the projector, where $\mathbf{c}_1 \in \mathbb{R}^3$ is one of the marked points on C.

With this data we are able to describe the image plane CI, lying 1 unit in front of $\mathbf{p_p}$ and having $\mathbf{d_p}$ as its normal. The next step is projecting the rays C and U onto CI which yields metric points $\mathbf{c}$ and $\mathbf{u}$ on the plane. At this point we can calculate the projector's up direction $\mathbf{d_u} = \frac{\mathbf{u}-\mathbf{c}}{|\mathbf{u}-\mathbf{c}|}$.

The vertical dimension of pixels and, therefore, the image on CI can be measured using $\mathbf{c}$ and $\mathbf{u}$. A final input request to the user is a 3D point lying along a ray R, cast by a pixel in the same row of the image as the center ray (pixel coordinates $(x,\frac{h}{2})^T$, with $x \neq \frac{w}{2}$). Re-projecting the marked point onto CI yields a point $\mathbf{r}$, which can be used to calculate the horizontal dimension of pixels and the image on CI. At this point we have all necessary data for estimating the projector's pose and its internal parameters with the procedures described above.

Refining the camera-projector calibration We have to deal with inaccurate data introduced at several stages of the pose estimation process and by the handheld nature of the projection screen. Those inaccuracies heavily impact the calibration result. Therefore, we need more data and a better procedure in order to establish a good calibration of the projector. Instead of the data described in the last paragraph, we collect 2 points, denoted with indices $id \in \{1,2\}$, along each

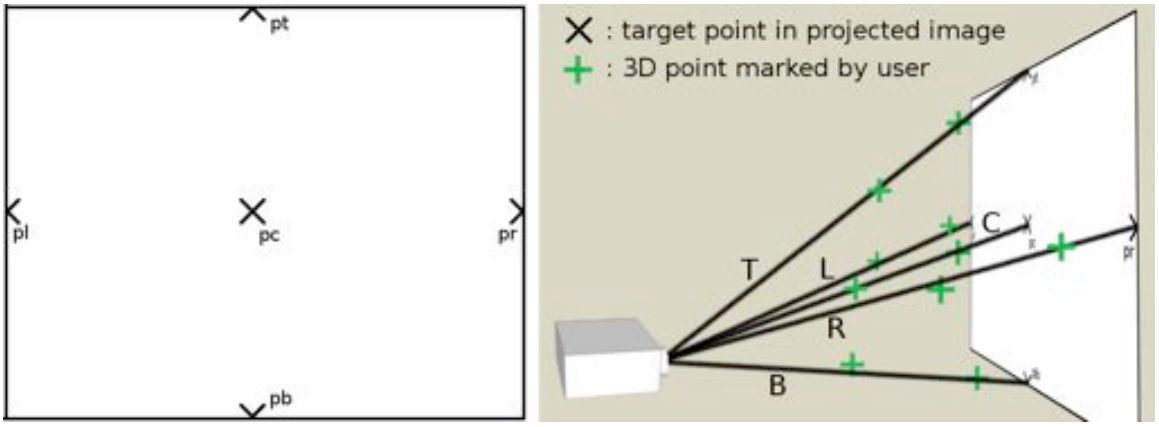


Figure 3.4: Geometric relationships of the projector calibration input. Left: Projected image, Right: 3D view on projected image and marked points

of 5 rays (green crosses in fig. 3.4): The points $\mathbf{c}_{id} \in \mathbb{R}^3$, on the ray C at the center of the screen, and the points $\mathbf{l_{id}}$, $\mathbf{r_{id}}$, $\mathbf{t_{id}}$ and $\mathbf{b_{id}} \in \mathbb{R}^3$ along the rays L, R, T and B. Those rays are again the projection rays of chosen pixels: C is produced by the center pixel $\mathbf{pc} \in \mathbb{Z}^2$of the image. L, R, T and B are produced from the center pixels $\mathbf{pl}$, $\mathbf{pr}$, $\mathbf{pt}$, $\mathbf{pb} \in \mathbb{Z}^2$ (see fig. 3.4, left) of the left, right, top and bottom borders of the image.

With those data we first establish an initial estimation of the projector parameters using the method described above. After this we computationally cast rays $\overline{C}$, $\overline{L}$, $\overline{R}$, $\overline{T}$ and $\overline{B}$ through the points on PI that correspond to $\mathbf{pc}$, $\mathbf{pl}$, $\mathbf{pr}$, $\mathbf{pt}$, $\mathbf{pb} \in \mathbb{Z}^2$, using the initial guess for the projector's intrinsic parameters. We compute the distances dc_{id}, dl_{id}, dr_{id}, dt_{id} and db_{id} between those rays and the 3D positions marked by the user in the calibration process ($\mathbf{c_{id}}$, etc.) (see appendix A for distance computation). Finally we conduct a steepest descent search to optimize the projector calibration by minimizing the sum of the squared distances:

$$\min(\sum_{i=1}^{2}[(dc_i)^2 + (dl_i)^2 + (dr_i)^2 + (dt_i)^2 + (db_i)^2])$$

Therefore, we codify the projector calibration by the following 8 parameters: A 3-vector codifying the position of the projector's focal point relative to the camera center, a 3-vector codifying the orientation of the projector's coordinate system relative to the camera coordinate system (using the procedure to rotate one orientation into another from appendix A, 3 angles can be obtained) and a 2-vector holding the dimensions of the projector pyramid on PI.

3.2 Rendering the Warped Image

Having calibrated the camera-projector system, we can deal with pre-warping the projected image to fit the projection screen. As input for this the pose estimation system delivers the camera centric 3D coordinates $\mathbf{c_{id}} \in \mathbb{R}^3$, $id \in \{tl, tr, bl, br\}$ of the 4 corners of the projection screen. Those can be transformed into projector centric coordinates $\mathbf{p_{id}} \in \mathbb{R}^3$ by using the transformation $C2P$ described above. Re-projecting those onto PI yields points $\mathbf{r}_{id} = (r_x^{id}, r_y^{id}, 1)^T \in \mathbb{R}^3$ on PI. From those the pixel positions $(u_{id}, v_{id})^T \in \mathbb{R}^2$ of the projection screen's corners $\mathbf{c_{id}}$ can be computed by the formulas given above. A homography could be obtained from the 4 point correspondences between the original image's corners and the target corners.

Fortunately we utilize a 3D-engine for rendering and thus do not have to perform the texture mapping manually. Instead we define the projection matrix of the rendering system to match the projection pyramid of the projector. This allows us to simply place a 3D quad at the locations $\mathbf{p_{id}}$ (in projector centric coordinates) of the projection screen corners. This quad is then textured with the pre-rendered image, that our application produced. Figure 3.1 on page 40 shows a normally rendered image from FiberMesh and the same image rendered on a 3D quad matching the projection screen's position. This technique has also been used by other researchers (e.g. [RB01]).

Chapter 4

Implementation and Technical Details

In this chapter we first discuss the hardware we used and different projection screen prototypes that have been built throughout the the project. Then we will analyze some parameters of the system that are mostly implied by technical properties of the hardware. Finally we will discuss implementation issues which arose mainly due to using programs written in different programming languages.

4.1 Hardware

This section presents and discusses the different projection screen prototypes we have build, the circuitry developed for signalling button presses and the the 3rd party hardware, we have been using.

4.1.1 Screen Prototypes

The pose estimation object shown in figure 4.1, left is simply a circuit board with 4 IR-LEDs (LD271L) on it. The fourth LED (center of the board) is held in place above the plane by a wire. The prototype shown in figure 4.1 is substantially different from the first one. The feature points that can be tracked by the IR-camera are 5 ordinary light bulbs.

The prototype shown in figure 4.2, left contains 5 high opening angle IR-LEDs (Sharp GL100MD1MP1), which have a very small SMD form factor. This prototype was an experiment for finding out if the Wiimote could track more than 4 LEDs. On the circuit board there is a circuit that switches back and forth between the 4 outer LEDs and the inner LED. Switching forth and back between the two circuits is realized using a bilateral switch (HEF4066B) and a frequency generator.

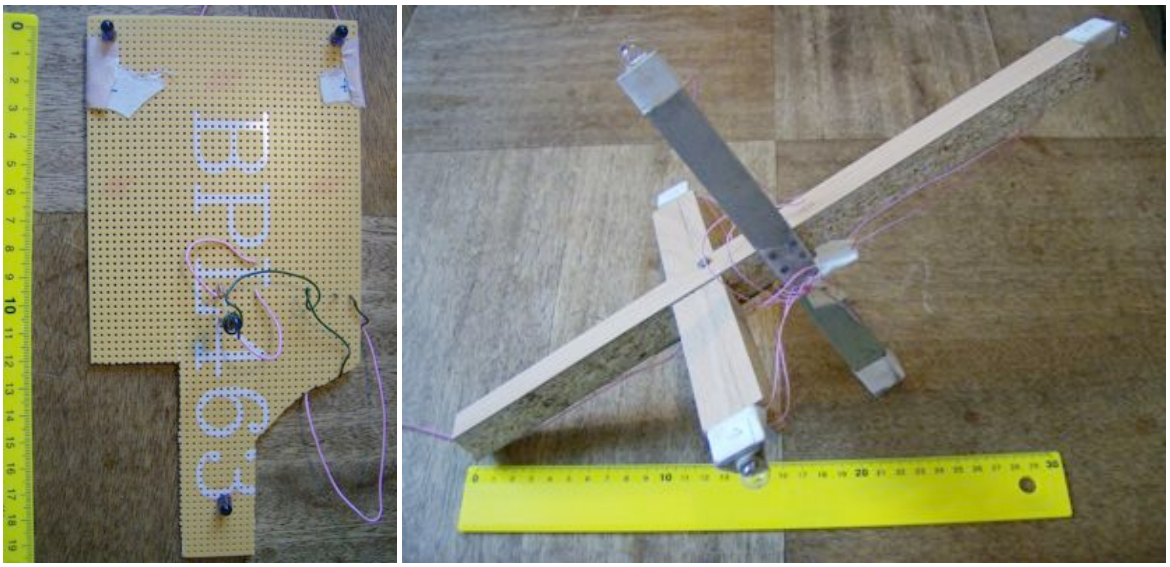


Figure 4.1: Left: Trackable circuit board with 4 narrow opening angles IR-LEDs, Right: Trackable object with light bulbs

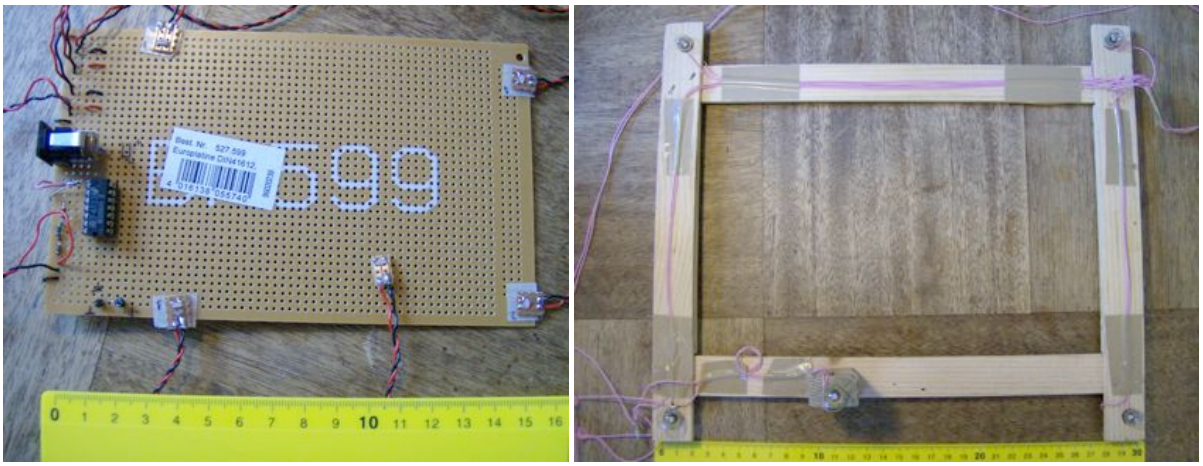


Figure 4.2: Left: Circuit board with 2 sets of alternately blinking LEDs, Right: Projection screen prototype with light bulbs

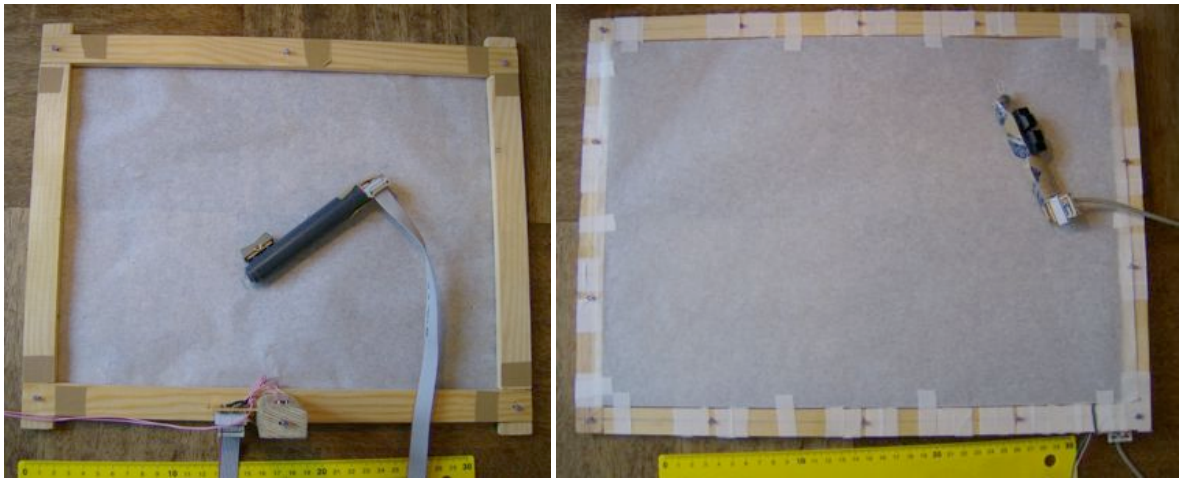


Figure 4.3: Projection screen prototypes with 6 and 14 (left and right) LEDs and pen

Figure 4.2, right shows our first prototype with projection screen form factor, which weights 122g. In the image no projection surface is attached to the frame. The screen has 5 regular light bulbs as feature points, one of which can be switched on and off independently for signalling a push button's state. The light bulb on the lower horizontal frame part is raised above the plane constituted by the other light bulbs by approximately 4cm.

The first projection screen that was tracked using IR-LEDs can be seen in figure 4.3, left. The LEDs used on this screen (SFH 487 P) have 3mm diameter and a high opening angle. Four of them are dedicated to pose tracking and 2 are used for signalling button presses, by being turned on or off. The LED on the bottom frame is raised by 7 cm above the other LEDs, which are roughly coplanar. The buttons are mounted on the light-pen, which is connected to the frame through a ribbon cable. The pen contains a low opening angle IR-LED as tip (TSHA 5203). Together with the pen, the screen weights 168g.

Figure 4.3, right shows the most recent projection screen prototype, having 14 coplanar IR-LEDs (SFH 487 P) on its frame. The middle LEDs along the horizontal frame parts are used for signalling button presses. The signalling method can be chosen between "blink-on-press" and "on-on-press" by jumpers at the user-side of the projection screen frame. An important feature of this prototype is that no materials producing highlights under the light of the projector were used. Therefore the adhesive tape on the frame is matte and visible under normal conditions. Together with the pen this screen weights 140g.

All projection screen prototypes supported by a wooden frame and the projection surface is made of translucent paper which has reasonable image quality (see fig. 1.1 on page 2) for a prototype at virtually no cost.

Discussion The first three prototypes are not suited for projecting on them, but important conclusions can be drawn from them: In terms of visibility regular light

bulbs can be used as features trackable by an IR camera, as they produce large blobs of IR-light which are visible from a high range of angles. The big blobs are on the one hand an advantage, as they can be easily found in the image. On the other hand, due to their size, computing their weighted barycenter takes longer than for the small light blobs produced by LEDs.

Also we found that "normal" IR-LEDs have opening angles that are too small for pose estimation. The first trackable object was only visible to the camera when the LEDs pointed almost directly towards the camera. The form factor of LEDs is also crucial for tracking them with an IR-camera: LEDs with SMD form factor are barely visible on the captured image due to the small size of their light source. Therefore, the usage of high opening angle LEDs with at least 3mm diameter (the SFH 487 P for example) is strongly recommended when using LEDs as trackable features.

Moreover, IR-LEDs have one important advantage over regular light bulbs: They are invisible to the human eye and therefore not interfering with the projected image. Light bulbs, in contrast, can render the projected image nearly unrecognizable due to their brightness.

Most of the prototype designs have at least one light source that is not coplanar to the other ones. The reason is a trigonometric one: Assume, the cameras viewing direction is perpendicular to the projection screen plane. Light sources that lie on the screen plane will have low changes in their image space position when rotating the plane for a fixed and small angle. The reason for this is that most of the rotation's movement for this case is parallel to the camera's viewing direction. In contrast to this, points that are raised above the screen plane will mainly move parallel to the camera's image plane, making this change clearly visible in image space. The only drawback of including a light source that is raised above the plane is the strange look and the additional weight of the part of the frame that holds the raised light source. For this reason and as we use an adaptation of the camera calibration algorithm of Zhang [Zha99] (that only handles a set of coplanar points) the inclusion of a raised light source is not desirable for the final prototype.

The last prototype has some clear advantages over the other ones: It is based on 3mm LEDs with high opening angle, and contains enough of those to minimize the effect of image noise. Additionally it has a pen and the capability to signal button presses with a method chosen by the user. Another clear advantage is that it does not produce any highlights in the camera image, when illuminated by the projector. Finally this screen is big enough to show a picture with decent size, while still being relatively lightweight.

4.1.2 Button Press Circuit

Our current projection screen prototype has 12 high opening angle IR-LEDs (SFH 487 P) on the frame that are dedicated to pose tracking. The LEDs are placed in 3 parallel circuits, each containing 4 LEDs in series. Additionally there are two

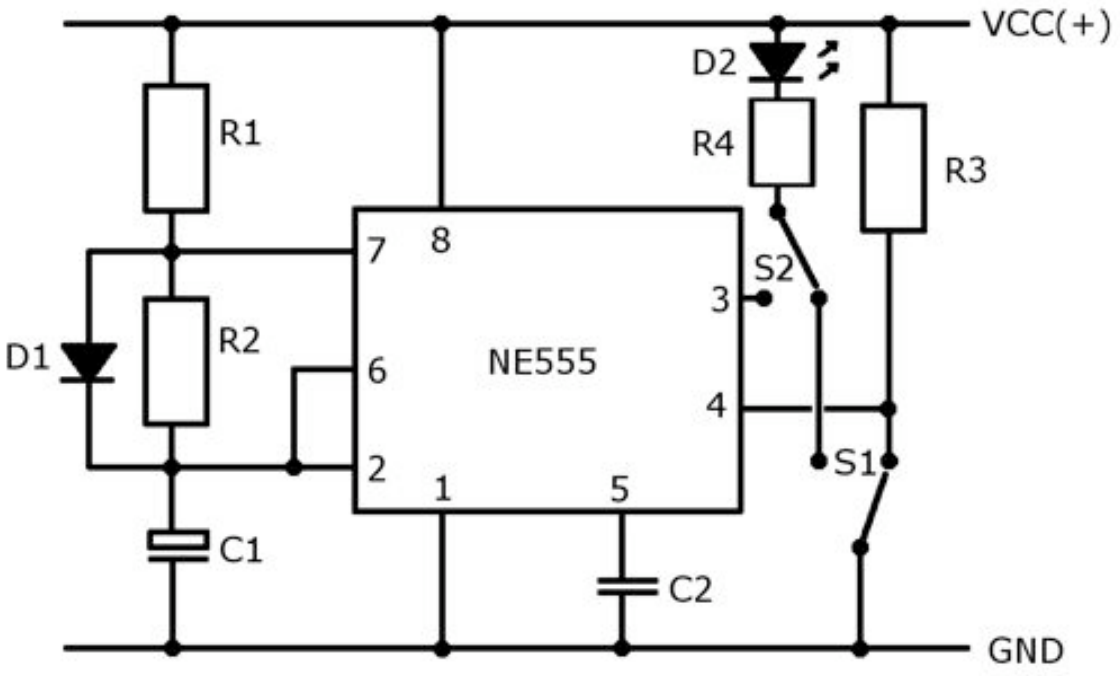


Figure 4.4: The timer circuit for button-press signalling

of those IR-LEDs in timer circuits (see fig. 4.4) that are parallel to the 3 LED-circuits. $S1$ is connected to the push button, that is associated with the IR-LED $D2$ in the circuit. The switch $S1$ is in the depicted position if the button is not pushed. The timer circuit has two states:

If S2 is in the state depicted in figure 4.4, switching S1 will make the LED shine constantly. As long as the button is not pressed, $S1$ is in the depicted state and the LED will not shine.

The other state is activated by switching S2 towards pin 3 of the NE555. In the depicted state, the LED will shine constantly as the reset pin of the NE555 is grounded. When the button is pressed, $S1$ gets switched which initially turns the LED off. Then it starts flashing with a frequency of ~16.5Hz (see sec. 2.4 for further details), where impulse- and break-times are equal. Here is a list with the components of the timer-circuit:

- Resistors: $R1 = R2 = 4.4\,k\Omega$, $R3 = 150\,k\Omega$, $R4 = 220\,\Omega$
- Diodes: $D1$ is a 1N4001 diode, $D2$ is a SFH 487 P Infrared LED.
- Capacitors: $C1$ is a $10\,\mu F$ electrolytic capacitor, $C2$ is a $10\,nF$ ceramic capacitor
- $S1$ is a push button (shown in unpressed state in fig.)
- $S2$ is a switch, implemented by 3 pins and a jumper

Figure 4.5: From left to right: Wiimote, USB Bluetooth receiver, FireWire camera, IR-pass filter, PCMCIA FireWire card, Power supply for the camera if using laptop

All the resistors in this circuit are $0.25W$ metal-film resistors. The supply voltage of $6V$, is provided by a wall power supply with variable voltages. But as the LEDs are not high power devices, a battery solution might work as well. As the push buttons, which implement $S1$ are located on the pen, a wire needs to connect the pen and the projection screen. For further development of the prototype a better solution for transmitting button press events might be more appropriate. One possible approach would be to use a Bluetooth transmitter to communicate those events.

4.1.3 Third Party Hardware

In this subsection we briefly mention the 3rd party hardware we use and have used, most of which can be seen in figure 4.5. Within the first phase of this diploma project, we used a standard Wii Remote (also known as Wiimote) together with a "ANYCOM USB-200" Bluetooth receiver and the "WIDCOMM Bluetooth Software 4.0.1.2400" of Kensington for image acquisition and blob detection. Although this combination works, we recommend looking for an alternative Bluetooth receiver on the WiiLi list of compatible devices[1], as setting up the Bluetooth drivers was a long series of trial and error.

The camera we use is a "DFK 21BF04" FireWire camera from "The Imaging Source", which we interface through the .NET library provided on the CD deliv-

[1] `http://www.wiili.org/index.php/Compatible_Bluetooth_Devices`, last accessed on 14.11.2008

ered with the camera. For connecting the camera to a laptop without FireWire (IEEE 1394) interface, we used the Dawicontrol "DC-1394 PCMCIA" card together with an "EGSTON N2EFSW3 12 VDC" power supply. The problem with laptops and FireWire cameras is that even laptops having a FireWire interface do not provide the power that most FireWire cameras expect to be delivered over the FireWire interface. Thus, an external power supply and an adapter are needed. One important part of the camera-driven image acquisition system is the IR-pass filter, which basically just consists of 2 exposed photo negatives glued onto the plastic cap of a mineral water bottle. Of course we had to cut a hole into the plastic cap in order to let the IR-light pass through the photo negatives onto the optics of the camera.

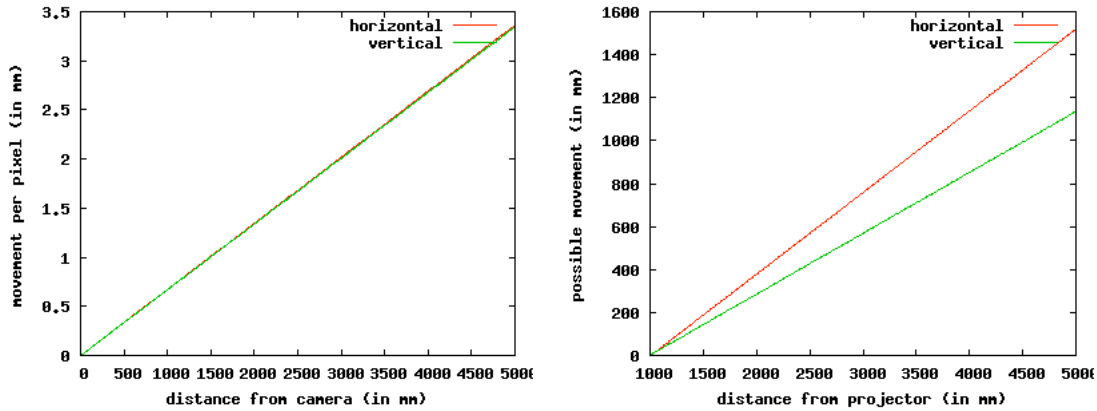
Our projector is a NEC LT35, but any other off-the-shelf projector with a high resolution is suited for projecting onto the projection screen as well.

4.2 Technical Parameters of the System

Beardsley et al. [BvBRF05] presented a hand-held projector which can be used to project on and interact with planar surfaces. In contrast to this our setup consists of a fixed projector and a hand-held projection screen. This imposes several constraints on the working area: The screen needs to stay within the projector beam for projecting on it and within the camera's field of view in order to estimate its pose. Therefore, the working area of the projection screen is limited to the intersection of the two fields of view, which is depicted in figure 3.2 on page 40. The setup that works best for our hardware is placing the camera approximately 1m behind the projector, with the viewing axes of the camera and the projector being roughly parallel.

To measure the parameters presented in this chapter, simulated poses have been utilized in order to obtain data for convenient analyzing. For authenticity the usage of real pose data would be preferable, but as the focus of this section is presenting strengths and limitations of our system we have opted for simulating poses. We used real calibration data for the simulation to stay as close to real applications as possible.

In order to relate the obtained data to real conditions we first want to present some measurements of noise in the estimated pose. Noise is already present in the camera image and in some cases amplified by numerical aspects of our algorithms. The data about noise was obtained while viewing the projection screen in a fixed pose, at a distance of approximately 2 meters from the camera. To measure the noise, we monitored blocks of images, each containing 100 subsequent camera images. From those blocks we took the 2 most distant measures or estimates. The two most distant pixel positions of the same blob normally differ about 0.2 to 0.3 pixels per block, where blocks with differences exceeding 0.5 pixels are rare. The noise in the estimated distance from the camera focal point typically ranged



(a) screen motion (mm) per pixel at different distances (mm) from the camera

(b) possible screen motion (mm) without leaving projector beam by distance (mm) from projector

Figure 4.6: Effects of distance movement within the projection screen's plane. Left: Movement in projection screen plane per movement by 1 pixel, Right: Possible movement before leaving projector beam

between 2 and 4 millimeters per block. The projection screen's estimated forward direction varied between 0.4° and 0.8° in most cases.

The following measurements assume that the center of the projection screen lies on the viewing axis of the camera or projector. When distance is the independent variable of a measurement, the projection plane normal is assumed to be parallel to the camera's or projector's viewing axis. All rotations are performed about the up-axis of the projector's coordinate system.

First of all we will examine some relations between camera-, projector- and real space. The two graphs in figure 4.6 show those relations: The left graph shows which movement within the plane of the projection screen is associated with a movement by one pixel in image space, depending on the distance from the camera. We can clearly see that the movement in 3D space that is imposed by displacing the image for 1 pixel grows linearly with increasing distance from the camera.

The graph on the right side of figure 4.6 shows how far the user can move the projection screen at a fixed distance from the projector from one side of the projector beam to the other side, without leaving it. With growing distance from the projector the allowed range of the projection screen grows linearly. Another observation we can make from this graph is that the projection screen will need to be at least 1m away from the projector in order to be completely filled by the projected image.

Next we will cover how the estimation of the the projection screen's orientation is influenced by distance d_c from and angle α_c to the camera. The left graph of figure 4.7 shows how far the positions of the LEDs move in image space when the projection screen is rotated by 5°. The dependent value is the average displace-

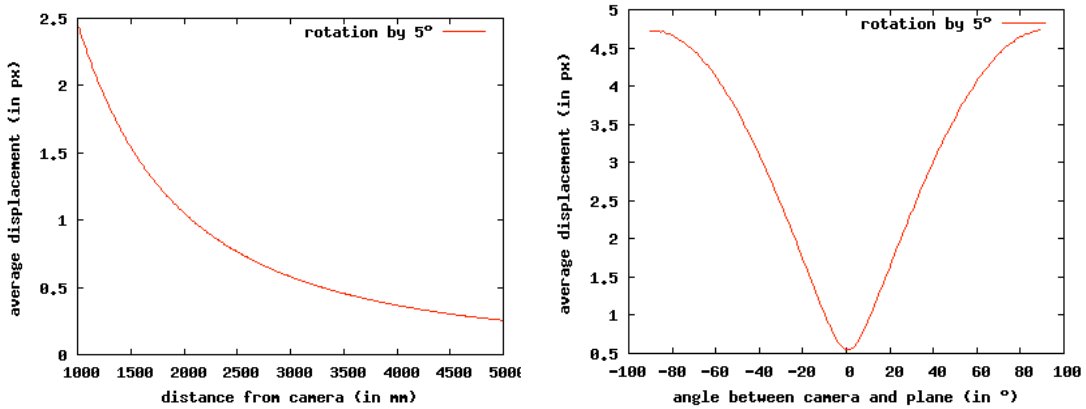


Figure 4.7: Effects of distance and angle on the pixel displacement for rotating the projection screen by 5°

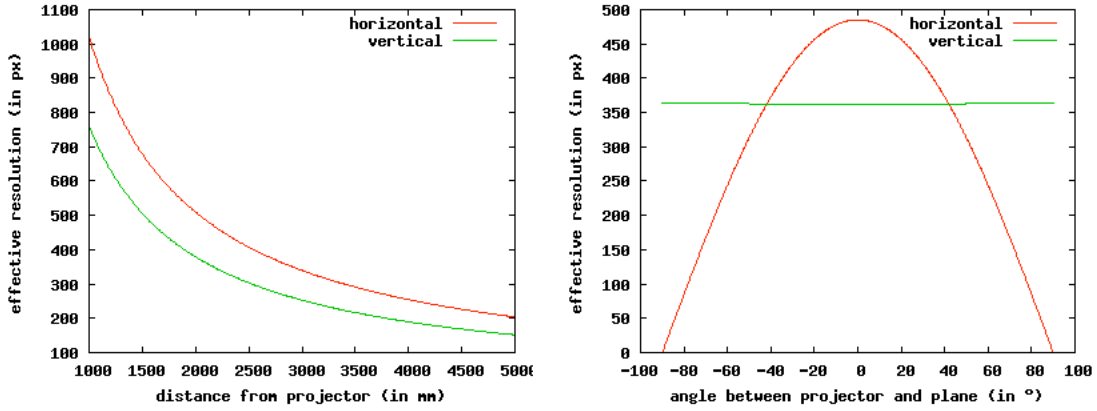


Figure 4.8: Effects of distance and angle on the effective resolution on the projection screen plane

ment of all LEDs of the projection screen in image space if applying this rotation. In real 3D space the average displacement remains the same, regardless of the distance to the camera. Therefore the graph additionally shows that for fixed displacements of points in 3D the displacement in image space drops at a rate of $\frac{1}{d_c}$ with increasing distance to the camera.

The aspect that is covered by the diagram on the right side of figure 4.7 is closely related to the geometry of the projection frame and the method of pose estimation: The current projection screen is a plane and all feature points on it are coplanar. The effect of this for orientation estimation can be seen in the diagram: on very low angles α_c (projection plane is nearly orthogonal to camera direction) changes in rotation are nearly imperceptible by means of pixel displacement. With growing α_c, the average displacement per pixel and 5° of rotation are strongly increasing.

The diagrams in figure 4.8 depict the influence of the distance d_p and angle α_p of the projection screen from the projector on the effective resolution on the

projection screen. The effective resolution is a measure for the achieved image quality and simply shows how many pixels of the projector beam fall onto the area of the projection screen. We measure it as the average distance of opposing vertices the projection screen. This value decreases with $\frac{1}{d_p}$, which is the same rate as the one for pixel displacement.

The right image of figure 4.8 shows the effect of rotation on the effective resolution, where α_P is the angle between projection screen plane and the projector's image plane: With increasing angles the effective horizontal resolution drops towards zero. The effect of rotation on the vertical resolution is a result of the perspective projection of the projector rays and it is small enough to be neglected for real applications.

Discussion Those measures have a high impact on interaction by the projection screen: The first two graphs are surprising, but they highlight several properties of the pose estimation and interaction with the projector: As the ratio of 3D movement per pixel displacement grows with increasing distance to the camera, noise in the image capturing process yields stronger noise in the screen position when the projection screen is farther away from the camera. Additionally, the precision of the pose estimation will decrease with increasing distance. On the other hand the freedom of horizontal and vertical projection screen movements increases with growing distance from the projector.

The same facts apply to the orientation estimate: Its precision suffers from distance to the camera. The impact of high distances on the orientation estimate is obvious when the average point position noise in image space with the average displacement for a rotation by 5° at high distances ($d_c > 4000$): The average noise (between 0.2 and 0.3 px) is half as big as the average displacement implied by a 5° rotation (< 0.5 px).

As the third graph shows, the orientation of the projection screen is a very important factor for the projection screen's orientation estimate as well. Holding the projection screen plane parallel to the camera's image plane has advantages as well as disadvantages: For the measurement of very fine orientation changes this is a bad setup, as small changes in the projection screen's orientation yield pixel displacements which can hardly be recognized. On the other hand noise will mainly be converted to position changes instead of orientation changes in this situation. If viewing an object from a distance, this is absolutely advantageous, but in case of viewing an object from very close noise in the rotation estimate is preferable over noise in the position estimate.

One fact that has not been measured directly is the effect of rotation on the dimension of our projection plane in the image. When rotating the plane towards being parallel to the rays towards the camera's focal point, one of the screen's dimensions nearly vanishes in the camera image. For position estimation this is a problem: The missing information increases the position estimations noise to values around 20mm within 100 frames (ground noise is 2-4 mm). Those changes

are dominated by the depth component of the position, as the information of one dimension can not be used to counteract the noise along the other dimension.

For the effective resolution there is not much to discuss: It decreases strongly with distance from the projector as well as with angle to the projector's image plane. The only thing we want to add here is that the implications of noise (in the camera image) of course also impact the projected image: which on the one hand shakes stronger due to higher noise in the pose estimation. On the other hand the 3D displacement for a fixed displacement in image space will be amplified with growing distances to the projector.

All those measures clearly show that with the current setup there are restrictions on the available working space: The projector beam and viewing area of the camera limit the movement parallel to their respective image planes. Those limits are less constraining when moving away from them, but with growing distance the effective resolution of the projected image and the precision of screen pose estimation decrease. Additionally the angle between the projection plane's normal and the viewing directions of camera and projector are in practice restricted to values below 90° as at that angle the LEDs become invisible to the camera. In addition to this absolute restriction, the impact of noise grows strongly at high angles.

Resulting from those observations we can formulate some guidelines for using the projection screen: At first, the angle between camera direction and projection direction should be low, as in general both devices yield optimal performance, when the projection screen plane is parallel to their image plane. Additionally the "base pose" of the projection screen should place the projection screen plane roughly parallel to the projector's and camera's image planes.

The second guideline is connected to distance from the projector and camera. The projection screen should be as close to the projector as possible in order to obtain a high effective resolution. Therefore the user should place the projection screen's base position at the closest distance to the projector that yields a comfortable freedom in horizontal and vertical movement of the projection plane. Additionally the camera and projector should be placed as close as possible to each other, respecting possible differences in the opening angle of their fields of view.

4.3 Software

In this section we describe some problems we have encountered while developing the software for our device. But first we briefly lay out the structure of our software system: The main application which contains most of the algorithms we presented is written in C#. FiberMesh, a tool for designing 3D freeform surfaces based on 2D sketches which can use our projection screen as interface is written in C++. The pose estimation module based on Zhang's camera calibration method [Zha99]

is written in C++, mainly for performance reasons. The reason for not using C++ is the experience of drastically increased ease of use and efficiency when working with C# in contrast to C++. In our case, this outweighs the slightly reduced performance, which can mainly be accounted to the garbage collection and run-time checks C# performs automatically.

The main problems that arose from using C# are due to its small age: As the language is not yet as established as for example C++, there are on the on hand less libraries than for for more established languages and on the other hand some of the existing libraries seem not to be tested thoroughly enough. In contrast to this for C++ there are many libraries which have proved their quality and performance through the years.

Additionally our target application, FiberMesh [NISA07], is written in C++ and needs an interface to communicate with the pose estimation library, which is written in C#. Another problem we had to face is the poor performance of C# when working on images through the safe indexer method (access through `[]`, as if it was an array) of `TIS.Imaging.ImageBuffer`. All imaging component used in the following, are in the namespace `TIS.Imaging`.

4.3.1 Performance Issues with C#

First we look at the performance of C# when reading the camera image through the indexer (array like access with `[]`) of `ImageBuffer`. Using this method on many pixels wastes enormous amounts of CPU-cycles, as each of the calls performs array bounds checks and probably some other checks as well. This problem can be solved by obtaining a pointer to the raw image data through `ICImagingControl.ImageActiveBuffer.GetImageDataPtr()`. Using the pointer to this array in unsafe mode yields a significant performance boost in comparison to accessing pixels by the indexer method.

The same is true for traversing or copying large arrays. Using the `fixed` keyword, a pointer to an array can be obtained. Using this pointer the data in the array can be accessed without array bounds checks and the like. Note that we do not recommend to use pointers in C# unless it is really necessary. As long as there are no performance problems usage of the normal array and image access methods is strongly recommended, as those will perform some useful run-time checks which will help to reveal common bugs quickly and without long debugging sessions.

4.3.2 Interoperability between C# and C(++)

Using C(++) code from within C# The scenario we want to examine in this part is using C or C++ code from within C#. Code from plain C dlls can be called from within C# and the mechanism for this is relatively simple: Assume, we have a dll called “PoseTrackerCParts.dll” and we want to call a method with the signature

```
__declspec(dllexport)
   void UpdateTrackerDll(float seen_by_object_id[])
```

that is defined and exported in this dll. In order to do this we define a method within our C# class that calls the desired method instead of defining an own method body. This can be achieved by the following C# code snippet:

```
[DllImport("PoseTrackerCParts.dll",
  EntryPoint = "UpdateTrackerDll")]
unsafe private extern static
  void UpdateTracker(float* seen_by_object_id);
```

By using the `DllImpport` attribute, we forward calls to `UpdateTracker` to the C method. Note that the array argument of the C method needs to be translated to a pointer and thus the method has to be `unsafe`. Before calling `UpdateTracker` in C# code we need to create a `float*` for usage as `seen_by_object_id`, which is big enough to hold the desired amount of floats. This can be done by using `stackalloc` and transferring the data to the allocated block in an `unsafe` environment in C#.

Using this approach it is possible to interface entire C libraries from C# in theory. Although this is possible we do not recommend it for the following reasons: For larger libraries writing all those forwarding methods is very tedious. This can be helped by using a framework like SWIG [Bea96] that automatically creates those methods for the user. Most likely extra rules will have to be defined in order to let SWIG handle uncommon code constructs of the library correctly. Additionally we experienced random problems with objects that have already been freed while using our SWIG-created interface to newmat10. Overhead is another issue that needs addressing: When calling C code from C#. Thus the approach is inappropriate for libraries, that are invoked very often as for instance newmat, that is invoked for each operation we perform on a matrix.

The approach we recommend is writing the main application in C# and outsourcing modules that make heavy usage of C(++) libraries to C(++). As interface between the C(++) code and C# a C-style dll with a small amount of interfacing functions can be utilized. Using this approach we have built a dll for the Camera calibration pose estimation that publishes an interface with 8 functions. Two of those functions are called once each update step. All functions called this way are not directly passing any references forth and back between C# and C code. Instead all communication is channeled through dedicated buffer objects. Calling C methods just once per update ensures that the overhead stays low and not sharing any permanent objects minimizes the amount of possible interop problems.

Using C# code from C++ code If the C++ application is managed it can directly access other .NET components by including them. If it is unmanaged accessing C# components is a bit more complicated. The solution we chose is communicating the results of pose estimation via networking protocols. Another method to access C# code from within C(++) is COM-object based interop.

An analysis of the necessary communication showed that we do not need to communicate very often per update. Therefore the overhead of using the network stack does not occur often. Additionally, using a networking protocol to communicate allows us to run both programs on different computers without stealing CPU cycles or graphics power from each other and without having to link them to one program.

The main information we need to communicate is the projection screen pose. This data has to be communicated for every frame. A pose can be completely described by either 3 3-vectors (position, forward direction and up direction) or by a 4×4 transformation matrix. Therefore, we decided to communicate this using the UDP protocol to minimize latency. Cursor position and button states are communicated using UDP as well. UDP of course has some drawbacks, that need to be taken into account: Normally the danger of packet-loss is the worst problem associated with UDP. As each message contains a complete description of the estimated screen pose, cursor position and the button states, this problem does not impact our implementation other than decreasing the temporal resolution of the tracking when packets are lost.

The second issue with UDP is far more interesting for our case: The messages do not necessarily arrive in the same order they have been send. This can cause problems for instance with button presses: Imagine a button is pressed and gets released. If the last message containing the "button down"-state arrives after a message signalling the "button up" state, the delayed message will be considered as a separate button press. This problem is easy to fix by adding successive numbers to the messages and dropping messages, that arrive after a successor in sending order has already been handled.

Information that is not as time-critical as the ongoing pose and cursor information is sent via TCP. As the TCP-protocol assures that all messages are received we only need to send those messages once and can be sure they will be received. Information being sent this way are for example the projector calibration and requests of the client to subscribe to the pose updates.

4.3.3 Third Party Tools and Libraries

There are several pieces of 3rd party software we use and they do of course deserve to be credited here:

We used the "Managed Library for Nintendo's Wiimote v1.1.0.0"[2] to acquire positions of IR-blobs as long as we used the Wiimote as tracking device. "The Imaging Source .NET"[3] is the software that comes with the FireWire camera we use. This library is utilized to set the FireWire camera into the appropriate

[2]Managed Library for Nintendo's Wiimote: `http://www.brianpeek.com/blog/pages/wiimotelib.aspx` , last accessed 14.11.08

[3]The Imaging Source: `http://www.theimagingsource.com/en/products/software/windows_sdks/icimagingcontrol/overview/` , last accessed 14.11.08

recording mode, extract image information and to show the live camera stream and augment it with information obtained in the blob extraction and pose estimation process.

newmat10D[4] is used throughout the code of pose estimation, which reimplements the camera calibration algorithm described in [Zha99]. ALGLIB [BB] is used for the multivariate optimizations needed to solve this camera calibration. During the reimplementation of this camera calibration method in C++, we used "CxImage"[5] to dump debug images to bitmaps.

SWIG[6] has been used to generate a C# wrapper for newmat10, which we discontinued in favor of the more convenient solution of writing the parts using newmat directly in C++.

[4]newmat: `http://www.robertnz.net/nm10.htm` , last accessed 14.11.08
[5]CxImage: `http://www.xdp.it/cximage.htm` , last accessed 14.11.08
[6]SWIG: `http://www.swig.org/` , last accessed 14.11.08

Chapter 5

User Interaction

This chapter deals with several aspects relevant for users that interact with software by using our hand-held projection screen. In order to discuss this subject a lot of different things have to be considered: One aspect of our projection screen is using the screen pose as 6DOF input device. A survey of existing techniques for 3D interaction was given by Hand in [Han97] and although various devices exist for 3 or more DOF input (some of them are shown and categorized in [Sub00]), controlling virtual cameras by mouse input still seems to be the dominant standard.

With our prototype we hope to enrich the range of possible interactions with virtual 3D worlds and objects. Our approach of using a fixed projector to project on a hand-held projection screen has strong impact on the working area of the projection screen. The implications of this on user interaction are investigated within the first section of this chapter. Thereafter, we will discuss how the an unstable pose estimate can be smoothed, as a stable projection screen pose is absolutely necessary for comfortable interaction. Finally we present some control schemes for the virtual camera employing the ability of our projection screen to be used as a 6DOF pose tracking device. Those camera control schemes are then evaluated in an informal user study.

5.1 Interaction within the Constrained Space

One issue with the limited working space is communicating those limits to the user. Most users notice when they leave the projector beam, as this turns parts of the projection screen black. Due to the lack of feedback most users do not notice when they leave the viewing space of the camera. One reason for this is that the pose estimation system can keep up a pose estimate with some missing points, although this decreases the precision and stability of the pose estimate. When

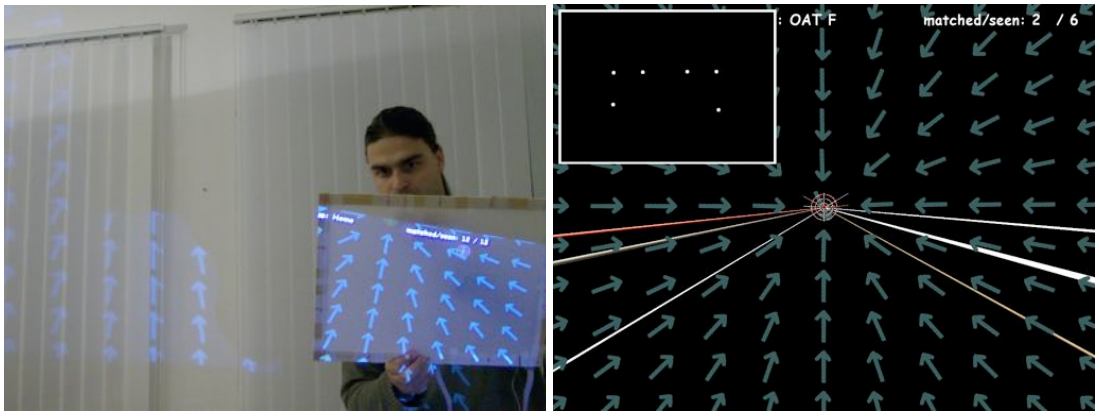


Figure 5.1: Left: User calibrating the projector pose, Right: "Good state" of the calibration process

leaving the camera viewing pyramid entirely, the pose estimation can obviously not be updated anymore.

One occasion where those problems are especially observable is the process of calibrating camera and projector pose (see sec. 3.1). During this process the system's user has to mark two sets of 5 points. The first set at a high distance from the projector and the second set at a smaller distance. If the pose estimation of the projection screen is imprecise when marking one of the points, i.e. due to LEDs leaving the camera image, the complete calibration process will yield a bad calibration. If the user calibrating the system does not re-mark the point the whole calibration process is useless and has to be restarted. Therefore, communicating the limits of the camera's field of view to the user is an important usability feature.

Before discussing how to help the user to stay within the camera's field of view, we shortly describe the calibration process from a user perspective. In figure 5.1, left a user calibrates the projector. The user can only see the part of the projection that falls onto the projection screen. The parts that can be seen on the wall, are not visible to him. Remember that the user needs to mark points on a ray in 3D space. The ray which the user is supposed to mark is visualized by projecting a small cross-hair with the image (as can be seen in fig. 5.1, right in the middle of the screen). This in connection with not seeing the whole projection leads to a problem: If the cross-hair is projected on the wall instead of showing up on the projection screen, The user has to search the whole area of the projector beam in order to find the target. To improve on this lack of usability we filled the whole projection screen with green arrows that guide towards the point the user is supposed to click on.

Back to the original problem of leaving the camera's sensing area: When one of the visible IR-LEDs is too close to the border of the camera image, we use arrows to guide the user away from the image's border. In order to underline the "unsafe" position of one of the LEDs those arrows are rendered in red. They point away from the camera image's border and the saturation of their red color increases with

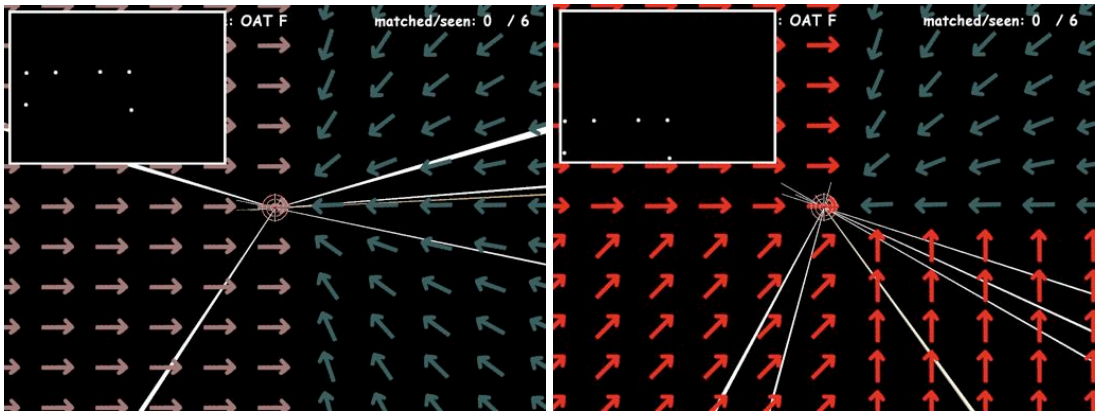


Figure 5.2: Optical warnings, communicating the restrained pose recognition space. Left: Projection screen close to camera image's left border, Right: Projection almost leaving the camera image

growing proximity to the camera image's border. The images in figure 5.1, right and figure 5.2 show screenshots of the calibration program along with possible camera images yielding the shown screen-shot. The lines that can be seen in the screenshots are representations of the observed camera rays.

The screen-shot in figure 5.1 on the facing page, right shows a "good state" of the calibration process. The blobs of the IR-LEDs are sufficiently far away from the borders of the camera's viewing area. The second image shows a situation in which the system starts to warn the user that at least one of the observed blobs is dangerously close to the border of the camera's image. The third image shows a situation with three of the IR-LEDs almost leaving the camera's field of view. Note, that showing the red arrows on the correct border of the projected image needs prior knowledge of the approximate relation between camera and projector pose. The screenshots are rendered under the assumption that camera's and projector's orientations are roughly the same.

5.2 Smoothing

As mentioned while discussing the working parameters of the system there is significant noise in the pose estimation delivered by our system. Noise is already recognized in the position of the blobs extracted directly from the camera image. In addition to this all operations the system performs can introduce numerical errors. Especially when using pose estimation with a low count of observed points the screen pose can vary strongly between two frames due noise and numerical instabilities. If those problems are not addressed users will not be able or willing to work with our system as the projected image shakes on the projection screen and the camera direction is uncomfortably unstable.

Exponential smoothing The computationally easiest approach to solve this problem is the "no trend, no seasons"-variant of exponential smoothing as described for example in [Gar06]: This method computes a new estimate e by using a weighted (with smoothing factor $s \in [0,..1]$) sum of the last estimate e_l and the new measure e_n of the value: $e = (1-s) \cdot e_l + s \cdot e_n$.

Moving average Another approach is to use a moving average, which we denote by $\overline{e}$, of the last n measures $e_k, k \in \{1,...,n\}$: To do this computationally quick, we no not recompute the whole average in each step. Instead, we maintain a list holding the last n estimates. When we get a new estimate e_{n+1}, we modify our average as follows: $\overline{e} = \overline{e} + \frac{e_{n+1}}{n} - \frac{e_1}{n}$. Additionally, we remove the oldest estimate e_1 from the head of our list and append e_{n+1} on its tail, which moves all entries 1 place forward (e_2 becomes the new e_1 etc.). To initialize the average to a value $\overline{e}_s$, we set $\overline{e} = \overline{e_s}$ and initialize the elements with $e_k = \overline{e_s}, \forall k \in \{1,...,n\}$.

Adaptive exponential smoothing In order to describe this extension to exponential smoothing let us define the difference $d = |e_n - e_s|$ between the current pose estimate e_n and the smoothed pose e_s. The noise threshold t_n and the signal threshold t_s, with $t_n < t_s$ are preset. Those parameters determine at which multiple of the average noise $\overline{d}$ in the smoothed variable the difference d will be considered to be a meaningful signal. If the difference is below the noise threshold ($d \leq t_n \cdot \overline{d}$) we use a high smoothing factor $s = s_n$ in order to keep the pose stable. If the difference is above the signal threshold ($t_s \cdot \overline{d} \leq d$) a low smoothing factor $s = s_s$ is applied in order to quickly adapt the smoothed pose to the new pose estimate. If $t_s \cdot \overline{d} < d < t_n \cdot \overline{d}$, we linearly interpolate between s_n and s_s.

When using this approach the estimated pose looks jumpy if the current frame's smoothing differs strongly from the last smoothing. Users observe the rapidly changing smoothing factor as discontinuity in the movement. This effect is especially annoying when the algorithm switches forth and back between a high and a low smoothing factor. To avoid this effect we restrict changes in the smoothing factor to 0.1 per frame.

This approach can easily be expanded to vectors by applying the smoothing to each of the components of a vector. For distance measurement we use the Euclidean distance between the new measure and the smoothed estimate.

Noise by average We have not yet specified, how we obtain the average noise $\overline{d}$ for adaptive smoothing. One approach to obtain $\overline{d}$ is using the moving average of the changes within the last x seconds. When using this approach we have set the thresholds t_n and t_s to 1 and 2 and the smoothing factors to $s_n = 0.9$ and $s_s = 0.1$.

Discussion Exponential smoothing and moving average smoothing both suffer from the same problem: If we have to deal with high noise, we need to use very

strong smoothing (high smoothing factor s or number n of estimates used for averaging). This in turn makes the smoothed values (projection screen pose estimate) adapt very slowly to real changes in the measured variable (real projection screen pose). This is a problem as we project the image onto the projection screen and thus the user experiences the slow adaption as a displacement between the real projection screen position and the position on that the image is projected.

Having discovered this problem, we evaluated several sophisticated approaches to smoothing in order to improve the user's experience. The evaluated algorithms include the well known Wiener filter [Wie64] and the Kalman Filter [Kal60]. The Wiener filter is only applicable for stationary stochastic processes. As noted while analyzing the physical limitations of our projection system our estimated pose has varying noise depending on its relative pose to the camera. Therefore the Wiener filter is not suited for our needs. In contrast to this the Kalman filter [Kal60] is able to cope with non-stationary stochastic processes and has previously been applied to different pose tracking problems [WW92, WBV+99]. The Kalman filter, however, seemed too complex for the given smoothing task and therefore we decided to try if an adaptive exponential smoothing approach could be sufficient. The most promising adaptive exponential smoothing method mentioned in [Gar06] has fixed parameters for the adaption. For a fixed distance of the projection screen to the camera and relatively small angles between screen and camera image, the approach yielded a stable pose. But when moving farther away from the camera or increasing the angle towards it, noise could not be filtered out sufficiently. Thus, this approach is not suited for our scenario as well.

As the noise for a fixed distance and angle is constant and user-conducted changes have a much higher amplitude than the noise, the adaptive exponential smoothing approach described above works good, provided the noise estimate is good. It smooths the pose estimate strongly as long as the currently smoothed pose is close enough to the last pose estimate. As soon as the two poses differ stronger than average noise can explain, the smoothing is decreased in order to adapt quickly to the new real pose of the projection screen.

When using the average difference between subsequent pose estimates as estimate $\overline{d}$ for the average noise, the number of frames about which the average change $\overline{d}$ in the smoothed variable is build is extremely important and implies a trade-off: When setting the frame amount too low, the noise estimate $\overline{d}$ increases for relatively short movement periods of the projection screen. This leads to a noise level filtering for the rest of the movement. If setting $\overline{d}$ too high the smoothing will adapt too slow to real increases in the average noise, as for example induced by high rotations of the projection screen against the camera. Currently we average over the last 600 differences.

Closing the discussion, we want to add that smoothing the positions of the extracted blobs yielded a smoothed pose which felt disconnected to the real changes in pose. The current implementation smooths the position, viewing direction and up-direction of the estimated screen pose separately. This decreases maximal noise in the position from 2 to 4 millimeters within 100 frames to 0.9 to 1.5.

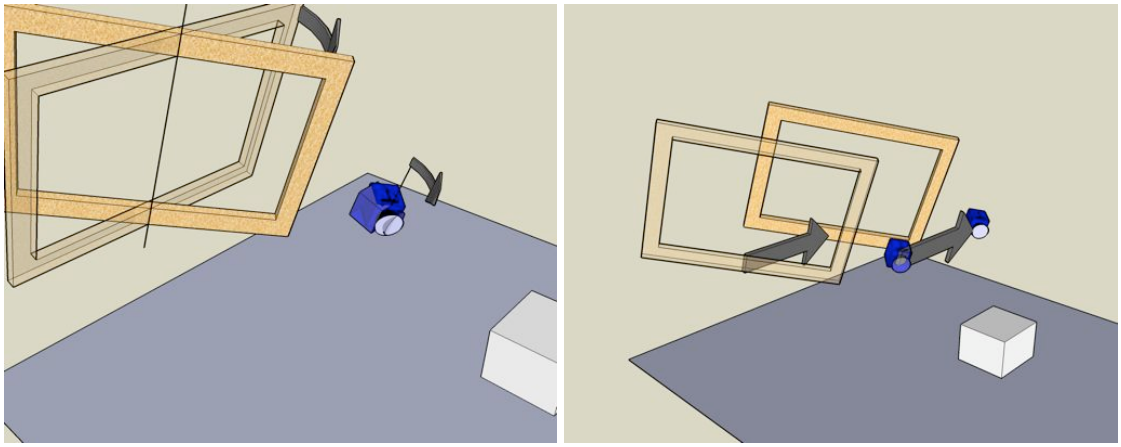


Figure 5.3: Absolute camera control: Projection screen frame, virtual camera and viewed object. Left: Rotation, Right: Translation

The noise in the forward direction can be reduced from 0.4° to 0.8° towards values between 0.1° to 0.3°. This results in a subjectively stable screen pose in most situations.

5.3 Camera Control

Before presenting the camera control algorithms we implemented we will give a quick overview of previous work in this area: The first researchers having investigated methods to use a 6 DOF interface for controlling the virtual camera in a non VR-environment seem to be Ware et al [WJ88, WO90]. Three years later Fitzmaurice et al. investigated methods to couple pose tracking with a hand-held display device [FZC93]. Another publication of relevance for exploring navigation through a spatial input device is the survey of Hinckley et al. [HPGK94] in which a consolidation of design issues for spatial input devices is given. Ongoing efforts to improve camera positioning algorithms through 2D input [HSH04, KKS+05, FMM+08, KMF+08] are additionally underlining the relevance of controlling virtual cameras intuitively and efficiently.

Absolute camera control Our first method to control the camera pose is directly mapping the absolute pose of the projection screen to the camera pose. This means that our projection screen can be understood as a hand-held camera for the virtual environment. Each movement or rotation of the projection screen is directly translated into movement or rotation of the camera from which our virtual scene is observed. Figure 5.3 shows two examples of this. Note, that the rotation axis of the virtual camera always contains the focal point of the camera. Another noteworthy thing is that translation is not re-targeted within a new coordinate system. This method follows the “Eyeball in hand” metaphor as described

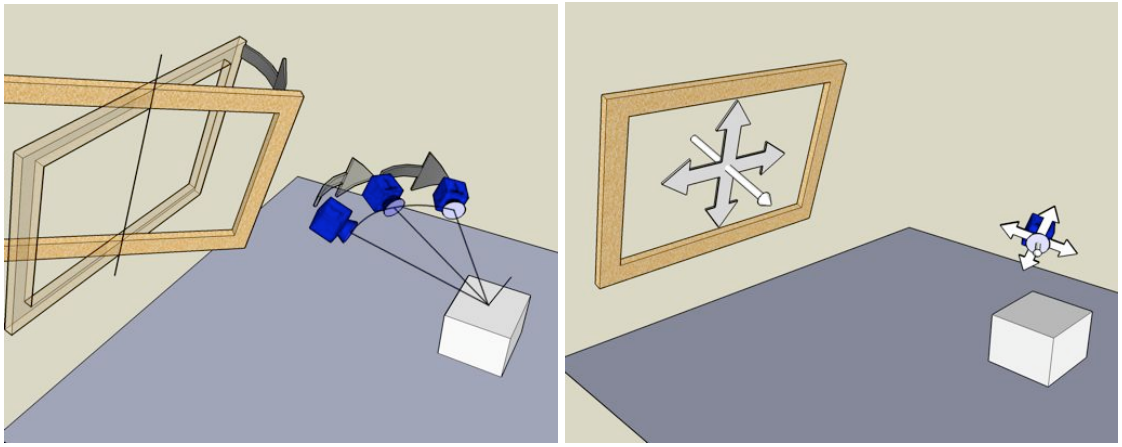


Figure 5.4: Relative camera control: Projection screen frame, virtual camera and viewed object. Left: Rotation continues over time, Right: Translation is mapped to the camera's reference frame

in [WO90].

A possible extension to absolute camera pose control is to give the user the possibility to fix the current camera pose relative to the object and therefore to move the projection screen without imposing changes on the view. A metaphor for this is "grabbing" the observed object, which is inverse to the approach presented by Rekimoto[Rek96]: Rekimoto changed the view using the palmtops orientation in a "World in hand" manner as long as a button on the palmtop was pressed.

Relative camera control This approach is based on orbiting the modeled object: For rotation we use a relative approach instead of absolutely coupling the cameras orientation to the projection screen pose. A centered orientation of the projection frame is defined by the user at first. As long as the projection screen leaves this orientation the camera constantly rotates around the modeled object (see fig. 5.4, left). The further the projection frame is rotated from its centered rotation, the faster the camera rotates. This follows Ware's "Flight vehicle" metaphor [WO90]. Let α denote the deviation of the projection screen's orientation from the centered pose and β be the turning speed of the virtual camera. We use a quadratic mapping from α to β, which allows precise camera orientation adjustments on low αs and rapid rotations on high values of α. In contrast to the absolute camera control, translation is remapped in the relative approach: When moving the projection screen, the movement is translated into the virtual camera's frame of reference (see fig. 5.4, right).

The rotation mapping of the relative camera control has one drawback: On low values of α the camera rotates at a speed that is hardly recognizable, but will nonetheless change the view over time. To let the user notice if and how fast the camera rotates we added arrows indicating the direction of the current camera rotation. The arrows are rendered translucent and the translucency is degraded

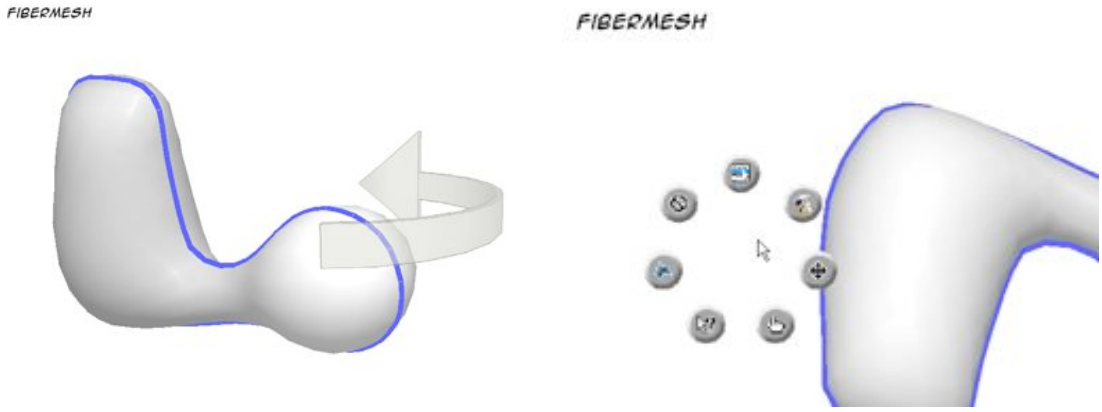


Figure 5.5: Left: Arrows indicating rotation around the object, Right: Circular menu in FiberMesh

(from 100% for no rotation) with increasing rotation speed (see fig. 5.5, left).

Another issue is that users change their posture and position over time, changing the pose of the projection screen as well. In order to allow for these changes without leading to constant rotation of the camera, our algorithm slowly adapts the centered pose towards the current pose.

One modification to the relative camera control is overlaying its rotation control with an absolute one. This gives an early indication in which direction the camera will start rotating as soon as the threshold for leaving the centered pose is left. This control method will be referred to as "Combined camera control 1".

The second modification does not change anything on the camera control method itself. Instead it slowly changes the model pose to fit the screen. Therefore the z-buffer of the graphics card is read after rendering, identifying pixels that are holding parts of the 3D model. Using this information, the modeled object is slowly translated to appear centered on the screen. Additionally it is moved towards the camera or away from it in order to use the screen space as efficiently as possible. We call this modification "Combined camera control 2".

Discussion Most discussion about camera control types is postponed to section 5.5, as it is highly subjective. As completely exploring a 3D object with the absolute camera control would be impossible without the modification of "holding" the observed object, we do not offer testers a separate control method without this modification.

Similarly, using the relative camera controls was absolutely awkward without communicating to the user when the camera rotates and when it does not. Therefore, the simple but effective solution of showing "rotation arrows" is included in all relative camera controls. A possible drawback of adapting the centered pose to the current screen pose are situations in which users want to rotate around the object for an extended period of time. This would mistakenly change the cen-

tered position. As such a situation is unlikely for 3D modeling applications, we included the centered pose adaption in all implementations shown to testers.

5.4 Usage of FiberMesh

FiberMesh's interface for input is based on two main components: Mouse interaction and keyboard shortcuts. Although most interaction with the program is mouse driven, for some tasks like switching to another tool it is necessary to use the keyboard. While the keyboard is still available when using the projection screen, it is not even arbitrarily convenient to access it while holding the projection screen.

Therefore, we use a circular menu that can by accessed by pressing the right pen button. The menu includes buttons for tool switching, resetting the program, toggling through the different camera control modes and for undoing the last change (see fig. 5.5, right). We evaluated the idea to use screen pose-driven menus like the one presented by Rekimoto [Rek96], but we decided to keep the circular menu. On the one hand, most people are used to interact with mouse-driven menus and on the other hand using the screen pose for menu interaction could have undesirable effects on the camera pose when leaving a menu. As we want to evaluate the performance of camera handling we decided not to add additional unfamiliar elements to FiberMesh which could have an influence on the overall usability of the system.

One interesting aspect of using FiberMesh with the screen pose driven camera control is that we suppress updates of the camera pose while drawing. This reduces the amount of information users have to take into account and allows them to fully focus on the 2D position of the pen on the screen while drawing.

On the technical side of the adaptation of FiberMesh there were two issues: First, we had to implement the projection mechanism described in section 3.1. Closely related to this, we have to render the mouse cursor onto the screen instead of relying on the operating system's cursor. Otherwise the user would not see the cursor at the right position of the image, as of course the operating system's cursor is not warped together with our application's image. The second issue is that the original FiberMesh does not have a mechanism to handle 3D camera pose. All the programming interface allows for is panning, zooming and rotating the object. As FiberMesh relies on the position of the object to be at the coordinate origin, we have to emulate real camera positioning by using those tools. For some users this emulation seems to feel wrong.

5.5 Informal User Study

We conducted an informal study with 7 users that were asked to evaluate the different camera control modes by filling out a questionnaire. Therefore, they were

introduced to FiberMesh and given time create 3D models while using the different camera control schemes. After giving their subjective overall rating for each camera control mode, they were asked to evaluate the different camera control types under different aspects. All questions in the questionnaire were posed together with an answer-scale of 5 different ratings. The lowest and highest rating were juxtaposed with a verbal description of the rating (see appendix B for the questions and answer-scales). In the following we present the results of the study and draw some first conclusions. As the number of participants is very low and the interviews with users were informal and without a fixed interview structure, our data is not sufficient for a formal study. Therefore we do not statistically analyze the data for more than mean values of the different ratings.

The first page of the questionnaire contains an introductory text together with the request to give a subjective rating of the different camera control modes. The following pages of the questionnaire contain the same set of 4 questions and a field for free-form feedback for each of the camera control modes. The participants of the user study did not see the detailed questions before filling in their subjective evaluations in order to not bias them in their judgment by proposing possible measurements for the control mechanisms.

The design of the detailed questions was guided by our understanding, which features users could expect from a good camera system. One commonly used model for measuring the performance of input methods is Fitt's law[Mac92]. As our study is informal, we did not perform any tests for conformity with Fitt's law. Nonetheless, we extracted two important parameters of the Fitt's law tests: task completion time and required exactness to complete a task. These are the basis for two of the four detail questions. Additionally our stated aim is to make the usage of our system as intuitive to novice users as possible while increasing the ease of use for users which are already experienced in 3D navigation. This lead to the other two detailed questions.

The first detailed question was how fast the user could become familiar with the camera control in question. The second question asked how clear the mechanics of the camera control were after getting familiar with it. In the third question the user was asked to evaluate the precision with that a desired pose could be established and the fourth asked for the quickness of establishing a desired pose (see fig. 5.7 for the results).

In figure 5.6 the mean subjective rating is shown against the mean of averaged (per user) detail ratings for each camera control scheme. Seemingly the detailed questions have a tendency to be answered slightly more positive than the overall ratings. It is noteworthy as well that the pen and the absolute control schemes have a significantly lower subjective rating than the average of their detail-ratings would suggest. Thus it seems that either our detailed measures are not exhaustive or that assuming equal weights for the detailed measurements is wrong. For conducting a formal study on the camera controls, we suggest analyzing the correlations between the subjective ratings and the detailed ratings.

Figure 5.7 shows the mean results of the detailed questions. Two things show

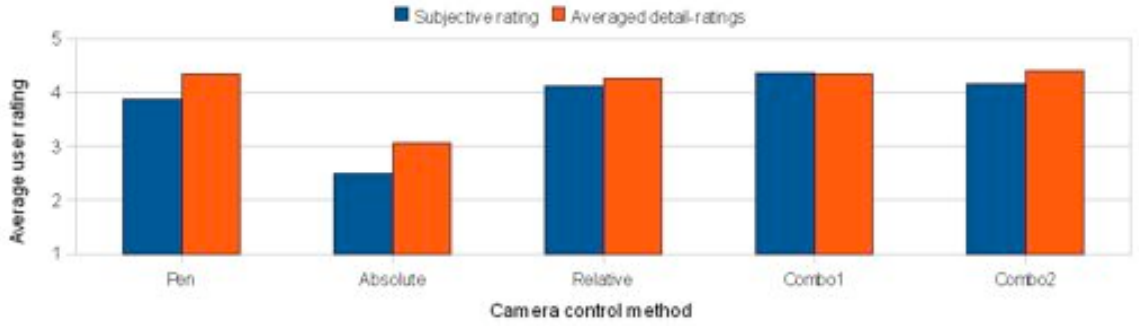


Figure 5.6: Subjective and average user ratings for the different camera control methods

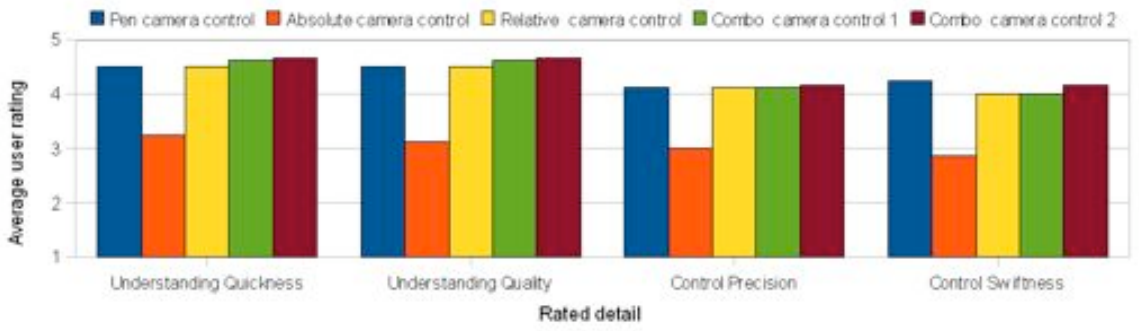


Figure 5.7: Rating means of detailed questions for the different camera control methods

up right away in this view: On the one hand the quickness of adapting to a given control scheme seems to correlate strongly to the overall understanding of the scheme after adaption. On the other hand it seems, the problem of making a control scheme understandable is not as hard to solve as establishing a feeling of precise and quick control.

Now we will examine the different control methods more detailed: The pen camera control was most of all thought as a reference control. It was a direct translation from the mouse control which is included in the standard version of FiberMesh. For translation the user can drag the object by right clicking on it and rotation can be achieved by right clicking and dragging outside of the model. The underlying control schemes are panning and the virtual trackball, which has been improved and analyzed in various publications (see [HSH04] for example). This methods for controlling the view on an object are widely established in mouse-controlled systems. Therefore, the good ratings for this methods, especially from experienced 3D modelers, were anticipated. This input scheme seems to be the most polarizing one: Four of the testers rated this control with their subjectively highest score and three users gave it the lowest of their subjective ratings.

The ratings of the absolute camera control mirror that it suffers from several problems: In spite of being able to fix the modeled object relative to the camera, most testers felt they needed very long to establish a desired camera pose. When trying to rotate around the model by 180°, using rotation and holding the object alternatingly seems to be a difficult mental task. This control scheme seems to be hard to grasp for most people and even precision seems to be an issue, here. This camera control was chosen to be the worst one by most users. Surprisingly one tester reported this to be his distinct favorite.

Most participants of the test could not name the features distinguishing the control schemes Relative, Combined1 and Combined2. As the differences are very subtle this was expected. Some of them reported to feel a slight subjective difference, although they could not name it. Surprisingly several users said things like "Why does the object grow?" while using Combined2, but did not attribute this to the camera control method. One tester even mentioned that the growing object confused him during the test, without attributing it to the camera control method. For some users the arrows that indicate camera rotation in the relative control schemes pointed in the subjectively wrong direction in the first place. Thus it seems to be a good idea to make the arrows invertible, as well as the rotation direction (which would be the relative pendant to Ware's "Scene in hand" metaphor [WO90]). Another tester suggested to allow panning the modeled object in the control schemes with relative camera rotation.

One finding of the study is that modeling with our current prototype for extended periods of time would not be possible. The testers reported strong fatigue of the fingers which held the projection screen. Another usability issue reported by the testers was that they could not hold the projection screen static enough for the camera pose to feel static. This suggests that fixing the screen in some way might be a good idea for modeling purposes.

Although for modeling the hand-held projection screen has some drawbacks, we believe that it works very well for other use-cases. After the usability study with FiberMesh our testers were given the chance to freely explore a big textured cube within the limits of the tracking and projection system. The control scheme for this was fixed to the absolute camera control. Most testers who are inexperienced with navigating virtual 3D environments reported this approach did not feel natural at first. A possible explanation: People are used to having the center of their visual sensation at their eyes and the camera pose in this case is controlled by their hand-movement. The testers who could overcome this initial problem, reported a convenient feeling of control over the camera pose for exploring the cube. Possible reasons for the discrepancy between this scenario and FiberMesh are the missing necessity to watch the object from all sides and not needing to exactly position the cursor at a given point on the object. Anther possible explanation for the discrepancy is the camera pose emulation we had to implement in FiberMesh (see section 5.4).

Especially for exploration tasks we want to suggest trying a setup, where the users head is tracked in addition to the screen pose. This way the virtual camera could be placed at the users head and the projection screen could be used as a window, instead of the current hand-held camera metaphor. This might cause less confusion than the current setup.

Nonetheless, in terms of camera control methods our relative approaches seem to be as usable as the pen driven camera control as far as we can tell from the user ratings. This suggests that further exploration of those methods may yield intuitive and at the same time efficient ways of camera pose control even for 3D modeling.

Chapter 6

Discussion and Future Work

Throughout this thesis we analyzed an approach to use a spatially aware display for 3D navigation. Our efforts are specifically aimed at increasing the ease of use of 3D modeling software. The pose of the projection screen, on which our hardware is based, is tracked using the camera image of IR-LEDs that are mounted on the projection screen's frame. After initially calibrating our system the 3D-modeling application's image can be pre-warped to fit the hand-held projection screen. Users can employ a pen to point, click and draw on the screen, which makes the screen a two handed input device having 8 degrees of freedom.

First we laid out how image coordinates for the IR-LEDs on the projection screen can be acquired algorithmically and how they can be matched to the known model points. Next we discussed different methods of estimating the projection screen's pose given image coordinates, model coordinates and the matching. The following part of the thesis dealt with using the estimated screen pose: Tracking of the pen's position has been explained and we have shown how to calibrate the camera-projector system and based on this pre-warp the image in order to fit the projection screen. After discussing those mathematical and algorithmic aspects of the system, we laid out the specifics of the projection screen hardware and some implementation issues we encountered. The final chapter was dedicated to the investigation of the projection screen's usability per se and as a means to control the view in a 3D modeling application. The usability section was concluded by presentation and discussion of an informal user study about using the frame as a camera controlling device for 3D modeling.

Our main testing application is FiberMesh [NISA07], an application for sketch based creation of free form surfaces, within which users can successfully utilize our system to create 3D models. Additionally, one of our camera controlling approaches seems to be as convincing as the original approach using pan and zoom for translation and a virtual sphere for rotation of the model. In spite of those facts we have found that in its current state our system is not yet perfectly

suited to meet the needs of interfacing a 3D modeling application. This has several reasons:

First of all, the calibration between camera and projector is not completely accurate. For pure projection of an image onto the screen it is absolutely sufficient and exploring a 3D object using the projection screen is a comfortable experience. As the user controls the cursor by pointing onto the projected image even the slightest calibration error is recognizable as an awkward displacement between the cursor and the pen's tip. Steps towards improving the calibration between projector and camera could either be collecting more data when initially calibrating, or improving the calibration automatically and continuously after calibration. The latter one would require a method to relate the pose estimation to the projected image. Approaches based on intermittent infrared projections [LHD07] could be used to acchieve this. A different way to relate the frame pose and projected image to each other is the usage of a tracking methods without IR-LEDs, which could for instance be based on extracting the projection screen's SIFT features [Low04]. One problem that needed to be solved in this setting is the brightness of the projected image: The extremely bright projected image could inhibit the extraction of meaningful information about the projection screen's features.

Another limitation of the system is that it works only within the relatively small portion of space covered by both the projector and the camera pyramid. In this area future work could investigate a setup with several cameras and projectors in order enlarge the operation space of the system. A second approach could involve a camera/projector system like the one presented by Pinhanez [Pin01] which could enlarge the usable area by projecting images using a rotatable mirror and a rotating camera.

The effective resolution of the image on the projection screen should also be improved for higher distances from the projector. Closely related to this, the size of the projection screen in the camera's image makes pose estimation less accurate and more sensitive to noise at high distances. Solutions to this problem could involve zoom optics in the camera and in the projector. While those would improve the pose estimation and the projection, they needed more sophisticated calibration methods for both the camera and the projector.

One last issue concerning the optical system is projector focus: The camera image can be slightly out of focus without significantly impacting the pose estimation, but an unfocused projection decreases the quality of the projected image notably. This could probably be fixed easily by using a projector that can be focused by software. In this case the pose estimation software could re-adjust focus if the projection screen leaves the focused area of the projection beam.

A good way to avoid optics- and calibration-related problems would be to use an active and touch sensitive display. This would on the one hand make the cursor position independent of the projection screen pose and on the other hand ensure a constantly high resolution. As a downside to this we do not know of any current displays which are both big enough for modeling applications and lightweight enough to be hand-held.

We have not yet optimized the ergonmic properties of the projection screen, therefore, when holding the screen for extended periods of time hand fatigue occurs and increases quickly. Possible solutions to this could be a comfortable handle on the screen or further reducing the weight of the screen by using lighter materials than wood. A last resort would be using a mechanic that supports the screen as long as the user does not change the screen's position.

Altogether our projection screen is not yet mature enough for 3D-modeling, mainly due to technical reasons. Therefore, other spatially aware displays or even normal desktop PCs might be suited better for this purpose at the time being. On the other hand the spatially aware projection screen has some unique properties: At first it is lightweight enough to be hand-held and have a reasonably big display area at the same time. Additionally, the form factor is comparable to a notebook and therefore makes the system accessible for novice users. If using a semi transparent screen material, it might be suitable for augmented reality applications and as long as users are not obstructing the projection it should be easily expandable to a multi-user system with multiple screens. Last but not least, the overall costs of the projection screen are negligible if a projector and a camera are already at hand: The costs of the current prototype are approximately 25 Euro.Therefore, we feel that there is much potential in further improving the hand-held projection screen's usability as a spatially aware, hand-held device.

Even with the current limitations and weaknesses the hand-held projection screen has shown to be a convenient tool for exploratory tasks. Being intuitive to handle and very responsive, the screen could be a good input and output device for some types of computer games. If used in a multi-camera and -projector setup, the system might turn out to be suited for exploring machine designs, medical 3D data or other virtual 3D objects as well.

Closing this thesis we want to point out that especially for 3D modeling and the camera control needed for this task we could probably learn a lot from spatially aware displays. Especially hand-held devices might not be the best approach for actual modeling as they are prone to noise inserted by the user's unsteady hand. Nevertheless, the ways in which users interact intuitively with such a display in order to change their view on a 3D object might be the key to find new ways of enhancing desktop environments with integrated devices for this purpose. Those devices should at the same time exact enough for modeling purposes and provide intuitive ways of manipulating the view on 3D objects. Although view manipulation through 2D input devices is nowadays very sophisticated and highly established we are sure that there must be more intuitive and effortless approaches than the existing ones.

Appendix A

Mathematical Basics

Many of the routines or formulas laid out here are basics and will be known to most readers. For the convenience of readers that are less experienced with computer graphics and computer vision, we added those nonetheless.

Planes

Hesse-Form In most cases we will use the Hesse-Form to represent planes. A plane is then described by a normal vector $\mathbf{n}$ and a real value d. A point $\mathbf{p}$ is in the plane if and only if (iff) it fulfills the following equation: $\mathbf{n} \cdot \mathbf{p} = d$

Creating a Hesse plane from a not normalized vector $\mathbf{v}$ orthogonal to the plane and an arbitrary point $\mathbf{p}$ within the plane is trivial: $\mathbf{n} = \frac{\mathbf{v}}{|\mathbf{v}|}$, $d = \mathbf{n} \cdot \mathbf{p}$

Intersections

Line - Hesse plane To find the intersection between a line (given by a point $\mathbf{p}$ on the line and its direction vector $\mathbf{v}$) and a Hesse plane (given by $\mathbf{n}$ and d) we have to distinguish 2 cases:

1. The plane and line are parallel ($\mathbf{n} = \frac{\mathbf{v}}{|\mathbf{v}|}$): If $\mathbf{n} \cdot \mathbf{p} = d$, then the line is embedded in the plane. Otherwise there is no intersection.
2. Plane and line are not parallel ($\mathbf{n} \neq \frac{\mathbf{v}}{|\mathbf{v}|}$): There is exactly one intersection point. We can get this intersection point, by substituting the line equation into the Hesse plane equation: $\mathbf{n} \cdot (a \cdot \mathbf{v} + \mathbf{p}) = d \Leftrightarrow a \cdot \mathbf{n} \cdot \mathbf{v} + \mathbf{n} \cdot \mathbf{p} = d \Leftrightarrow a = \frac{d - \mathbf{n} \cdot \mathbf{p}}{\mathbf{n} \cdot \mathbf{v}}$. Now all we have left to do is to resubstitute a into the line equation: $\mathbf{c} = a \cdot \mathbf{v} + \mathbf{p}$

Line - Ball Assume we have given a line (by a contained point $\mathbf{p}$ and its direction $\mathbf{v}$) and a ball (given by its center $\mathbf{c}$ and its radius r). To get the intersection points we have to solve the quadratic equation $(\mathbf{v} \cdot \mathbf{v}) \cdot x^2 + (\mathbf{v} \cdot (\mathbf{p} - \mathbf{c})) \cdot x + (\mathbf{p} - \mathbf{c}) \cdot (\mathbf{p} - \mathbf{c}) - r^2 = 0$. Each non-imaginary result $x_{1/2}$ to this quadratic equation yields an intersection point **cut** between the plane and the ball $\mathbf{cut_{1/2}} = \mathbf{p} + x_2 \cdot \mathbf{v}$.

Distances

Point - Line Let $\mathbf{q}$ be a point and L be a line (given by the point $\mathbf{p}$ and the direction $\mathbf{v}$). The distance d of $\mathbf{q}$ to L is then computed as follows: $d = \frac{|(\mathbf{q}-\mathbf{p})\times\mathbf{v}|}{|\mathbf{d}|}$.

Closest Points

Point - Line Let $\mathbf{q}$ be a point and L be a line (given by a point $\mathbf{p}$ and its direction $\mathbf{v}$). The following procedure yields the point $\mathbf{c}$ on L, that lies closest to $\mathbf{q}$:

Algorithm 7 Closest: Point and Line

Compute the distance d between $\mathbf{q}$ and L. (see p. 82)
Compute the angle $\alpha = \arcsin(\frac{d}{|\mathbf{q}-\mathbf{p}|})$
between $\mathbf{v}$ and the direction $\mathbf{q}-\mathbf{p}$ to the point.
Compute the distance $d_L = \cos(\alpha)\cdot|\mathbf{q}-\mathbf{p}|$ between $\mathbf{c}$ and $\mathbf{p}$
Compute $\mathbf{c} = \mathbf{p} + \frac{\mathbf{v}}{|\mathbf{v}|}\cdot d_L$

Line - Line Let a first line L be given by its direction $\mathbf{v}$ and the point $\mathbf{p}$ and a second line M by the direction $\mathbf{u}$ and the point $\mathbf{q}$. We determine the point on M that is closest to L by laying a plane $\overline{L}$ through L, with normal $\mathbf{u}$. The intersection point $\mathbf{v}$ between $\overline{L}$ and M (see p. 81) is the point on M that is closest to L.

Algorithm 8 Closest: Line and Line

Compute the direction $\mathbf{d}$ from $\mathbf{c}$ to $\mathbf{p}$: $\mathbf{d} = \frac{\mathbf{p}-\mathbf{c}}{|\mathbf{p}-\mathbf{c}|}$
Get the point on the ball: $\mathbf{b} = \mathbf{d}\cdot r$.

Coordinate Transformations & Angles

Cartesian coordinates to polar coordinates Let $\mathbf{v} = (x,y)^T \in \mathbb{R}^2$ be a Cartesian vector. The corresponding polar coordinates are $(r,\theta)^T \in \mathbb{R}^2$ with $r = \sqrt{x^2+y^2}$ and $\theta = \begin{cases} \arccos x & , y \geq 0 \\ 2\pi - \arccos x & , y < 0 \end{cases}$

Cartesian coordinates to rotational coordinates Let $\mathbf{v} = (x,y,z)^T \in \mathbb{R}^3$ be a Cartesian vector. The corresponding rotational coordinates are $(r,\theta,\varphi)^T \in \mathbb{Z}^3$, where r is the length of the vector, θ is rotation around the y-axis and φ is rotation by θ around the z-axis. The rotational coordinates are computed as follows: $r = \sqrt{x^2+y^2+z^2}$, $\varphi = \arcsin(\frac{y}{r})$ and $\theta = \Theta$, where $(R,\Theta)^T$ are the polar coordinates (see p. 82) of $(x,-z)^T$. This is similar to spherical coordinates, but in order to make using those values easier for our purpose, φ starts at the level of the xz-plane. Figure A.1 (left) depicts the geometry of those values.

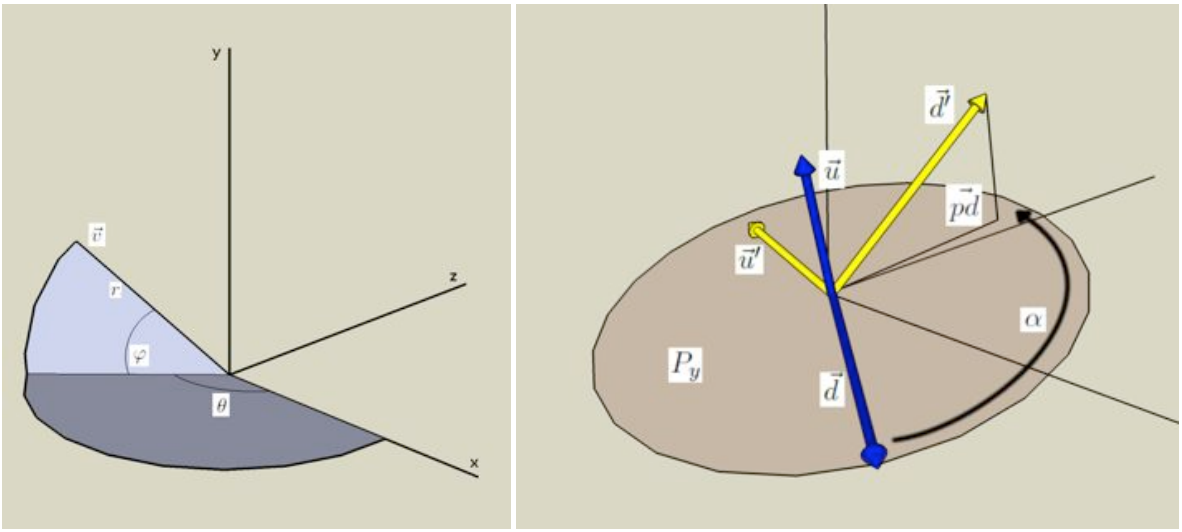


Figure A.1: Left: Vector in "rotational coordinates", Right: State before aligning two orientations

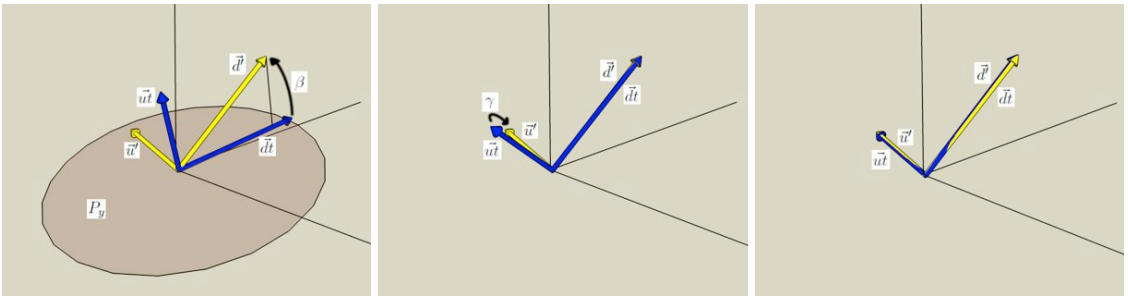


Figure A.2: Aligning two orientations, after 1st, 2nd and 3rd step

Rotation matrix from two orientations Let $\mathbf{d}$ and $\mathbf{d}'$ be the viewing directions of two orientations and $\mathbf{u}$ and $\mathbf{u}'$ the respective upward directions. Both orientations are centered in the coordinate origin. We now want to compute a 3×3 matrix R that transforms the orientation given by $\mathbf{d}$ and $\mathbf{u}$ into the orientation given by $\mathbf{d}'$ and $\mathbf{u}'$. The algorithm builds R successively by transforming $\mathbf{d}$ and $\mathbf{u}$ until they match $\mathbf{d}'$ and $\mathbf{u}'$. This is done using a temporary rotation matrix T (initialized to the identity matrix) and the temporary transformations of the first orientation $\mathbf{dt} = T \times \mathbf{d}$ and $\mathbf{ut} = T \times \mathbf{u}$. In figure A.1 (right) the initial situation can be seen. Figure A.2 shows the results of the three steps performed in algorithm 9.

Algorithm 9 Transforming one orientation into another one

```
//Move dt into the rotation plane of t' around dt × du
Let P_y be the plane with normal u and offset 0 (see p.81).
Compute the projection pd of d' onto P_y.
Determine the angle α between d and pd (see p. 84).
Create the matrix T rotating by α around u (see p. 85).
//Align dt with d'
Determine the angle β between dt and d' around the axis d × u.
Compute the matrix R1 rotating around d × u by β.
Concatenate the two rotations: T := R1 × T
//Align ut with u'
Determine the angle γ between ut and u' around d'.
Compute the rotation R2 around d' by γ.
The complete rotation is obtained by concatenation:
  R := R2 × T .
```

Rotation angle between 2 vectors Assume we have two points **a** and **b** and a rotation axis given only the direction **r**. We want to determine the angle that we have to rotate **a** around **r** (assuming **r** is orthogonal to **a** and **b**) in order to transform it into **b**. First we determine the angle α between **a** and **b**:

$$\alpha = \begin{cases} \arccos(\mathbf{a}\cdot\mathbf{b}) & ,\text{if } (\mathbf{a}\times\mathbf{b})\cdot\mathbf{r} < 0 \\ -\arccos(\mathbf{a}\cdot\mathbf{b}) & ,\text{otherwise} \end{cases}$$

In a mathematical sense this would suffice, but due to numerical instabilities in all involved values, we pre-rotate **a** by $\frac{\pi}{2}$, to obtain more stable results in the numerically most unstable area around $\mathbf{a}\cdot\mathbf{b} < -0.95$.

Transformation matrices

Those definitions are commonly known and will be found in all introductory works about computer graphics or in lecture notes as for example in [Ale07].

Translation A homogeneous translation matrix, moving for $(x,y,z)^T$:

$$\begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around the x axis (pitch) A homogeneous rotation matrix for rotation by α around the x axis:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around the y axis (yaw) A homogeneous rotation matrix for rotation by α around the y axis:

$$\begin{pmatrix} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around the z axis (roll) A homogeneous rotation matrix for rotation by α around the z axis:

$$\begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation around a vector Assume we have a rotation axis $\mathbf{v}$and an angle α, by which we want to rotate around $\mathbf{v} = (v_x, v_y, v_z)^T \in \mathbb{R}^3$. For brevity we are using the following definitions: $s := \sin\alpha$, $c := \cos\alpha$, $u := (1 - \cos\alpha)$. With those the rotation matrix looks as follows:

$$\begin{pmatrix} v_x^2 u + c & v_y v_x u - v_z s & v_z v_x u + v_y s & 0 \\ v_x v_y u + v_z s & v_y^2 u + c & v_z v_y u - v_x s & 0 \\ v_x v_z u - v_y s & v_y v_z u + v_x s & v_z^2 u + c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Finding Roots

Secant Search Assume we have a function $f : \mathbb{R} \to \mathbb{R}$ and two arguments x_1 and x_2. To find a root of f, we first approximate the differential of f through $(x_1, f(x_1))$ and $(x_2, f(x_2))$. Then we use the intersection of it with the x-axis, to estimate the root of the function: $x_2' = \frac{f(x_2)\cdot x_1 - f(x_1)\cdot x_2}{f(x_2) - f(x_1)}$. The estimated root x_2' and the last estimate $x_1' = x_2$ are then recursively given into the secant search procedure again. The recursion can be stopped, if one of the function values is close enough to zero (root found) or if x_1 and x_2 are to close to each other. Of course it is more efficient to iterate this iteratively.

Matrix operations

Pseudo Inverse Let $A = U\Sigma V^*$ be the singular value decomposition of the input matrix A. The pseudo-inverse A^+ is then

$$\begin{aligned} A^+ &= U\Sigma^+ V^* \\ \text{with } (\Sigma)^+_{ij} &= \begin{cases} \frac{1}{\sigma_i} & , \text{ if } i = j \wedge \sigma_i \neq 0 \\ 0 & , \text{ otherwise} \end{cases} \end{aligned}$$

Appendix B

Qestionnaire of the Informal User Study

This chapter shows the questions the participants of our inormal user studies were asked to answer. The questions are posed in german, as the participants of the user study were all native german speakers and some of them are not fluent in English. The detailed questions were asked once for each evaluated camera pose control.

General Questions

The general questions were answered once by each participant. They were presented to the participants of the study from the beginning of the test. On the one hand thy are collecting data about the user's experience with navigating virtual 3D spaces. On the other hand the participant's subjective evaluation of the different camera control methods are queried:

Angaben zum Nutzer

Haben Sie Erfahrung mit der Navigation in dreidimensionalen virtuellen Räumen? Hierzu zählen alle 3D Applikationen, in denen Sie die Kamera frei bewegen können.

Ja O Nein O

Haben sie Erfahrung mit 3D-modelling Software?

Ja O Nein O

Gesamtwertung

Bitte geben sie zuerst ihre subjektive Gesamtwertung für die unterschiedlichen Sichtkontrollen an.

	nicht so gut				exzellent
Stift	O	O	O	O	O
Absolut	O	O	O	O	O
Relativ	O	O	O	O	O
Kombiniert 1	O	O	O	O	O
Kombiniert 2	O	O	O	O	O

Detailed Questions

Those questions were posed for each of the 5 different camera control techniques: Absolute and Relative camera control through the projection screen, the combination of absolute and relative camera control and the combined, the combined camera control with centering of the object and the pen driven reference control:

Wie schnell haben Sie sich an diese Art der Sichtkontrolle gewöhnt?

gar nicht				sofort klar
O	O	O	O	O

Wenn sie eine bestimmte Änderung der Sicht wünschten, war Ihnen klar wie Sie diese erreichen können? Hier nehmen wir an, dass Sie sich bereits an die Sichtkontrolle gewöhnt haben.

komplett unklar				ohne Überlegen klar
O	O	O	O	O

Wie präzise können Sie mit der vorliegenden Methode eine gewünschte Sicht herstellen?

weit von Wunsch-sicht entfernt				absolut präzise
O	O	O	O	O

Wie schnell können Sie gewünschte Sichten durchschnittlich herstellen?

dauert lange				geht sehr schnell
O	O	O	O	O

Möchten Sie diese Methode der Sichtkontrolle kommentieren? Auf dem Rest dieser Seite haben Sie Platz dafür.

Appendix C

Obsolete Pose Estimation Algorithms

C.1 Local Search

This basic approach for pose estimation utilizes a steepest descent [Hei06] search. The idea of our algorithm is running several instances of steepest descent with different target functions and neighborhood definitions. In theory all the work could be done within a single steepest descent run. The reason for using multiple runs is mainly computation time: By running several instances with small neighborhoods and simple target functions we converge towards a good pose estimate quickly. Before laying out the stages of this approach we present the parameters which we use throughout the different steepest descent instances.

The **coding of a solution** consists of 6 components: $trans_x$, $trans_y$, $trans_z$, rot_x, rot_y, rot_z, where **trans** $= (trans_x, trans_y, trans_z)^T \in \mathbb{R}^3$ is the estimated position and **rot** $= (rot_x, rot_y, rot_z)^T \in \mathbb{R}^3$ contains a yaw-pitch-roll style codification of the rotation part of the pose.

Neighborhoods

We have 3 different possibilities of building a neighborhood for a given solution. All the neighborhoods are based on fixed-step alterations to one or more of the solution parameters. These parameters are:

Translation. An obvious set of neighbors of a solution can be obtained by translating it. For time-complexity reasons the neighborhood only consists of two neighbors along each axis. We apply a fixed translation along the altered axis ($trans_x$, $trans_y$ or $trans_z$), yielding a neighborhood of 6 elements.

Single axis rotation. Neighbors of this kind are only altered in exactly one of the rotation components (rot_x, rot_y or rot_z), resulting in a rotation around only one of the camera coordinate system axes. This neighborhood contains 6 elements as well.

Full rotation. This neighborhood contains all rotations that can be built by applying a combination of increments or decrements (by a fixed value) to any of the 3 rotation parameters. This makes up a neighborhood of 26 elements.

Target functions

We have different measures for the quality of solutions. Those measures will be used as target functions for the optimization process. As they are error measures, their values are decreasing with increasing quality of the solution. The error functions are named as follows:

Barycenter. Let $\overline{cp}$ be the barycenter of the matched re-projected points and $\overline{cr}$ be the barycenter of the matched image ray-points. Then the barycentric error is $e_b = |\overline{cp} - \overline{cr}|^2$.

$$\overline{cp} = \sum_{p \in \hat{CP}} \frac{p}{|\hat{CP}|}$$
$$\overline{cr} = \sum_{r \in \hat{CR}} \frac{r}{|\hat{CR}|}$$

Re-projection error. This is the sum e_r of the re-projection errors of all matched points.

$$e_r = \sum_{p^l \in \hat{CP}} e_r^l$$

Discourage rotation. This error is increased if the step to a neighbor is rotational. This measure is used if we want to allow turning in general, but prefer translation. Let us therefore denote the last solution's rotation components by rot'_x, rot'_y and rot'_z. The rotational error is then $e_{rot} = (rot_x - rot'_x)^2 + (rot_y - rot'_y)^2 + (rot_z - rot'_z)^2$.

Dimensions. This error is used as a cue for having found the correct distance from the camera. To calculate it, we need the minimal and maximal x- and y-components on that one of the re-projected points lies (see fig. 2.3 on page 24, right): $px_{min} = \min(p_x^l \,|\, (p_x^l, p_y^l, 1) \in \hat{CP})$, $py_{min} = \min(p_y^l \,|\, (p_x^l, p_y^l, 1) \in \hat{CP})$, $px_{max} = \max(p_x^l \,|\, (p_x^l, p_y^l, 1) \in \hat{CP})$ and $py_{max} = \max(p_y^l \,|\, (p_x^l, p_y^l, 1) \in \hat{CP})$. The same measures are taken for the seen point's rays in order to obtain rx_{min}, ry_{min},

rx_{max} and ry_{max}. The dimension error $e_d = ((px_{max} - px_{min}) - (rx_{max} - rx_{min}))^2 + ((py_{max} - py_{min}) - (ry_{max} - ry_{min}))^2$ reflects how well the dimensions of the reprojected and observed images match.

Algorithm

For detailing the algorithm, let us assume we have image rays *CR* (implied by the observed points), know the model points *CM* and the matching functions $i2m$ and $m2i$. Furthermore, we need an initial screen pose, from which we can initialize our local search.

The procedure of estimating a pose consists of a sequence of steepest descent runs with different settings. Each instance of steepest descent will use the output of the last step as initial solution. In the first run, the reprojected pose barycenter will be aligned with the barycenter of the image points (see fig. 2.3 on page 24) by translation. This is a preprocessing step in order to move the re-projection of the estimated pose roughly to the area of the image where the LEDs have been observed.

The second steepest descent run uses the heuristic of image dimensions for depth estimation. By only allowing translation, movement along the z-axis is forced, as under perspective projection this translation has the highest impact on an object's dimension in the image (see fig. 2.3 on page 24, right and 2.4 on page 25, left with the lines indicating image and model dimensions). Note that this heuristic terminates with a solution that is worse than the initial one, if the orientations of the current solution and the real pose differ strongly.

In the third step, rotations are allowed but discouraged. Allowing translation and rotation for the pre-positioned pose estimate leads to a good pose estimate within some iterations. Discouraging of rotation biases the search to adapt the screen pose's translation after each rotation step.

In some cases the third step ends in a local minimum of the re-projection error, which can be overcome by allowing more general rotations. Thus the fourth steepest descent pass, adds multiple axis rotations to the examined neighbors. This could already have been included in the previous step. But as the multiple axis rotation neighborhood is considerably bigger than the other ones, it has been introduced in a separate pass. This way computation time is saved through delaying the examination of this big neighborhood to a point at that smaller neighborhoods can not overcome local minima anymore. The resulting pose estimation of the last two steepest descent runs can be seen in figure 2.4 on page 25.

Algorithm 10 shows the complete sequence of steepest descent runs.

Algorithm 10 Steepest descent pose estimation

Input: An initial pose estimation **init** $= (it_x, it_y, it_z, ir_x, ir_y, ir_z)^T \in \mathbb{R}^6$
Output: A pose estimation $(trans_x, trans_y, trans_z, rot_x, rot_y, rot_z)^T \in \mathbb{R}^6$

```
Start a steepest descent search with the init as initial so-
lution.
  Restrain the neighborhood of a solution to translation and
  minimize the barycentric error (e_b → min).
Using the output of the first step as initialization,
  start another steepest descent run
  restrained to the translation neighborhood,
  minimizing the dimension error (e_d → min).
Improve the pose estimate of step two,
  by minimizing a combination of re-projection
  and rotational error (w_1·e_r + w_2·e_rot → min, w_1,w_2 ∈ ℝ).
  The neighborhood is made up by translation
  and one axis rotation.
Refine the output of the previous step
  by additionally allowing multiple axis rotations.
```

C.2 Single Parameter

This approach is an attempt to develop a closed-form mathematical solution to pose estimation. The idea is to reduce the parameter set needed to solve the pose estimation problem to one parameter. The motivation behind that is to reduce the overall computing time by either finding a formula that can be directly computed or by restricting the neighborhood of a search step to two neighbors.

Determining two positions on 2D rays through an angle α The first step is solving a related two dimensional problem (see fig 2.5 on page 26): Assume we know the angle γ between two Rays A and B, that are originating in a point $\mathbf{F}$, which is without loss of generality the coordinate origin. Additionally we know the distance $|\mathbf{ab}| = |\mathbf{a} - \mathbf{b}|$ between two points $\mathbf{a} \in \mathbb{R}^2$ and $\mathbf{b} \in \mathbb{R}^2$ on those rays. Furthermore we know the normalized directions $\mathbf{dA} \in \mathbb{R}^2$ and $\mathbf{dB} \in \mathbb{R}^2$ of the rays. Now we want to compute the position of $\mathbf{a}$ and $\mathbf{b}$ on the rays, given an angle α between the line connecting the points $\mathbf{a}$ and $\mathbf{b}$ and the ray A.

The basis of computing those positions are the relationships of the triangle described by the vertices $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{F}$. As a first step we compute the length of the altitude of the triangle through $\mathbf{b}$, which is annotated by bRa in figure 2.5 on page 26. This can be done using the sine of the angle at $\mathbf{a}$ (which is $\pi - \alpha$) and the length of $\mathbf{b} - \mathbf{a}$. Knowing γ and the length of $\mathbf{bRa}$, we can calculate the distance bF between $\mathbf{b}$ and $\mathbf{F}$, again based on the sine of γ. The cosine of γ together with the length bF will yield the distance between $\mathbf{F}$ and $\mathbf{c}$. Together with the distance

ca between **c** and **a**, the length aF is obtained, where $ca = \sqrt{|\mathbf{bRa}|^2 - |\mathbf{ab}|^2}$. The lengths aF and bF with the ray directions **dA** and **dB** yield the positions **a** and **b**. Algorithm 11 shows those computations.

Algorithm 11 computation of the positions **a** and **b** from

Input: vector **ab**; angles γ, α; directions **dA** and **dB**.
Output: positions **a** and **b**.

Compute the length of **bRa**: $|\mathbf{bRa}| := |\mathbf{ab}| \cdot \sin(|\pi - \alpha|)$
Compute the length bF: $bF := \frac{|\mathbf{bRa}|}{\sin(\gamma)}$
Compute the length aF: $aF := bF \cdot \cos(\gamma) + \sqrt{|\mathbf{bRa}|^2 - |\mathbf{ab}|^2}$
Compute the positions **a** and **b**: $\mathbf{a} := \mathbf{dA} \cdot aF$, $\mathbf{b} := \mathbf{dB} \cdot bF$

How does this 2D problem relate to our pose estimation problem? Assume the rays A and B are obtained from the (1D) image coordinates of seen LEDs. If this is the case, the according LEDs can only lie on the rays A and B in 2D space. In order to obtain the rays A and B, the positions of those LEDs on the image metric image plane P_o (see sec. 1.4) need to be computed, using the intrinsic camera paramters. If we additionally know the matching model points for those LEDs, the distance $|\mathbf{ab}|$ can be obtained from the model coordinates of the LEDs. The determination of **a** and **b** estimates positions of the two LEDs in 2D space.

Determining a rotation circle for a third point by fixing α in 3D Next we utilize this 2D algorithm in 3D space to determine the position of 3 points. The 2D algorithm can be applied in 3D, as computing distances and angles in 3D is easily performed (see appendix A). Let us assume we have extracted rays A, B and C from the camera image. Now we start estimating the screen pose by choosing an angle α which fixes positions for the two LEDs on A and B through the 2D algorithm. As our projection screen is a rigid object and the LEDs are fixed at points **a** and **b** on their rays the third LED's position can only rotate around **ab**, the axis going through the two fixed LEDs. This is depicted in figures 2.6 on page 27 and 2.7 on page 27.

Denoting the possible positions of the third LED by the rotation circle $R \in \mathbb{R}^3$, we can relate the geometry shown in figures 2.6 on page 27 and 2.7 on page 27 to a pose estimation as follows. The LEDs lying on the rays A and B are already in a position that matches the observed points by construction. Our chosen angle α is correct, regarding the data obtained from our image, if there is a possible position of the third LED, that lies on the associated ray C (as seen in fig. 2.7 on page 27). As R describes all possible locations of this third LED, α is correct, if R and C have a common point. If this is not the case (as in fig. 2.6 on page 27) the estimated pose does not fit the observed point positions.

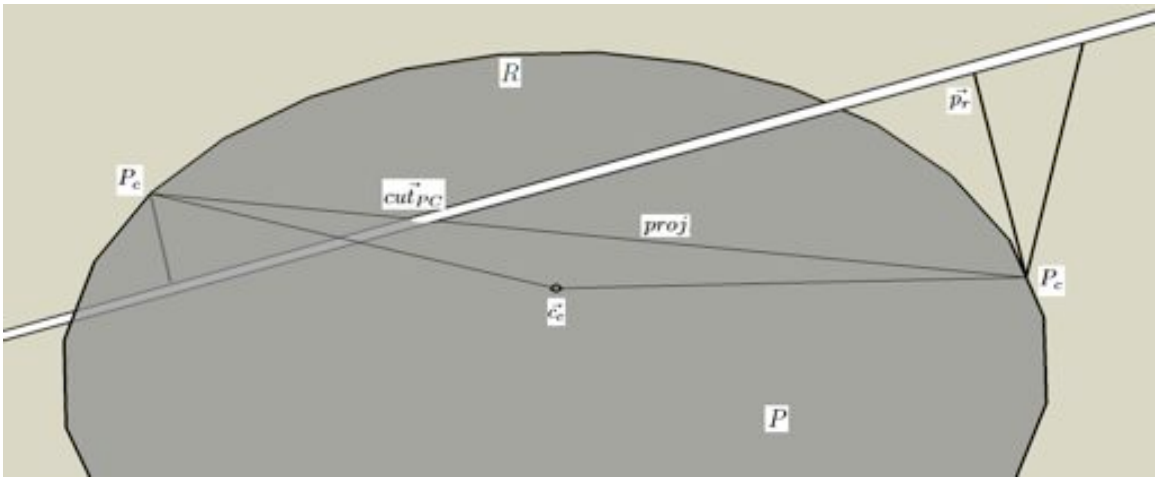


Figure C.1: Computing distance between circle and ray, *proj* intersects *R*

Approximating the distance between the rotation circle and the ray As we have just seen, we can use the distance between *C* and R as a measure of quality for our guessed angle α. In order to systematically optimize α we need a way to compute the distance between *C* and R. As this task is not trivial, we use some heuristics to build a set of candidate points on *C* for being closest to *R*. The closest candidate is then used to estimate the distance between *C* and *R*.

Let **dC** be the normalized direction of the ray *C*. Knowing the fixed positions **a** and **b**, we can obtain a possible location of **c**, by executing algorithm 13 as far as possible with the currently known data. Let *P* be the plane in which R is embedded. By intersecting the plane *P* with the line induced by **a** and **b** (see appendix A), the rotation center $\mathbf{c_c}$ of *R* can be obtained. The radius *r* can simply be obtained as the distance between the possible position of **c** and $\mathbf{c_c}$: $r = |\mathbf{c_c} - \mathbf{c}|$.

Knowing those values, we can start building the set $Cand \in P(\mathbb{R}^3 \times \mathbb{R})$, in which we will store the candidate points on *C* with the according distances to *R*. The geometric relationships utilized by this algorithm are shown in figure C.1. The intersection point $\mathbf{cut_{PC}}$ between the rotation plane *P* and the ray *C* is one of the heuristically interesting points and is added to *Cand*. If the ray *C* is perpendicular to *P*, then $\mathbf{cut_{PC}}$ is point on *C* that is closest to *R*.

Another set of interesting points can be obtained, using the projection *proj* of *C* onto *P*. Next we obtain the intersecting points P_C of *proj* with *R*: As *P* and *R* are coplanar, we can use the method for calculating the intersection between a ball (with center $\mathbf{c_c}$ and radius *r*) and *proj* in order to obtain those intersection points (see appendix A). This is numerically more stable than computing the intersection between the 3D-circle and 3D-line.

If *proj* and *R* do not intersect ($P_C = \emptyset$), then the point $\mathbf{p_c}$ on *R* that is closest to *proj* will be used instead of the intersection points (see fig. C.2). This point can be obtained by first getting $\mathbf{p_p}$, the point on *proj* that is closest to the center $\mathbf{c_c}$ of *R*. Then the point $\mathbf{p_c}$ can be obtained by using the normalized direction $\mathbf{d_{p_p}}$ from

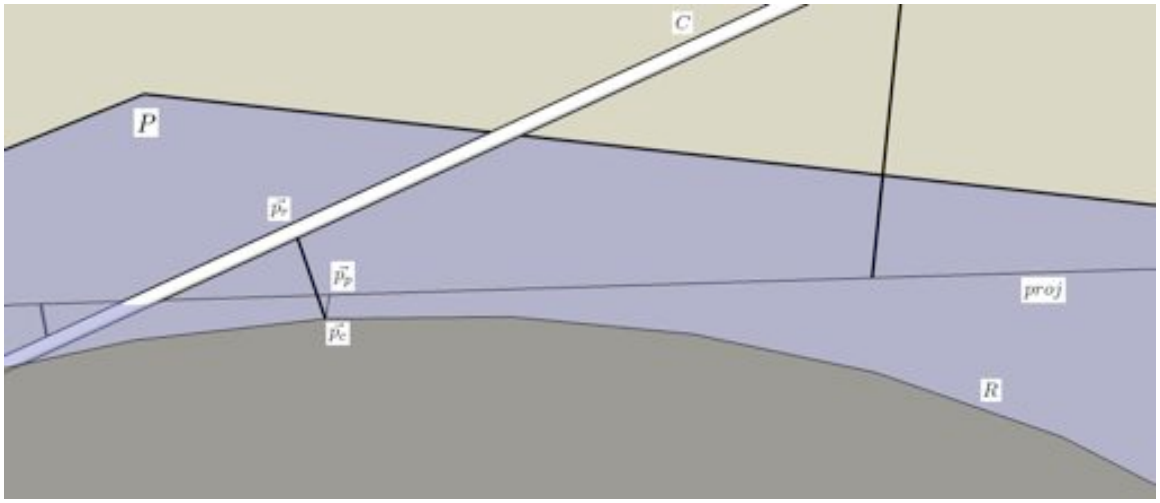


Figure C.2: Computing distance between circle and ray, *proj* and R do not intersect

$\mathbf{c_c}$ to $\mathbf{p_p}$ and the radius r of R: $\mathbf{p_c} = \mathbf{c_c} + \mathbf{d_{p_p}} \cdot r$.

Now P_C contains at least one point. For each of those points $\mathbf{p}$ we obtain the point $\mathbf{p_r}$ on C that is closest to $\mathbf{p}$ (see appendix A). Finally we compute the distance $d = |\mathbf{p_r} - \mathbf{p}|$ between $\mathbf{p_r}$ and $\mathbf{p}$. If $\mathbf{p}$ lies in the area enclosed by R ($|\mathbf{p_r} - \mathbf{c_c}| < r$) then we negate d. This will serve as a help to find extrema in the function that maps from angles to distances later on. Finally we add $(\mathbf{p_r}, d)$ to *Cand*. If C is parallel to P, then the points $\mathbf{p_r}$ are the points on C that are closest to R.

Algorithm 12 approximates the distance between the ray C and the rotation circle R, using the results we just described.

Constructing the matrix representing the pose estimation From algorithm 12 we obtained a set *Cand* of 1 to 3 points, on the ray C that are candidates for being closest to the rotation circle R of $\mathbf{c}$. If one of those points ($\mathbf{c_{cand}}$ with the according distance d_{cand}) is very close to R ($d_{cand} < \varepsilon$, where ε is a small threshold) then we we can consider $\mathbf{c_{cand}}$ to be a valid position for the third LED, given the positions $\mathbf{a}$ and $\mathbf{b}$ of the LEDs which are determined by the angle α. In this case we want to extract a pose estimation from $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c_{cand}}$.

This pose estimation is embodied by a matrix T which transforms the points from the model coordinate system into points in 3D space, fitting $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c_{cand}}$. T is constructed in a step by step procedure, that is closely related to the procedure of distance estimation described above. In order to construct T we will have to differentiate between the model positions of points $\mathbf{a_m}$, $\mathbf{b_m}$ and $\mathbf{c_m}$ and the (seen) target position $\mathbf{a_s}, \mathbf{b_s}$ and $\mathbf{c_s}$. The target positions are determined by the estimated screen pose ($\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c_{cand}}$). Additionally we define vectors $\mathbf{a}_t, \mathbf{b_t}$ and $\mathbf{c_t}$ that will fulfill $T \times \mathbf{x_m} = \mathbf{x_t}$ at any time, where x is a placeholder for a, b and c. Throughout algorithm 13 the update of $\mathbf{a}_t, \mathbf{b_t}$ and $\mathbf{c_t}$ is done implicitly whenever we add a new transformation (T changes). At the end of the procedure we will have $\mathbf{x_t} = \mathbf{x_s}$.

Algorithm 12 Estimation of distance between C and R

Input: ray C, center $\mathbf{c_c}$ and radius r of R, the normal $\mathbf{n_p}$ of the plane P
Output: set of candidate points on C for being closest to R, with according distance to R

Initialize $Cand$: $Cand = \emptyset$
Get the intersection point $\mathbf{cut_{PC}}$ of P and the ray C.
IF C is not parallel to P THEN
  Set $Cand = Cand \cup \{(\mathbf{cut_{PC}}, |\mathbf{cut_{PC}} - \mathbf{c_c}| - r)\}$.
Project the ray C onto the rotation plane P (using $\mathbf{n_p}$):
  The projected line $proj$ is described by
  the point $\mathbf{p_{proj}}$ and the direction $\mathbf{v_{proj}}$.
Compute the intersection points $P_c \in P(\mathbb{R}^3)$ of $proj$ with R
IF $proj$ does not intersect R ($P_c = \emptyset$) THEN
  Get the point $\mathbf{p_p}$ on $proj$, that is closest to $\mathbf{c_c}$
  Get the point $\mathbf{p_c}$ on the circle, that is closest to $\mathbf{p_p}$
  Add $\mathbf{p_c}$ to P_c
FOR EACH intersecting point $\mathbf{p} \in P_c$ DO
  Get the point $\mathbf{p_r}$ on the ray C that is closest to $\mathbf{p}$.
  Compute the distance d of $\mathbf{p_r}$ to R:

$$d = \begin{cases} -|\mathbf{p_r} - \mathbf{p_p}| & \text{if } |\mathbf{p_r} - \mathbf{c}| < r \\ |\mathbf{p_r} - \mathbf{p_p}| & \text{otherwise} \end{cases}.$$

  Add $\mathbf{p_r}$ with the according distance d to $Cand$:
    $Cand = Cand \cup \{(\mathbf{p_r}, d)\}$

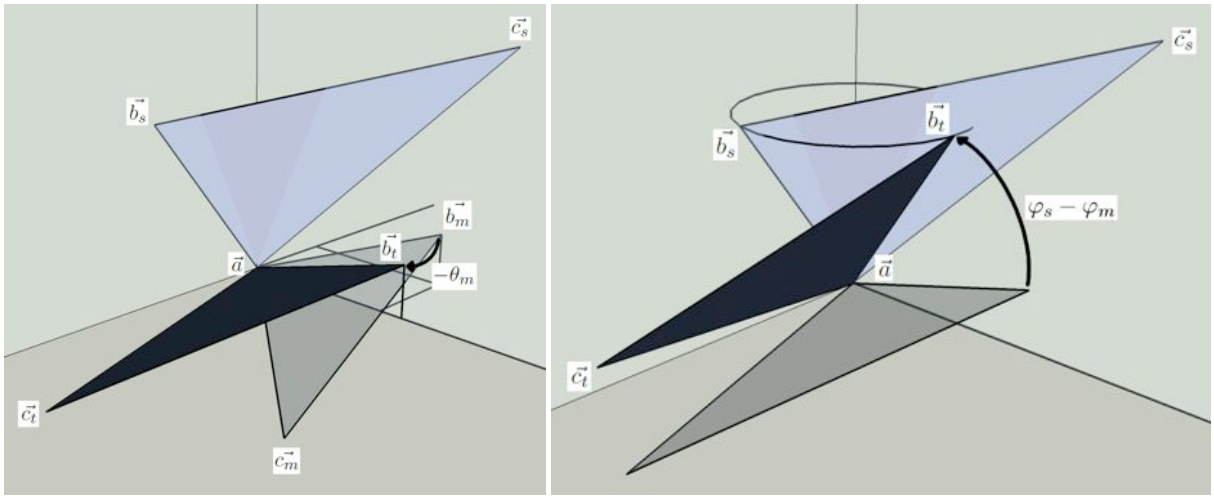


Figure C.3: Transformation to desired pose. Left: Aligning **b** with xy-plane, Right: Matching φ_t with φ_s

The first step to achieve this, is to align the directions from **a** to **b**: We need to find transformations in that match the transformed direction $\mathbf{d_t} = \mathbf{b_t} - \mathbf{a_t}$ with the target direction $\mathbf{d_s} = \mathbf{b_s} - \mathbf{a_s}$. Therefore, we first determine the rotational coordinates $(r_m, \theta_m, \varphi_m)^T$ (see appendix A) of $\mathbf{d_m}$ and the according rotational coordinates of the target direction $\mathbf{d_s}$: $(r_s, \theta_s, \varphi_s)^T$.

In order to rotate around $\mathbf{a_m}$, we move $\mathbf{a_m}$ to the coordinate origin, with a matrix that translates by $-\mathbf{a_m}$ (see appendix A). Now we can start aligning the two directions: First we rotate around the y-axis by $-\theta_m$, which aligns b_t with the xy-plane (see fig. C.3, left). The next step is to adjust φ_t to equal φ_s by rotating around the z-axis by $\varphi_s - \varphi_m$ (see fig. C.3, right). After rotating around the y-axis by θ_s, the directions $\mathbf{d_t}$ and $\mathbf{d_m}$ are matched (see fig. C.4, left).

The last rotation we need to perform, adjusts the direction $\mathbf{dc_t} = \mathbf{c_t} - \mathbf{a_t}$ to the desired one $\mathbf{dc_s} = \mathbf{c_s} - \mathbf{a_s}$. In order to get the angle β, by that we have to rotate, we determine the normals of two planes triangles defined by **a**, **b** and **c**: $\mathbf{n_t}$ for the transformed triangle and $\mathbf{n_s}$ for the target triangle. The angle β is the angle between $\mathbf{n_t}$ and $\mathbf{n_s}$ with respect to $\mathbf{d_s}$ (see appendix A). Finally we can rotate around $\mathbf{d_s}$ by β, which aligns $\mathbf{dc_t}$ with $\mathbf{dc_s}$ (see appendix A). The last step to fully align the the transformed pose with the target pose is to transform by $\mathbf{a_s}$.

Now we know how we can obtain pose candidates by fixing the angle $a2$ between $\mathbf{ab} = \mathbf{b} - \mathbf{a}$ and A. We can evaluate the quality of such a candidate, by estimating the distance of the rotation circle R to the ray C. This mapping from an angle to the according distance will from now on be called $a2d : \mathbb{R} \to \mathbb{R}$. Additionally we can compute a matrix that transforms our model into a pose that matches given 3D target points $\mathbf{a_s}$, $\mathbf{b_s}$ and $\mathbf{c_s}$, if the target points are induced by the model.

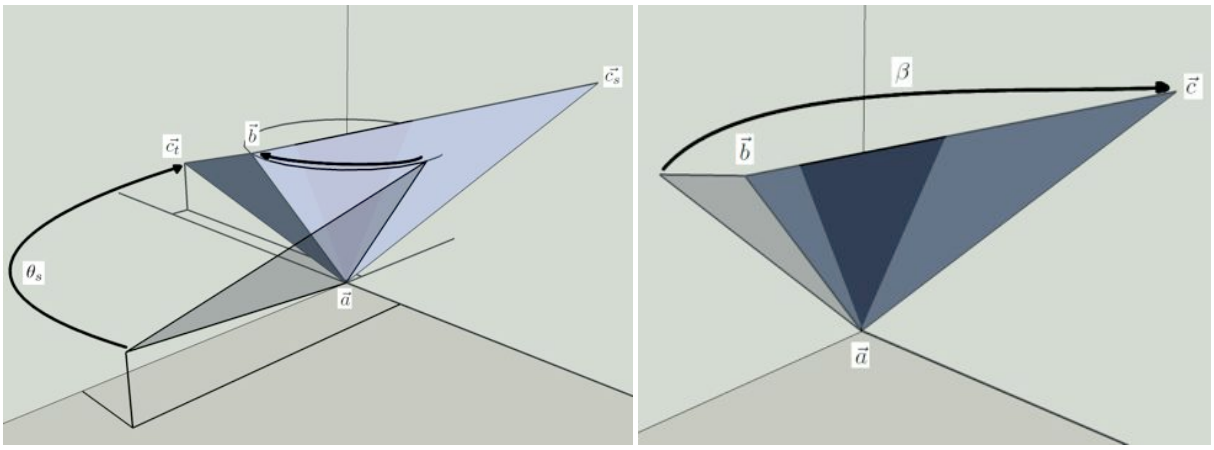


Figure C.4: Transformation to desired pose. Left: Rotation to align **b**, Right: Rotation to align **c**

Algorithm 13 pose matrix from target points $\mathbf{a_s}$, $\mathbf{b_s}$ and $\mathbf{c_s}$

Input: target points $\mathbf{a_s}$, $\mathbf{b_s}$ and $\mathbf{c_s}$ and model points $\mathbf{a_m}$, $\mathbf{b_m}$ and $\mathbf{c_m}$
Output: Matrix T, transforming model points into target points

Set $T = T_1$, where T_1 is a matrix, translating by $-\mathbf{a}$.
Compute the rotational coordinates $(r_s, \theta_s, \varphi_s)^T$ of $\mathbf{d_s} = \mathbf{b_s} - \mathbf{a_s}$.
Compute the rotational coordinates $(r_m, \theta_m, \varphi_m)^T$ of $\mathbf{d_m} = \mathbf{b_m} - \mathbf{a_m}$.
Set $T = T_2 \times T$, with T_2 being a matrix
  rotating around the y-axis by $-\theta_m$.
Set $T = T_3 \times T$, where T_3 is rotating the z-axis by $\varphi_s - \varphi_m$.
Set $T = T_4 \times T$. T_4 rotates by θ_s around the y-axis.
Compute the angle β between the normals
  $\mathbf{n_t}$ and $\mathbf{n_s}$ of the transformed and target triangle.
Set $T = T_5 \times T$, where T_5 rotates around β, using $\mathbf{d_m}$ as axis.
Output result $T := T_6 \times T$, with the matrix T_6 translating by $\mathbf{a_s}$.

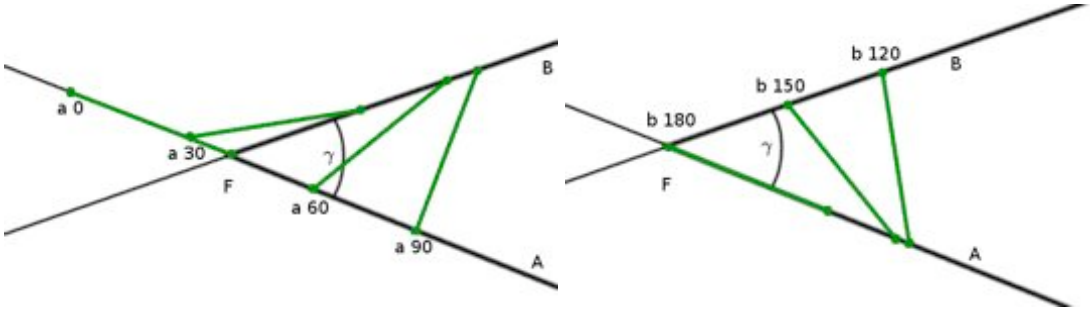


Figure C.5: Positions of **a** and **b** juxtaposed with the angles α determining them

Finding roots of *a2d* The last open problem of this pose estimation method is to find roots of $a2d$. Finding one of those will yield a screen pose estimation as shown above. Assume we have rays A, B and C. Additionally we compute the opening angle γ between A and B and choose the plane for fixing the angle γ between **ab** and A so that γ is positive in this plane. Let us first determine the range for α (see fig. C.5): If $\alpha = 0$, the estimated **a** will lie behind the focal point F of our camera and **b** will be on F. As we increase α, **a** stays behind the focal point until $\alpha = \gamma$. This will be the angle, at which **a** is on F and starts moving in front of it when further increasing α. Rotating further, at some point **b** will start to move towards the focal point again with increasing α and reach it at $\alpha = \pi$ (180°). As the pictures taken with our camera imply that all points are in front of the image plane and therefore in front of the focal point as well, we will search through the range of angles in $[\gamma, ..., \pi]$.

The main method we use for conducting this search is an adaption of secant search (see appendix A). We start at one of the borders of the interval for α and use a small offset towards the other border to initialize the search. Now we want to introduce the modifications, we made to the standard secant search: The first modification is to restrict the difference in α per step. This is necessary as with unlimited step range some existing roots can not be found.

Additionally we added a mechanism, that allows us to use certain features of $a2d$ in order to find roots easier. Those features are changes in the direction of the gradient and in the sign of the function value. Assume we have the two input angles α_1 and α_2 and we have computed our estimation α_2' for the next iteration of secant search. If the distance function changes its sign between the angles for the next iteration ($sign(a2d(\alpha_2)) \neq sign(a2d(\alpha_2'))$) or the direction of the gradient changes ($sign(a2d(\alpha_2') - a2d(\alpha_2)) \neq sign(a2d(\alpha_2) - a2d(\alpha_1))$) we know that there has to be a root or extremal point in between α_2 and α_2'. As especially in those areas $a2d$ can be very flat we obtained bad estimates of a_2', oftentimes. Therefore, when detecting those features, we use a bisection method to narrow the interval of the feature down, instead of directly applying our modified secant search.

With the secant search approach that we just described briefly, we can find the

first root α_f and the last root α_l within the interval. In most cases there are 4 roots. To identify the third and fourth root, the secant algorithm is not suitable, as it would have to be modified further to overcome local extrema. So we use another approach to find the roots located in between the ones we just found: This approach relies on the nature of $a2d$: Empirically we found that $a2d$ starts at a negative value and then goes to a maximum, which is followed by a minimum, another maximum and then falls negative again. Knowing this, we can assume that between α_f and α_l we will have two maxima which are enclosing a minimum.

Based on this observation we built an algorithm that searches the interval between α_f and α_l for the two maxima, which are enclosing a minimum. The detection of the extrema is purely based on differences between neighboring data points: A local maximum is detected if subsequent angles α_1, α_2 and α_3 with $\alpha_1 < \alpha_2 < \alpha_3 \wedge a2d(\alpha_1) < a2d(\alpha_2) > a2d(\alpha_3)$ are detected. The idea of the algorithm is to incrementally build a list of angles between α_f and a_l until two maxima can be detected in the list. From the existence of two different maxima we can deduct the existence of a minimum in between them. The list is expanded by adding data points in between each pair of successive existing points. While iterating over the list in order to add data points, the existence of 2 different maxima in the new data set is tested. Once we found those, we can start two instances of secant search starting at the maxima and searching towards the minimum in between the maxima. Algorithm 14 is based on this idea. For the sake of readability of the algorithm we are not performing the maxima detection in parallel to adding new data points here, although it can be implemented with just one traversal of the list per iteration.

Algorithm 14 roots in between α_f and α_l

Input: starting angle α_f and last angle α_l
Output: 2 roots in between α_f and α_l

```
Initialize a list of known angles K with K = (α_f, α_l)
WHILE not 2 local maxima found DO
  FOR EACH angle α in the list K (with successor α_s) DO
    Add new angle α_n = (α+α_s)/2 in between α and α_s
  Find first angle in list α_max1 for that α_max1 > α_s,
    where α_s is the successor of α_max1.
  Find last angle in list α_max2 for that α_max2 > α_p,
    where α_p is the predecessor of α_max2.
  IF α_max1 ≠ α_f ∧ α_max2 ≠ α_l ∧ α_max1 ≠ α_max2 THEN
    We found 2 local maxima.
Apply modified secant search with α_max1 and α_s as parameters,
  where α_s is the successor of α_max1
Apply modified secant search with α_max2 and α_p as parameters,
  where α_p is the predecessor of α_max2
Return roots found by secant search runs
```

Bibliography

[AD03] Adnan Ansar and Kostas Daniilidis. Linear pose estimation from points or lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):578–589, 2003.

[Ale07] Marc Alexa. Computer graphics i. lecture slides, Technical University Berlin, 2007. http://www-alt.cg.tu-berlin.de/cg1_07.html.

[BB] Sergey Bochkanov and Vladimir Bystritsky. Alglib. http://www.alglib.net/optimization/levenbergmarquardt.php. last accessed 14.11.08.

[Bea96] David M. Beazley. Swig: an easy to use tool for integrating scripting languages with c and c++. In *TCLTK'96: Proceedings of the 4th conference on USENIX Tcl/Tk Workshop, 1996*, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[BvBRF05] P. Beardsley, J. van Baar, R. Raskar, and C. Forlines. Interaction using a handheld projector. *Computer Graphics and Applications, IEEE*, 25(1):39–43, Jan.-Feb. 2005.

[BWR93] J. Brian Burns, R. S. Weiss, and E. M. Riseman. View variation of point-set and line-segment features. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(1):51–68, 1993.

[Fio01] Paul D. Fiore. Efficient linear solution of exterior orientation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):140–148, 2001.

[FL88] OD Faugeras and F. Lustman. Motion and structure from motion in a piecewise planar environment. *INT. J. PATTERN RECOG. ARTIF. INTELL.*, 2(3):485–508, 1988.

[FMM+08] George Fitzmaurice, Justin Matejka, Igor Mordatch, Azam Khan, and Gordon Kurtenbach. Safe 3d navigation. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 7–15, New York, NY, USA, 2008. ACM.

[FZC93] George W. Fitzmaurice, Shumin Zhai, and Mark H. Chignell. Virtual reality for palmtop computers. *ACM Trans. Inf. Syst.*, 11(3):197–218, 1993.

[Gar06] Jr. Everette S. Gardner. Exponential smoothing: The state of the art–part ii. *International Journal of Forecasting*, 22(4):637–666, 00 2006.

[Han97] C. Hand. A survey of 3d interaction techniques. In *Computer Graphics Forum*, volume 16, pages 269–281, 1997.

[Har94] Richard I. Hartley. Self-calibration from multiple views with a rotating camera. In *ECCV '94: Proceedings of the third European conference on Computer vision (vol. 1)*, pages 471–478, Secaucus, NJ, USA, 1994. Springer-Verlag New York, Inc.

[HCLL89] Radu Horaud, Bernard Conio, Oliver Leboulleux, and Bernard Lacolle. An analytic solution for the perspective 4-point problem. *Comput. Vision Graph. Image Process.*, 47(1):33–44, 1989.

[Hei06] H.-U. Heiß. Informatik 3. lecture scriptum, Technical University Berlin, 2006. http://www.kbs.cs.tu-berlin.de/teaching/ws2006/info3/.

[Hor86] Berthold K. Horn. *Robot Vision*. McGraw-Hill Higher Education, 1986.

[HPGK94] Ken Hinckley, Randy Pausch, John C. Goble, and Neal F. Kassell. A survey of design issues in spatial input. In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 213–222, New York, NY, USA, 1994. ACM.

[HSH04] Knud Henriksen, Jon Sporring, and Kasper Hornb? Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics*, 10(2):206–216, 2004.

[Kal60] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[KKS+05] Azam Khan, Ben Komalo, Jos Stam, George Fitzmaurice, and Gordon Kurtenbach. Hovercam: interactive 3d navigation for proximal object inspection. In *I3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, pages 73–80, New York, NY, USA, 2005. ACM.

[KMF+08] Azam Khan, Igor Mordatch, George Fitzmaurice, Justin Matejka, and Gordon Kurtenbach. Viewcube: a 3d orientation indicator and controller. In *SI3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 17–25, New York, NY, USA, 2008. ACM.

[Lee08] Johnny Chung Lee. Projects - wii. http://www.cs.cmu.edu/ johnny/projects/wii/, 2008. last acessed: 08.11.2008.

[LH81] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981.

[LHD07] Johnny Lee, Scott Hudson, and Pau Dietz. Hybrid infrared and visible light projection for location tracking. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 57–60, New York, NY, USA, 2007. ACM.

[LHM00] Chien-Ping Lu, Gregory D. Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622, 2000.

[LHSD05] Johnny C. Lee, Scott E. Hudson, Jay W. Summet, and Paul H. Dietz. Moveable interactive projected displays using projector based tracking. In *UIST '05: Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 63–72, New York, NY, USA, 2005. ACM.

[Lin93] Tony Lindeberg. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: a method for focus-of-attention. *International Journal of Computer Vision*, 11:283–318, 1993.

[Lin94] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, Norwell, MA, USA, 1994.

[Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[Mac92] I.S. MacKenzie. Fitts'Law as a Research and Design Tool in Human-Computer Interaction. *Human-Computer Interaction*, 7(1):91–139, 1992.

[MPL+02] Wendy E. Mackay, Guillaume Pothier, Catherine Letondal, Kaare Boegh, and Hans Erik Sorensen. The missing link: augmenting biology laboratory notebooks. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 41–50, New York, NY, USA, 2002. ACM.

[NISA07] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26(3):41, 2007.

[Pin01] Claudio S. Pinhanez. The everywhere displays projector: A device to create ubiquitous graphical interfaces. In *UbiComp '01: Proceedings of the 3rd international conference on Ubiquitous Computing*, pages 315–331, London, UK, 2001. Springer-Verlag.

[RB01] Ramesh Raskar and Paul Beardsley. A self-correcting projector. In *In Proceedings of Computer Vision and Pattern Recognition*, pages 504–508, 2001.

[Rek96] Jun Rekimoto. Tilting operations for small screen interfaces. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 167–168, New York, NY, USA, 1996. ACM.

[Rob63] Lawrence Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.

[RvBB+05] Ramesh Raskar, Jeroen van Baar, Paul Beardsley, Thomas Willwacher, Srinivas Rao, and Clifton Forlines. ilamps: geometrically aware and self-configuring projectors. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, page 5, New York, NY, USA, 2005. ACM.

[SG97] Zsolt Szalavári and Michael Gervautz. The personal interaction panel - a two-handed interface for augmented reality. In *Computer Graphics Forum*, pages 335–346, 1997.

[Sub00] Sriram Subramanian. Survey and classification of spatial object manipulation techniques. In *Proceedings of OZCHI 2000, Sydney, Australia*, pages 330–337, 2000.

[TFK+02] Michael Tsang, George W. Fitzmaurice, Gordon Kurtenbach, Azam Khan, and Bill Buxton. Boom chameleon: simultaneous capture of 3d viewpoint, voice and gesture annotations on a spatially-aware display. In *UIST '02: Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 111–120, New York, NY, USA, 2002. ACM.

[Tsa87] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *Robotics and Automation, IEEE Journal of [legacy, pre - 1988]*, 3(4):323–344, Aug 1987.

[WBV+99] Greg Welch, Gary Bishop, Leandra Vicci, Stephen Brumback, Kurtis Keller, and D'nardo Colucci. The hiball tracker: high-performance wide-area tracking for virtual and augmented environments. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 1–ff., New York, NY, USA, 1999. ACM.

[Wie64] N. Wiener. *Extrapolation, Interpolation, and Smoothing of Stationary Time Series: With Engineering Applications*. MIT Press, 1964.

[WJ88] C. Ware and D. R. Jessome. Using the bat: a six dimensional mouse for object placement. In *Proceedings on Graphics interface '88*, pages 119–124, Toronto, Ont., Canada, Canada, 1988. Canadian Information Processing Society.

[WO90] Colin Ware and Steven Osborne. Exploration and virtual camera control in virtual three dimensional environments. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 175–183, New York, NY, USA, 1990. ACM.

[WW92] J. Wang and W.J. Wilson. 3d relative position and orientation estimation using kalman filter for robot control. *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*, pages 2638–2645 vol.3, May 1992.

[WZC06] Jingtao Wang, Shumin Zhai, and John Canny. Camera phone based motion sensing: interaction techniques, applications and performance study. In *UIST '06: Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 101–110, New York, NY, USA, 2006. ACM.

[Yee03] Ka-Ping Yee. Peephole displays: pen interaction on spatially aware handheld computers. In *CHI '03: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–8, New York, NY, USA, 2003. ACM.

[Zha99] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 1:666–673 vol.1, 1999.