TECHNISCHE UNIVERSITÄT DRESDEN

FAKULTÄT INFORMATIK
INSTITUT FÜR SOFTWARE- UND MULTIMEDIATECHNIK
PROFESSUR FÜR COMPUTERGRAPHIK UND VISUALISIERUNG
PROF. DR. STEFAN GUMHOLD

Bachelorarbeit

zur Erlangung des akademischen Grades
Bachelor of Science

Emulation eines Mauszeigers zur Präsentationssteuerung an der Powerwall

Nico Schertler
(Geboren am 18. Dezember 1989 in Leisnig)

Betreuer: Dipl.-Medieninf. Frank Michel

Dresden, 7. September 2012

Aufgabenstellung

Bei der Arbeit an der Powerwall ist die Verwendung einer handelsüblichen Maus nur sehr bedingt sinnvoll, da die Steuerung typischerweise freihändisch vor der Präsentationsfläche erfolgen muss. Deshalb soll in dieser Arbeit eine Maussteuerung mithilfe eines Laserpointers emuliert werden. Dazu ist die Rückseite der Powerwall mit einer Kamera zu beobachten und aus dem Kamerabild die Laserpointerposition zu tracken. Als Laserpointer soll ein Presenter eingesetzt werden, der über Mausknöpfe und eine schnurlose Anbindung über USB verfügt. Es ist eine Anwendung zur Steuerung der Mausbewegung zu entwickeln die die Kombination aus Presenterknöpfen und getrackten Laserpointerposition im Windows Betriebssystem wie eine Maus behandelt.

Die Bearbeitungszeit beträgt 12 Wochen.

Teilziele:

- Literaturrecherche zum Video Tracking und zur Steuerung von Eingabegeräten
- Recherche zu bestehenden Presenter-Systemen
- Implementierung der Verfolgung des Laserpointers im Kamerabild
- Emulieren der Mausbewegung anhand des getrackten Laserpointerposition
- Umsetzung der Presentertasten als Mauseingaben
- Unterstützung von mindestens zwei verschiedenen Kameraschnittstellen (z.B. USB und IEEE1394)
- Evaluation der Genauigkeit sowie der Latenz des entwickelten Systems

Erweiterungen:

- gleichzeitiges Tracken mehrerer Laserpointer
- Untersuchung zum Nutzen des Trackens der Handposition (z.B. mittels MS Kinect Sensors) um somit einen gerichteten Zeigestab zu emulieren

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Emulation eines Mauszeigers zur Präsentationssteuerung an der Powerwall

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 7. September 2012

Nico Schertler

Kurzfassung

Präsentationen an Leinwänden haben den Nachteil, dass sie nur bedingt gesteuert werden können. Interaktionen beschränken sich meist auf das Wechseln von Folien. Die Benutzung von Maus und Tastatur bringt während einer Demonstration Probleme mit sich.

Im Rahmen dieser Arbeit soll deshalb eine Emulationssoftware erstellt werden, die es ermöglicht, mit der Präsentation durch Benutzung eines Presenters zu interagieren. Dabei soll durch den Laserpointer der Mauszeiger gesteuert und durch die Knöpfe auf dem Presenter Klickaktionen ausgeführt werden.

Diese Arbeit betrachtet verwandte Projekte, führt in die wesentlichen Grundlagen zum Thema ein und stellt das verwendete Konzept sowie die Implementierung des Systems vor. Die Software und alle Materialien befinden sich auf der beigelegten CD.

Abstract

Presentations on large screens suffer from the disadvantage of limited control capabilities. Most interactions are limited to changing slides. With mouse and keyboard arise additional problems when used within a demonstration.

For the purposes of this work, an emulation software will be created, which allows the user to interact with the presentation using a presenter. A laser pointer will control the system's cursor position and the presenter's buttons allow click operations.

Topics of this work are related projects, theoretical fundamentals, the developed concept, as well as the system's implementation. The software and all related materials are provided on the enclosed CD.

Inhaltsverzeichnis

1 Einleitung

Projektionsleinwände stehen heute in vielen Einrichtungen zur Verfügung. Ihr Hauptzweck ist in den meisten Fällen die Darstellung und Präsentation von Inhalten. Da für diesen Zweck oft eine eingeschränkte beziehungsweise keine Interaktion mit der Präsentationsfläche nötig ist, ist meist nicht die entsprechende Hardware vorhanden, die weitergehende Möglichkeiten offenlegen würde.

Einer der weitverbreitetsten Einsatzwecke ist die Anzeige von Präsentationen oder ähnlichem Material. Für dessen Bedienung ist ein Presenter ausreichend, mit dem die Präsentation gestartet, gestoppt und die Folien gewechselt werden können. Im einfachsten Fall ist nicht einmal die Verwendung eines Presenters notwendig, da die Interaktion über die Tastatur erfolgen könnte. Dies hat aber den Nachteil, dass sich der Präsentator von der Präsentation oder dem Publikum abwenden muss. In der weiteren Arbeit ist mit einem Presenter das kabellos an den Computer angebundene Gerät gemeint, womit Präsentationen gesteuert werden können. Ein Presenter ist oft mit einem Laserpointer ausgestattet.

Eine direkte Interaktion mit der Leinwand ist meistens nicht möglich, da sie lediglich ein Anzeigemedium ist. Weitere Einsatzmöglichkeiten, die Aktionen mit und an der Projektionsfläche voraussetzen, können zum Beispiel folgende sein:

- Direkte Auswahl und Modifikation von angezeigten Objekten
- Zeichnen auf der Leinwand (Verwendung als Whiteboard)
- Steuern von beliebigen Programmen

In Einzelplatzszenarien können viele dieser Einsatzmöglichkeiten durch die Verwendung von Maus und Tastatur realisiert werden. Jedoch ist dies, wie bereits erwähnt, für eine Präsentation suboptimal.

Eine weitere Möglichkeit ist die Verwendung von 3D- beziehungsweise Gyromäusen. Durch solche Geräte wird es dem Nutzer ermöglicht, den Mauszeiger des Betriebssystems sehr genau zu steuern. Unter Umständen ist hier eine Eingewöhnung durch den Nutzer nötig.

1.1 Motivation

Ziel dieser Arbeit ist es, eine Eingabemöglichkeit zu entwickeln, die die Genauigkeit von Gyromäusen und die Intuitivität von Presentern vereinigt. Durch Zeigen mit einem Laserpointer auf die Leinwand soll der Nutzer den Mauszeiger an die anvisierte Stelle verschieben können. Weitere Mausinteraktionen wie Klicks kann er mit den entsprechenden Tasten des Presenters ausführen. Für Nutzer, die öfter mit einem Presenter und dessen Laserpointer arbeiten, sollte die Bedienung dieses Konzepts intuitiv verwendbar sein.

Die Software soll für einen Windows-Rechner entwickelt werden. Für die Umsetzung des Konzepts ist außerdem eine Kamera notwendig, die die Leinwand erfassen kann.

1.2 Aufbau der Arbeit

Die Arbeit ist folgendermaßen gegliedert. Zunächst werden in Kapitel zwei verwandte Arbeiten vorgestellt, da einige der darin verwendeten Konzepte in dieser Arbeit Anwendung finden. Kapitel drei beschäftigt sich mit den theoretischen Grundlagen, die zum Verständnis notwendig sind und führt einheitliche Notationen ein. Im vierten Kapitel werden die Vorgehensweisen der Software erklärt, deren Implementation in Kapitel fünf erläutert wird. Abschließend erfolgt eine Evaluation und Auswertung des entwickelten Systems in Kapitel sechs.

2 Verwandte Arbeiten

Das Konzept, einen Laserpointer als Eingabemöglichkeit zu verwenden, wurde bereits in einigen Projekten aufgegriffen. Nachfolgend werden drei Vertreter vorgestellt, bei denen Quellcode und / oder Dokumentation vorhanden ist. Gute Ansätze aus diesen Systemen wurden in diese Arbeit übernommen und problematische Implementierungen verbessert. Außerdem werden Veröffentlichungen vorgestellt, in denen verwendete Verfahren präsentiert werden.

2.1 Projekte zur Mausemulation mithilfe eines Laserpointers

2.1.1 Presenter Mouse

Dieses am Israelischen Institut für Technologie (Technion) entwickelte Projekt [PKG03] dient der Verfolgung eines roten Laserpointers. Andersfarbige Laser werden nicht unterstützt. Da bei vielen Kameras Laserpunkte als weiße Stellen wahrgenommen werden, können die meisten Presenter verwendet werden, jedoch ist die Erkennungszuverlässigkeit dann niedriger.

Zur Detektion des Pointers wird eine Hintergrundsubtraktion ausgeführt, wie sie auch in dieser Arbeit verwendet wird (siehe Kapitel 4.1). Anschließend wird der hellste Punkt ermittelt, der den Laserpointerkandidaten darstellt. Weitere Tests ermitteln, ob es sich um einen Laser handelt. Im positiven Fall wird der Mauscursor entsprechend gesetzt.

In den durchgeführten Testläufen verhielt sich die Erkennung jedoch sehr unzuverlässig. Fehlerkennungen führten zum Flackern des Mauszeigers.

Die Kalibrierung erfolgt durch Einzeichnen der Bildschirmecken im Kamerabild. Die Positionierung der Kamera hinter der Leinwand ist nicht möglich, da dadurch die Kalibrierung nicht mehr durchführbar ist, weil der Desktop schlecht beziehungsweise nicht sichtbar ist. Verzerrungen durch die Linse werden, wie bei allen nachfolgend vorgestellten Projekten, vernachlässigt. Dies ist für viele Webcam-Modelle legitim, da die Entzerrung in der Kamera oder vom Treiber vorgenommen wird. Andere Geräte sind aber nicht geeignet.

2.1.2 Laser Interaction

Mit diesem Projekt [Col08] sollen Bewegungen mit einem Laserpointer auf den Mauszeiger übertragen werden. Die Ausmaße des Kamerabilds werden dabei mit denen des Desktops gleichgesetzt, sodass der Mauszeiger in die obere linke Ecke des Bildschirms verschoben wird, wenn sich der Laserpunkt oben links im Bild befindet. Dadurch kann die Kamera einen beliebigen Raum überwachen und muss nicht auf die Leinwand gerichtet sein. Eine Kalibrierung ist nicht notwendig. Der verwendete Maximumdetektor wird in dieser Arbeit in einer ähnlichen Form genutzt.

Aufgrund dieses Konzepts ist es allerdings nicht möglich, dass ein auf die Leinwand gerichteter Laserpunkt direkt auf die Stelle auf dem Desktop projiziert wird, die dieser Position entspricht. Dieses Verhalten ähnelt der Bedienung mit einer WiiMote.

Da zur Auswertung des Kamerabilds nur wenige Verarbeitungsschritte notwendig sind, ist dieses System sehr schnell. Bei einer Kameraauflösung von 640 x 480 Pixeln wird auf dem Testsystem eine Framerate von etwa 30 fps erreicht.

2.1.3 Laser Tag

Das Laser Tag-Projekt [WL08] verfolgt das Ziel, mit einem Laserpointer auf einer beliebigen Fläche zu zeichnen. Dazu erfolgen eine Kalibrierung, sowie eine Detektion des Laserpunkts.

Die Kalibrierung wird ähnlich wie bei dem Projekt Presenter Mouse durchgeführt. Die Kanten des Projektionsbereichs müssen im Kamerabild markiert werden, sodass die Korrespondenzen hergestellt werden können. Auch hier ist es nicht möglich, die Kamera hinter der Leinwand zu positionieren.

Zur Detektion wird das Kamerabild in den HSV-Farbraum konvertiert und eine Blob-Detektion ausgeführt. Dazu wird das Bild mit einem Schwellwert versehen, der es in ein binäres Bild konvertiert. Dieses Verfahren wird auch in dieser Arbeit genutzt und genauer in Kapitel 4.1.2 erläutert. Der größte Blob wird als Laserpunkt interpretiert und entsprechende Linien gezeichnet. Eine Erkennung von mehreren Punkten ist prinzipiell möglich, jedoch nicht vorgesehen.

Das Projekt nutzt einige OpenSource-Frameworks. Unter anderem die in dieser Arbeit verwendete Video Input Library.

2.2 Kamerakalibrierung

Kamerabilder sind im Allgemeinen Verzerrungen unterworfen, die vor einer Verarbeitung entfernt werden müssen. Es existieren viele Ansätze, um dieses Problem zu adressieren. [Zha00] und [Bou10] zeigen in ihren Arbeiten ein Verfahren, bei dem ein bekanntes Referenzobjekt genutzt wird, um Korrespondenzen zwischen Szenen- und Bildraum aufzustellen. Aus diesen werden mit einem Optimierungsverfahren Parameter zur Entzerrung des Kamerabilds ermittelt werden. Das Vorgehen zur Kalibrierung der Kamera nach [Zha00] wird in Kapitel 3.3 erläutert.

Ein weiterer Ansatz ist die Selbstkalibrierung [FLM92] [MF92]. Dabei sind keine Referenzobjekte wie bei der Methode nach [Zha00] notwendig. Im Kamerabild werden markante Punkte gesucht und während der Bewegung der Kamera verfolgt. Daraus ergeben sich Korrespondenzen in verschiedenen Bildern, aus denen die Kalibrierungsparameter berechnet werden.

2.3 Videotracking

Videotracking bezeichnet den Prozess, ein Objekt in einem Bild zu finden und in einem darauffolgenden Video zu verfolgen. Ein Überblick über die verbreitetsten Verfahren wird in [YJS06] gegeben.

Der Vorgang des Trackings beginnt mit der Auswahl einer geeigneten Repräsentation des zu suchenden Objekts. Da im Rahmen dieser Arbeit ein Laserpunkt gesucht werden muss, der wenige Pixel groß ist, eignet sich hauptsächlich die Darstellung als Punkt [VRB01]. Zur Detektion ist temporär die Repräsentation als Kontur [Yil04] und Ellipse [CRM03] hilfreich.

Zur Erkennung von Punkten werden oft Ecken beziehungsweise Feature Points genutzt. Diese zeichnen sich dadurch aus, dass die Ableitungen des Bilds in allen Richtungen hoch sind [TH98], womit eine Erkennung ermöglicht wird. Im Rahmen dieser Arbeit durchgeführte Versuche zeigen aber, dass diese Verfahren schlecht geeignet sind, um Laserpunkte zu erkennen. Stattdessen wird ein ähnliches Hintergrundsubtraktionsverfahren verwendet wie in [IDB97].

Zusätzlich zur Erkennung erfolgt beim Videotracking eine Zuordnung der gefundenen Elemente zu den eigentlichen Objekten im Bild, sodass deren Bewegung verfolgt werden können. Dieser Schritt entfällt in dieser Arbeit, weil mehrere Laserpunkte zwar erkannt werden können, aber eine Umsetzung auf die Mausbewegung nur für den ersten gefundenen Punkt erfolgt.

3 Grundlagen

Im folgenden Kapitel werden die Hintergründe der eingesetzten Verfahren erläutert. Dazu gehören neben der Verwendung und Berechnung von perspektivischen Transformationen Verfahren zur Kamerakalibrierung und Algorithmen, mit denen Konturen in einem Bild gefunden, die Zugehörigkeit eines Punkts zu einem Polygon ermittelt, sowie der Flächeninhalt eines Polygons berechnet werden kann. Auf diesen Grundlagen bauen alle Schritte auf, die die Software zur Erfüllung ihrer Aufgabe ausführen muss. Dabei werden einheitliche Notationen eingeführt, die im weiteren Verlauf der Arbeit Anwendung finden.

3.1 Perspektivische Transformationen

Transformationen dienen dazu, Vektoren aus einem Koordinatensystem in ein anderes zu überführen. Im Rahmen dieser Arbeit werden sie unter anderem dazu verwendet, um Punkte im Kamerabild auf ihre Originalpositionen abzubilden. Für alle weiteren Verfahren wird der zweidimensionale Vektorraum $\mathbb{R}^2$ über den reellen Zahlen zugrunde gelegt. Elemente dieses Raums sind zweielementige Spaltenvektoren, die folgendermaßen bezeichnet werden:

$$v = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \in \mathbb{R}^2$$
$$v_x, v_y \in \mathbb{R}$$

Für perspektivische Transformationen (Homographien) muss dieser zugrundeliegende Körper um eine Dimension erweitert werden und bildet damit den homogenen zweidimensionalen Vektorraum $\mathbb{R}^3$. Dessen Elemente werden wie folgt dargestellt:

$$\underline{v} = \begin{pmatrix} \underline{v_x} \\ \underline{v_y} \\ \underline{v_w} \end{pmatrix} \in \mathbb{R}^3$$
$$\underline{v_x}, \underline{v_y}, \underline{v_w} \in \mathbb{R}$$

Die w-Komponente bezeichnet das Inverse des Skalierungsfaktors des Vektors. Ein homogener Vektor kann demnach wie folgt in einen inhomogenen Vektor überführt werden:

$$v = \frac{1}{\underline{v_w}} * \begin{pmatrix} \underline{v_x} \\ \underline{v_y} \end{pmatrix}$$
$$\varphi : \mathbb{R}^3 \to \mathbb{R}^2 : \underline{v} \mapsto v$$

Die Funktion φ bezeichnet diese Abbildung eines homogenen auf seinen inhomogenen Vektor. Da diese Abbildung nicht injektiv ist, können mehrere homogene Vektoren Repräsentanten desselben inhomogenen Vektors sein.

Perspektivische Transformationen werden durch Matrizen aus dem Raum $\mathbb{R}^{3\times3}$ beschrieben. Matrizen werden mit Großbuchstaben bezeichnet. Die Transformation mit einer Matrix entspricht folgender Multiplikation:

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix} \in \mathbb{R}^{3\times 3}$$
$$\underline{v'} = M * \underline{v}$$

Die Multiplikation zweier Matrizen entspricht der Hintereinanderausführung der beiden Transformationen.

Da durch perspektivische Abbildungen Elemente des ursprünglichen Vektorraums $\mathbb{R}^2$ transformiert werden sollen, wird nachfolgend diese Operation definiert. Weil es sich bei den Vektoren um Ortsvektoren handelt, werden die inhomogenen Vektoren in den homogenen Raum überführt, indem „1“ als w-Komponente hinzugefügt wird:

$$* : \mathbb{R}^{3\times 3} \times \mathbb{R}^2 \to \mathbb{R}^2 : (M, v) \mapsto \varphi\left[M * \begin{pmatrix} v_x \\ v_y \\ 1 \end{pmatrix}\right]$$
$$v' = M * v$$

3.2 Ermitteln von Transformationen aus Punktkorrespondenzen

Im vorigen Kapitel wurde der Umgang mit perspektivischen Transformationen erklärt. Nachfolgend wird erläutert, wie eine Homographie auf der Basis von Punktkorrespondenzen berechnet werden kann. Dieses Verfahren wird beispielsweise dazu verwendet, um die Ausrichtung der Kamera bezüglich der Projektionsfläche zu ermitteln.

Eine Korrespondenz ist ein Paar bestehend aus einem Punkt p im Ausgangskoordinatensystem und seinem Bild p' im Zielsystem. Ziel ist es, eine optimale Transformation H zu finden, die alle Punkte der Korrespondenzen auf Positionen abbildet, sodass der kumulierte Fehler zu den Bildern minimal ist.

Aufgrund der Mehrdeutigkeit von homogenen Vektoren, gibt es zu einer homogenen Matrix weitere Matrizen, die zu dieser äquivalent sind. Alle Vielfachen (ausgenommen das 0-fache) verhalten sich gleichwertig bezüglich der Abbildung von inhomogenen Vektoren. Aufgrund dessen kann die Anzahl der Freiheitsgrade der zu berechnenden Homographie von neun auf acht gesenkt werden, indem ein Eintrag festgesetzt wird:

$$H = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix}$$

Wie in Kapitel 3.1 beschrieben, wird ein Punkt p zu folgendem Bild transformiert:

$$p' = \frac{1}{\underline{p'_w}} * \begin{pmatrix} \underline{p'_x} \\ \underline{p'_y} \end{pmatrix}$$
$$\underline{p'} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{pmatrix} * \begin{pmatrix} p_x \\ p_y \\ 1 \end{pmatrix} = \begin{pmatrix} h_{11}p_x + h_{12}p_y + h_{13} \\ h_{21}p_x + h_{22}p_y + h_{23} \\ h_{31}p_x + h_{32}p_y + 1 \end{pmatrix}$$
$$p' = \frac{1}{h_{31}p_x + h_{32}p_y + 1} \begin{pmatrix} h_{11}p_x + h_{12}p_y + h_{13} \\ h_{21}p_x + h_{22}p_y + h_{23} \end{pmatrix}$$

Diese Abbildung kann wie folgt auf eine lineare Gleichung gebracht werden [PKG03]:

$$p' * (h_{31}p_x + h_{32}p_y + 1) = \begin{pmatrix} h_{11}p_x + h_{12}p_y + h_{13} \\ h_{21}p_x + h_{22}p_y + h_{23} \end{pmatrix}$$

$$p'h_{31}p_x + p'g_{32}p_y + p' = \begin{pmatrix} h_{11}p_x + h_{12}p_y + h_{13} \\ h_{21}p_x + h_{22}p_y + h_{23} \end{pmatrix}$$

$$p' = \begin{pmatrix} h_{11}p_x + h_{12}p_y + h_{13} - p'_x h_{31}p_x - p'_x h_{32}p_y \\ h_{21}p_x + h_{22}p_y + h_{23} - p'_y h_{31}p_x - p'_y h_{32}p_y \end{pmatrix}$$

$$p' = \begin{pmatrix} p_x & p_y & 1 & 0 & 0 & 0 & -p'_x p_x & -p'_x p_y \\ 0 & 0 & 0 & p_x & p_y & 1 & -p'_y p_x & -p'_y p_y \end{pmatrix} * \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix}$$

Jede Korrespondenz fügt zu der Koeffizientenmatrix die angegebenen zwei Zeilen hinzu. Da die gesuchte Homographie acht Freiheitsgrade hat, sind vier Korrespondenzen nötig, um die Gleichung zu lösen. Deren Lösung definiert die Transformationsmatrix, die Vektoren aus dem Ausgangskoordinatensystem in das Zielsystem abbildet.

Um Messfehler beim Ermitteln der Korrespondenzen auszugleichen, wird mit der Methode der kleinsten Quadrate eine optimale Lösung gesucht. Dazu wird die transponierte Koeffizientenmatrix von links an beide Seiten der Gleichung multipliziert. Diese Gleichung hat in jedem Fall eine Lösung. Eine sinnvolle Interpretation ist jedoch erst möglich, wenn es mindestens vier Korrespondenzen gibt, die zu linear unabhängigen Gleichungen führen. Stehen mehr Paare zur Verfügung, erhöht dies im Allgemeinen die Genauigkeit der gefundenen Homographie.

3.3 Kamerakalibrierung

Bei vielen Kameras treten neben der linearen Transformation weitere Verzerrungen auf, die durch die Linse beeinflusst werden [BK08]. Hauptsächlich handelt es sich dabei um radiale und tangentiale Verzerrungen. Im folgenden Abschnitt werden die beiden Arten erläutert und Verfahren zur Entzerrung vorgestellt.

Bei radialer Verzerrung handelt es sich um Verschiebungen der Pixel entlang von Strahlen, die durch den Bildmittelpunkt verlaufen [Ste97]. Punkte, die im Originalbild denselben Abstand zum Mittelpunkt haben, haben auch nach der Verzeichnung denselben Abstand. Abbildung 3.1 zeigt zwei Beispiele für radiale Verzerrungen mit unterschiedlichen Parametern.

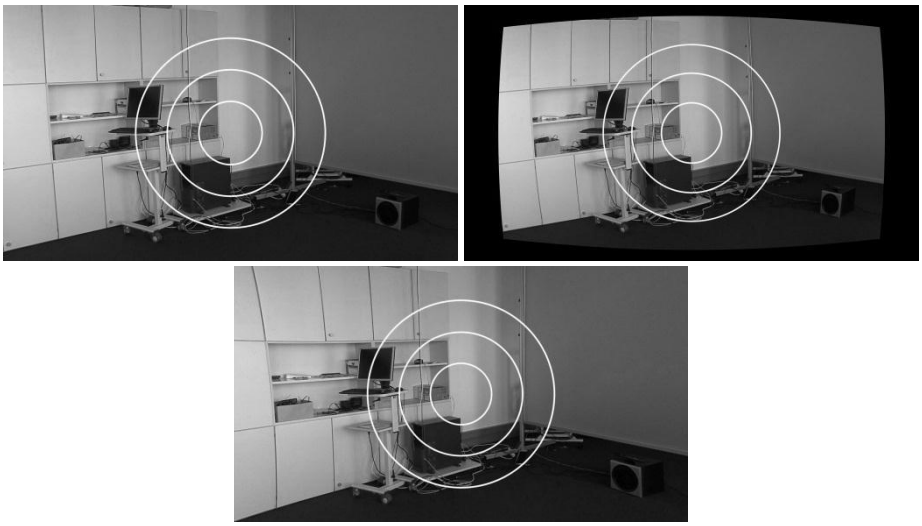

Abbildung 3.1 Beispiel für radiale Verzerrung; Links: Originalbild; Rechts: Tonnenförmige Verzerrung; Unten: Kissenförmige Verzerrung

Zur Korrektur der radialen Verzeichnung müssen die Pixel entlang des Mittelpunktstrahls zurück an ihre ursprüngliche Position verschoben werden. Die Stärke der Verschiebung ist abhängig vom Abstand zum Bildmittelpunkt und kann durch folgende Taylorentwicklung angenähert werden:

$$x_{korrigiert} = x * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{korrigiert} = y * (1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Der Parameter r bezeichnet den Abstand eines Pixels zum Bildmittelpunkt. Dieser liegt auf dem Koordinatenursprung. Die Koeffizienten k_1 bis k_3 müssen während einer Kalibrierung ermittelt werden. Bei einer tonnenförmigen Verzeichnung sind diese Parameter positiv, bei einer kissenförmigen Verzeichnung negativ. Mischformen können existieren.

Tangentiale Verzeichnungen ergeben sich aus der fertigungsbedingten Verschiebung des Bildmittelpunkts und der optischen Achse. Ihre Korrektur erfolgt auf Grundlage von zwei Parametern p_1 und p_2 durch folgende Gleichung:

$$x_{korrigiert} = x + \left(2p_1 y + p_2(r^2 + 2x^2)\right)$$
$$y_{korrigiert} = y + (p_1(r^2 + 2y^2) + 2p_2 x)$$

Neben den fünf Parametern muss während der Kalibrierung noch die sogenannte intrinsische Matrix M_i der Kamera ermittelt werden. Diese Matrix transformiert das aufgenommene Kamerabild dahingehend, dass der Koordinatenursprung in der oberen linken Ecke liegt und dessen Ausmaße dem Pixelmaß entsprechen. Diese Matrix ist folgendermaßen aufgebaut:

$$M_i = \begin{pmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix}$$

Zur Kalibrierung werden Referenzobjekte (zum Beispiel ein Schachbrett) in verschiedenen Ausrichtungen aufgenommen und Korrespondenzen zwischen dem Originalkoordinatensystem des Objekts und dem Bildraum definiert. Mit dem in Kapitel 3.2 vorgestellten Verfahren können die zugehörigen Transformationsmatrizen bestimmt werden. Die Bilder der Ausgangspunkte sind definiert durch:

$$\begin{pmatrix} p'_x \\ p'_y \end{pmatrix} = H * \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

Die Homographie H enthält einen perspektivischen Anteil, der die Vektoren auf die Bildebene projiziert. Wird dieser Anteil entfernt, verbleiben folgende Einträge:

$$H' = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ 0 & 0 & 1 \end{pmatrix}$$

Diese Matrix transformiert Punkte aus dem Szenenraum in den Kameraraum ohne Projektion auf die Bildebene. Somit ist sie eine Kombination aus intrinsischer und extrinsischer Matrix. Letztere definiert die Ausrichtung der Kamera in der Szene und ist demnach auf eine Rotation und Translation beschränkt:

$$H' = M_i * M_e$$
$$M_e = (r_1 \quad r_2 \quad t)$$

Die intrinsische Matrix M_i ist für alle Blickwinkel gleich, während die extrinsische Matrix M_e spezifisch pro Ansicht ist. In [Zha00] wird gezeigt, wie aus den aufgenommenen Homographien die intrinsischen und extrinsischen Parameter extrahiert werden können.

Diese Parameter dienen als Ausgangspunkt für die Abschätzung der Verzerrungsparameter. Mit einem iterativen Verfahren werden diese sowie die intrinsische Matrix optimiert, sodass für alle Ansichten des Referenzobjekts ein minimaler Gesamtfehler entsteht.

3.4 Konturendetektion

Das Auffinden von Konturen in einem binären Bild, dargestellt als Matrix $M \in \{0,1\}^{höhe \times breite}$, ist ein wesentlicher Schritt für die Implementation. Er wird zum Beispiel dazu verwendet, um im Kamerabild diejenigen Polygone zu finden, die einen Laserpunkt darstellen. Nachfolgend wird ein Algorithmus vorgestellt, der dies für einfache Polygone leisten kann [SA85].

Ausgangspunkt für die Detektion ist ein zeilenweiser Scan des gesamten Bilds. Sobald auf einen Pixel mit dem Wert 1 getroffen wird, der einen linken Nachbarn mit dem Wert 0 hat, wird eine Kantenverfolgungsprozedur gestartet:

Algorithmus 3.1 Scan zur Konturendetektion

```
konturen := ∅
inPolygon := false
for y from 0 to höhe − 1
        inPolygon := false
        for x from 0 to breite − 1
                if Pixel(x, y) == 1 && Pixel(x − 1, y) == 0 && NOT inPolygon then
                        Kantenverfolgung(x, y)
                        inPolygon := true
                end if
                *
        next
next
return konturen
```

In der Variablen $konturen$ werden alle gefundenen Konturen gespeichert. Die Variable $inPolygon$ gibt an, ob sich der aktuelle Pixel innerhalb eines bereits gefundenen Polygons befindet. In diesem Fall wird keine Kantenverfolgung gestartet. Dadurch werden Konturen, die sich innerhalb von anderen Konturen befinden ignoriert, da nur alle äußeren Kanten ermittelt werden sollen. Zu Beginn jeder Zeile wird diese Variable auf $\boldsymbol{false}$ zurückgesetzt, da die linke Bildkante nie innerhalb einer Kontur ist. Sie kann höchstens selbst eine solche sein. An der mit $*$ markierten Stelle wird ein weiterer Codeabschnitt eingefügt, mit dem die $inPolygon$-Variable zurückgesetzt wird, nachdem das Polygon passiert wurde. Vorher wird die Kantenverfolgung erklärt.

Bei dieser wird der aktuelle Pixel mit einer „2“ markiert. Wenn der rechte Nachbarpixel den Wert 0 hat, so handelt es sich um die rechte Kante der Kontur, was mit einer „-2“ gekennzeichnet wird. Danach wird zum nächsten Pixel auf der Kante vorangeschritten und gleichermaßen verfahren. Die abgeschrittene Kante wird zum globalen Kantenpuffer hinzugefügt:

Algorithmus 3.2 Kantenverfolgung

```
Parameter x, y
kontur := ∅
while valid((x, y))
        append (x, y) to kontur
        if Pixel(x + 1, y) == 0 then Pixel(x, y) := −2
        else Pixel(x, y) := 2
        (x, y) := adjazenter Pixel mit Wert 1, der einen Nachbarpixel mit Wert 0 hat
end while
add kontur to konturen
```

Die rechten Kanten des Polygons können an der „-2“ erkannt werden. Somit kann Algorithmus 3.1 durch Einfügen des folgenden Codes vervollständigt werden:

Algorithmus 3.3 Ergänzung zu Algorithmus 3.1 (einfügen an $*$)

```
if Pixel(x, y) == −2 then inPolygon := false
if Pixel(x, y) == 2 then inPolygon := true
```

3.5 Algorithmen auf Polygonen

Viele der verwendeten Verfahren bauen auf Polygonen auf. Beispielsweise kann die projizierte Umrandung des Desktops oder eine Kontur im Bild als Polygon aufgefasst werden. Die Eckpunkte von Polygonen werden durch Vektoren repräsentiert. Nachfolgend sollen zwei wesentliche Verfahren vorgestellt werden, die beim Umgang mit Polygonen auftreten.

3.5.1 Punktlokation

Dieses Verfahren soll ermitteln, ob sich ein Punkt im Innern des Polygons befindet. Es baut auf dem in [Sal78] vorgestellten Algorithmus auf. Durch Punktlokation kann zum Beispiel festgestellt werden, ob sich ein potentieller Laserpunkt in einem Bereich befindet, der zum Desktop gehört.

Die Kanten des Polygons trennen jeweils das Innere vom Äußeren. Um festzustellen, ob ein Punkt in dem Polygon liegt, wird ein Hilfsstrahl definiert, der im zu untersuchenden Punkt startet und in eine beliebige Richtung zeigt. Um Berechnungen zu sparen, wird oft ein waagerechter oder senkrechter Strahl verwendet.

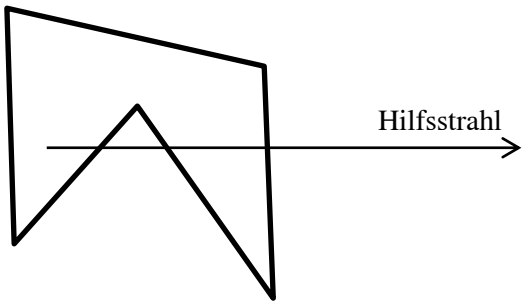

Abbildung 3.2 Hilfsstrahl bei der Punktlokation

Da die Eckpunkte des Polygons endliche Koordinaten haben und der Strahl im unendlichen endet, befindet sich der Endpunkt des Strahls außerhalb des Polygons. Wird ein Punkt auf dem Strahl verschoben, so ändert sich die Position des Punkts relativ zum Polygon (innerhalb oder außerhalb) jedes Mal, wenn eine Kante überschritten wird. Befinden sich zwischen zwei beliebigen Punkten auf dem Strahl eine ungerade Anzahl an Kantenübergängen, so liegen beide Punkte in unterschiedlichen Bereichen des Polygons. Andernfalls sind sie im selben Bereich. Da der (unendliche) Endpunkt des Strahls außerhalb des Polygons liegt, kann anhand der Anzahl der Übergänge auf dem Strahl auf die Position des zu untersuchenden Punkts geschlossen werden. Dementsprechend gilt stets, dass sich der Punkt innerhalb des Polygons befindet, wenn es eine ungerade Anzahl von Schnitten des Strahls mit den Kanten des Polygons gibt. In diesem Fall sind Anfangs- und Endpunkt des Strahls in unterschiedlichen Bereichen. Andernfalls befindet sich der Punkt außerhalb. Numerische Ungenauigkeiten sind in dem angegebenen Algorithmus vernachlässigt.

Algorithmus 3.4 Ermitteln, ob sich ein Punkt p innerhalb eines Polygons befindet

$n \coloneqq 0$
$strahl \coloneqq beliebiger\ Strahl\ ausgehend\ von\ p$
$\boldsymbol{for\ all}\ Kante\ k\ \boldsymbol{in}\ polygon$
    $\boldsymbol{if}\ Schnitt(k, strahl)\ \boldsymbol{then}\ n \coloneqq n + 1$
$\boldsymbol{next}$
$\boldsymbol{return}\ n\ mod\ 2 == 1$

3.5.2 Flächeninhalt von Polygonen

Der Flächeninhalt von Polygonen wird im Rahmen dieser Arbeit dazu verwendet, um festzustellen, ob ein Polygon einen Laserpunkt darstellen kann. Dazu werden Hilfsdreiecke erzeugt, deren Flächeninhalt mit dem Kreuzprodukt berechnet werden kann [Bou88].

Vorerst wird ein Hilfspunkt definiert, von dem ausgehend die Dreiecke aufgespannt werden. In Abbildung 3.3 sind diese Hilfsdreiecke beispielhaft für den Hilfspunkt O und die beiden Kanten x_2x_3 und x_1x_2 gezeigt. Es wird jeweils ein Dreieck pro Kante aufgespannt. Die Wahl des Hilfspunkts ist beliebig, sodass im einfachsten Fall der Koordinatenursprung genutzt werden kann.

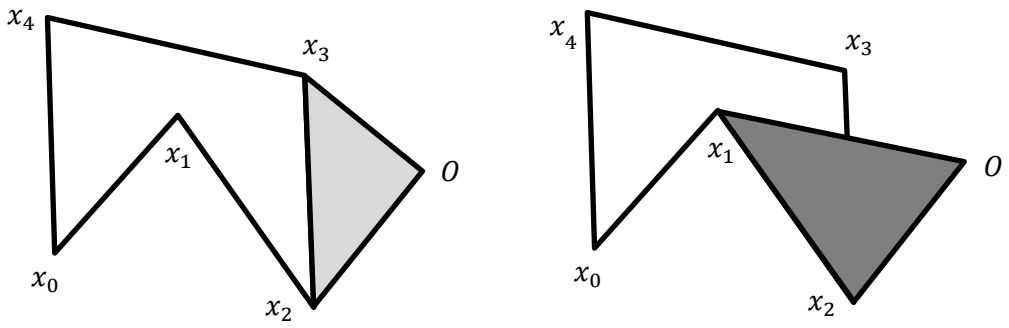

Abbildung 3.3 Zwei der fünf Hilfsdreiecke zur Bestimmung des Flächeninhalts von Polygonen

Der Flächeninhalt der Hilfsdreiecke ist definiert durch den Betrag des Kreuzprodukts zweier Kanten. Um das Kreuzprodukt berechenbar zu machen, wird die z-Komponente 0 eingefügt. Da beide Kanten dann parallel zur xy-Ebene sind, sogar in ihr liegen, sind die x- und y-Komponenten des Kreuzprodukts 0 und der Betrag reduziert sich auf die z-Komponente:

$$A_{Dreieck} = \frac{1}{2}|((x_i - O) \times (x_{i+1} - O))|$$
$$A_{Dreieck} = \frac{1}{2}|(x_i \times x_{i+1})|$$
$$A_{Dreieck} = \frac{1}{2}\left|\begin{pmatrix} x_{i_x} \\ x_{i_y} \\ 0 \end{pmatrix} \times \begin{pmatrix} x_{i+1_x} \\ x_{i+1_y} \\ 0 \end{pmatrix}\right|$$
$$A_{Dreieck} = \frac{1}{2}\left|x_{i_x} * x_{i+1_y} - x_{i_y} * x_{i+1_x}\right|$$

Die Hilfsdreiecke können zwei verschiedene Orientierungen aufweisen. Im Beispiel ist das hellgraue Dreieck Ox_2x_3 im Uhrzeigersinn angeordnet. Die z-Komponente des Kreuzprodukts ist somit negativ. Im dunkelgrauen Hilfsdreieck Ox_1x_2 ist diese positiv, da es gegen den Uhrzeigersinn angeordnet ist. Werden alle Hilfsdreiecke addiert, heben sich die Flächen außerhalb des Polygons auf und es bleiben nur die Flächen innerhalb übrig. Somit ist der Flächeninhalt des Polygons unter Zuhilfenahme der vorzeichenbehafteten Dreiecksflächen:

$$A_{Polygon} = \sum_i A_{Dreieck_{i_\pm}}$$
$$A_{Polygon} = \frac{1}{2}\sum_i \left(x_{i_x} * x_{i+1\ mod\ N_y} - x_{i_y} * x_{i+1\ mod\ N_x}\right)$$
$$mit\ N \stackrel{\text{def}}{=} Anzahl\ der\ Eckpunkte$$

Algorithmus 3.5 Flächeninhalt eines Polygons

```
A := 0
for all Kante k in polygon
        a := k.Anfangspunkt
        e := k.Endpunkt
        A := A + (a.x * e.y − a.y * e.x)
next
return A/2
```

4 Mausemulation mittels eines Laserpointers

Im folgenden Kapitel wird darauf eingegangen, wie die Emulation einer Maus umgesetzt wurde. Zuerst werden die zugrundeliegenden Konzepte und Strategien erläutert und anschließend die Implementierung erklärt.

Die Emulation besteht aus zwei getrennten Prozessen: der Steuerung des Mauszeigers und dem Auslösen der Mausknöpfe. Letzterer Prozess erfordert keine weiteren Grundlagen. Deswegen wird im ersten Teil dieses Kapitels nur auf die Mauszeigersteuerung eingegangen.

Dazu sind folgende Teilschritte notwendig:

1. Detektion des Laserpointers im Kamerabild
2. Ermitteln der Position auf dem Desktop
3. Setzen der Mauszeigerposition

Abbildung 4.1 zeigt den Aufbau des Systems mit allen Komponenten. Im Beispiel befindet sich die Kamera vor der Leinwand. Eine Positionierung dahinter ist ebenso möglich und sogar empfehlenswert. In diesem Fall hat das angezeigte Bild auf der Leinwand einen geringeren Einfluss auf die Detektion, was zu einer höheren Zuverlässigkeit führt.

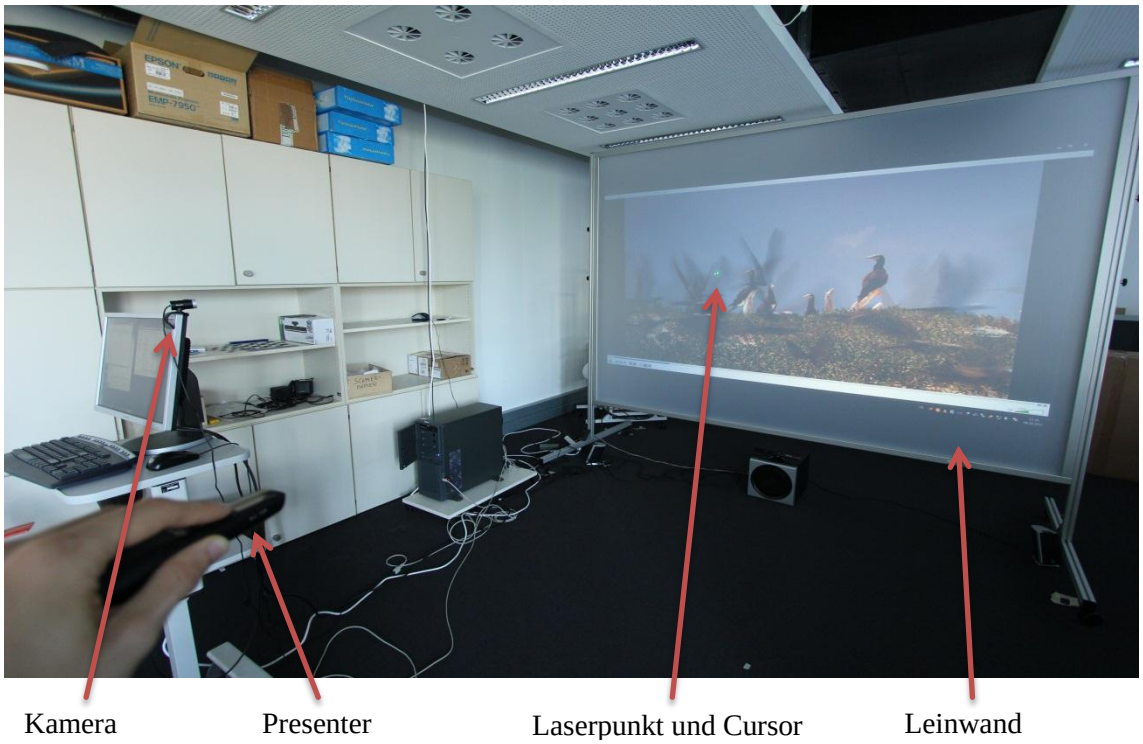

Abbildung 4.1 Aufbau des Systems

4.1 Detektion des Laserpointers

Der erste Schritt ist die Erkennung der Position, auf die der Nutzer mit dem Laserpointer zeigt. Dafür haben sich ein Maximum- und ein Blobdetektor als sinnvoll herausgestellt, die auf ähnlichen

Vorverarbeitungen des Kamerabilds aufbauen. Nachfolgend werden diese Gemeinsamkeiten vorgestellt. Eine detaillierte Betrachtung der beiden Verfahren erfolgt in den Kapiteln 4.1.1 und 4.1.2.

Der erste Vorverarbeitungsschritt ist der Existenz verschiedenfarbiger Laserpointer geschuldet. Häufig sind rote und grüne Laser im Einsatz, aber auch andere Farben sind möglich und sollen genauso zuverlässig erkannt werden. Alle Laser haben eines gemeinsam: Sie sind sehr hell. Um dieser Eigenschaft nachzugehen, wird das Ausgangsbild in den HSV-Farbraum überführt [SCB87]. Die V-Komponente (Value) ist genau dann maximal, wenn einer der Farbkanäle im RGB-Bild maximal ist. Sie entspricht der Farbintensität. Mit ihrer hohen Helligkeit haben Laser demnach eine hohe Intensität und können so ermittelt werden. Daher wird in allen weiteren Schritten nur noch der V-Kanal betrachtet. Eine Überführung in den YCbCr-Farbraum und weitere Nutzung des Y-Kanals wäre auch denkbar. Allerdings würden dann z.B. rote Pixel mit dem RGB-Wert (1, 0, 0) nicht die volle Helligkeit haben. Solche Pixel können aber zu einem roten Laserpointer gehören, wenn die verwendete Kamera entsprechend hochwertig ist und den Pixel nicht als weiß wahrnimmt. Im HSV-Farbraum haben sowohl der rote als auch der weiße Pixel maximale Intensität V, was diesen geeigneter macht.

Zur Erkennung des Laserpunkts wird ausgenutzt, dass sich bei Präsentationen das angezeigte Projektorbild oft nur geringfügig ändert. Eine weitere Möglichkeit der Änderung des Bilds ist der komplette Neuaufbau von einem Frame zum nächsten. Daher kann während der Emulation der Hintergrund (das angezeigte Bild) vom Vordergrund (der Laserpointer) getrennt werden. Ergebnis dieser Hintergrundsubtraktion ist ein Vordergrundbild, in dem im Idealfall nur noch der Laserpunkt enthalten ist, was dessen Erkennung vereinfacht.

Zur Hintergrundberechnung werden eine bestimmte Anzahl der vorhergehenden Bilder mit einem Box-Filter gefaltet. Dieser Filter erlaubt eine effiziente und performante Implementierung und erreicht ausreichend gute Resultate. Die Verwendung eines anderen Filters (zum Beispiel einer aufsteigenden Gerade, bei der das letzte Bild stärker gewichtet wird) ist ebenso möglich. Die Faltung mittels Box-Filter wird durch folgende Formel beschrieben:

$$\bar{M} = \frac{1}{n} \sum_{i=k-n+1}^{k} Bild_i$$

Dabei bezeichnet n die Breite des Filters und k den Index des zuletzt aufgenommenen Bilds.

Das so berechnete Mittelwertbild kann als Näherung für den Hintergrund des nächsten Bilds angesehen werden. Somit lässt sich der Vordergrund ermitteln, indem vom Gesamtbild das Hintergrundbild subtrahiert wird. Da in dem Bild nur Punkte mit hoher Intensität für weitere Betrachtungen interessant sind, muss das Vordergrundbild nicht auf den Wertebereich der Bilddatenstruktur skaliert werden. Negative Werte können ohne Verlust relevanter Daten auf „0“ abgeschnitten werden.

Um einen vollständigen Bildwechsel zu erkennen, wird das aktuelle Bild mit dem vorhergehenden Bild an einigen Stützstellen auf Unterschiede untersucht. Wenn die daraus resultierende Abweichung der Pixel

$$\sigma^2 = \frac{1}{n} \sum_{i} (p_i - p_i^*)^2$$

einen bestimmten Betrag überschreitet, wird das Hintergrundbild zurückgesetzt und der Boxfilter entsprechend verkürzt. Das Symbol n bezeichnet hier die Anzahl der zu untersuchenden Stellen, p_i den Wert eines aktuellen Pixels und p_i^* den desselben Pixels im vorhergehenden Bild.

Im Idealfall enthält das so ermittelte Vordergrundbild nur den oder die Laserpointer, falls vorhanden. Ein Nachteil ist, dass die Intensität des Lasers im Vordergrundbild durch die Subtraktion geringer wird. Werden Laser und Projektorbild von der Kamera annähernd ähnlich intensiv wahrgenommen (zum Beispiel wenn der Laser auf einem weißen Hintergrundbild ist), ist eine Erkennung nicht mehr möglich. Dieses Problem hätten aber auch andere Verfahren, da auf dem Ausgangsbild der Laserpointer nicht mehr erkennbar ist. Somit ist diese Tatsache nicht als Unzulänglichkeit des Verfahrens anzusehen. Abhilfe würde in diesem Fall die Verwendung einer hochwertigeren Kamera, ein dunkleres Projektorbild oder ein hellerer Laser schaffen.

Abbildung 4.2 zeigt beispielhaft die Ergebnisse der Vorverarbeitungsschritte für den Fall, dass die Kamera vor der Leinwand positioniert ist. Auf dem Desktop wird ein Video angezeigt, weswegen das Hintergrundbild leicht unscharf wirkt. Das subtrahierte Vordergrundbild enthält hauptsächlich den Laserpunkt und einige Artefakte, die bei der Hintergrundsubtraktion übrig geblieben sind.

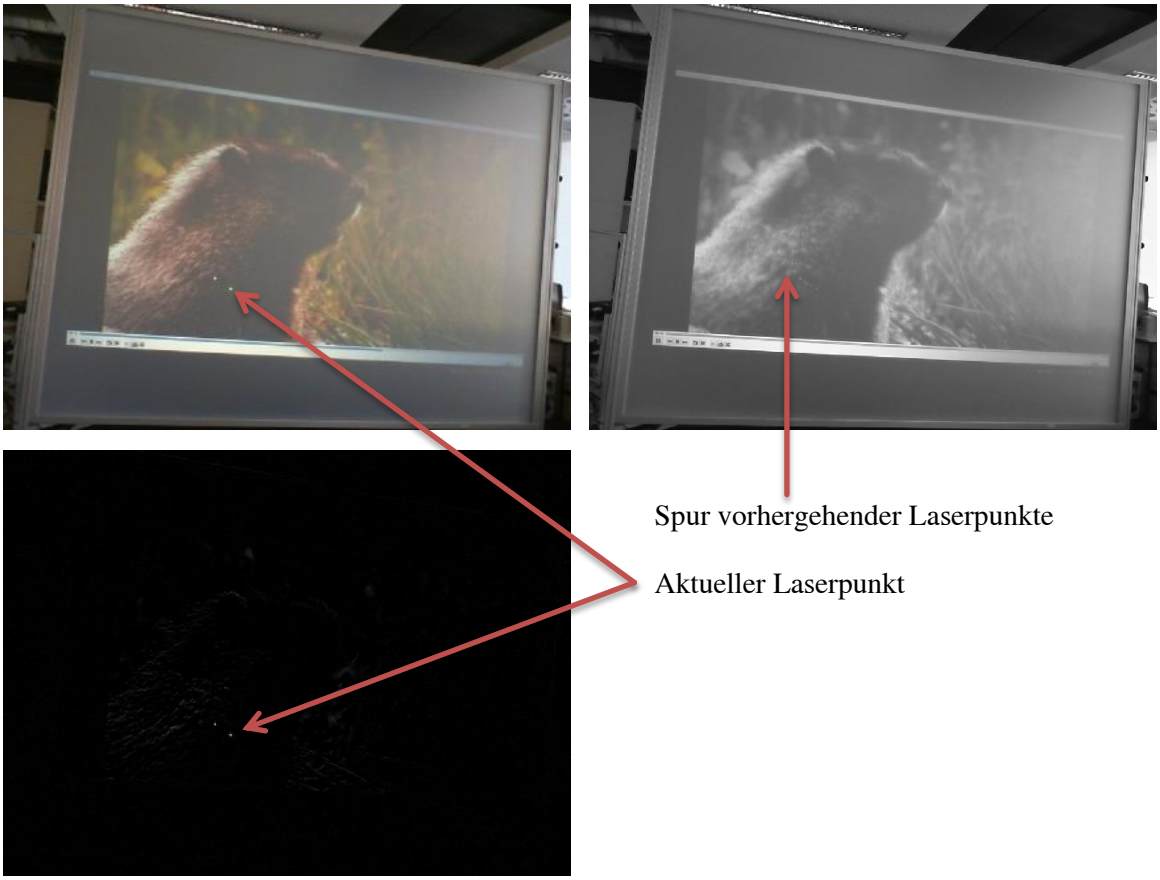

Abbildung 4.2 Ergebnisse der Vorverarbeitungsschritte. Links: aufgenommenes Kamerabild im RGB-Raum; Rechts: gefaltetes Hintergrundbild im HSV-Raum (V-Kanal); Unten: Vordergrundbild (V-Kanal)

Damit sind die Vorverarbeitungsschritte abgeschlossen. Es folgt die eigentliche Erkennung des Laserpointers. Die beiden implementierten Verfahren werden nachfolgend vorgestellt.

4.1.1 Maximumdetektion

Im vorliegenden Vordergrundbild erscheinen die Laserpunkte als helle Pixel. Der hellste Pixel im Bild ist mit hoher Wahrscheinlichkeit ein Pixel im Laserpunkt. Daher wird auf dem Bild eine Suche nach dem globalen Maximum ausgeführt. Ist der Wert des Maximums größer als ein bestimmter Schwellwert, wird davon ausgegangen, dass der Laserpunkt gefunden wurde. Andernfalls ist er nicht existent. Die Maximumsuche wird nur auf dem Bereich ausgeführt, der zum Projektorbild gehört. Näheres zur Bestimmung dieses Bereichs folgt im Kapitel 4.2.3.

Mit diesem Verfahren kann allerdings nur die Position eines Laserpunkts ermittelt werden, was für die meisten Anwendungsfälle ausreicht. Existieren zwei Pixel mit der maximalen Intensität, wird nichtdeterministisch einer der beiden ausgewählt. Wenn mehrere Laserpointer erkannt werden sollen, muss auf das zweite Verfahren zurückgegriffen werden.

Vorteilhaft bei diesem Verfahren ist, dass es sehr stabil und schnell läuft.

4.1.2 Blobdetektion

Das zweite Verfahren nutzt aus, dass ein Laserpunkt immer als eine Ansammlung von hellen Pixeln auftritt. Diese Ansammlungen werden ermittelt, überprüft und im positiven Fall als Laserpunkt interpretiert.

Der erste Schritt dazu ist die Überführung des Vordergrundbilds in ein Binärbild. Dies erfolgt mithilfe eines Schwellwerts. Alle Pixel, die kleiner als dieser Schwellwert sind, werden auf „0“ gesetzt, alle anderen Pixel auf „1“. Das Bild enthält nach diesem Schritt alle potentiellen Laserpunkte als weiße Flecken (Blobs).

Es folgt die Ermittlung der Konturen dieser Blobs, wie in Abschnitt 3.4 beschrieben. Anhand der von diesen Konturen eingeschlossenen Fläche (vgl. Abschnitt 3.5.2) lässt sich bestimmen, ob es sich tatsächlich um einen Laserpunkt handelt. Kleine Blobs gehören mit einer größeren Wahrscheinlichkeit zu einem Laserpunkt als größere. Bei zu großen Punktansammlungen handelt es sich oft um Objekte im Kamerabild, die durch die Hintergrundsubtraktion nicht herausgefiltert werden konnten.

Zur Bestimmung des Zentrums wird um das Polygon eine möglichst optimale Ellipse gelegt, deren Mittelpunkt dem des Polygons weitestgehend entspricht. Sofern sich dieser Punkt innerhalb des Desktopbereichs im Bild befindet (vgl. Abschnitt 3.5.1), wird eine Mausposition an dieser Stelle registriert.

Beide Verfahren unterstützen sogenannte Tote Bereiche. Dabei handelt es sich um Flächen im Kamerabild, in denen keine Detektion ausgeführt wird. Sie wurden eingeführt, da die Rückseite der Powerwall stark reflektiert, wodurch teilweise die Lampe des Projektors sichtbar wird. Durch Hinzufügen eines Toten Bereichs an der Stelle der Lampe können Fehlerkennungen vermieden werden. Da in den Bereichen der Lampe maximale Intensität auftritt, ist eine Erkennung des Laserpunkts in deren Umgebung nicht möglich. Deswegen mindert die Einführung der Toten Bereiche nicht die Erkennungsqualität. Sie wurden zur Stabilisierung der Verfahren in den vorgegebenen Rahmenbedingungen implementiert und stellen damit ein weiteres Softwarefeature dar.

4.2 Berechnung der Position auf dem Desktop

Nach den bisherigen Berechnungsschritten steht fest, an welchen Positionen im Kamerabild sich ein Laserpunkt befindet. Ziel des nächsten Schritts ist es, diese Positionen aus dem Kamerabild auf Positionen auf dem Desktop abzubilden, um den Mauscursor an diese Stelle setzen zu können.

Zu Beginn wird betrachtet, wie das Originalbild des Desktops von der Kamera wahrgenommen wird. Ohne Beschränkung der Allgemeinheit sei in folgender Abbildung der Projektor vor der Leinwand und die Kamera dahinter. Die dargestellten Transformationen sind äquivalent für komplementäre Positionen.

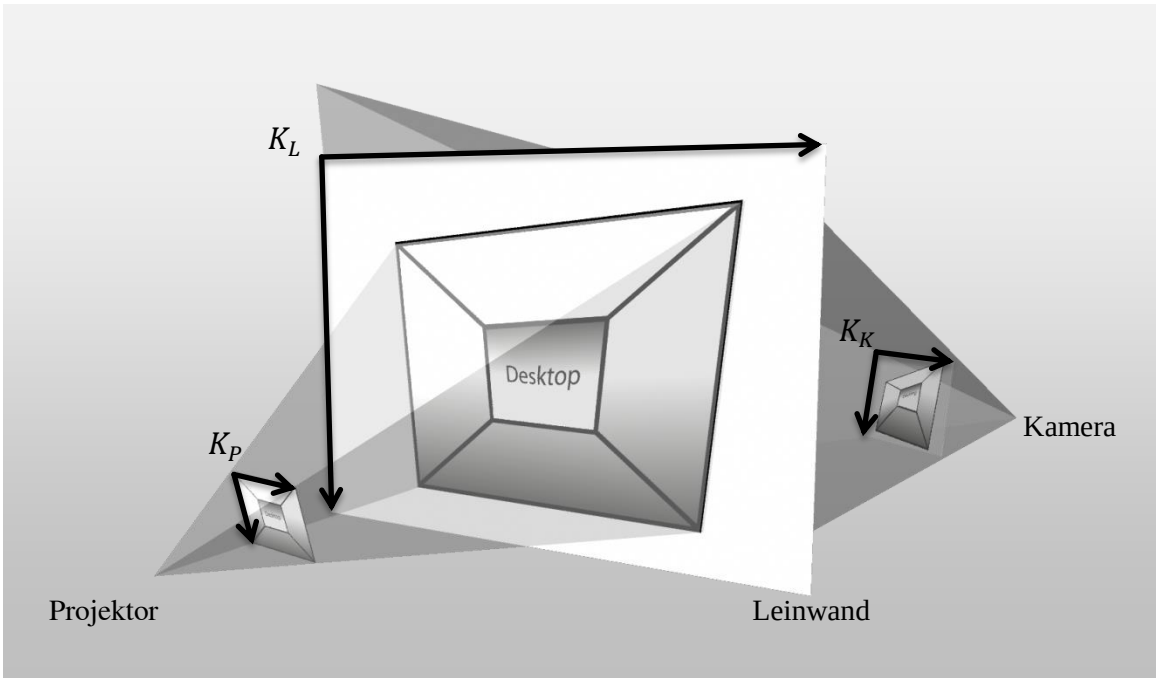

Abbildung 4.3 Transformation des Originalbilds in den Kameraraum

Das Originalkoordinatensystem des Desktops ist das System, das vom Projektor angezeigt wird. Es wird mit K_P bezeichnet. Das Leinwandsystem K_L und das Kamerasystem K_K sind die Systeme an den dargestellten Stellen. Um einen Vektor von einem Koordinatensystem in ein anderes zu überführen können perspektivische Transformationen verwendet werden. Die Transformation $T_{K_L \leftarrow K_P}$ überführt einen Vektor aus dem Koordinatensystem des Projektors in einen Vektor, der diesem im Leinwandsystem entspricht. Dementsprechend wird ein Punkt im Originalsystem auf folgenden Punkt im Kamerasystem abgebildet:

$$v_{K_K} = T_{K_K \leftarrow K_L}\left(T_{K_L \leftarrow K_P} * v_{K_P}\right)$$
$$v_{K_K} = \left(T_{K_K \leftarrow K_L} * T_{K_L \leftarrow K_P}\right) * v_{K_P}$$
$$v_{K_K} = T_{K_K \leftarrow K_P} * v_{K_P}$$

Durch die Zusammenfassung der beiden Abbildungsschritte kann die gesamte Transformation mit einer Matrix dargestellt werden. Die Projektion auf die Leinwand ist für alle weiteren Schritte unerheblich, da diese nur einen Zwischenschritt darstellt.

Zur Berechnung des Punkts im Originalsystem, der einem Punkt im Kamerasystem entspricht, kann die inverse Matrix genutzt werden:

$$\left(T_{K_K \leftarrow K_P}\right)^{-1} * v_{K_K} = \left(T_{K_K \leftarrow K_P}\right)^{-1} * T_{K_K \leftarrow K_P} * v_{K_P}$$
$$\left(T_{K_K \leftarrow K_P}\right)^{-1} * v_{K_K} = v_{K_P}$$
$$T_{K_P \leftarrow K_K} * v_{K_K} = v_{K_P}$$

Anschließend erfolgt eine Verzerrung des Kamerabilds, die durch deren Linse verursacht wird. Bei dieser Verzerrung handelt es sich nicht um eine lineare Transformation, sodass diese nicht mit in die Abbildungsmatrix integriert werden kann. Die Entzerrung erfolgt wie in Kapitel 3.3 beschrieben. Nachfolgend wird erläutert, wie die benötigten Parameter ermittelt werden.

4.2.1 Ermitteln der Verzerrungsparameter

Die Verzerrungsparameter sind für eine Kamera konstant, sofern ihre physikalischen Eigenschaften, wie zum Beispiel die Brennweite, nicht verändert werden. Daher können diese durch eine einmalige Kalibrierung ermittelt werden. Wie in Kapitel 3.3 beschrieben werden dazu Korrespondenzen zwischen dem verzerrten und dem zu berechnenden unverzerrtem Bild aufgestellt. Als Hilfsmittel dient ein Schachbrettmuster, dessen innere Ecken die Punkte für die Korrespondenzen bilden. Als Zielpositionen der Punkte im entzerrten Bild dient immer dieselbe Menge von Punkten, die auf der xy-Ebene als Schachbrett angeordnet sind. Mit dem in Kapitel 3.3 vorgestellten Verfahren werden aufgrund der Korrespondenzen die intrinsischen und Verzerrungsparameter der Kamera berechnet.

4.2.2 Ermitteln der Transformationsparameter

Um die Transformationsmatrix vom Kameraraum in den Desktopraum zu ermitteln wird ein ähnliches Verfahren wie bei der Ermittlung der Verzerrungsparameter angewandt: In beiden Räumen werden äquivalente Punkte gesucht und so Korrespondenzen hergestellt. Die gesuchte Matrix ist diejenige, für die gilt:

$$v_i' = T * v_i \; \forall \; i$$

Diese Matrix wird, wie in Kapitel 3.2 beschrieben, berechnet.

Die Herstellung der Korrespondenzen erfolgt auf zwei unterschiedlichen Wegen, je nachdem, ob sich die Kamera vor oder hinter der Leinwand befindet.

Befindet sie sich vor der Leinwand, ist das Projektorbild deutlich sichtbar. Daher kann auf der Leinwand ein Schachbrettmuster angezeigt werden, dass im Kamerabild automatisch gefunden werden kann. Die Positionen sind dann sowohl im Originalraum als auch im Kameraraum bekannt, woraufhin die Transformationsmatrix berechnet werden kann.

Im Falle, dass sich die Kamera hinter der Leinwand befindet, ist das Bild, das auf die Leinwand projiziert wird, oft schlecht oder nicht sichtbar. Hier werden dem Nutzer vier Punkte vorgegeben, die er mit dem Laserpointer markieren soll. Der Laserpunkt ist dann gut sichtbar, sodass dadurch die Korrespondenzen hergestellt werden können.

4.2.3 Ermitteln des Desktopbereichs im entzerrten Kamerabild

Ein weiterer Datensatz, der während der Kalibrierung erfasst wird, sind die Ausmaße des Desktops im Originalsystem. Diese werden als Koordinaten des Rechtecks abgespeichert, das auf der Leinwand

angezeigt wird. Das ist notwendig, da durch Multidisplayeinstellungen des Betriebssystems die linke obere Ecke des Originalsystems nicht immer (0, 0) ist.

Somit ist nach der Kalibrierung das Desktop-Polygon bekannt. Um dieses in den Kameraraum zu überführen, werden dessen Eckpunkte mit der ermittelten Matrix transformiert:

$$v' = T_{K_K \leftarrow K_P} * v$$

Da es sich bei der Matrix um eine perspektivische Transformation handelt, werden Strecken auf Strecken abgebildet. Dadurch ist es ausreichend, die Eckpunkte zu transformieren. Das so entstandene Polygon entspricht den Ausmaßen des Desktops im Kamerabild.

5 Implementation der Emulationssoftware

Im folgenden Kapitel soll aufgezeigt werden, wie die Software implementiert ist. Grundlage für die meisten grafischen Algorithmen bildet die OpenSource-Bibliothek OpenCV [Ope12] in der Version 2.4. Alle Programme wurden mit dem Microsoft Visual Studio entwickelt. Die verwendeten Programmiersprachen sind C++ und C#. Nachdem die gesamte Architektur vorgestellt wurde, werden die einzelnen Teile genauer vorgestellt.

5.1 Gesamtarchitektur

Die Software besteht aus zwei Teilen: einer Bibliothek, die in C++ implementiert ist und die als DLL kompiliert wird, sowie einer grafischen Benutzeroberfläche (GUI), die mit C# / WPF erstellt wurde. Die Bibliothek übernimmt alle algorithmischen Aufgaben, wie zum Beispiel das Abrufen und die Analyse der Kamerabilder. Die GUI stellt dem Nutzer eine Schnittstelle bereit, mit der er das Programm steuern kann. Darüber hinaus sind hier einige Funktionalitäten implementiert, die nicht Teil der Kameraanalyse sind, zum Beispiel das Abfangen der Presenter-Knöpfe.

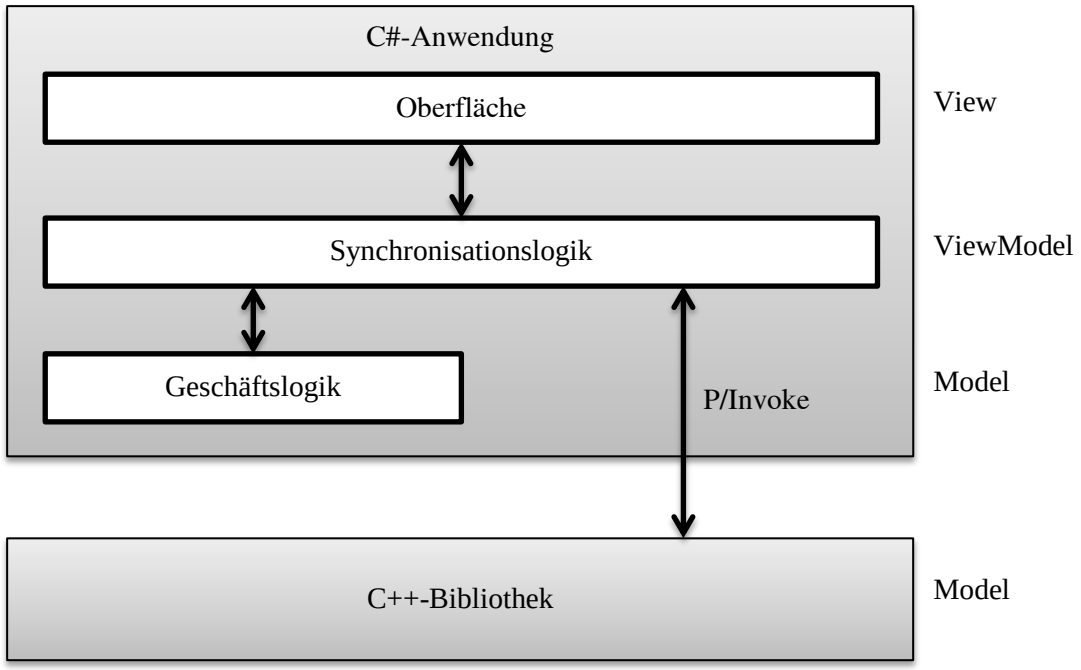

Abbildung 5.1 Architekturdiagramm der gesamten Software

Abbildung 5.1 zeigt die Architektur des Programms und damit das Zusammenspiel der Softwarekomponenten. Das Programm baut auf dem Model-View-ViewModel (MVVM) Entwurfsmuster auf. Die Synchronisationsschicht (ViewModel) bezieht ihre Daten teilweise aus der im selben Programm implementierten Geschäftslogikschicht und teilweise aus der C++-Bibliothek. Bei beiden Schichten handelt es sich um Datenmodelle. Die C++ Bibliothek wird über Platform Invocation Services (P/Invoke) aufgerufen, die die Kommunikation zwischen .Net- und COM-Programmen ermöglichen.

5.2 Aufbau der C++-Bibliothek

Die Aufgabe der C++-Bibliothek ist das Abrufen und die Analyse der Kamerabilder. Das Programm ist in folgende Namensräume unterteilt:

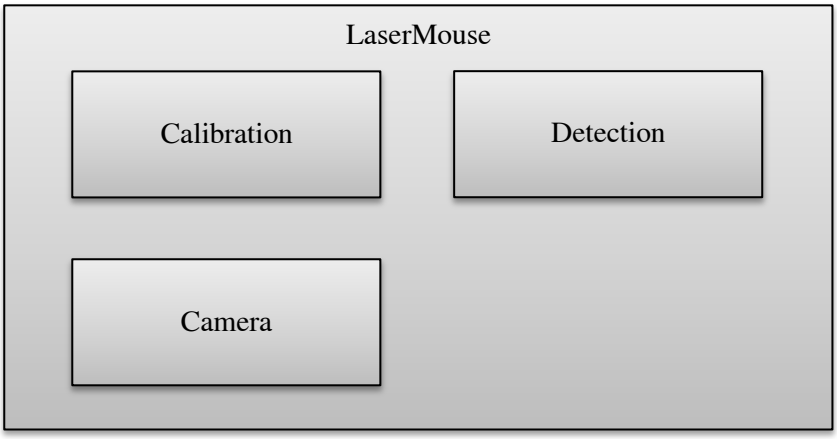

Abbildung 5.2 Namensräume der C++-Bibliothek

Der Lasermouse-Namensraum enthält allgemeine Definitionen und Methoden wie zum Beispiel Einstellungen für den zu verwendenden Schwellwert. Viele dieser Methoden werden in die DLL exportiert und können so von der GUI-Anwendung aufgerufen werden. Diese Definitionen werden durch den Calibration-Namensraum erweitert. Auch dessen Funktionen sind von außen sichtbar und dienen der Kamerakalibrierung. In den Namensräumen Detection und Camera sind Klassen definiert, die nicht in der DLL veröffentlicht werden. Während die Klassen aus „Detection“ der Analyse der Kamerabilder dienen, werden die Inhalte von „Camera“ zum Abrufen und Aufbereiten der Kameradaten verwendet.

5.2.1 Der LaserMouse-Namensraum

Bei den Methoden in diesem Namensraum handelt es sich größtenteils um Setter, Getter, sowie um Hilfsmethoden. Auf eine genaue Betrachtung wird daher verzichtet.

5.2.2 Der Calibration-Namensraum

Dieser Namensraum übernimmt die in Kapitel 4.2 beschriebenen Vorgehensweisen. Er ermittelt die benötigten Transformationsparameter und ist dafür zuständig, diese in Dateien zu persistieren und zu laden.

Zur Kalibrierung der Kamerakonstanten sind zwei Methoden vorhanden. Die calibrateCamera-Methode nimmt zwei Callbacks entgegen. Mit diesen wird der aufrufenden Applikation der aktuelle Kalibrierungsstatus in Form des aktuellen Kamerabilds und der Anzahl der schon bearbeiteten Bilder mitgeteilt. Während der Kalibrierung wird in einer Endlosschleife der Bildstrom auf das

Vorhandensein eines Schachbrettmusters untersucht[1]. Wurde ein solches gefunden, werden dessen Koordinaten zwischengespeichert. Der Nutzer kann den Kalibrierungsprozess jederzeit durch Aufruf der zweiten Methode endCalibrateCamera() beenden. Dadurch wird die Schleife abgebrochen und die eigentliche Kalibrierung findet statt, bei der die Verzerrungsparameter ermittelt werden[2].

Zur Bestimmung der Projektionsmatrix werden die Methoden calibrateProjectionPlaneFront() für die Kamerapositionierung vor der Leinwand und calibrateProjectionPlaneBack() für die Positionierung dahinter genutzt. Die entsprechenden Korrespondenzen werden entweder über die Schachbrettdetektion oder mithilfe des Nutzers ermittelt. In letzterem Fall wird der hellste Punkt als Laserpunkt gedeutet, wenn er einen Schwellwert überschreitet. Ist dieser Punkt für eine bestimmte Zeit an einer ähnlichen Stelle, wird dieser Punkt als Korrespondenz markiert. Sind alle Korrespondenzen ermittelt, wird die Matrix berechnet, die die Positionen auf ihre Bilder abbildet[3].

5.2.3 Der Camera-Namensraum

In diesem Namensraum befindet sich eine abstrakte Klasse, sowie deren konkrete Ausprägungen, die zusammen das Strategy-Entwurfsmuster implementieren:

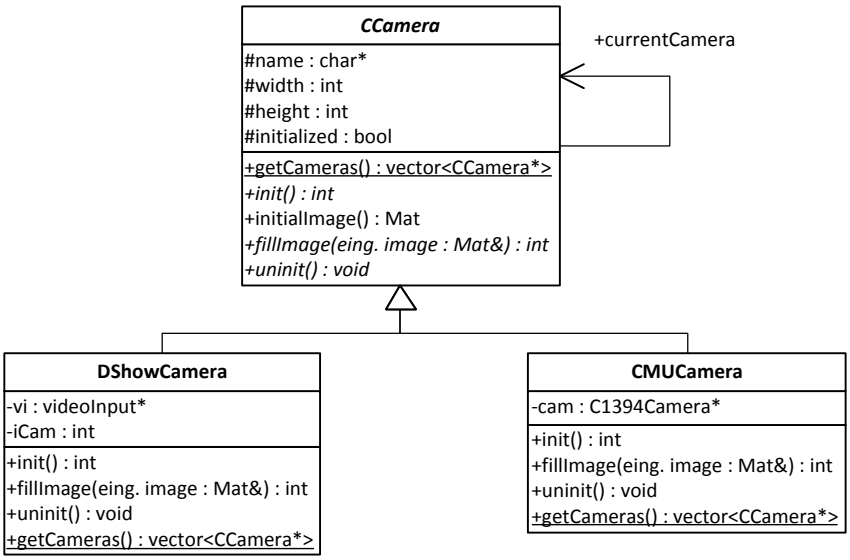

Abbildung 5.3 Klassendiagramm des Camera-Namenraums

Die konkreten Klassen CMUCamera und DShowCamera sind jeweils für IEEE1394-Kameras (Firewire) und für über DirectShow ansprechbare Kameras (USB) zuständig. Der DShowCamera liegt die videoInputLibrary [Vid12] zugrunde, die die Ansteuerung der Kameras übernimmt. Die CMUCamera nutzt den CMU Kameratreiber der Carnegie Mellon University [Bak11]. Die statische Methode getCameras() der konkreten Klassen überprüft die angeschlossenen Geräte und gibt sie

[1] vgl. OpenCV Funktion findChessboardCorners()
[2] vgl. OpenCV Funktion calibrateCamera()
[3] vgl. OpenCV Funktion findHomography()

zurück. Die entsprechende Methode der CCamera-Klasse vereinigt die Ergebnisse der konkreten Klassen zu einem Gesamtergebnis. Die Funktionen init() und uninit() bereiten die Datenübertragung einer Kamera vor beziehungsweise geben verwendete Ressourcen frei. Die Methode fillImage() befüllt eine OpenCV-Matrix mit dem aktuellen Kamerabild.

Die Nutzung des Strategy-Patterns ermöglicht es, sehr einfach neue Kameraarten zu unterstützen. Durch die Abstraktion der Kameras können alle Algorithmen mit einem beliebigen ausgewählten Gerät ausgeführt werden.

5.2.4 Der Detection-Namensraum

Um unabhängig verschiedene Algorithmen entwickeln zu können, wurde auch im Detection-Namensraum auf ein Strategy-Pattern zurückgegriffen. Da die beiden bisher implementierten Verfahren einige Codeabschnitte gemeinsam haben (Kamerabild abrufen, Hintergrundsubtraktion…), wurden sie als TemplateMethod implementiert:

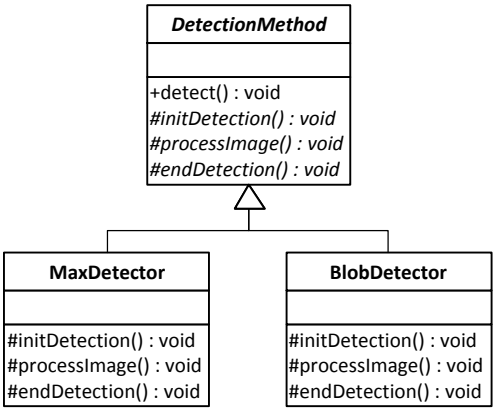

Abbildung 5.4 Klassendiagramm des Detection-Namensraums

Um die Emulation zu starten wird die detect()-Methode einer Instanz aufgerufen werden. Die Emulation läuft dann solange bis eine vorher spezifizierte Taste gedrückt wird. Dabei kann es sich auch um eine Taste auf dem Presenter handeln. Zu Beginn können von den konkreten Klassen benötigte Initialisierungen ausgeführt werden (initDetection()). Gleiches gilt für das Ende (endDetection()). Die processImage()-Methode wird aufgerufen, sobald ein Bild von der Kamera empfangen wurde. Während der Emulation wird die Position des Mauszeigers über den API-Aufruf SetCursorPos() geändert. Die dafür zuständige Methode glättet die Bewegung des Mauszeigers um dem Nutzer eine angenehme Bedienung zu ermöglichen.

5.3 Aufbau der C#-Anwendung

Die GUI-Anwendung besteht aus einem Hauptfenster, sowie einem System-Tray-Symbol. Über das Symbol können Informationen zum Status der Emulation übermittelt werden (mit dessen Farbe und über BalloonTips). Die drei Schichten der Architektur wurden bereits in Abbildung 5.1 vorgestellt. Im folgenden Kapitel werden diese Komponenten genauer erklärt. Um einen ersten Überblick zu

schaffen, wird nachfolgend die Klassenübersicht abgebildet. Hilfsklassen, wie zum Beispiel Commands für das Binding, werden aufgrund Übersichtlichkeit vernachlässigt.

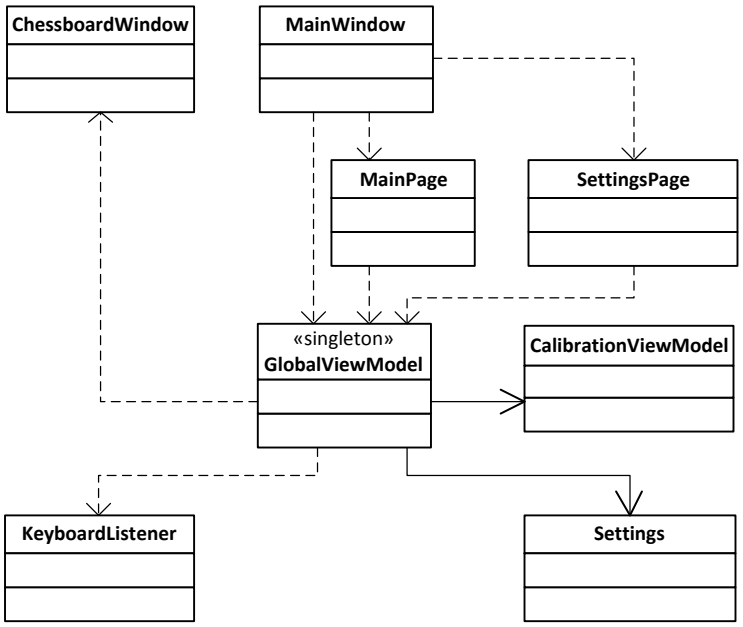

Abbildung 5.5 Klassenübersicht der C#-Anwendung

5.3.1 Model

Zum Datenmodell gehört neben der C++-Bibliothek ein KeyboardListener. Dieser baut auf [Key10] auf und wurde geringfügig angepasst. Der Listener ist dafür zuständig, global auf Tastatur-Eingaben zu reagieren, also nicht nur auf Eingaben, bei denen ein Anwendungsfenster den Fokus hat. Er findet Verwendung beim Abfangen der Presenter-Tasten, die von dessen Treiber als Tastaturanschläge übersetzt werden. Wurde ein Anschlag registriert, werden die entsprechenden Events KeyDown und KeyUp ausgelöst, die in der ViewModel-Schicht verarbeitet werden.

Weiterhin gehört zum Datenmodell eine Einstellungsklasse, die für jeden Benutzer dessen Optionen persistiert und lädt. Dazu gehört zum Beispiel die Tastenbelegung des Presenters. Die Settings-Klasse bildet in der Architektur eine Sonderrolle. Da sie ihre Daten schon View-gerecht zur Verfügung stellt, agiert sie gleichzeitig als ViewModel.

5.3.2 ViewModel

Im MVVM-Entwurfsmuster hat das ViewModel die Aufgabe, die Daten aus dem Model aufzubereiten und in einer passenden Form dem View bereitzustellen. Hauptklasse des ViewModels ist die GlobalViewModel-Klasse, die als Singleton implementiert ist. Die Realisierung mittels Singleton-Entwurfsmuster ermöglicht es, von jeder Stelle des Programms auf das ViewModel zuzugreifen, ohne dass eine Referenz darauf gespeichert werden muss.

Die GlobalViewModel-Klasse tritt auch als Abonnent der Ereignisse des KeyboardListeners auf und setzt die assoziierten Befehle um. Meistens handelt es sich dabei um Klick-Aktionen der Maustasten. Diese werden über den Aufruf der Windows-API (user32.dll) realisiert.

Alle Eigenschaften der ViewModels verfügen über Synchronisationsmechanismen (INotifyPropertyChanged), um die DataBinding-Möglichkeiten von WPF nutzen zu können. Außerdem werden Command-Hilfsklassen verwendet, die für die Ausführung bestimmter Aufgaben zuständig sind. Diese Commands werden direkt in den Views gebunden.

5.3.3 View

Die Views bilden die Schnittstelle, mit der der Nutzer mit dem Programm interagieren kann. Sie treten in Form von Windows-Fenstern auf. Die verwendeten Elemente und Konzepte (wie zum Beispiel Buttons und Menüs) halten sich an die Windows-Richtlinien, sodass der Nutzer das Programm intuitiv durch Anwenden seiner Bediengewohnheiten verwenden kann. Da die Fenster sehr stark auf eine deklarative Beschreibung mittels XAML aufbauen, worin auch die Datenbindung der Felder an die entsprechenden Eigenschaften des ViewModels definiert sind, enthalten sie relativ wenig Code, weshalb hier nur kurz auf die drei Fenster eingegangen wird.

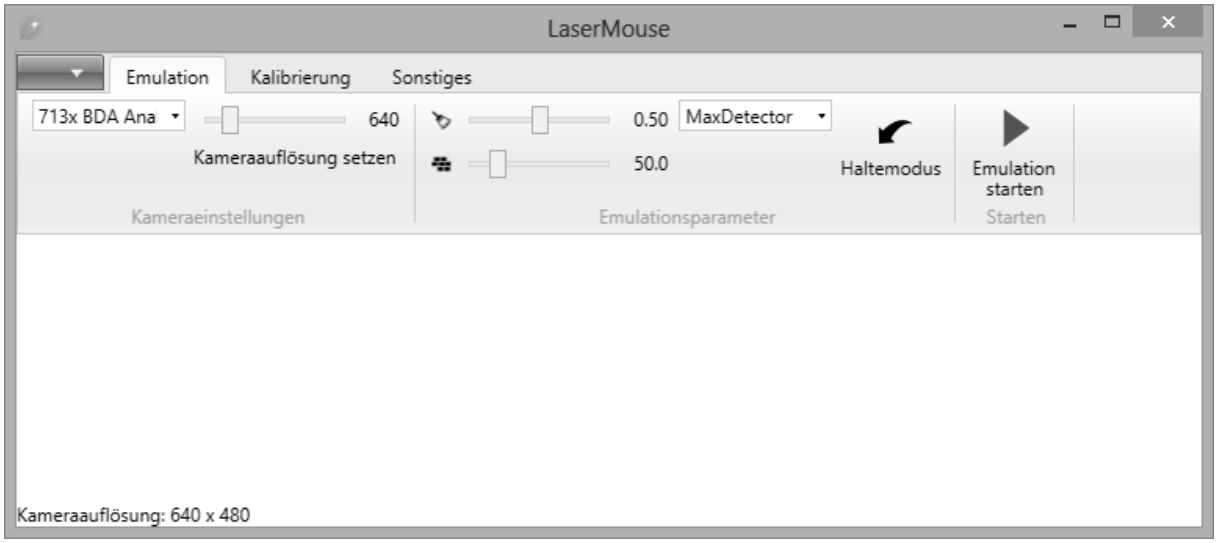

Abbildung 5.6 Screenshot des Hauptfensters

Das MainWindow bildet den Einstiegspunkt des Programms. Am oberen Rand befindet sich ein Menüband (Ribbon), mit dem der Nutzer zwischen verschiedenen Aufgabengebieten wechseln kann. Im Emulationsbereich können die Parameter der Emulation eingestellt werden (verwendete Kamera und Algorithmus, sowie die dargestellten Parameter). Der Kalibrierungsabschnitt erlaubt den Aufruf der verschiedenen Kalibrierungsmethoden während der dritte Tab das Anzeigen des Kamerabilds und Einzeichnen von toten Bereichen erlaubt. Der verbleibende Platz des Fensters wird von einem Frame eingenommen, in dem dynamisch verschiedene Seiten dargestellt werden können. Die Startseite ist die MainPage, in der das Kamerabild dargestellt werden kann und im unteren Bereich ausgewählte Informationen in Statusleisten angezeigt werden.

Über das Menü kann zur SettingsPage gewechselt werden, mit der Programmeinstellungen angepasst werden können.

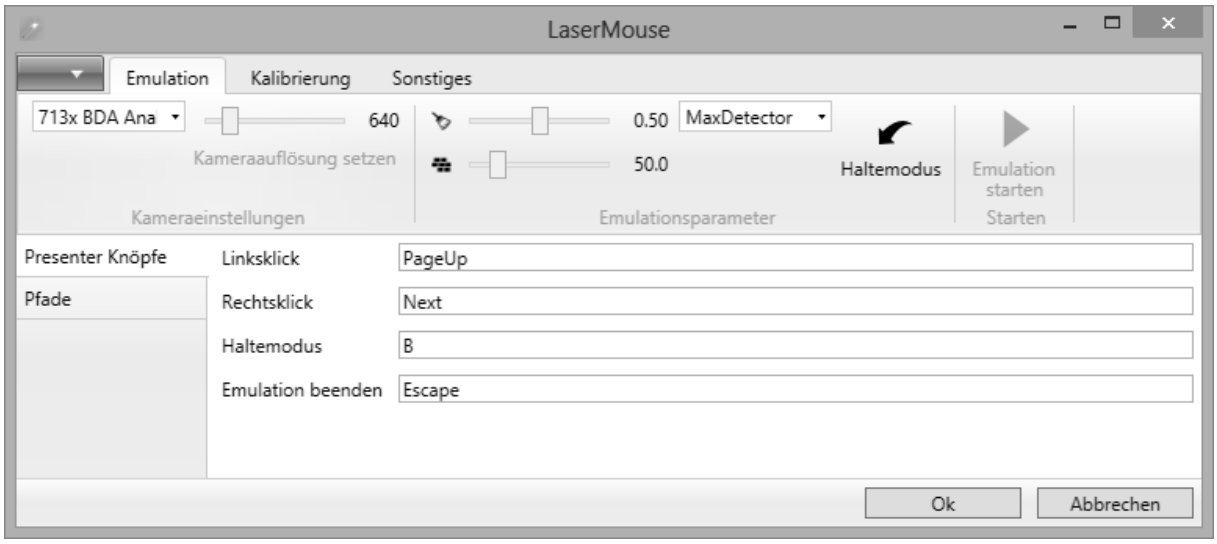

Abbildung 5.7 Screenshot des Hauptfensters mit SettingsPage

Links in der Seite befindet sich eine Auswahl von Karteikarten (Tabs), mit denen der Nutzer das Aufgabengebiet wählen und die zugehörigen Einstellungen anpassen kann. Mit dem Klick auf Ok werden sie persistiert, während bei der Wahl des Abbrechen-Knopfs die geänderten Einstellungen verworfen werden.

Die Kalibrierungsmethoden verwenden teilweise das ChessboardWindow:

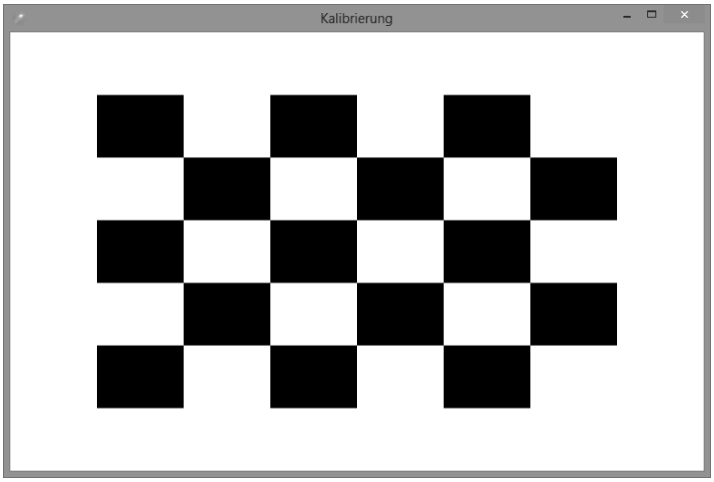

Abbildung 5.8 Screenshot des ChessboardWindows

Dieses Fenster erlaubt die Schachbretterkennung durch die C++-Bibliothek. Sobald das Fenster angezeigt wird, kann es vom Nutzer zurechtgerückt werden, woraufhin die jeweilige Kalibrierungsmethode gestartet wird. Folgende Hotkeys sind implementiert:

- Escape: blendet das Fenster aus
- 1 – 4: ändert die Hintergrundfarbe des Schachbretts (führt gegebenenfalls zur besseren Erkennung)
- F: Vollbildmodus umschalten
- M: Messen, Kalibrierung starten

Für die Kalibrierung von hinten muss der Nutzer bestimmte Ecken innerhalb des Schachbretts markieren. Dabei handelt es sich jeweils um die äußersten inneren Ecken in der Reihenfolge oben links, oben rechts, unten rechts, unten links. Der erste Punkt ist somit die rechte untere Ecke des ersten schwarzen Blocks in der ersten Zeile.

Im Benachrichtigungsbereich (System Tray) des Betriebssystems wird ein Symbol registriert, womit dem Nutzer weitere Informationen zur Verfügung gestellt werden.

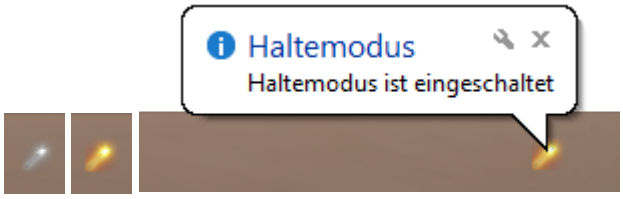

Abbildung 5.9 Varianten des Symbols im Benachrichtigungsbereich; Links Ausgangszustand; Mitte: Emulation läuft; Rechts: BalloonTip

6 Evaluation

Im folgenden Kapitel wird das erstellte System einer Evaluation unterzogen. Dabei werden verschiedene Einstellungen und Verfahren verglichen. Abschließend folgt ein Fazit, in dem Erweiterungsmöglichkeiten der Software aufgezeigt werden. Die erhobenen Messdaten befinden sich auf der beigelegten CD. Das Testsystem ist folgendermaßen charakterisiert:

- Dual Core AMD Opteron 1218
- 3,00 GB verwendbarer Arbeitsspeicher
- Windows 7 32 bit
- Microsoft LifeCam Cinema

6.1 Evaluation der Rechengeschwindigkeit

Die Verarbeitungsgeschwindigkeit ist eine wesentliche Größe zur Bewertung der Benutzbarkeit des Systems. Da die Berechnungsschritte während der Emulation in einer unterbrechungslosen Schleife ausgeführt werden, wirkt sich die Bearbeitungszeit direkt auf die Framerate aus. Außerdem hat sie Einfluss auf die Latenz des Systems. Allerdings wird diese auch durch die Latenz der Kamera beeinflusst. Die Mausbewegung wird erfahrungsgemäß ab Frameraten von 10 bis 15 fps als flüssig wahrgenommen. Geringere Raten führen zu einem Ruckeln, was eine Bedienung weiterhin ermöglicht, jedoch die Akzeptanz durch den Nutzer senkt.

Im Rahmen dieser Arbeit wurden zwei Detektionsalgorithmen implementiert. Abbildung 6.1 vergleicht die Berechnungszeit der beiden Verfahren bei unterschiedlichen Kameraauflösungen. Dabei wird nur der spezifische Detektionsschritt betrachtet. Vorverarbeitungen, die für beide Algorithmen gleich sind, entfallen.

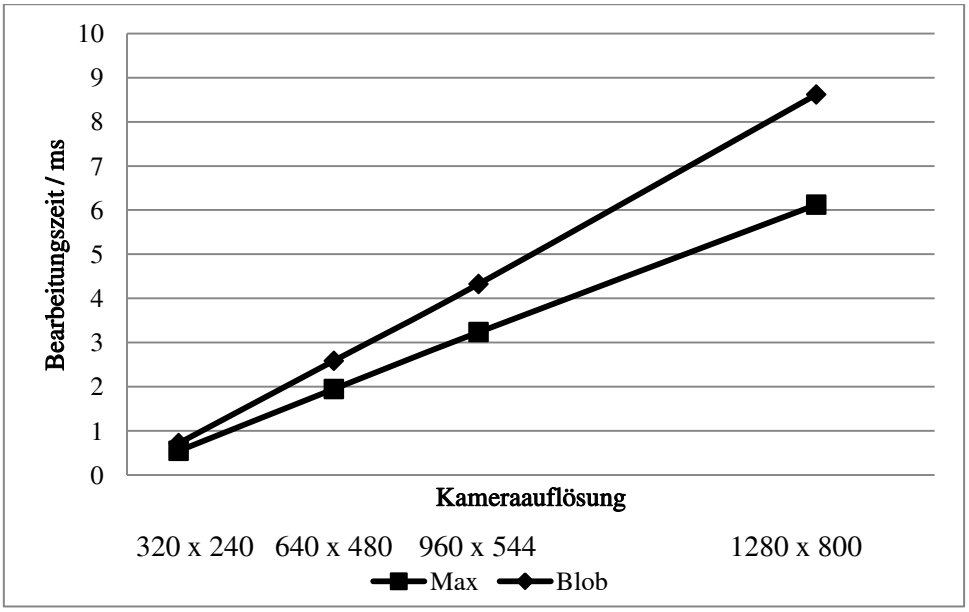

Abbildung 6.1 Vergleich der beiden implementierten Algorithmen; Zusammenhang zwischen Kameraauflösung und Bearbeitungszeit des spezifischen Detektionsschritts

Aus den ermittelten Daten wird deutlich, dass der Maximumdetektor bei allen Auflösungen um den Faktor 1,3 bis 1,4 schneller ist als der Blobdetektor. Dies hat jedoch kaum einen Einfluss auf die Gesamtbearbeitungszeit. Abbildung 6.2 stellt dar, wie sich die Gesamtbearbeitungszeit eines Frames zusammensetzt.

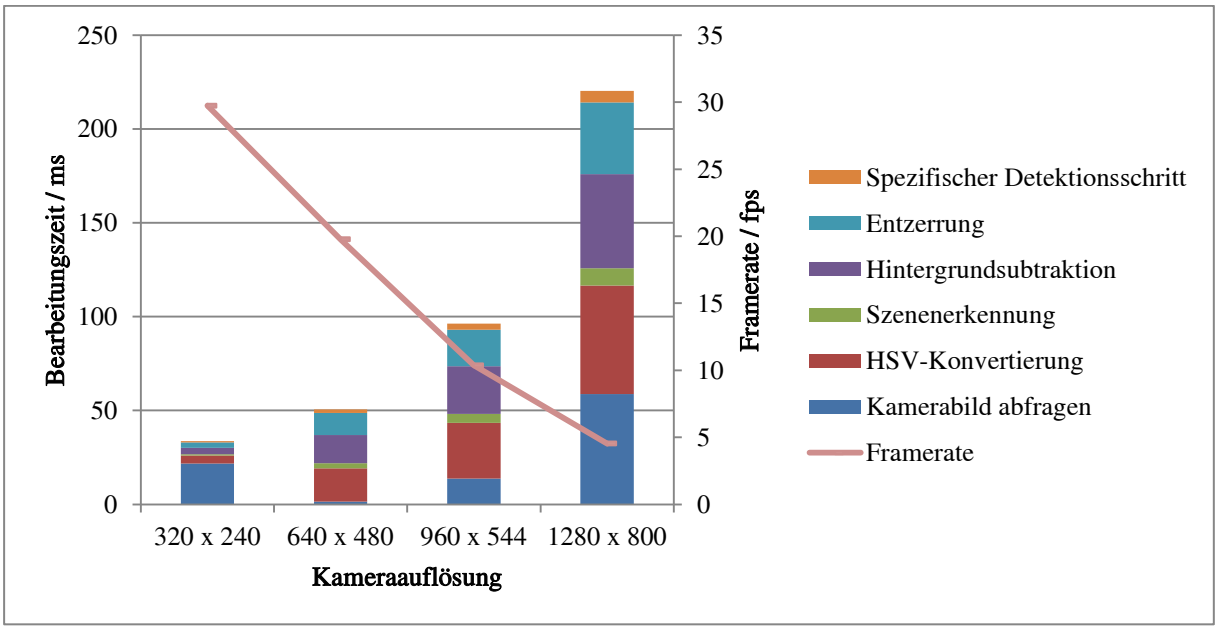

Abbildung 6.2 Zusammensetzung der Bearbeitungszeit des Maximumdetektors bei verschiedenen Kameraauflösungen

Das Diagramm zeigt, dass der spezifische Bearbeitungsschritt, in dem sich Maximumdetektor und Blobdetektor unterscheiden, kaum Einfluss auf die Gesamtrechenzeit hat. Deswegen erreichen beide Verfahren ähnliche Frameraten. Stellen, an denen die Framerate erhöht werden kann, sind die drei größten Bereiche (HSV-Konvertierung, Hintergrundsubtraktion und Entzerrung). Unter der Annahme, dass das Kamerabild bereits entzerrt vorliegt, kann der Entzerrungsschritt weggelassen werden. Aufgrund der Allgemeingültigkeit des Systems wurde dies aber nicht umgesetzt.

6.2 Evaluation der Latenz

Die Gesamtlatenz ist ein Maß für die wahrnehmbare Reaktivität des Systems. Zu ihr zählen hauptsächlich die Latenz der Kamera und die Bearbeitungszeit. Zur Messung der Latenz wurde die Leinwand mit einer Kamera beobachtet und die Zeit zwischen dem Auftauchen des Laserpunkts und der Verschiebung der Maus gemessen. Die entstandenen Videos sind mit vierfacher Zeitlupe aufgenommen, was zu einer Framerate von 100 fps führt.

Abbildung 6.3 zeigt das Ergebnis der Messung. Zum Vergleich wurden die Bearbeitungszeiten hinzugefügt.

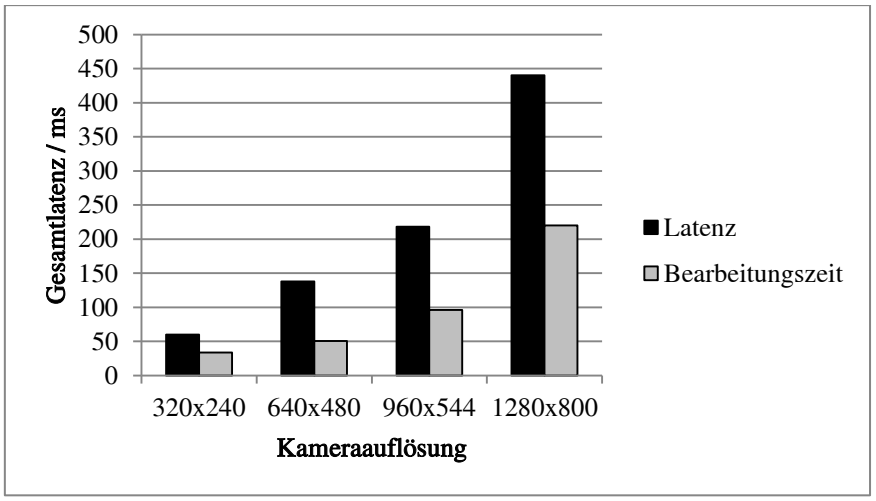

Abbildung 6.3 Zusammenhang zwischen Kameraauflösung und Gesamtlatenz des Systems mit dem Blobdetektor

Diese Latenzen sind erfahrungsgemäß bis zu einer Auflösung von 960x544 kaum wahrnehmbar und haben keinen negativen Einfluss auf die Bedienbarkeit. Bei größeren Auflösungen entsteht ein deutlicher Versatz zwischen den Aktionen des Nutzers und der Mausbewegung. Dadurch ist die Verwendung dieser Auflösungen weniger geeignet.

6.3 Evaluation der Genauigkeit

Zur Evaluation der Genauigkeit der Software wurde das System mit der Kamerapositionierung hinter der Leinwand kalibriert. Während der Messung hatte der Desktop eine Auflösung von 1400 x 1050 px. Der Mauscursor hat dabei eine Breite von 10 px, was zu einer Größe von etwa 1,8 cm auf der Leinwand führt. Die Evaluation wurde durchgeführt, indem Nahaufnahmen des Laserpunkts und des Mauszeigers an verschiedenen Stellen auf dem Desktop erstellt wurden. In den Bildern wurden die Breite des Mauszeigers, sowie der Abstand des Laserpointers zu dessen Spitze gemessen. Aus dem Verhältnis beider Werte wurde die Ungenauigkeit auf dem Desktop errechnet.

Die Messdaten befinden sich auf der beigelegten CD. Da diese stark schwanken, kann über die Genauigkeit des Systems keine exakte Aussage getroffen werden. Im Allgemeinen lässt sich beobachten, dass die mittlere Ungenauigkeit zwischen zwei und vier Pixeln liegt und mit steigender Kameraauflösung sinkt. Abbildung 6.4 zeigt diesen Zusammenhang.

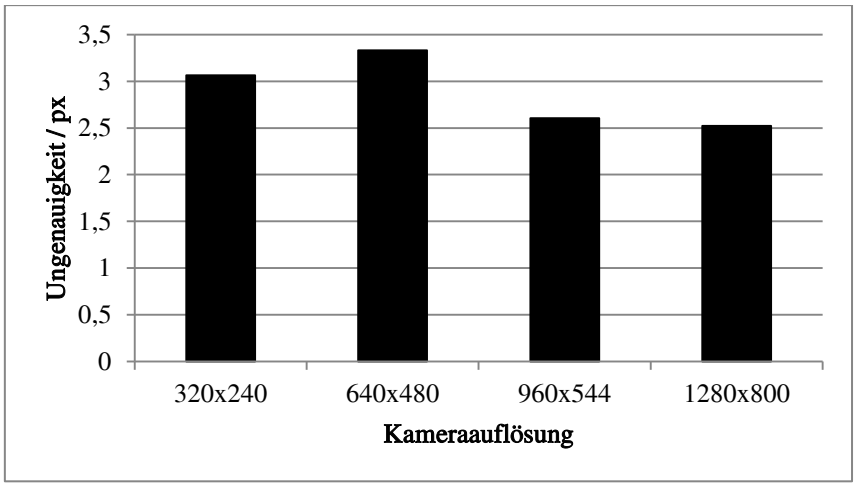

Abbildung 6.4 Zusammenhang zwischen Kameraauflösung und Ungenaugkeit der Erkennung mit dem Maximumdetektor

Das Maximum bei der Kameraauflösung von 640 x 480 Pixeln ist auf den hohen Einfluss des Zufalls bei dem verwendeten Messverfahren zurückzuführen. Da Messungen an diskreten zufälligen Stellen durchgeführt wurden, ist nicht garantiert, dass diese über das Kamerabild gleichverteilt sind. Offensichtlich reicht die Anzahl von zehn Messungen nicht aus, um die Gleichverteilung anzunähern.

Zusätzlich zur Genauigkeit sinkt mit abnehmender Kameraauflösung die Erkennungszuverlässigkeit, was überwiegend bei einer Auflösung von 320x240 px deutlich wird. Dies hat den Grund, dass die Fläche des Laserpunkts auf dem Kamerabild bei geringeren Auflösungen kleiner ist. Ist sie zu klein, nimmt er weniger als einen Pixel ein, was die Erkennung schwieriger macht. Erkennungszuverlässigkeit bezeichnet die Wahrscheinlichkeit, dass ein Laserpunkt erkannt wird. Sie wird in Abbildung 6.4 nicht dargestellt, da sie sich schwer messen lässt.

Die Evaluationsergebnisse zeigen, dass bei einer Kameraauflösung von 640x480 px die günstigste Kombination aus Latenz, Genauigkeit und Zuverlässigkeit erreicht wird.

6.4 Fazit und Ausblick

Das entwickelte System erlaubt es dem Nutzer, die meisten Aktionen, die mit einer Maus möglich sind, mit einem Presenter nachzuvollziehen. So können Präsentationen von beliebigen Positionen ohne die Notwendigkeit einer festen Unterlage gesteuert werden. Die implementierten Verfahren sind weitestgehend unempfindlich gegenüber Störeinflüssen und entsprechend performant und genau.

Durch die strukturierte Architektur der Software und der Verwendung von bewährten Entwurfsmustern der Softwaretechnologie ist es möglich, weitere Algorithmen und andere Funktionalitäten hinzuzufügen, um die Störanfälligkeit noch weiter zu verbessern oder die Leistung zu erhöhen. Um dem Nutzer weitere Möglichkeiten zu bieten, kann beispielsweise eine Gestenerkennung implementiert werden.

Da die Software den Mauszeiger des Betriebssystems setzt, kann jedes andere Programm, das diesen nutzt, mit der Emulation gesteuert werden. Für andere Programme, die direkt auf den Maustreiber zugreifen (zum Beispiel über DirectInput), kann es sinnvoll sein, den Emulationscode in einen Softwaretreiber umzuwandeln. Dadurch ist es auch möglich, Multitouch-Nachrichten an das

Betriebssystem zu senden, die mit den geeigneten Programmen ausgewertet werden können. Dadurch kann es erforderlich werden, eine Verfolgung der Punkte zu implementieren.

Das entwickelte System erfüllt somit die Anforderungen an ein produktiv eingesetztes Produkt. Gleichzeitig bietet es eine Vielzahl von Möglichkeiten zur Weiterentwicklung.

Literaturverzeichnis

[PKG03] Popovich, Evgeny, Karni, Zachi und Gotsman, Craig. PresenterMouse. [Online] 2003. http://www.cs.technion.ac.il/~zachik/presentermouse/alg.htm.

[Col08] Collier, Robert. Control your computer using a laser or IR pen. [Online] 2008. http://www.instructables.com/id/Control-your-computer-using-a-laser-or-IR-pen/?ALLSTEPS.

[WL08] Watson, Theodore und Lieberman, Zachary. Laser Tag 2.0 - How To - Download and Source Code. [Online] 2008. http://www.muonics.net/blog/index.php?postid=26.

[Zha00] *A Flexible New Technique for Camera Calibration.* Zhang, Zhenyou. November 2000, IEEE Transactions on pattern analysis and machine intelligence, Vol. 22.

[Bou10] Bouguet, Jean-Yves. Camera Calibration Toolbox for Matlab. [Online] 9. Juli 2010. http://www.vision.caltech.edu/bouguetj/calib_doc/.

[FLM92] Faugeras, O., Luong, Q. und Maybank, S. Camera self-calibration: Theory and experiments. *Computer Vision - ECCV '92; Lecture Notes in Computer Science.* 1992.

[MF92] Maybank, S. und Faugeras, O. A theory of self-calibration of a moving camera. *International Journal of Computer Vision.* 1992.

[YJS06] Yilmaz, Alper, Javed, Omar und Shah, Mubarak. Object tracking: A survey. *ACM Computing Surveys; Vol. 38, Is. 4.* 2006.

[VRB01] Veenman, Cor J., Reinders, Marcel J. T. und Backer, Eric. Resolving Motion Correspondence. *IEEE Transactions on pattern analysis and machine intelligence; Vol. 23; No. 1.* 2001.

[Yil04] Yilmaz, A. Contour-based object tracking with occlusion handling in video acquired using mobile cameras . *IEEE Transactions on Pattern Analysis and Macine Intelligence; Vol. 26, Is. 11.* 2004.

[CRM03] Comaniciu, D., Ramesh, V. und Meer, P. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence; Vol. 25; Is. 5.* 2003.

[TH98] Trajkovic, Miroslav und Hedley, Mark. Fast corner detection. *Image and Vision Computing; Vol 16, Is. 2.* 1998.

[IDB97] Intille, Stephen S., Davis, James W. und Bobick, Aaron F. Real-Time Closed-World Tracking. *Computer Vision and Pattern Recognition; IEEE Computer Society Conference.* 1997.

[BK08] Bradski, Gary und Kaehler, Adrian. Learning OpenCV. s.l. : O'Reilly Media, 2008.

[Ste97] Stein, G. P. Lens Distortion Calibration Using Point Correspondences. *Computer Vision and Pattern Recognition; IEEE Computer Society Conference.* 1997.

[SA85] *Topological Structural Analysis of Digitized Binary Images by Border Following.* Suzuki, Satoshi und Abe, Kenchi. 1985, Computer Vision, Graphics, and Image Processing 30, S. 32-46.

[Sal78] *An Efficient Point-In-Polygon Algorithm.* Salomon, Kenneth B. 1978, Computers & Geosciences, Vol. 4, S. 173-178.

[Bou88] Bourke, Paul. Calculating the area and centroid of a polygon. [Online] Juli 1988. [Zitat vom: 30. Juli 2012.] http://paulbourke.net/geometry/polyarea/.

[SCB87] *An experimental comparison of RGB, YIQ, LAB, HSV, and opponent color models.* Schwarz, Michael W., Cowan, William B. und Beatty, John C. April 1987, ACM Transactions on Graphics Vol. 6, S. 123-158.

[Ope12] OpenCV. [Online] http://opencv.willowgarage.com/wiki/.

[Vid12] VideoInput. [Online] https://github.com/ofTheo/videoInput.

[Bak11] Baker, Christopher R. CMU 1394 Digital Camera Driver. [Online] 26. September 2011. http://www.cs.cmu.edu/~iwan/1394/.

[Key10] github: C# Keyboard Listener. [Online] 11. Juli 2010. https://gist.github.com/471698.

Abbildungsverzeichnis