Sheffield Hallam University

SHEFFIELD HALLAM UNIVERSITY

SCHOOL OF ENGINEERING

Design and Implementation of Human Interface with Computer System Using a Camera Projector

BY

UshaKiran Soutapalli

MSc in COMPUTER AND NETWORK ENGINEERING

FULL TIME

2006-2007

Sheffield Hallam University

PREFACE

This report describes project work carried out in the school of engineering at Sheffield Hallam University between June 2006 and September 2006.

The submission of the report is in accordance with the requirements for the award of the degree of **MSc in computer and Network Engineering**, under the auspices of the University.

ACKNOWLEDGMENT

I consider it a privilege to present this thesis as a student of MSc in Computer and Network Engineering of the School Of Engineering, Sheffield Hallam University, Sheffield, UK. I express my heartfelt gratitude to the school for giving me this opportunity.

This work is based on all that I have learnt from my current and previous teachers, advisors and mentors. My contribution is only an indication to my learning ability. I wish to express my special thanks to Mr. Jan Wedekind my supervisor, mentor and guide in the true sense who made this project realisable for me. I also wish to express my heart felt thanks to my family and friends for their tremendous support.

Table Of Contents Page No.

Chapter 1: Introduction…………………………………………………1

1.1 Introduction……………………………………………………………………1

1.2 Background……………………………………………………………………2

1.3 Motivation…………………………………………………………………..2

1.4 Applications……………………………………………………………………3

1.5 Thesis Description………………………………………………………...4

  1.5.1 System Requirements………………………………………………4

  1.5.2 Thesis Time/Schedule…………………….……….…………………5

1.6 Potential Hazards……………………..……………………………………...6

1.7 Report Guideline……………………..………………………………………6

1.8 Summary…………………………..………………………………………6

Chapter 2: Relevant Theory and Analysis…………………………7

2.1 Introduction………………………………………………………………7

2.2 Computer and Machine Vision……………………………………………..7

2.3 Image Processing………………………………………………………...9

2.3.1 Digital Image Processing…………………………………………………10

2.3.2 Examples of Fields that Use Digital Image Processing……………………14

2.3.3 Features of Digital Image Processing……………………………………..15
2.3.4 Components of an Image Processing System…………………………..18
2.4 Mimas……………………..……………………………………………20
2.4.1 Algorithms and methods in Mimas………………………………………21
2.5 Summary………………………………………………………….....22

Chapter 3: The Open Source and Qt4 Environment…………….23

3.1 Introduction …..23
3.2 What is Open Source……………………………………………...23
3.3 Qt Cross Platform C++ Development……….………………………23
3.3.1 Qt is Open Source ………………………………………………..24
3.3.2 The Qt Class Libraries……………………………………….…......25
3.3.3 Signals and Slots………………………………………………….25
3.3.4 GUI Design with Qt Designer ……………………………………25
3.3.5 Qt/X11 - The Standard for Desktop Linux Development………………...26
3.4 Summary ……………………………………………………………27

Chapter 4: Camera Calibration……………………………….28

4.1 Introduction……………………………………………………….28

4.2 What is Calibration………………………………………………….28

4.3 Homography… ………………………………………………………29

4.4 Distortion free Camera Calibration…………………………………30

4.5 Summary……………………………………………………………34

Chapter 5: Edge Detection, Pattern Search, Object Recognition and Segmentation…………………………………………………….35

5.1 Introduction………………………………………………………...35

5.2 Edge Detection……………………………………………………..35

5.3 Pattern Search and Recognition……………………………………36

5.4 Object Recognition…………………………………………………37

5.4.1 Two-dimensional Object Recognition…………………………...40

5.4.2 Three-dimensional Object Recognition…………………………41

5.4.3 Using Cross Correlation to Recognize an Object……………….42

5.4.4 Object Tracking Criteria…………………………………………44

5.5 Segmentation………………………………………………………45

5.5.1 Graph-Based Segmentation..46

5.6 Summary…………………………………………………………...47

Table Of Contents Page No.

Chapter 6: Design, Implementation and Results.........................48

6.1 Introduction...48

6.2 Design and Implementation ……………………………………………..48

6.3 UML Diagrams……………………………………………………………..50

6.4 Design of User Interface…………………………………………………52

6.4.1 StackedWidget named stackexample……………………………………53

6.5 Searching and Loading a Projected Pattern……………………………..54

6.5.1 GU Interface for Searching and Loading a Projected Pattern…………..56

6.5.2 Results of the Search and Load Projected Pattern………………………57

6.6 Design and Implementation of Calibration……………………………..59

6.7 Implementation Results…………………………………………………62

6.8 Summary………………………………………………………………...64

Chapter 7: Conclusion and Further Work..................................65

7.1 Discussion………………………………………………………………..65

7.2 Further Work……………………………………………………………..66

7.3 Conclusion………………………………………………………………..66

Reference………………………………………………………………67

Appendix: Source Code Used in this Thesis………………………70

List of Figures and Snapshots

[1] Image Depicting Homography…………………………………………..29

[2] UML State Diagram……………………………………………………..50

[3] UML Collaboration Diagram……………………………………………51

[4] Snap shots of Pattern Search and Recognition……………………………..56

[5] Snap shots of Calibration……………………………………………….....61

[6] Snap shots of Design …………………………………………………….53

[7] Snap shots of Implementation…………………………………………….58

[8] Snap shots of Results …………………………………………………….63

ABSTRACT

Human interface with Computer using a camera projector is a project developed using Qt4 in Linux; it involves topics like computer vision, image processing and Mimas which are currently in extensive research and are subject to dynamic changes day by day.

Trying to communicate with a computer system using human interaction and eliminating traditional keyboard and mouse from the scene are subjected to research from long time, but implementing the same in Qt4 platform makes it more flexible and at the same time an Open Source Technology.

Creating an interface for Human Interaction is mainly based on the noise free calibration, pattern search and capturing the reference image.

The Calibrate function calibrates using the homography between the projected pattern and the camera image.

Chapter 1

Introduction

1.1 Introduction:

The aim of the thesis is to design, develop and implement the software for human interface with a computer system using a camera projector.

The following issues will be addressed.

- Study and investigate various Open Source Technologies
- Study the Qt environment
- Study and make effective use of Mimas tool kit
- Design an interactive User interface in Qt.
- Understand Calibration, Homography, Object Recognition techniques.
- Implement these techniques one after the other sequentially in the Design.

1.2. Background:

Interactive Camera Projector System which is the basis for Human interface with Computer System Using a Camera Projector is one among the Fun Projects developed in Microsystems and Machine Vision Laboratory (4311) by Juan Roldan an Erasmus student from Spain. This software makes use of a low cast webcam that is calibrated against a standard projector screen. The webcam used determines the position of the physical pointer such as a pencil which is then used to virtually move the X11 pointer. Point-and-click has also been implemented.

1.3 Motivation:

With much support of my supervisor I have learned that OpenGL offers even best methods to develop live video within Qt4 than the one used in Interactive Camera Projector System such as OpenGL pixel-transfer and Simulating X11 mouse events.

1.4. Applications:

- Oral Presentations for lectures

  Oral Presentations would be flexible enough for long lectures as the presenter can jus point the reference image on the screen and there by move on to slide after slide rather than clicking mouse or using keyboard every time when there is a need to change in the slide.

- Interactive interface with the computer since it be integrated in the graphical environment like Windows, Linux etc operating systems.

  When working with Graphical Environments like Windows, Linux etc., an interactive interface like this would enable the user to point to the objects and images by a pen or pencil rather than giving a command or by clicking mouse buttons.

- Convert a TFT screen in to a touching panel only with a camera.

A TFT Screen can be flexibly converted in to a touch panel just by using a low-cost camera.

1.5 Thesis Description:

The aim of this thesis is to design, develop and implement a user interface for "Human Interface with Computer System using a Camera Projector"; an interactive system is developed to use a pencil as a reference object to point the object on a TFT screen. Prior to such implementation a detailed study on computer vision, machine vision and Image processing has been conducted.

1.5.1 System Requirements:

The most basic requirements to achieve the described design are detailed as follows:

- Hardware Requirements
- Software Requirements

Hardware Used:

- A desktop computer with basic configuration
- Low cost Web camera
- Camera Projector

Software Used:

- Linux Operating System (Suse)
- Open Source Technology

- Qt4 environment
- Mimas toolkit

1.5.2 Time/Schedule

Table 1.1 shows the definitions of the tasks and timescale of the thesis.

Task Time Period	
Literature study and investigation	June 2006
Familiarization of the Qt4 environment	June 2006
Learn and experiment the Gui C++ language	July 2006
Design UML Diagrams	August 2006
Design the User Interface of the System	September 1st 2006 to September 15th 2006
Implementation and Report writing	September 15th 2006 to September 29th 2006

Table 1.1: Tasks to be completed for the thesis and their schedule

1.6 Potential Hazards

The precautions that were strictly followed during the entire thesis period were

- An erect sitting posture was maintained while working on the computer
- Breaks were taken at regular intervals to avoid cramps and sprains that can be caused due to sitting in front of the computer for long hours.

1.7 Report Guideline

Chapter 2 contains the theory behind Machine and Computer Vision and Image Processing which is very essential to this thesis. This chapter discusses most of the theory used in the project. Chapter 3 is on the Qt 4 environment. An overview of the GUI C++ language has also been presented here. Chapter 4 talks about Calibration, Homography, Pattern Search and Object Recognition which form a major part of this thesis. Chapter 5 shows the design and implementation done in the project. It also shows results obtained in the project. Finally Chapter 6 concludes the thesis work and has some very interesting future work that can be done.

1.8 Summary:

This chapter contains the background and motivation behind this project. It also briefly summarizes the Computer vision and Image Processing issues.

Chapter 2

Relevant Theory and Analysis

2.1 Introduction:

This chapter details the theoretical research and analysis of Machine vision, Computer vision, Image processing with fundamental issues and few examples.

2.2 Machine Vision and Computer Vision:

Machine vision is the process whereby a machine, usually a digital computer, automatically processes an image and identifies the “contents of the image”. That is it recognizes what the image consists of, often the content may be a machined part and the objective is not only to locate the part but to inspect it as well.

The terms “computer vision” and “image understanding” are often concluded to be machine vision.

Computer vision has been the most prominent area research for more than three decades. This field reveals the facts that image processing and pattern recognition have been successful at large in terms of delivering operational systems. Day to day barcode scanners used in supermarkets, and pattern recognition techniques used in identification, bill recognition, and address recognition signifies the changes evolved in this field.

Increase and dynamism in computer power and the widespread use of multimedia technology has led to such progress in this field. The use of imaging technology for day to day tasks and the vast use of images as part of the Web has resulted in a rigorous reduction in price and delivery of images to virtually every desktop. With respect to large scale computer vision applications that involve motion estimation, depth recovery, and scene interpretation, considerable progress has been achieved.

2.3 Image Processing:

In general “image processing” refers to processing information for which the data input is in the form of images or frames of video and the resultant output is also in the form of images or frames of video. Most image processing techniques involve manipulation of an image to improve or change the quality of the image using algorithms to modify the data representing an image or video to improve and enhance them.

A current trend in the image processing involves modifying the content of the image to give it a totally new and better look.

An image may be defined as a two-dimensional function, f(x, y), where x and y are plane coordinates, and the amplitude of f at any pair of coordinates (x, y) is known as the *intensity* or *gray level* of the image at that point. When x, y, and the amplitude values of f are all finite, discrete quantities, such as image is called a *digital image*.

2.3.1 Digital Image Processing:

The term '*digital image processing'* refers to processing digital images using a digital computer. As mentioned above a digital image is composed of a finite number of elements, each of these elements are assigned a particular location and value. These elements are called as *picture elements*, *image elements*, and *pixels*. *Pixel* is the term most widely used to represent the elements of a digital image.

Vision is the most advanced of our senses, and this is the reason why images play the vital task in human perception. Imaging machines cover the entire Electro Magnetic spectrum, right from gamma to radio waves. They can handle complex images generated by sources that humans cannot associate to. These include ultrasound, electron microscopy, and computer-generated images.

Thus, digital image processing encompasses a wide and varied field of applications.

Sometimes a distinction is made by defining image processing as a discipline in which both the input and output of a process are images. Sometimes this is considered as we believe this to be a limiting and somewhat artificial boundary. For example, under this definition, even the trivial task of computing the average intensity of an image (which yields a single number) would not be considered an image processing operation. On the other hand, there are fields such as computer vision whose ultimate goal is to use computers to emulate human vision, including learning and being able to make inferences and take actions based on visual inputs. This area itself is a branch of artificial intelligence (AI) whose objective is to emulate human intelligence. The area of image analysis (also called image understanding) is in between image processing and computer vision. There are no clear-cut boundaries in the continuum from image processing at one end to computer vision at the other. It is useful to consider the paradigm that there are three types of computerized processes in this continuum: low-, mid-, and high-level processes. Low-level processes involve primitive operations such as image preprocessing to reduce noise, contrast enhancement, and image sharpening. [1],[2] A low-level process is characterized by the fact that both its inputs and outputs are images. Mid-level processing on images involves tasks such as segmentation (partitioning an image into regions or objects), description of those objects to reduce them to a form suitable for computer processing, and classification (recognition) of individual objects. A mid-level process is characterized by the fact that its inputs generally are images, but its outputs are attributes extracted from those images (e.g., edges, contours, and the identity of individual objects). Finally, higher-level processing involves “making sense” of an ensemble of recognized objects, as in image analysis, and, at the far end of the continuum, performing the cognitive functions normally associated with vision. Based on the preceding inspection, a logical place of overlap between image processing and image analysis is the area of recognition of individual regions or objects in an image. Thus, we can conclude that *digital image processing* encompasses processes whose inputs and outputs are images and, in addition, encompasses processes that extract attributes from images, up to and including the recognition of individual objects.

Successful applications of image processing concepts can be found in astronomy, biology, nuclear medicine, law enforcement, defense, and industrial applications. The second major areas of application of digital image processing techniques are being used in solving problems dealing with machine perception. In this case, interest focuses on procedures for extracting from image information in a form suitable for computer processing. Often, this information bears little resemblance to visual features that humans use in interpreting the content of an image. Examples of the type of information used in machine perception are statistical moments, Fourier transform coefficients, and multidimensional distance measures. Typical problems in machine perception that routinely utilize image processing techniques are automatic character recognition, industrial machine vision for product assembly and inspection, military recognizance, automatic processing of fingerprints, screening of X-rays and blood samples, and machine processing of aerial and satellite imagery for weather prediction and environmental assessment. The continuing decline in the ratio of computer price to performance and the expansion of networking and communication bandwidth via the World Wide Web and the Internet have created unprecedented opportunities for continued growth of digital image processing. [4],[5]

2.3.2 Examples of Fields that Use Digital Image Processing:

Today, there is almost no area of technical endeavor that is not impacted in some way by digital image processing. The areas of application of digital image processing are so varied that some form of organization is desirable in attempting to capture the breadth of this field. One of the simplest ways to develop a basic understanding of the extent of image processing applications is to categorize images according to their source (e.g., visual, X-ray, and so on).The principal energy source for images in use today is the electromagnetic energy spectrum. Other important sources of energy include acoustic, ultrasonic, and electronic (in the form of electron beams used in electron microscopy). Synthetic images, used for modeling and visualization, are generated by computer.

Images based on radiation from the EM spectrum are the most familiar, especially images in the X-ray and visual bands of the spectrum. Electromagnetic waves can be conceptualized as propagating sinusoidal waves of varying wavelengths, or they can be thought of as a stream of massless particles, each traveling in a wavelike pattern and moving at the speed of light. Each massless particle contains a certain amount (or bundle) of energy. Each bundle of energy is called a *photon*.

2.3.3 Features of Digital Image Processing:

Image acquisition is the first step in developing of digital images. Generally, the image acquisition stage involves preprocessing, such as scaling.

Image enhancement is the simplest and most appealing areas of digital image processing. Basically, the idea behind enhancement techniques is to bring out detail that is obscured, or simply to highlight certain features of interest in an image. A familiar example of enhancement is when we increase the contrast of an image because “it looks better.” It is important to keep in mind that enhancement is a very subjective area of image processing.

Image restoration is an area that deals with improving the appearance of an image. Enhancement, on the other hand, is based on human subjective preferences regarding what constitutes a “good” enhancement result.

Color image processing is an area that has been gaining in importance because of the significant increase in the use of digital images over the Internet. *Wavelets* are the foundation for representing images in various degrees of resolution.

Compression, as the name implies, deals with techniques for reducing the storage required to save an image, or the bandwidth required to transmit it. Although storage technology has improved significantly over the past decade, the same cannot be said for transmission capacity. This is true particularly in uses of the Internet, which are characterized by significant pictorial content. Image compression is familiar to most users of computers in the form of image file extensions, such as the jpg file extension used in the JPEG (Joint Photographic Experts Group) image compression standard.

Segmentation procedures partition an image into its constituent parts or objects. In general, independent segmentation is one of the most difficult tasks in digital image processing. An uneven segmentation procedure brings the process a long way toward successful solution of imaging problems that require objects to be identified individually. On the other hand, weak or erratic segmentation algorithms almost always guarantee eventual failure. In general, the more accurate the segmentation, the more likely recognition is to succeed. [1],[2],[3]

Representation and description almost always follow the output of a segmentation stage, which usually is raw pixel data, constituting either the boundary of a region (i.e., the set of pixels separating one image region from another) or all the points in the region itself. In either case, converting the data to a form suitable for computer processing is necessary. The first decision that must be made is whether the data should be represented as a boundary or as a complete region. Boundary representation is appropriate when the focus is on external shape characteristics, such as corners and inflections. Regional representation is appropriate when the focus is on internal properties, such as texture or skeletal shape. In some applications, these representations complement each other.

Choosing a representation is only part of the solution for transforming raw data into a form suitable for subsequent computer processing. A method must also be specified for describing the data so that features of interest are highlighted. *Description*, also called *feature selection*, deals with extracting attributes that result in some quantitative information of interest or are basic for differentiating one class of objects from another.

Recognition is the process that assigns a label (e.g., "Pen") to an object based on its descriptors. Digital image processing involves with the development of methods for recognition of individual objects. Knowledge about a problem domain is coded into an image processing system in the form of a knowledge database. This knowledge may be as simple as detailing regions of an image where the information of interest is known to be located, thus limiting the search that has to be conducted in seeking that information. The knowledge base also can be quite complex, such as an interrelated list of all major possible defects in a materials inspection problem or an image database containing high-resolution satellite images of a region in connection with change-detection applications.

2.3.4 Components of an Image Processing System:

With respect to *sensing, an important component of Image Processing System*, two elements are required to acquire digital images. The first is a physical device that is sensitive to the energy radiated by the object to be imaged. The second, called a *digitizer*, is a device for converting the output of the physical sensing device into digital form. For instance, in a digital video camera, the sensors produce an electrical output proportional to light intensity. The digitizer converts these outputs to digital data.

Software for image processing consists of specialized modules that perform specific tasks. A well-designed package also includes the capability for the user to write code that, as a minimum, utilizes the specialized modules.

Digital storage for image processing applications falls into three principal categories: (1) Short-term storage for use during processing, (2) on-line storage for relatively fast recall, and (3) archival storage, characterized by infrequent access. Storage is measured in bytes (eight bits), Kbytes (one thousand bytes), Mbytes (one million bytes), Gbytes (meaning giga, or one billion, bytes), and Tbytes (meaning tera, or one trillion, bytes).

One method of providing short-term storage is computer memory. Another is by specialized boards, called *frame buffers*, that store one or more images and can be accessed rapidly, usually at video rates (e.g., at 30 complete images per second).The latter method allows virtually instantaneous image *zoom*, as well as *scroll* (vertical shifts) and *pan* (horizontal shifts). Online storage generally takes the form of magnetic disks or optical-media storage. The key factor characterizing on-line storage is frequent access to the stored data. Finally, archival storage is characterized by massive storage requirements but infrequent need for access. Magnetic tapes and optical disks housed in "jukeboxes" are the usual media for archival applications.

2.4 Mimas:

Mimas was originally conceived as a platform for real-time machine vision research. Its aim was and still is to reduce the implementation time of new research into the application workspace. It is written in C++ and is released in source code form subject to the GNU Lesser General Public License (LGPL).

Mimas has been used to build a number of vision systems including for two European Union sponsored projects, namely MINIMAN (completed in 2002) and MiCRoN (completed 2005). It is currently being used in the EPSRC funded Nanorobotics contract. Mimas is also being used to build a number of customised vision solutions for academia and industry. The core development team has huge experience in delivering real-time vision solutions. If you do require a customised vision-based solution then please contact the core team of this software. [26]

2.4.1 Algorithms and methods in Mimas:

- Localization of objects using colour in Mimas generic image class (greylevel and colour)
- low level image processing
- frequency domain processing
- variety of recognition methods
- variety of tracking methods
- active contours
- comprehensive matrix library
- variety of statistical operations

- associative neural network
- multi-layer perceptrons ANN
- image capture
- various example interfaces

Mimas is designed to be platform independent from the ground-up. Hence a user interface is not built-in. Rather Mimas acts as the engine of a vision system. Since it is written in C++, we recommend that you use the GPL-ed version of the cross-platform Qt toolkit or the Mozilla XP toolkit for building user interfaces. [26]

2.5 Summary:

A brief study of Computer and Machine vision is presented. A study on Image processing and various issues related to digital image processing are presented with few examples. Lastly a brief study of Mimas and Mimas algorithms and methods are stated. The study involved in gathering the information in this chapter reflects in the following chapters.

Chapter 3

Open Source and Qt4 Environment

3.1 Introduction:

This chapter presents the essential features of Open Source Technology and Qt4 environment which are the basis of this thesis.

3.2 What is Open Source? :

In general Open Source is software available for free for both viewing and modifying, it a Standard provided and maintained by “Open Group”. Any Software firm or an individual agreeing to the standards provided by Open Group can view as well as modify the software provided. The software under Open Source is redistributed as and when it is modified. Examples of Open Source applications are MySQL and PHP[27]

3.3 Qt - Cross-Platform C++ Development:

Qt sets the standard for high-performance, cross-platform application development. It includes a C++ class library and tools for cross-platform development and internationalization.

3.3.1 Qt is Open Source:

Qt comprises all of the advantages of Open Source in a commercially-supported, proven framework:

- **Open source benefits** include an active open source developer community contributes to the ongoing development of Qt while complete code transparency allows Qt developers to "see under the hood", customizing and extending Qt to meet their unique needs.

- **The assurances of a commercial product** include customer-acclaimed product support, a dedicated Qt development team, and a growing ecosystem of 3rd party tools, components and services.

Qt sets the standard for high-performance, cross-platform application development. It includes a C++ class library and tools for cross-platform development and internationalization.[27]

3.3.2 The Qt Class Libraries:

The Qt Class libraries form the foundation of Qt. The libraries makes available approximately 400 fully object-oriented classes with most of the infrastructure functionality needed to build cross-platform server and rich client applications.

The libraries contain classes for GUI, layout, database, internationalization, networking, XML, and much more. [27]

3.3.3 Signals and Slots:

A common and recurring source of problems and crashes in development of GUIs is this issue of how to communicate between different components. The Qt solution to this problem is the Signals and Slots mechanism. Signals and Slots is a central feature of Qt - it provides a typesafe way of facilitating inter-object communication, and is probably the part that differs most from the features provided by other frameworks. [27]

3.3.4 GUI Design with Qt Designer:

Qt Designer is a full-fledged GUI builder. Using Qt Designer, application designers can lay out and preview the GUI of their applications.

Cross-Platform Builds

Writing software for multiple platforms can be tedious and error-prone. Maintaining makefiles can be even more so, especially if several makefiles are required for different compiler and platform combinations. Qt addresses this challenge by including the qmake tool, which takes care of generating correct makefiles for the target platforms.

3.3.5Qt/X11 - The Standard for Desktop Linux Development:

Qt/X11 is the Qt edition for Unix and Unix-like operating systems that run the X Window System, Version 11 or short: X11. It provides all the graphical Qt tools and the entire Qt application framework API, including but not limited to Unicode support, drag'n'drop, 3D OpenGL graphics and network programming.

Qt/X11

Qt for X11 is the fastest, most powerful way of creating powerful, C++-based cross-platform rich client and console applications running on Linux. Development teams using Qt experience a boost to their productivity, while the commercial support and the backing of a commercial development organization also ensures predictability, risk reduction and product development momentum. [27]

Qt/X11 is source compatible with Qt/Windows and Qt/Mac. Code written for either one compiles and runs with the other ones.

3.4 Summary:

This chapter describes Open Source and Qt4 Environment covering Qt class libraries, GUI Designer, Signals and Slots and Qt X11 which have been used in the software development.

Chapter 4

Camera Calibration

4.1 Introduction:

This Chapter emphasizes on calibration techniques and how a distortion free calibration is achieved.

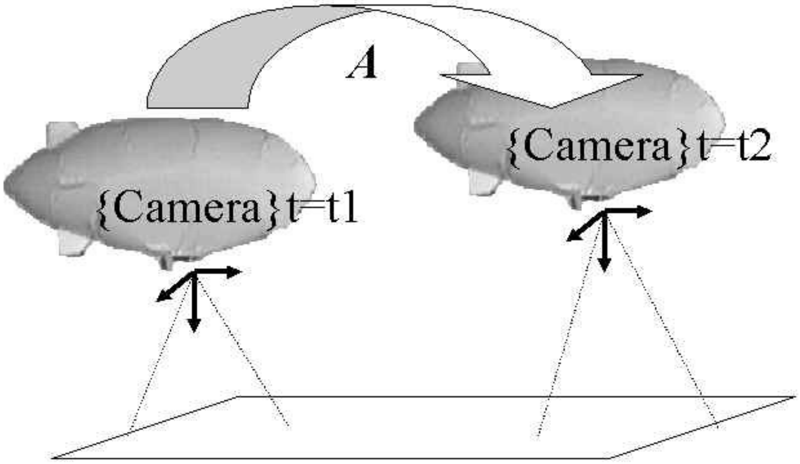
4.2 Calibration:

Calibration is the process where the definition or magnitude of the output or a response of a measuring instrument is compared with the value of the input quantity or attribute, a measurement standard. Calibration is often regarded as the process of adjusting the output or indication on a measurement instrument to agree with value of the applied standard, within a specified accuracy. In simple words it can be termed as Checking and adjusting systematically the operations of a device.

4. 3 Homography:

Homography in mathematical terms is defined as a circle preserving transformation of about an even number of inversions. In general, it is a relationship defined over two figures where any point in one figure corresponds to one and only one point in the other figure and vice-versa.

- Image projections of points belonging to a planar surface, at t=t1 and t=t2 are related by the homography A



- The homography A can be decomposed into rigid motion parameters:

  A = distance x Rotation + translation x normal_to_plane

- The following plots show some examples of applying this technique

4.4 Distortion Free Camera:

“Let $\vec{m}_i \in \mathbb{R}^3,\ i \in \{1, 2, \ldots, N\}$ be the homogeneous coordinate of the *i*th point on the planar calibration-object and let $\vec{m}'_i \in \mathbb{R}^3$ be the homogeneous coordinate of the corresponding pixel in the camera-image. Further let $(\mathbb{R}^n, \simeq)$ be an equivalence relation defined by

$$\vec{a} \simeq \vec{b} \ :\Leftrightarrow\ \exists \lambda \in \mathbb{R}/\{0\} : \ \lambda \vec{a} = \vec{b}$$

If the camera-system does an ideal central projection (e.g. no distortion), the projective transformation (the Homography) can be modeled using $\mathcal{H} \in \mathbb{R}^{3\times 3}$ as follows

$$\vec{m'}_i + \vec{\epsilon}_i \simeq \mathcal{H}\vec{m}_i$$

or more elaborately $\lambda \begin{pmatrix} m'_{i1} \\ m'_{i2} \\ m'_{i3} \end{pmatrix} + \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \\ \epsilon_{i3} \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ m_{i3} \end{pmatrix}$ where $(\epsilon_{i1}, \epsilon_{i2})^\top$ is the zero-mean error-vector in the observation of the *i*th point in the camera-image.

Using $m_{i3} = m'_{i3} = 1,\ \epsilon_{i3} = 0$ and $\vec{h_i}^\top = (h_{i1} \quad h_{i2} \quad h_{i3}),\ i \in \{1, 2, 3\}$ the model can be reformulated to

$\begin{pmatrix} m'_{i1} \\ m'_{i2} \end{pmatrix} = \frac{1}{\vec{h_3}^\top \cdot \vec{m}_i} \begin{pmatrix} \vec{h_1}^\top \\ \vec{h_2}^\top \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix} - \begin{pmatrix} \epsilon_{i1} \\ \epsilon_{i2} \end{pmatrix}$ or

$\begin{pmatrix} m'_{i1} \\ m'_{i2} \end{pmatrix} (\vec{h_3}^\top \cdot \vec{m}_i) = \begin{pmatrix} \vec{h_1}^\top \\ \vec{h_2}^\top \end{pmatrix} \begin{pmatrix} m_{i1} \\ m_{i2} \\ 1 \end{pmatrix} - \begin{pmatrix} \epsilon'_{i1} \\ \epsilon'_{i2} \end{pmatrix}$ with $\vec{\epsilon'}_i = [\vec{h_3}^\top \cdot \vec{m}_i]\, \vec{\epsilon_i}$

and using $|\mathcal{H}| \neq 0$

It is assumed, that the vectors $\vec{\epsilon'}_i$ have equal variances (*i.e.* $\vec{h_3}^\top \cdot \vec{m}_1 \approx \vec{h_3}^\top \cdot \vec{m}_2 \approx \ldots$) so that the Gauss-Markov theorem can be applied using $(\epsilon'_{i1}, \epsilon'_{i2})^\top$ as error-vectors. In this case the minimum least-squares estimator is the best linear estimator.

Each point-pair yields the following system of two linear equations

$$\begin{pmatrix} h_{11}\, m_{i1} + h_{12}\, m_{i2} + h_{13} \\ h_{21}\, m_{i1} + h_{22}\, m_{i2} + h_{23} \end{pmatrix} - \begin{pmatrix} m'_{i1}\, m_{i1}\, h_{31} + m'_{i1}\, m_{i2}\, h_{32} + m'_{i1}\, h_{33} \\ m'_{i2}\, m_{i1}\, h_{31} + m'_{i2}\, m_{i2}\, h_{32} + m'_{i2}\, h_{33} \end{pmatrix} = \begin{pmatrix} \epsilon'_{i1} \\ \epsilon'_{i2} \end{pmatrix}$$

Isolating the elements of the unknown matrix $\mathcal{H}$ gives

$$\begin{pmatrix} m_{i1} & m_{i2} & 1 & 0 & 0 & 0 & m'_{i1}\,m_{i1} & m'_{i1}\,m_{i2} & m'_{i1} \\ 0 & 0 & 0 & m_{i1} & m_{i2} & 1 & m'_{i2}\,m_{i1} & m'_{i2}\,m_{i2} & m'_{i2} \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{pmatrix} = \begin{pmatrix} \epsilon'_{i1} \\ \epsilon'_{i2} \end{pmatrix}$$

The combined system of all linear equations is

$$\underbrace{\begin{pmatrix} m_{11} & m_{12} & 1 & 0 & 0 & 0 & m'_{11}m_{11} & m'_{11}m_{12} & m'_{11} \\ 0 & 0 & 0 & m_{11} & m_{12} & 1 & m'_{12}m_{11} & m'_{12}m_{12} & m'_{12} \\ m_{21} & m_{22} & 1 & 0 & 0 & 0 & m'_{21}m_{21} & m'_{21}m_{22} & m'_{21} \\ 0 & 0 & 0 & m_{21} & m_{22} & 1 & m'_{22}m_{21} & m'_{22}m_{22} & m'_{22} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ m_{N1} & m_{N2} & 1 & 0 & 0 & 0 & m'_{N1}m_{N1} & m'_{N1}m_{N2} & m'_{N1} \\ 0 & 0 & 0 & m_{N1} & m_{N2} & 1 & m'_{N2}m_{N1} & m'_{N2}m_{N2} & m'_{N2} \end{pmatrix}}_{=:\mathcal{M}} \underbrace{\begin{pmatrix} h_{11} \\ h_{12} \\ \vdots \\ h_{33} \end{pmatrix}}_{=:\vec{h}} = \begin{pmatrix} \epsilon'_{11} \\ \epsilon'_{12} \\ \epsilon'_{21} \\ \epsilon'_{22} \\ \vdots \\ \epsilon'_{N1} \\ \epsilon'_{N2} \end{pmatrix}$$

To avoid the trivial solution $\vec{h} = \vec{0}$ the constraint $||\vec{h}|| = 1$ is introduced *without loss of generality*.

The calibration problem now has been reduced to the problem of finding $\widehat{\vec{h}} \in \mathbb{R}^9$ such that

1. $||\mathcal{M}\,\widehat{\vec{h}}||$ is minimal and
2. $||\widehat{\vec{h}}|| = 1$

$\widehat{\vec{h}}$ can be computed using the Singular value decomposition $\mathcal{M} = \mathcal{U}\,\Sigma\,\mathcal{V}^*$, because these are the properties of the right handed singular vector $\vec{v}_1$ with the smallest singular value σ_1 (where $\mathcal{V} = (\vec{v}_1\,\vec{v}_2\,\cdots)$). I.e. $\widehat{\vec{h}} = \vec{v}_1$. “ [20],[21],[12].

4.5 Summary:

This chapter summarizes the techniques like homography and distortion free camera calibration which form an essential part of the system developed.

Chapter 5

Edge Detection, Pattern Search, Object Recognition and Segmentation

5.1 Introduction:

This chapter emphasizes on topics such as Edge Detection, Pattern Search, Object Recognition and Segmentation.

5.2 Edge Detection:

Discrete Differentiation forms the basis for most of the applications in computer vision. One among them is edge detection. An edge is defined as an extended region of the image which undergoes a drastic directional change in intensity. Differential techniques measure these changes.

Basic edge detection begins by computing first-order spatial derivatives of an image f [x, y]:

$$f_x[x, y] = (f[x, y] * h^{1}[x] * h[y]$$

$$f_y[x, y] = (f[x, y] * h[x] * h^{1}[y] ,$$

where $h^1[.]$ and $h\,[.]$ are the derivative and prefilter

The “strength” of an edge at each spatial location is defined to be the magnitude of the gradient vector

$$\nabla[x, y] = (\ f_x[x, y]\ \ f_y[x, y]\ ), \text{ defined as:}$$

$$|\ \nabla[x, y]\ | = \sqrt{(\ f_x^2[x, y] + f_y^2[x, y]}$$

5.3 Pattern Search and Recognition:

"Pattern recognition is the research area that studies the operation and design of systems that recognize patterns in data. It encloses sub disciplines like discriminant analysis, feature extraction, error estimation, and cluster analysis sometimes called statistical pattern recognition, grammatical inference and parsing sometimes called syntactical pattern recognition. Important application areas are image analysis, character recognition, speech analysis, man and machine diagnostics, person identification and industrial inspection."

A complete pattern recognition system consists of a sensor that assembles the observations to be classified or to be described, a feature extraction technique that computes numeric or symbolic information such as images from the observations and a classification or description mechanism that does the actual job of classifying or describing observations, relying on the extracted features.

5.4 Object Recognition:

Object recognition is the process of finding a real world object from an image of the world, mainly utilizing the object models which are pre known. Though prior known it is one of the complicated task. A brief discussion of different steps in object recognition and some techniques that have been used for object recognition in many applications is given below.

The object recognition problem can be termed as a labeling problem based on models of known objects. Basically, given an image containing one or more objects of interest (and background) and a set of labels corresponding to a set of models known to the system, the system should assign correct labels to regions, or a set of regions, in the image. The object recognition problem is closely related with the segmentation problem, which is without recognizing an object at least to a partial extent, segmentation cannot be done, and without segmenting an object, recognition is not possible.

Feature selection or feature extraction is an important aspect of Object recognition as it the features which are compared to the pre known values of the obtained from the models of the objects and the newly extracted features are added to the database of the object models.

Many factors must be considered in the selection of appropriate methods for a particular application. The central issues that should be considered in designing an object recognition system are:

- Object *or model representation*: In what way should the objects be represented in the model database? What would be the important characters or features of the objects that must be captured in these models? .The representation of an object should capture all relevant information without any redundancies and should organize this information in a form that allows easy access by different components of the object recognition system.

- *Feature extraction*: Prominent features should be detected, and so that they can be detected reliably. Most features can be computed in two-dimensional images but they are related to three-dimensional characteristics of objects. Due to the nature of the image formation process, some features are easy to compute reliably while others are very difficult.

- *Feature-model matching*: Methods and Algorithms to match the features in images to models in the database need to be compiled carefully. In most object recognition tasks, there are many features and numerous objects. An exhaustive matching approach will solve the recognition problem but may be too slow to be useful. Effectiveness of features and efficiency of a matching technique must be considered in developing a matching approach.
- *Hypotheses formation*: How can a set of likely objects based on the feature matching be selected, and how can probabilities be assigned to each possible object? The hypothesis formation step is basically a heuristic to reduce the size of the search space. This step uses knowledge of the application domain to assign

some kind of probability or confidence measure to different objects in the domain. This measure reflects the likelihood of the presence of objects based on the detected features.

- Object *verification*: How can object models be used to select the most likely object from the set of probable objects in a given image? The presence of each likely object can be verified by using their models. One must examine each plausible hypothesis to verify the presence of the object or ignore it. If the models are geometric, it is easy to precisely verify objects using camera location and other scene parameters. In other cases, it may not be possible to verify a hypothesis. [13]

5.4.1 Two-dimensional Object Recognition:

In many applications, images are captured from a distance enough to consider the projection to be orthographic. If the objects are always in one static position in the picture, then they can be considered two-dimensional. In these applications, we use a two-dimensional model base. There are two possible cases:

- Objects will not be occluded, as in remote sensing and many industrial applications.
- Objects may be occluded by other objects of interest or be partially visible.

In some cases, though the objects may be far, they may appear in different positions resulting in multiple stable views. In such cases also, the problem may be considered inherently as two-dimensional object recognition.

5.4.2 Three-dimensional Object Recognition:

If the images of objects can be obtained from arbitrary viewpoints, then an object may appear very different in its two views. For object recognition using three-dimensional models, the perspective effect and viewpoint of the image have to be considered. The fact that the models are three-dimensional and the images contain only two-dimensional information affects object recognition approaches. Again, the two factors to be considered are whether objects are separated from other objects or not.

For three-dimensional cases, one should consider the information used in the object recognition task. Two different cases are:

- Intensity: There is no surface information available explicitly in intensity images. Using intensity values, features corresponding to the three-dimensional structure of objects should be recognized.

- 2.5-dimensional images: In many applications, surface representations with viewer-centered coordinates are available, or can be computed, from images. This information can be used in object recognition. Range images are also 2.5-dimensional. These images give the distance to different points in an image from a particular viewpoint.

5.4.3 Using Cross Correlation to Recognize an Object:

If there is an object *g* [*i, j*] and we need to detect its instances in a plane *f*[*i*,*j*]. An obvious thing to do is to place the object at a location in the plane and to detect its presence at that point by comparing intensity values in the object with the corresponding values in the plane. Hardly will there exists a match in the intensities, we need a measure of dissimilarity between the intensity values of the object and the corresponding values of the plane. [18],[25]

Different measures may be defined as:

where *R* is the region of the object.

The sum of the squared errors is the most popular measure. In the case of object matching, this measure can be computed indirectly and computational cost can be reduced.

This can be simplified:

Assume that *f* and *g* are fixed, they then give a measure of mismatch. A reasonable strategy for obtaining all locations and instances of the object is to shift the object and use the match measure at every point in the plane. Thus, for an m × n object, compute

where *k* and *l* are the displacements with respect to the object in the plane. This operation is called the cross-correlation between *f* and *g*.

We need to find the locations that are local maxima and are above a certain threshold value. However, a small problem in the above computation was introduced when we assumed that *f* and *g* are constant. When applying this computation to images, the template *g* is constant, but the value of *f* will be varying. The value of M will then depend on *f* and hence will not give a correct indication of the match at different locations. This problem can be solved by using normalized cross-correlation. The match measure M then can be computed using

It can be shown that *M* takes maximum value for [*i, j*] at which $g = cf$.

5.4.4 Object Tracking Criteria:

Many object tracking applications share the following properties:

- the inter frame motion of the object is tiny so that the object in the next frame is in the neighborhood of the object in the current frame
- the same point on the object has a consistent color/gray scale in all the frames of the sequence
- the boundary of the moving object has a relatively large motion field
- The boundary of the object has a large image gradient value.

The boundary of an object is tracked using the following criteria:

- shape similarity: object shapes are similar in two successive frames
- region similarity: the properties (color, texture) of a region in the object remain constant throughout the sequence
- motion cue: the object boundary should be attracted to pixels with a large amount of motion
- gradient cue: the object boundary should be attracted to pixels with a large image gradient.[14],[15].

5.5 Segmentation:

Segmentation in general is breaking up an object into several readable parts so that even the minute hidden parts in that object could be read in a better way. It means that looking through an object in details by dividing or segmenting the object into readable parts.
The most primary issue considered with image segmentation is differentiation of the object from the background.[23],[24].

5.5.1 Graph-Based Segmentation:

In a graph-based approach to segmentation Let G = (V, E) be an undirected graph with vertices $v_i \in V$, the set of elements to be segmented, and edges $(v_i, vj) \in E$ corresponding to pairs of neighboring vertices. Each edge $(v_i, v_j) \in E$ has a corresponding weight $w((v_i; v_j))$, which is a non-negative measure of the dissimilarity between neighboring elements v_i and v_j.

But where as in the case of image segmentation, the elements in V are pixels and the weight of an edge is some measure of the dissimilarity between the two pixels connected by that edge (e.g., the difference in intensity, color, motion, location or some other local attribute).

In general the elements in a component need to be similar, and elements in different components to be dissimilar. This means that edges between two vertices in the same component should have relatively low weights, and edges between vertices in different components should have higher weights.

5.6 Summary:

This Chapter Summarizes how Edge Detection, Pattern Search, Object Recognition and Segmentation is achieved for computer vision based applications.

Chapter 6

Design, Implementation and Results

6.1 Introduction:

This Chapter emphasizes the design and implementation along with few results.

6.2 Design and Implementation:

Designing a Graphical User Interface for Human Interaction with Computer System Using a Camera Projector

An Interface is developed in Qt4 Environment which has been installed in Suse Linux. Qt4 Designer is very flexible and easy that it allows users to create and develop user-interactive interface in no time and also allows to making connections among components which are famously called as “Widgets”.

Firstly a Stacked Widget is implemented, a Stacked Widget is just as a wizard which can take as many pages as needed which are implemented sequentially one after the other, it can be in other terms same as Stack in Data Structures.

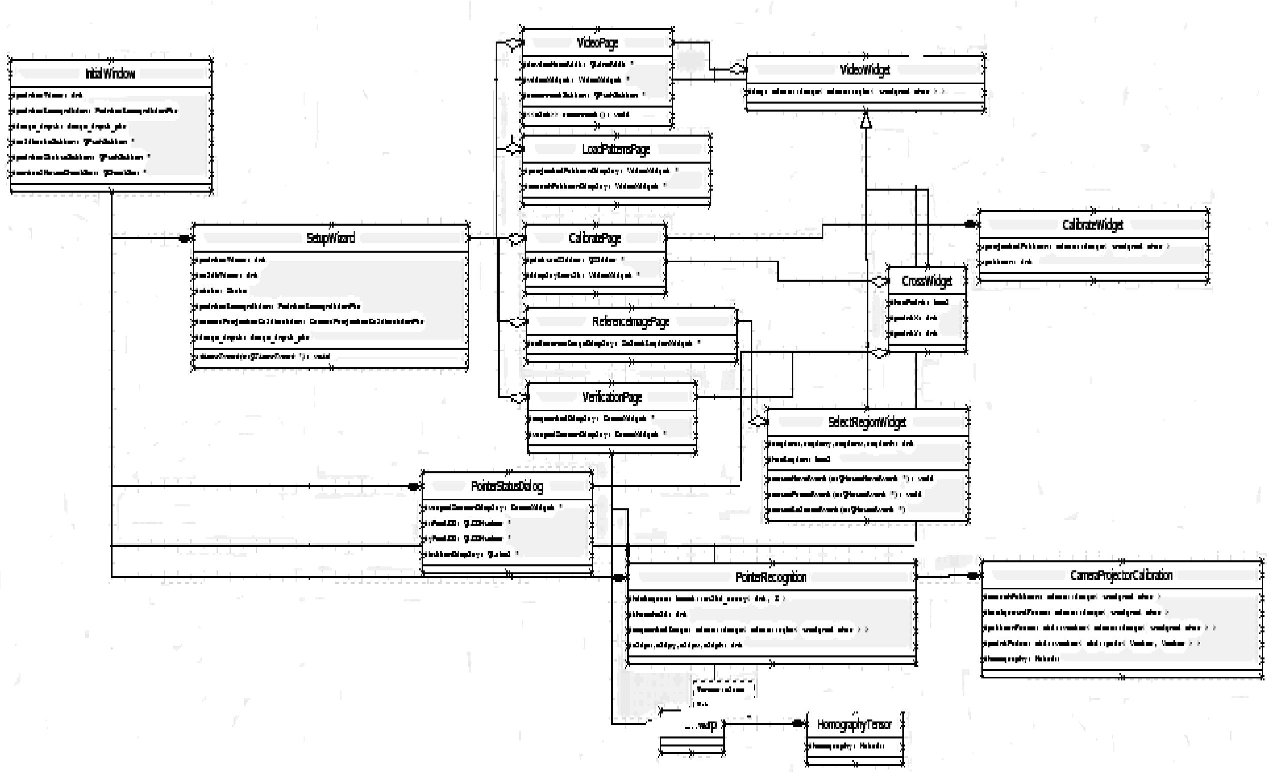
Other Widgets like Pushbuttons, Line Editor, and Labels etc., can be implemented on the same Stacked Widget.

Events are triggered by making Connections between widgets using Signals and Slots.

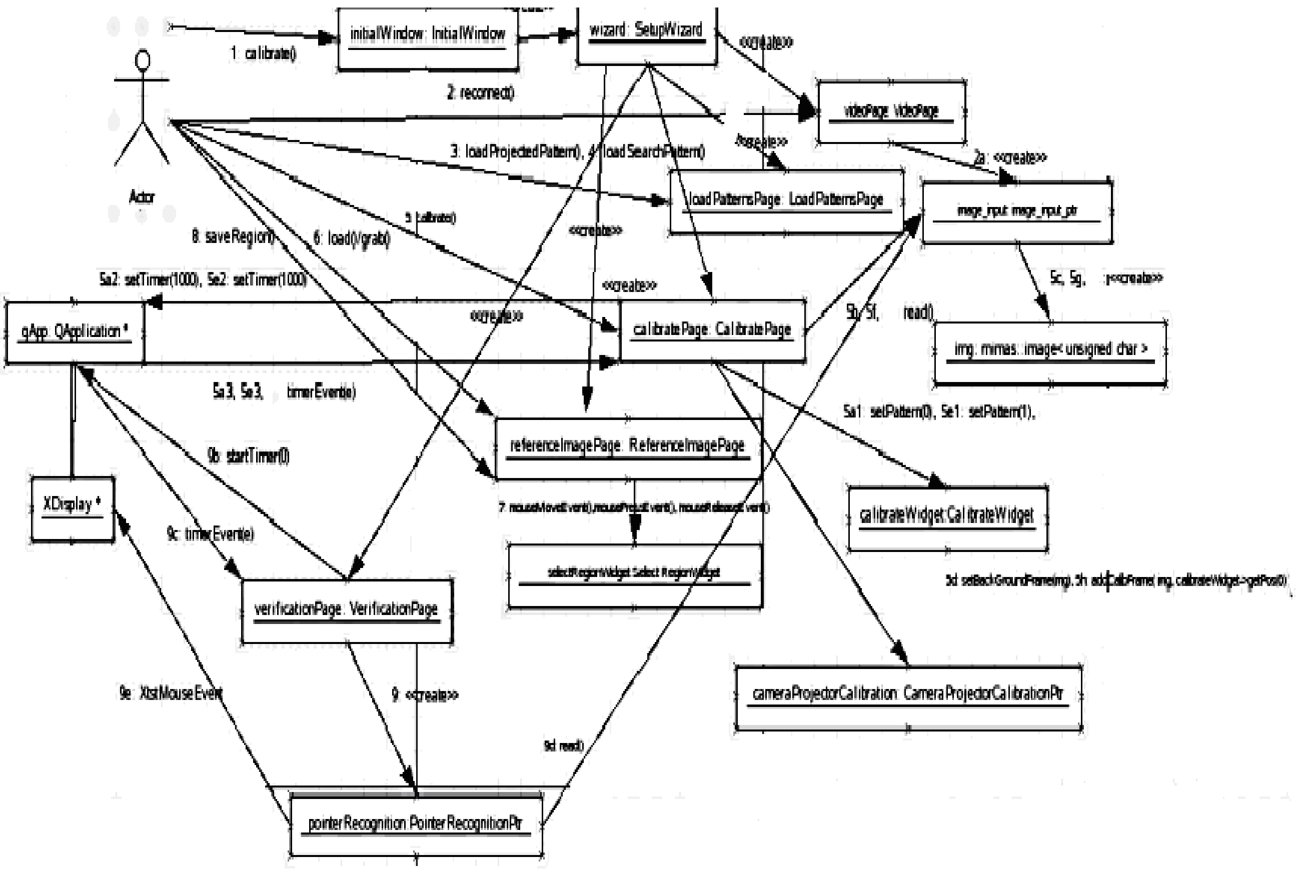
6.3 UML Diagrams:

Below given are the UML Diagrams that depict the static design of the interface.

Static Diagram of Human Interface with Computer System Using a Camera Projector.


InitialWindow
VideoPage
VideoWidget
LoadPatternsPage
SetupWizard
CalibratePage
CalibrateWidget
CrossWidget
ReferenceImagePage
VerificationPage
SelectRegionWidget
PointerStatusDialog
PointerRecognition
CameraProjectorCalibration
...warp
HomographyTensor

Collaboration Diagram of Human Interface with Computer System Using a Camera Projector.

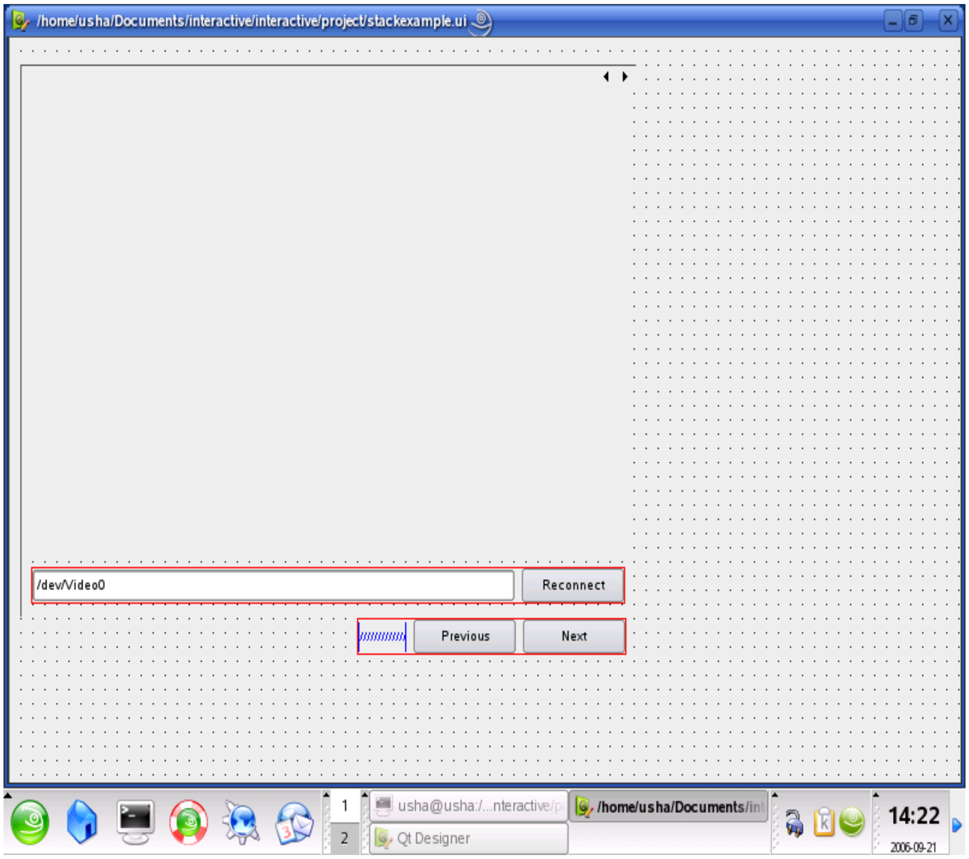

Actor
1: calibrate()
initialWindow: InitialWindow
wizard: SetupWizard
<<create>>
2: reconnect()
videoPage: VideoPage
<<create>>
2a: <<create>>
3: loadProjectedPattern(), 4: loadSearchPattern()
loadPatternsPage: LoadPatternsPage
image_input: image_input_ptr
5: calibrate()
<<create>>
8: saveRegion()
6: load()/grab()
5a2: setTimer(1000), 5e2: setTimer(1000)
<<create>>
5c, 5g, <<create>>
<<create>>
qApp: QApplication *
calibratePage: CalibratePage
5b, 5f, read()
img: mimas::image< unsigned char >
5a3, 5e3, timerEvent(e)
5a1: setPattern(0), 5e1: setPattern(1),
referenceImagePage: ReferenceImagePage
9b: startTimer(0)
XDisplay *
7: mouseMoveEvent(),mousePressEvent(), mouseReleaseEvent()
calibrateWidget:CalibrateWidget
9c: timerEvent(e)
selectRegionWidget:SelectRegionWidget
5d: setBackGroundFrame(img), 5h: addCalibFrame(img, calibrateWidget->getPos())
verificationPage: VerificationPage
9e: XtstMouseEvent
9: <<create>>
cameraProjectorCalibration: CameraProjectorCalibrationPtr
9d: read()
pointerRecognition:PointerRecognitionPtr

6.4 Design of User Interface:

In the Qt Designer a new widget is created and a Qstacked Widget is added on to it, a Stacked Widget is just same the Tab Widget Layout, pages such as Widgets can be layed on the Stacked Widget one after the other as depicted below :

```
class Ui_StackExample

   QStackedWidget *stackedWidget;
   QWidget *page;
   QWidget *page2;
   QWidget *widget;
    stackedWidget = new QStackedWidget(StackExample);
            stackedWidget-
         setObjectName(QString::fromUtf8("stackedWidget"));
   stackedWidget->setCurrentIndex(2);
   page = new QWidget();
    stackedWidget->addWidget(page1);
   page2 = new QWidget();
   widget = new QWidget(page2);
    stackedWidget->addWidget(page2);
   retranslateUi(StackExample);
```

6.4.1 StackedWidget named stackexample:


/home/usha/Documents/interactive/interactive/project/stackexample.ui
Reconnect
Previous
Next
1
2
usha@usha:/...nteractive/p
Qt Designer
/home/usha/Documents/in
14:22
2006-09-21

/dev/Video0

6.5 Searching and Loading a Projected Pattern:

Pattern search and projection as discussed above, is the next step in the program.

A pattern is already agreed upon, now a code to search and project the pre-defined pattern has to be set.

A button click would open a file dialog and upon selection of the pattern it is set.

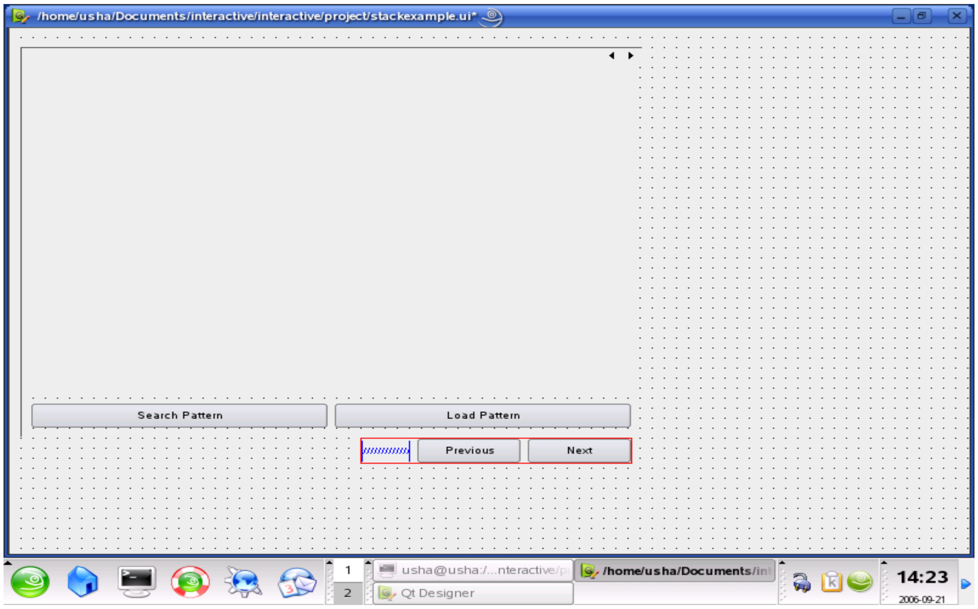
Search Pattern

```
        QString s = QFileDialog::getOpenFileName(
                this, "Search pattern",
                "../../icons",
                "Images (*.png)" );
           if( s != QString::null ) {
                image< rgba< unsigned char > > img;
                ifstream f( s.toLatin1(), ios::binary );
                f >> img;
                ui.searchPatternDisplay ->setImage( img );
```

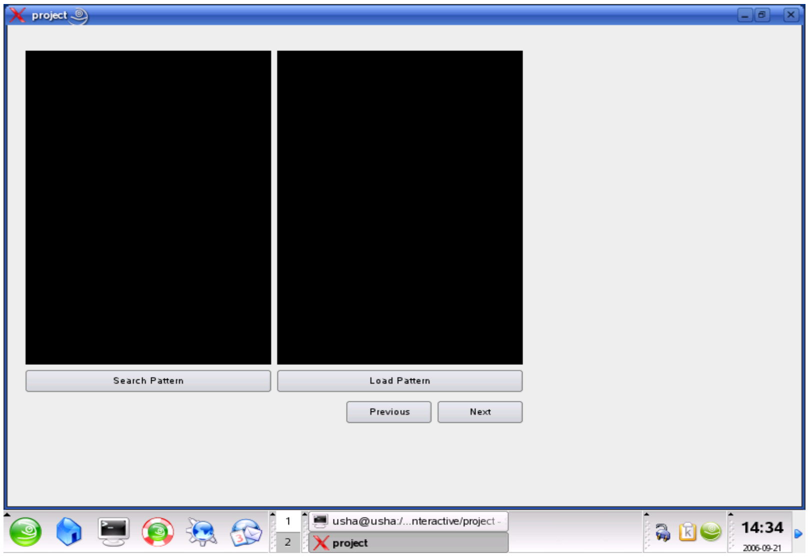
Projected Pattern

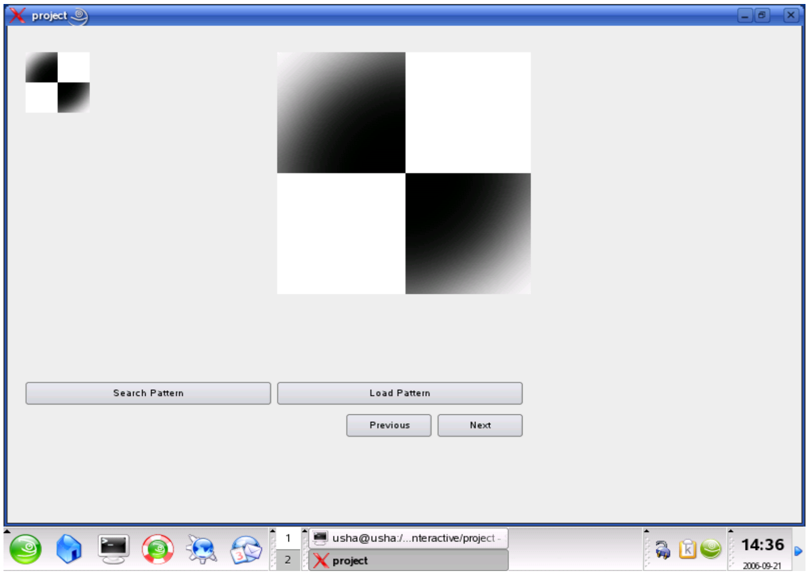
```
QString s = QFileDialog::getOpenFileName(
                        this, "Load pattern",
                        "../../icons",
                        "Images (*.png)" );
            if ( s != QString::null ) {
                  image< rgba< unsigned char > > img;
                  ifstream f( s.toLatin1(), ios::binary );
                  f >> img;

                  ui.projectionPatternDisplay ->setImage( img );
```

6.5.1 GU Interface for Searching and Loading a Projected Pattern:


/home/usha/Documents/interactive/interactive/project/stackexample.ui*
Search Pattern
Load Pattern
Previous
Next
usha@usha:/...nteractive/p
/home/usha/Documents/in
Qt Designer
14:23
2006-09-21

6.5.2 Results of the Search and Load Projected Pattern:


project
Search Pattern
Load Pattern
Previous
Next
1
2
usha@usha:/...nteractive/project -
project
14:34
2006-09-21

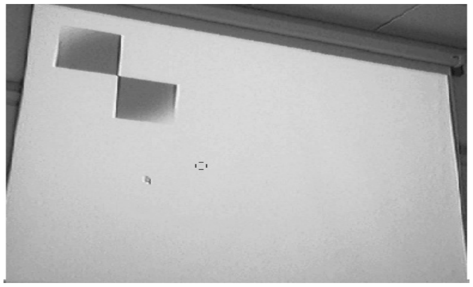

project
Search Pattern
Load Pattern
Previous
Next
usha@usha:/...nteractive/project -
project
14:36
2006-09-21

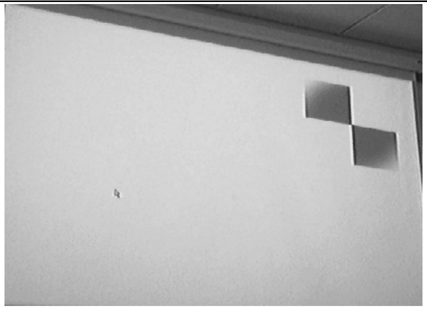
6.6 Design and Implementation of Calibration:

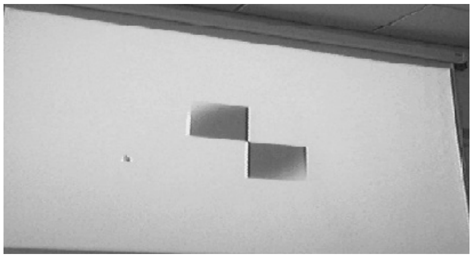
Calibration includes functions likes initializing the window while clearing the color on it or setting it to white color, then resize the window with coordinates like width and height to project it in orthogonal manner, then paint the window with the projected pattern pixel by pixel and finally calibrate the projected pattern on the window on the vector points that is topleft, top right, bottomleft, bottomright and centre points of the Window.

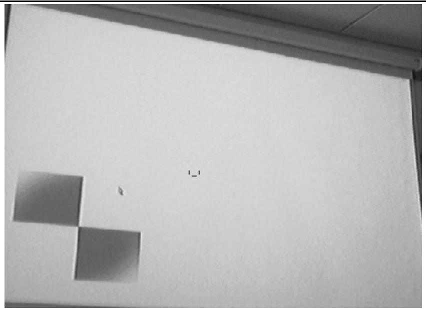
CalibrateWidget::initializeGL
CalibrateWidget::resizeGL
CalibrateWidget::paintGL
CalibrateWidget::Vector CalibrateWidget::getPos (int i)

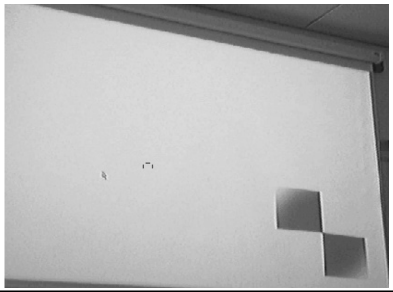
The below shown pictures depict the calibration from blank background to top left corner, top right corner, bottom right corner, bottom left corner and centre of the screen



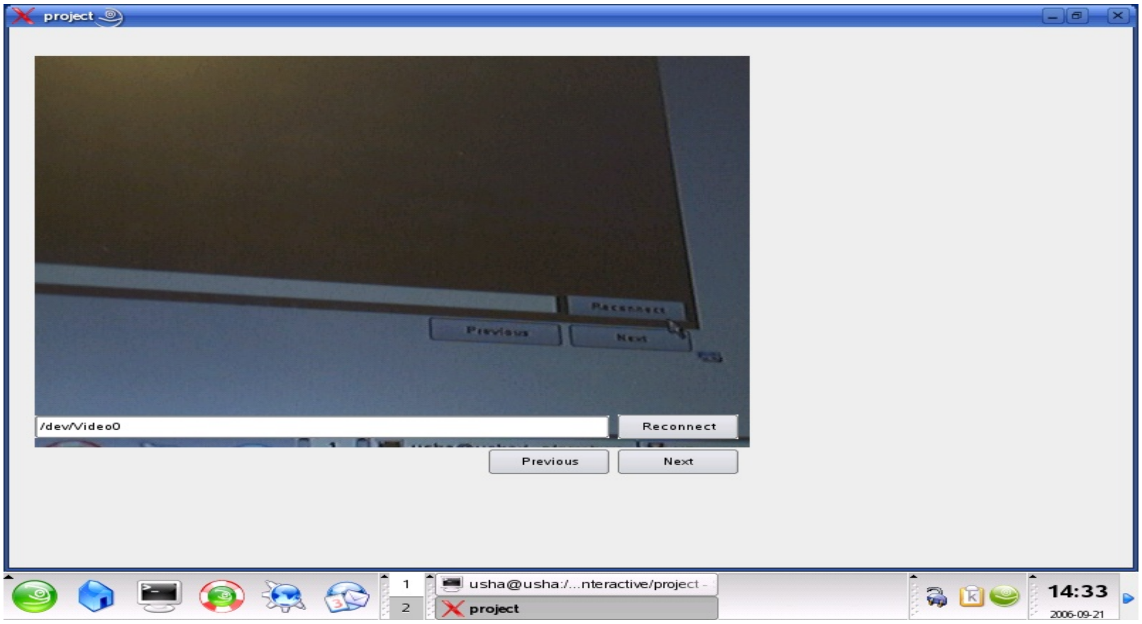






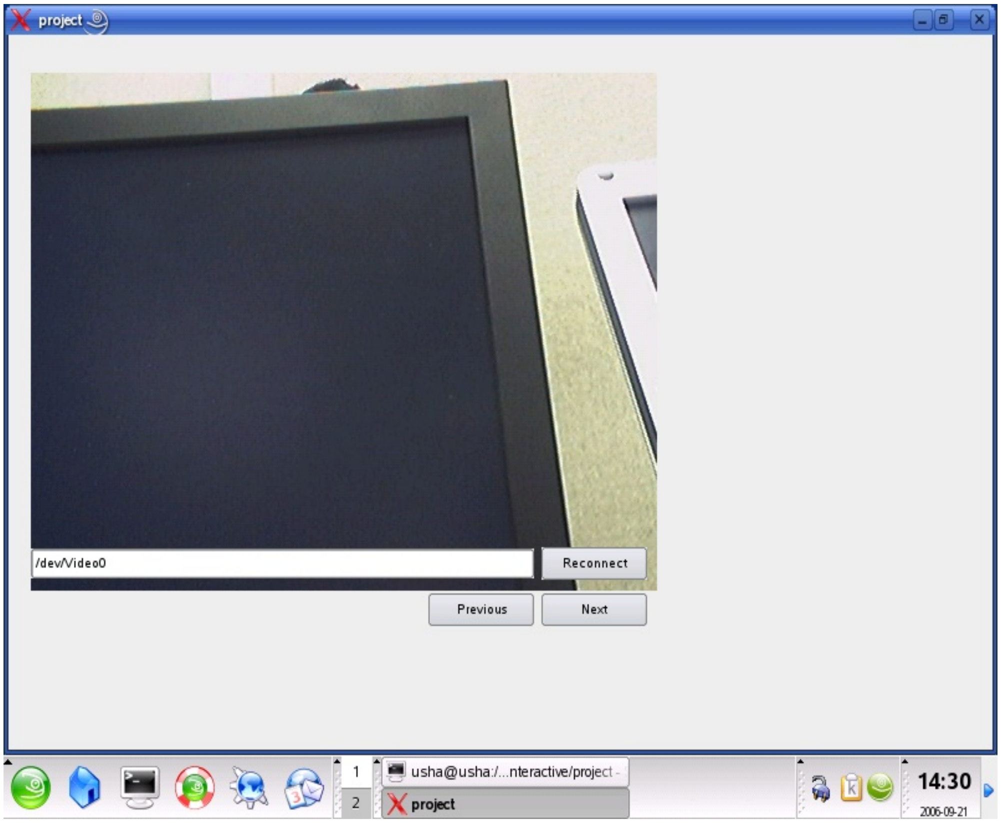


6.7 Implementation Results:



/dev/Video0

project
Reconnect
Previous
Next
1
2
usha@usha:/...nteractive/project -
project
14:33
2006-09-21



/dev/Video0

project
Reconnect
Previous
Next
1
2
usha@usha:/...nteractive/project -
project
14:30
2006-09-21

6.8 Summary:

This Chapter summarizes necessary code snippets along with results of implementation as snapshots

Chapter 7

Conclusion and Further Work

7.1 Discussion:

Thesis on Human interface with Computer System Using a Camera Projector discuss the main aspect of developing a User Interactive System where in a User can interact with a Computer either by selecting objects or navigating through icons and successful trigger events without using a key board or a mouse rather than use a reference object such a laser pointing device or an ordinary pen.

The thesis covers all the fundamental and major relevant aspects of Computer Vision, Machine Vision and Image Processing.

7.2 Conclusion:

This project is one among the fun project developed in Microsystems and Machine Vision Laboratory, though a fun project this system can be effectively put to use for presentation applications. Using Qt4 makes it robust and at the same time flexible.

7.3 Further Work:

More mouse events can be achieved and a virtual key board may also be implemented taking this implementation to more creative levels.

References:

[1] Milan Sonka, Vaclav Hlavac, and Roger Boyle : Image Processing, Analysis and Machine Vision.

[2] Wesley E. Synder and Hairong Qi : Machine Vision.

[3] Loius J. Galbiati, JR. :Machine Vision and Digital Image Processing Fundamentals.

[4] Ramesh Jain, Rangachar Kasturi and Brian G. Schunck : Machine Vision.

[5] Mark S. Nixon and Alberto S. Aguado : Feature Extarction and Image Processing

[6] C H Chen and P S P Wang : Hand book of Pattern Recognition and Computer Vision.

[7] Forsyth and Ponce: Computer Vision A Modern Approach.

[8] E. R. Davies: Machine Vision.

[9] http://www.experience.epson.com.au/help/understandingcolour/COL_G/0507_2.htm

[10] http://www.icaen.uiowa.edu/~dip/LECTURE/ImageProperties2.html

[11] http://www.cs.iitm.ernet.in/~sdas/courses/CV_DIP/PDF/PAT_RECOGN.pdf

[12] http://vision.eng.shu.ac.uk/mediawiki-1.4.10/index.php/Mimas_Camera_Calibration -This is the Calibration technique used in the Program

[13] Pattern Recognition Group at Delft University of Technology - description of their research area

[14] http://www.cse.msu.edu/prip/Files/deformableTemplateTracking.pdf

[15] http://66.249.93.104/search?q=cache:3Za0F7Hu-44J:www.netnam.vn/unescocourse/computervision/chap8.doc+object+Recognition+with+cross+correlation&hl=en&gl=in&ct=clnk&cd=7

[16] http://www.aaai.org/AITopics/html/pattern.html

[17] http://en.wikipedia.org/wiki/Image_processing

[18] D. Comaniciu and P. Meer. Robust analysis of feature spaces: color image segmentation.
Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pages 750-755, 1997.

[19] D. Comaniciu and P. Meer. Mean shift analysis and applications. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, pages 1197-1203, 1999.

[20] Zhengyou Zhang, “Flexible camera calibration by viewing a plane from unknown orientation,” IEEE International Conference on Computer Vision, pp. 666–673, Sep. 1999.

[21] Zhengyou Zhang, “A flexible new technique for camera calibration,” Microsoft Research TechnicalReport, http://research. microsoft.com/~zhang/calib/, 1998.

[22] Richard I. Hartley, “In defense of 8-point algorithm,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 19, no. 6, pp. 580–593, June 1997.

[23] J. Heikkil and O. Silvn, “A four-step camera calibration procedure with implicit image correction,” in IEEE Computer Society Conference on Computer Vision and Pattern Recognition, San Juan, Puerto Rico, 1997, pp. 1106–1112.

[24] P. Sturm and S. Maybank, “On plane-based camera calibration: a general algorithm, singularities, applications,” Proceedings of the Conference on Computer Vision and Pattern Recognition, pp. 432–437, June 1999.

[25] R. Tsai. An efficient and accurate camera calibration technique for 3D machine vision. In *Proceedings of Computer Vision and Pattern Recognition*, 1986.

[26] MMVL HomePage

[27] Qt Trolltech Documentation.

Appendix:

Appendix:

“Program for Interface and Camera Calibration”

C++ Source File

CalibrateWidget.cc Program

```
#include "calibrateWidget.hh"
```

```
using namespace std;
using namespace mimas;

CalibrateWidget::CalibrateWidget( QWidget *parent, Qt::WFlags f ):
  QGLWidget( parent ), pattern(0)
{
}
```

```
CalibrateWidget::CalibrateWidget( image< unsigned char > &_projectedPattern,
                                  QWidget *parent, Qt::WFlags f ):
  QGLWidget( parent ), projectedPattern( _projectedPattern ),
  pattern(0)
{
}


void CalibrateWidget::initializeGL(void)
{
  qglClearColor( Qt::white );
}

void CalibrateWidget::resizeGL( int w, int h )
                                                 {
```

```
  glViewport( 0, 0, (GLint)w, (GLint)h );
  glMatrixMode( GL_PROJECTION );
  glLoadIdentity();
  glOrtho( 0, width(), height(), 0, -1, 1 );
}

void CalibrateWidget::paintGL(void)
{
  glClear( GL_COLOR_BUFFER_BIT );
  glPixelStorei( GL_UNPACK_ALIGNMENT, 1 );
  if ( projectedPattern.initialised() && pattern >= 1 && pattern <= 5 ) {
    glPixelZoom( 1, -1 );
    Vector pos( getPos( pattern ) );
    glRasterPos2d( pos[0] - projectedPattern.getWidth() / 2,
                   pos[1] - projectedPattern.getHeight() / 2 );
    glDrawPixels( projectedPattern.getWidth(), projectedPattern.getHeight(),
                  GL_LUMINANCE, GL_UNSIGNED_BYTE,
                  projectedPattern.rawData().data() );
  };
};
```

```
CalibrateWidget::Vector CalibrateWidget::getPos( int i )
{
  assert( projectedPattern.initialised() );
  assert( pattern >= 1 && pattern <= 5 );
  int
    xmax = width() - projectedPattern.getWidth(),
    ymax = height() - projectedPattern.getHeight(),
    x,
    y;


  switch ( pattern ) {
  case 1:
    x = 0;
    y = 0;
    break;
  case 2:
    x = xmax;
    y = 0;
    break;
  case 3:
    x = 0;
    y = ymax;
    break;
  case 4:
    x = xmax;
    y = ymax;
    break;
  default:
    x = xmax / 2;
    y = ymax / 2;
  };
```

```
  Vector retVal( 3 );
  retVal[ 0 ] = x + projectedPattern.getWidth()  / 2;
  retVal[ 1 ] = y + projectedPattern.getHeight() / 2;
  retVal[ 2 ] = 1.0;
  return retVal;
}

void CalibrateWidget::setPattern( int _pattern )
{
  if ( pattern != _pattern ) {
    pattern = _pattern;
    update();
  };
}
```

Appendix:

"Program for Interface and Camera Calibration"
C++ Header File
CalibrateWidget.hh Program

```
#ifndef __CALIBRATEWIDGET_HH
#define __CALIBRATEWIDGET_HH

#include <boost/array.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/smart_ptr.hpp>
#include <QtOpenGL/QGLWidget>
#include <mimas/image.h>
#include <mimas/image_input.h>

/// Widget for displaying calibration pattern.
class CalibrateWidget: public QGLWidget
{
  Q_OBJECT
```

```
public:
  ///
  typedef boost::numeric::ublas::vector< double > Vector;
  ///
  CalibrateWidget( QWidget *parent = 0, Qt::WFlags f = 0 );
  ///
  CalibrateWidget( mimas::image< unsigned char > &_projectedPattern,
                   QWidget *parent = 0, Qt::WFlags f = 0 );

  Vector getPos( int i );
```

```
public slots:
  /** Display pattern at specified position
      @see pattern
      @see getPos */
  void setPattern(int);
protected:
  ///
  void initializeGL(void);
  ///
  void resizeGL( int w, int h );
  ///
  void paintGL(void);
                                   /// Image storing projection pattern
  mimas::image< unsigned char > projectedPattern;
  /** Index of current pattern-position.
      @see setPattern */
  int pattern;
};

#endif
```

Appendix:

“Program for Interface and Camera Calibration”
C++ Source File
Stackexample.cc Program

```
#include <QtGui/QApplication>
#include <GL/gl.h>
#include<cassert>
#include<complex>
#include <mimas/image.h>
#include <mimas/image_v4linput.h>
#include <QtGui/QFileDialog>
#include <QtGui/QMessageBox>
#include "stackexample.hh"
#include "ui_stackexample.h"
#include <fstream>
#include <mimas/image_funcs.h>
#include <GL/glut.h>
#include <mimas/image_fileinput.h>
#include <mimas/image_mesaoutput.h>
```

```
#include "calibrateWidget.hh"
#include "VideoWidget.hh"
#include "displayWidget.hh"
#include "cameraProjectorCalibration.hh"
#include "cameraProjectorCalibration.cc"
#include "pointerRecognition.hh"
#include "pointerRecognition.cc"
```

```
using namespace mimas;
using namespace std;

StackExample::StackExample( QWidget *parent ): QWidget( parent ),
state( Init ), calibTimer(0), pointerTimer(0), delay( 500 ),
      input( new image_v4linput< rgba< unsigned char > >( "/dev/video0" ))
{

       ui.setupUi( this );
      startTimer( 0 );
   connect(ui.nextButton,SIGNAL(clicked()),this,SLOT(nextPage()));
   connect(ui.previousButton,SIGNAL(clicked()),this,SLOT(prevPage()));
   connect(ui.LoadPatternButton,SIGNAL(clicked()),this,SLOT(loadPattern()));
   connect(ui.SearchPatternButton,SIGNAL(clicked()),this,SLOT(searchPattern()));
   connect(ui.CalibrateButton,SIGNAL(clicked()),this,SLOT(calibrate()));

       connect( ui.pointerCheckBox,
SIGNAL(toggled(bool)),this,SLOT(controlPointer(bool)) );
   connect( ui.pictureSlider, SIGNAL(valueChanged(int)),
        this, SLOT(displayPatternFrame(int)) );


}
```

```
void StackExample::nextPage(void)
{
        ui.stackedWidget->setCurrentIndex( ui.stackedWidget->currentIndex()+1);

}
```

```
void StackExample::calibrate(void)
{
        try {
           image< unsigned char > gimg(ui.projectionPatternDisplay->getImage());
           calibrateWidget= new CalibrateWidget(gimg);
           calibrateWidget->showFullScreen();

            }
       catch ( exception &e ) {
                QMessageBox::critical( this, "Calibration Error", e.what() );
            };
}

void StackExample::controlPointer( bool on )
{
  if ( ( pointerTimer != 0 ) != on ) {
    if ( on ) {

      assert( cameraProjectorCalibration );
      // Start timer (every 100 milliseconds).
      pointerTimer = startTimer( 100 );
    } else {
      // Stop timer.
      killTimer( pointerTimer );
      pointerTimer = 0;
    };
  };
}
```

```
void StackExample::displayPatternFrame( int frame )
{

      assert( cameraProjectorCalibration );
  if ( frame >= 0 && frame < cameraProjectorCalibration->getNumCalibFrames() )
    {
      ui.displayWidget->show();
//     ( cameraProjectorCalibration->getCalibFrame( frame ),
  //     cameraProjectorCalibration->getCalibPointPair( frame ).second );
                                        91,1
};

}
StackExample::image_input_ptr StackExample::getVideoInput(void)
  throw (mimasexception)
{
  if ( !videoInput )
    videoInput =
      image_input_ptr( new image_v4linput< rgba< unsigned char > >( device ) );
  return videoInput;
}

image< unsigned char > StackExample::grabFrame(void)
  throw (mimasexception)
```

```
{
  return image< unsigned char >( grabColourFrame() );
}
```

```
image< rgba< unsigned char > > StackExample::grabColourFrame(void)
  throw (mimasexception)
{
  image< rgba< unsigned char > > retVal;
  *getVideoInput() >> retVal;

 *getVideoInput() >> retVal;
  return retVal;
}

void StackExample::timerEvent( QTimerEvent *e )
{
 refresh();

  const int fringe = 40;
  try {

    if ( e->timerId() == calibTimer ) {

      assert( calibrateWidget != NULL );
      assert( cameraProjectorCalibration );

      // Discarding timer to allow time for cross-correlation.
      killTimer( calibTimer );
      calibTimer = 0;
    switch ( state ) {
    case Init:
      calibrateWidget->setPattern( 0 );
      state = First;

         break;
    case First:
      cameraProjectorCalibration->setBackgroundFrame( grabFrame() );
      calibrateWidget->setPattern( 1 );
      state = Second;
      break;
    case Second:
      cameraProjectorCalibration->addCalibFrame( grabFrame(),
                                   calibrateWidget->getPos( 1 )
);
      calibrateWidget->setPattern( 2 );
      state = Third;
      break;
    case Third:
      cameraProjectorCalibration->addCalibFrame( grabFrame(),
                                   calibrateWidget->getPos( 2 )
);
      calibrateWidget->setPattern( 3 );
      state = Fourth;
      break;
    case Fourth:
      cameraProjectorCalibration->addCalibFrame( grabFrame(),
                                            calibrateWidget->getPos( 3 )

);
```

```
        calibrateWidget->setPattern( 4 );
        state = Fifth;
        break;
 case Fifth:
        cameraProjectorCalibration->addCalibFrame( grabFrame(),
                                            calibrateWidget->getPos( 4 ) );
        calibrateWidget->setPattern( 5 );
        state = Closing;
        break;
      case Closing:
        cameraProjectorCalibration->addCalibFrame( grabFrame(),
                                            calibrateWidget->getPos( 5 ) );
        calibrateWidget->setPattern( 0 );
        state = Finished;
        break;
      default:
        state = Init;
        // Pass "active" area for recognition to recognition object.
        // This is only done at this point, because directly after
        // initialisation of calibrateWidget, calibrateWidget->width() and
                                                      // ..->height() will not have the correct valuies.
        pointerRecognition->setClip( fringe, fringe,
                              calibrateWidget->width() - 2 * fringe,
                              calibrateWidget->height() - 2 * fringe );
 delete calibrateWidget;
        calibrateWidget = NULL;
        ui.CalibrateButton->setEnabled( true );
```

```
/*      ui.pictureSlider->setEnabled( true );
       ui.pointerCheckBox->setEnabled( true );
       // Update display with new result of calibration.
       displayPatternFrame( ui.pictureSlider->value() );*/
      };
```

```
       // Start timer again.
      if ( state != Init )
        calibTimer = startTimer( delay );

    } else if ( e->timerId() == pointerTimer ) {

      assert( pointerRecognition );
      Vector camPos( 3 ), pos( 3 );
      // Perform recognition step and update mouse-cursor position (if found).
      if ( pointerRecognition->findPointer( grabColourFrame(), camPos, pos ) ) {
                                                   // Display *d = XOpenDisplay( NULL );
//       XTestFakeMotionEvent( d, DefaultScreen( d ),
  //    (int)pos[0], (int)pos[1], 0 );
    //   XCloseDisplay( d );
   // Display segmented camera image with estimated cursor position.
   //ui.displayWidget->setInfo( pointerRecognition->getSegmentedImage(), camPos
);
      };
    };

  } catch ( exception &e ) {
    delete calibrateWidget;
    calibrateWidget = NULL;
    cameraProjectorCalibration.reset();
    pointerRecognition.reset();
    state = Init;
   // Kill calibration timer (if present).
    if ( calibTimer != 0 ) {
      killTimer( calibTimer );
      calibTimer = 0;
    };
    // Kill the other timer as well ...
                                          controlPointer( false );
    QMessageBox::critical( this, "Error", e.what() );
  };
}


void StackExample::loadPattern(void)
{
        try {
                QString s = QFileDialog::getOpenFileName(
                            this, "Load pattern",
                            "../../icons",
                            "Images (*.png)" );
                if ( s != QString::null ) {
                        image< rgba< unsigned char > > img;
                        ifstream f( s.toLatin1(), ios::binary );
                        f >> img;

                        ui.projectionPatternDisplay ->setImage( img );
        }
                                    } catch ( exception &e ) {
        QMessageBox::critical( this, "Error loading image", e.what() );
    };
}
```

```
void StackExample::searchPattern(void)
{
    try {

      QString s = QFileDialog::getOpenFileName(
            this, "Search pattern",
            "../../icons",
            "Images (*.png)" );

        if( s != QString::null ) {
            image< rgba< unsigned char > > img;
            ifstream f( s.toLatin1(), ios::binary );
            f >> img;

            ui.searchPatternDisplay ->setImage( img );

            }
    }
    catch ( exception &e ) {
                                QMessageBox::critical( this, "Error loading
image", e.what() );
    };


}
void StackExample::prevPage(void)
{
    ui.stackedWidget->setCurrentIndex(0);
}
```

Appendix:

```
void StackExample::refresh(void)
{

      try
        {
               image< rgba< unsigned char > > img;
               *input >> img;
               ui.videoWidget->setImage( img );
        }

        catch(exception &e)
        {
        };
                                        }
```

Appendix:

“Program for Interface and Camera Calibration”

C++ Header File

Stackexample.hh Program

```
# ifndef STACKEXAMPLE_H
#define STACKEXAMPLE_H

#include <boost/smart_ptr.hpp>
#include <QtOpenGL/QGLWidget>
#include <mimas/image_v4linput.h>
#include <mimas/image.h>
#include <mimas/image_input.h>
#include <mimas/rgba.h>
#include "ui_stackexample.h"
#include "cameraProjectorCalibration.hh"
#include "pointerRecognition.hh"
#include "VideoWidget.hh"
#include "displayWidget.hh"

class CalibrateWidget;

class StackExample: public QWidget
{
	Q_OBJECT
public:
	StackExample( QWidget *parent = 0 );

typedef boost::shared_ptr< mimas::image_input< mimas::rgba< unsigned char > >
```

```
   >image_input_ptr;
```

```
typedef boost::numeric::ublas::vector< double > Vector;
protected:
        void timerEvent(QTimerEvent*);
        image_input_ptr getVideoInput(void) throw (mimas::mimasexception);
        void refresh(void);
        mimas::image< mimas::rgba< unsigned char > > grabColourFrame(void) throw
(mimas
::mimasexception);

        mimas::image< unsigned char > grabFrame(void) throw (mimas::mimasexception);

  std::string device;
  image_input_ptr videoInput;

protected slots:
        void nextPage(void);
        void prevPage(void);
        void loadPattern(void);


  void searchPattern(void);
        void calibrate(void);
        void displayPatternFrame(int);
        void controlPointer(bool);

private:

        int delay;
        boost::shared_ptr< mimas::image_v4linput< mimas::rgba< unsigned char > > >
input;

        mimas::image< mimas::rgba< unsigned char > > img;
        CameraProjectorCalibrationPtr cameraProjectorCalibration;
         enum { Init = 0, First, Second, Third, Fourth, Fifth, Closing,
         Finished } state;
        int calibTimer;
        int pointerTimer;

        PointerRecognitionPtr pointerRecognition;

        CalibrateWidget *calibrateWidget;
        Ui::StackExample ui;
  Ui::StackExample displayWidget;
};
#endif
```

Appendix:

“Program for Pointer Recognition”

C++ Source File

pointerRecognition.cc Program

```
#include <mimas/linalg.h>
#include "pointerRecognition.hh"
//#include "Qt/qursor.h"

using namespace boost;
using namespace mimas;
using namespace std;

PointerRecognition::PointerRecognition
  ( CameraProjectorCalibrationPtr _cameraProjectorCalibration,
    const image< rgba< unsigned char > > &_refImg ):
    cameraProjectorCalibration( _cameraProjectorCalibration ),
    x(0), y(0), w(320), h(200)
{
  setReferenceImage( _refImg );
}
```

```
void PointerRecognition::setReferenceImage
                                        ( const image< rgba< unsigned char > > &_refImg )
{
  // Resize to zero (otherwise old content is preserved).
  histogram.resize( extents[ 0 ][ 0 ][ 0 ] );
  histogram.resize( extents[ 8 ][ 8 ][ 8 ] );
  threshold = _refImg.getWidth() * _refImg.getHeight() / 40;
  cerr << "threshold = " << threshold << endl;
  for ( const rgba< unsigned char > *p = _refImg.rawData().data();
        p != _refImg.rawData().data() + _refImg.rawData().num_elements(); p++ )
    histogram[ (int)p->getRed() / 32 ][ (int)p->getGreen() / 32 ][ (int)p->getBl
ue() / 32 ]++;
}

bool PointerRecognition::findPointer
   ( const image< rgba< unsigned char > > &_frame, Vector &camPos, Vector &pos )
  throw (mimasexception)
{
  MMERROR( histogram.num_elements() > 0, mimasexception, ,
           "Reference image was not initialised." );
  segmentedImage = _frame;

  const rgba< unsigned char > black( 0, 0, 0 );
                                        for ( rgba< unsigned char > *p =
segmentedImage.rawData().data();
        p != segmentedImage.rawData().data() +
segmentedImage.rawData().num_elements(); p++ )
    if ( histogram[ (int)p->getRed() / 32 ][ (int)p->getGreen() / 32 ][ (int)p->getBlue() /
32 ]
         < threshold )
      *p = black;
  bool retVal = false;
  for ( int i=0; i<segmentedImage.getWidth(); i++ ){
    for ( int j=0; j<segmentedImage.getHeight(); j++ ){
      if ( segmentedImage.pixel( i, j ) != black ){
        camPos[0] = i;
        camPos[1] = j;
        camPos[2] = 1.0;


        pos = prod( inv( cameraProjectorCalibration->getHomography() ),
                    camPos );

        pos[0] /= pos[2];
        pos[1] /= pos[2];

pos[2] = 1.0;

        if ( pos[0] >= x && pos[1] >= y &&
             pos[0] < x + w && pos[1] < y + h ) {
          // cerr << camPos[0] << ", " << camPos[1] << " -> " << endl;
          // cerr << pos[0] << ", " << pos[1] << endl;
          retVal = true;
          i=segmentedImage.getWidth(); j=segmentedImage.getHeight();
        };
      }
    }
  }
  return retVal;
}
```

Appendix:

<u>“Program for Pointer Recognition”</u>
<u>C++ Header File</u>
<u>pointerRecognition.hh Program</u>

```
ifndef STACKEXAMPLE_H
#define STACKEXAMPLE_H

#include <boost/smart_ptr.hpp>
#include <QtOpenGL/QGLWidget>
#include <mimas/image_v4linput.h>
#include <mimas/image.h>
#include <mimas/image_input.h>
#include <mimas/rgba.h>
#include "ui_stackexample.h"
#include "cameraProjectorCalibration.hh"
#include "pointerRecognition.hh"
#include "VideoWidget.hh"
#include "displayWidget.hh"

class CalibrateWidget;
class StackExample: public QWidget
{
      Q_OBJECT
public:
      StackExample( QWidget *parent = 0 );

typedef boost::shared_ptr< mimas::image_input< mimas::rgba< unsigned char > >
   >image_input_ptr;
```

```
typedef boost::numeric::ublas::vector< double > Vector;
protected:
        void timerEvent(QTimerEvent*);
        image_input_ptr getVideoInput(void) throw (mimas::mimasexception);
        void refresh(void);
        mimas::image< mimas::rgba< unsigned char > > grabColourFrame(void) throw
(mimas
::mimasexception);

        mimas::image< unsigned char > grabFrame(void) throw (mimas::mimasexception);

  std::string device;
  image_input_ptr videoInput;

protected slots:
        void nextPage(void);
        void prevPage(void);
        void loadPattern(void);


  void searchPattern(void);
        void calibrate(void);
        void displayPatternFrame(int);
        void controlPointer(bool);

private:

        int delay;
        boost::shared_ptr< mimas::image_v4linput< mimas::rgba< unsigned char > > >
input;

        mimas::image< mimas::rgba< unsigned char > > img;
        CameraProjectorCalibrationPtr cameraProjectorCalibration;
         enum { Init = 0, First, Second, Third, Fourth, Fifth, Closing,
         Finished } state;
        int calibTimer;
        int pointerTimer;

        PointerRecognitionPtr pointerRecognition;

        CalibrateWidget *calibrateWidget;
        Ui::StackExample ui;
  Ui::StackExample displayWidget;
};
#endif
```

Appendix:

“Program for cameraProjectorCalibration”

C++ Source File

cameraProjectorCalibration.cc Program

```
#include <boost/multi_array.hpp>
#include <fstream>
#include <mimas/fourier.h>
#include <mimas/linalg.h>
#include <mimas/image_op.h>
#include "cameraProjectorCalibration.hh"

using namespace boost;
using namespace mimas;
using namespace std;

CameraProjectorCalibration::CameraProjectorCalibration
  ( const image< unsigned char > &_searchPattern ):
    searchPattern( _searchPattern ),
    pointPairsModified(false)
{
}
```

```
void CameraProjectorCalibration::addCalibFrame
   ( const mimas::image< unsigned char > &_frame,
     const Vector &_pos )



{
  patternFrame.push_back( _frame );
 pointPairs.push_back( make_pair( _pos, findPatternImg( _frame ) ) );
  pointPairsModified = true;
}

CameraProjectorCalibration::Matrix
   CameraProjectorCalibration::getHomography(void)
{
  if ( pointPairsModified ) {
    homography = genHomography( pointPairs );
    pointPairsModified = false;
  };
  return homography;
}

multi_array< double, 2 >::array_view< 2 >::type CameraProjectorCalibration::view
   ( multi_array< double, 2 > &in, int x, int y, int w, int h )
{
  typedef multi_array< double, 2 >::index_range range;
  typedef multi_array< double, 2 >::array_view< 2 >::type array_view;
  multi_array< double, 2 >::index_gen indices;
  array_view out =
    in[ indices[ range( y, y + h ) ][ range( x, x + w ) ] ];
  return out;
                            }
```

```
CameraProjectorCalibration::Matrix CameraProjectorCalibration::genHomography
  ( const std::vector< std::pair< Vector, Vector > > &pointPairs )
{
 Matrix system( 2 * pointPairs.size(), 9 );
 Matrix Vt;

 // Fill in the matrix to solve the system
 for(int i=0; i<(signed)pointPairs.size(); i++ ) {

  double
   x1  = pointPairs[i].first[0],
   x2  = pointPairs[i].first[1],
   xs1 = pointPairs[i].second[0],
   xs2 = pointPairs[i].second[1];

  system(2*i,0) = x1;
  system(2*i,1) = x2;
  system(2*i,2) = 1;
  system(2*i,3) = 0;
  system(2*i,4) = 0;
  system(2*i,5) = 0;
                    system(2*i,6) = -x1 * xs1;
  system(2*i,7) = -x2 * xs1;
  system(2*i,8) = -xs1;

  system(2*i+1,0) = 0;
  system(2*i+1,1) = 0;
  system(2*i+1,2) = 0;
  system(2*i+1,3) = -x1;
  system(2*i+1,4) = -x2;
  system(2*i+1,5) = -1;
  system(2*i+1,6) = x1 * xs2;
  system(2*i+1,7) = x2 * xs2;
  system(2*i+1,8) = xs2;
 }
```

```
  // calculate the SVD of the matrix A = U.sigma.Vt
  // we don't need to compute U or sigma.
  gesvd< double >( system, NULL, &Vt );

  // the result is the last column of the matrix V, which is the
  // last line of Vt.
  Matrix retVal( 3, 3 );
  // we put the result into the homography matrix
                                          retVal(0,0) = Vt(8,0);
  retVal(0,1) = Vt(8,1);
  retVal(0,2) = Vt(8,2);
  retVal(1,0) = Vt(8,3);
  retVal(1,1) = Vt(8,4);
  retVal(1,2) = Vt(8,5);
  retVal(2,0) = Vt(8,6);
  retVal(2,1) = Vt(8,7);
  retVal(2,2) = Vt(8,8);



  return retVal;
}
```

```
CameraProjectorCalibration::Vector CameraProjectorCalibration::findPatternDiffImg
   ( const image< double > &diffImg,
     const image< double > &tpl )
{
  multi_array< double, 2 > imgField
    ( extents[ diffImg.getHeight() + tpl.getWidth() ]
             [ diffImg.getWidth() + tpl.getHeight() ] );
  view( imgField, 0, 0, diffImg.getWidth(), diffImg.getHeight() ) =
    diffImg.rawData();
                                    multi_array< double, 2 > tplField
    ( extents[ diffImg.getHeight() + tpl.getWidth() ]
             [ diffImg.getWidth() + tpl.getHeight() ] );
  view( tplField, 0, 0, tpl.getWidth(), tpl.getHeight() ) = tpl.rawData();

  multi_array< double, 2 > crossCorrelation
    ( invrfft( rfft( imgField ) *
               conj( rfft( tplField ) ) ) );
  int maxIndex =
    max_element( crossCorrelation.data(),
                 crossCorrelation.data() + crossCorrelation.num_elements() ) -
    crossCorrelation.data();

  Vector retVal( 3 );
  retVal[0] = maxIndex % crossCorrelation.shape()[1] + tpl.getWidth() / 2;
  retVal[1] = maxIndex / crossCorrelation.shape()[1] + tpl.getHeight() / 2;
  retVal[2] = 1.0;
  return retVal;
}
```

```
CameraProjectorCalibration::Vector CameraProjectorCalibration::findPatternImg
   ( const image< unsigned char > &img )
{
                                        assert( searchPattern.initialised() );
  return findPatternDiffImg( image< double >( backgroundFrame ) -
                             image< double >( img ),
                             128.0 - image< double >( searchPattern ) );
}
```

Appendix:

“Program for cameraProjectorCalibration”

C++ Header File

cameraProjectorCalibration.hh Program

```
#ifndef __CAMERAPROJECTORCALIBRATION_HH
#define __CAMERAPROJECTORCALIBRATION_HH

#include <boost/numeric/ublas/matrix.hpp>
#include <boost/numeric/ublas/vector.hpp>
#include <boost/smart_ptr.hpp>
#include <mimas/image.h>
#include <mimas/rgba.h>
#include <algorithm>
#include <vector>

/// Class for calibrating the camera-projector system.
class CameraProjectorCalibration
{
public:
  ///
  typedef boost::numeric::ublas::vector< double > Vector;
  ///
  typedef boost::numeric::ublas::matrix
    < double, boost::numeric::ublas::column_major > Matrix;
  /** Constructor.
      @param _searchPattern Pattern used in cross-correlation to locate
      projected pattern. */
```

```
CameraProjectorCalibration( const mimas::image< unsigned char > &_searchPattern );
 ///
 int getNumCalibFrames(void) const { return patternFrame.size(); }
 /** Get currently stored background frame.
    Camera image of screen without projected pattern. */
 const mimas::image< unsigned char > &getBackgroundFrame(void) const
 { return backgroundFrame; }
 /// Get ith stored calibration frame.
 const mimas::image< unsigned char > &getCalibFrame( int i ) const
 { return patternFrame[i]; }
 ///
 const std::pair< Vector, Vector > &getCalibPointPair( int i ) const
 { return pointPairs[i]; }
 ///
 void setBackgroundFrame( const mimas::image< unsigned char > &img )
 { backgroundFrame = img; }
 /** Add a frame for calibration.
    The method \c findPatternImg is called to locate the projected pattern
          in the camera image. \c pointPairsModified is set to \c true .
    @param _frame Frame for calibration (camera image).
    @param _pos Screen-coordinates of the projected pattern.
    @see findPatternImg */
                              void addCalibFrame( const mimas::image< unsigned char >
&_frame,
                const Vector &_pos );
```

```
  /** Compute calibration-matrix.
      The homography is computed if a new camera-image was added using
      \c addCalibFrame (i.e. \c pointPairsModified is \c true) using the point
      pairs acquired so far (at least 5 point-pairs are required). */
  Matrix getHomography(void);
protected:
  /** Compute homography.
      See <A HREF="http://research.microsoft.com/%7Ezhang/Papers/TR98-
71.pdf">paper by Zhengyou Zhang</A> */
  static Matrix genHomography( const std::vector< std::pair< Vector, Vector > >
                               &pointPairs );
  /// Access a part of a boost::multi_array.
  static boost::multi_array< double, 2 >::array_view< 2 >::type view
   ( boost::multi_array< double, 2 > &in, int x, int y, int w, int h );
  /** Search a pattern in a difference image.
      The search is performed by searching the maximum of the
      cross-correlation. */
  static Vector findPatternDiffImg( const mimas::image< double > &diffImg,
                                    const mimas::image< double > &tpl );
  /** Search pattern in image.
      @see findPattern */
                                     Vector findPatternImg( const mimas::image< unsigned char >
&img );
  ///
  mimas::image< unsigned char > searchPattern;
  /// Background image to subtract from acquired images.
  mimas::image< unsigned char > backgroundFrame;
  /// The vector of acquired images.
  std::vector< mimas::image< unsigned char > > patternFrame;
  /// Vector with estimated locations of points.
  std::vector< std::pair< Vector, Vector > > pointPairs;
  /// Indication wether the homography needs to be recomputed.
  bool pointPairsModified;
  /// Storing the homography matrix.
  Matrix homography;
};
```

```
/** Smart-pointer.
   @see CameraProjectorCalibration */
typedef boost::shared_ptr< CameraProjectorCalibration >
CameraProjectorCalibrationPtr;

#endif
```

“Program for GUI Design”

Qt4 Designer File

ui_stackexapmle.cc Program

```
#ifndef UI_STACKEXAMPLE_H
#define UI_STACKEXAMPLE_H

#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QCheckBox>
#include <QtGui/QGridLayout>
#include <QtGui/QHBoxLayout>
#include <QtGui/QLineEdit>
#include <QtGui/QPushButton>
#include <QtGui/QSlider>
#include <QtGui/QSpacerItem>
#include <QtGui/QStackedWidget>
#include <QtGui/QVBoxLayout>
#include <QtGui/QWidget>
#include "VideoWidget.hh"
//#include "ui_displayWidget.h"
```

```
class Ui_StackExample
{
public:
   QWidget *layoutWidget;
   QHBoxLayout *hboxLayout;
   QSpacerItem *spacerItem;
   QPushButton *previousButton;
   QPushButton *nextButton;
   QStackedWidget *stackedWidget;
   QWidget *page;
   QVBoxLayout *vboxLayout;
   VideoWidget *videoWidget,*displayWidget;
   QHBoxLayout *hboxLayout1;
   QLineEdit *LineEditor;
   QPushButton *ReconnectButton;
   QWidget *page1;
   QGridLayout *gridLayout;
   QPushButton *SearchPatternButton;
   VideoWidget *searchPatternDisplay;
   QPushButton *LoadPatternButton;
   VideoWidget *projectionPatternDisplay;
   QWidget *page2;
   QWidget *widget;
   QHBoxLayout *hboxLayout2;
   QSpacerItem *spacerItem1;
                                    QCheckBox *pointerCheckBox;
   QPushButton *CalibrateButton;
   QSlider *pictureSlider;
   void setupUi(QWidget *StackExample)
   {
   StackExample->setObjectName(QString::fromUtf8("StackExample"));
   StackExample->resize(QSize(530, 439).expandedTo(StackExample-
>minimumSizeHint()));
   layoutWidget = new QWidget(StackExample);
   layoutWidget->setObjectName(QString::fromUtf8("layoutWidget"));
   layoutWidget->setGeometry(QRect(290, 410, 224, 26));
   hboxLayout = new QHBoxLayout(layoutWidget);
   hboxLayout->setSpacing(6);
   hboxLayout->setMargin(0);
   hboxLayout->setObjectName(QString::fromUtf8("hboxLayout"));
   spacerItem = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);

   hboxLayout->addItem(spacerItem);

   previousButton = new QPushButton(layoutWidget);
                                       previousButton-
>setObjectName(QString::fromUtf8("previousButton"));

   hboxLayout->addWidget(previousButton);

   nextButton = new QPushButton(layoutWidget);
   nextButton->setObjectName(QString::fromUtf8("nextButton"));

   hboxLayout->addWidget(nextButton);
```

```
stackedWidget = new QStackedWidget(StackExample);
stackedWidget->setObjectName(QString::fromUtf8("stackedWidget"));
stackedWidget->setGeometry(QRect(10, 20, 512, 389));
stackedWidget->setCurrentIndex(2);
page = new QWidget();
page->setObjectName(QString::fromUtf8("page"));
vboxLayout = new QVBoxLayout(page);
vboxLayout->setSpacing(6);
vboxLayout->setMargin(8);
vboxLayout->setObjectName(QString::fromUtf8("vboxLayout"));
videoWidget = new VideoWidget(page);
videoWidget->setObjectName(QString::fromUtf8("videoWidget"));
vboxLayout->addWidget(videoWidget);

hboxLayout1 = new QHBoxLayout();
hboxLayout1->setSpacing(6);
hboxLayout1->setMargin(0);
hboxLayout1->setObjectName(QString::fromUtf8("hboxLayout1"));
LineEditor = new QLineEdit(page);
LineEditor->setObjectName(QString::fromUtf8("LineEditor"));

hboxLayout1->addWidget(LineEditor);

ReconnectButton = new QPushButton(page);
ReconnectButton->setObjectName(QString::fromUtf8("ReconnectButton"));

hboxLayout1->addWidget(ReconnectButton);


vboxLayout->addLayout(hboxLayout1);

stackedWidget->addWidget(page);
page1 = new QWidget();
page1->setObjectName(QString::fromUtf8("page1"));
gridLayout = new QGridLayout(page1);
```

```
    gridLayout->setSpacing(6);
                                    gridLayout->setMargin(8);
    gridLayout->setObjectName(QString::fromUtf8("gridLayout"));
    SearchPatternButton = new QPushButton(page1);
    SearchPatternButton->setObjectName(QString::fromUtf8("SearchPatternButton"));

    gridLayout->addWidget(SearchPatternButton, 1, 0, 1, 1);

    searchPatternDisplay = new VideoWidget(page1);
    searchPatternDisplay->setObjectName(QString::fromUtf8("searchPatternDisplay"));

    gridLayout->addWidget(searchPatternDisplay, 0, 0, 1, 1);

    LoadPatternButton = new QPushButton(page1);
    LoadPatternButton->setObjectName(QString::fromUtf8("LoadPatternButton"));

    gridLayout->addWidget(LoadPatternButton, 1, 1, 1, 1);

    projectionPatternDisplay = new VideoWidget(page1);
    projectionPatternDisplay-
>setObjectName(QString::fromUtf8("projectionPatternDisplay"));

    gridLayout->addWidget(projectionPatternDisplay, 0, 1, 1, 1);

    stackedWidget->addWidget(page1);
    page2 = new QWidget();
    page2->setObjectName(QString::fromUtf8("page2"));
    widget = new QWidget(page2);
    widget->setObjectName(QString::fromUtf8("widget"));
    widget->setGeometry(QRect(280, 350, 230, 26));
    hboxLayout2 = new QHBoxLayout(widget);
    hboxLayout2->setSpacing(6);
    hboxLayout2->setMargin(0);
    hboxLayout2->setObjectName(QString::fromUtf8("hboxLayout2"));
    spacerItem1 = new QSpacerItem(40, 20, QSizePolicy::Expanding,
QSizePolicy::Minimum);
```

```
   hboxLayout2->addItem(spacerItem1);

   pointerCheckBox = new QCheckBox(widget);
   pointerCheckBox->setObjectName(QString::fromUtf8("pointerCheckBox"));

   hboxLayout2->addWidget(pointerCheckBox);

                                  CalibrateButton = new QPushButton(widget);
   CalibrateButton->setObjectName(QString::fromUtf8("CalibrateButton"));

   hboxLayout2->addWidget(CalibrateButton);

   pictureSlider = new QSlider(page2);
   pictureSlider->setObjectName(QString::fromUtf8("pictureSlider"));
   pictureSlider->setGeometry(QRect(490, 30, 20, 301));
   pictureSlider->setOrientation(Qt::Vertical);
   stackedWidget->addWidget(page2);
   retranslateUi(StackExample);

   QMetaObject::connectSlotsByName(StackExample);
   } // setupUi

   void retranslateUi(QWidget *StackExample)
   {
   previousButton->setText(QApplication::translate("StackExample", "Previous", 0,
QApplication::UnicodeUTF8));
   nextButton->setText(QApplication::translate("StackExample", "Next", 0,
QApplication::UnicodeUTF8));
   LineEditor->setText(QApplication::translate("StackExample", "/dev/Video0", 0,
QApplication::UnicodeUTF8));
                        ReconnectButton->setText(QApplication::translate("StackExample",
"Reconnect", 0, QApplication::UnicodeUTF8));
   SearchPatternButton->setText(QApplication::translate("StackExample", "Search
Pattern", 0, QApplication::UnicodeUTF8));
   LoadPatternButton->setText(QApplication::translate("StackExample", "Load Pattern",
0, QApplication::UnicodeUTF8));
```

```
   pointerCheckBox->setText(QApplication::translate("StackExample",
"ControlPointer", 0, QApplication::UnicodeUTF8));
   CalibrateButton->setText(QApplication::translate("StackExample", "Calibrate", 0,
QApplication::UnicodeUTF8));
   Q_UNUSED(StackExample);
   } // retranslateUi

};


namespace Ui {
   class StackExample: public Ui_StackExample {};
} // namespace Ui

#endif // UI_STACKEXAMPLE_H
```

Make File and Project Files”

```
############################################################################
#####
# Makefile for building: project
# Generated by qmake (2.00a) (Qt 4.0.1) on: Wed Sep 13 11:57:38 2006
# Project:  project.pro
# Template: app
# Command: /usr/bin/qmake -unix -o Makefile project.pro
############################################################################
#####

####### Compiler, tools and options

CC            = gcc
CXX           = g++
LEX           = flex
YACC          = yacc
DEFINES       = -DQT_NO_DEBUG -DQT_CORE_LIB -DQT_GUI_LIB
-DQT_OPENGL_LIB -DQT_SH
ARED
CFLAGS        = -pipe -fno-strict-aliasing -O2 -Wall -W -D_REENTRANT $(DEFINES)
CXXFLAGS      = -pipe -fno-strict-aliasing -O2 -Wall -W -D_REENTRANT
$(DEFINES)
LEXFLAGS      =
YACCFLAGS     = -d
```

```
INCPATH       = -I/usr/share/qt/mkspecs/default -I. -I/usr/include/QtOpenGL -I/u
sr/include/QtGui -I/usr/include/QtCore -I/usr/include -I. -I/usr/X11R6/include -
I. -I.
                              LINK          = g++
LFLAGS        =
LIBS          = $(SUBLIBS) -L/usr/lib -L/usr/X11R6/lib -lmimas -L/usr/lib -L
/usr/src/packages/BUILD/qt-x11-opensource-src-4.0.1/lib -L/usr/X11R6/lib -lQtOpenGL
-lQtGui -lpng -lSM -lICE -lXi -lXrender -lXrandr -lXcursor -lXinerama -lfreetype
 -lfontconfig -lXext -lX11 -lm -lQtCore -lz -ldl -lGLU -lGL -lpthread          AR            =
ar cqs
RANLIB        =
QMAKE         = /usr/bin/qmake                                        TAR           = tar -cf
COMPRESS      = gzip -9f
COPY          = cp -f                                         COPY_FILE     = $(COPY)
COPY_DIR      = $(COPY) -r
INSTALL_FILE  = $(COPY_FILE)                                            INSTALL_DIR   =
$(COPY_DIR)
DEL_FILE      = rm -f
SYMLINK       = ln -sf                                          DEL_DIR       = rmdir
MOVE          = mv -f
                        CHK_DIR_EXISTS= test -d
MKDIR         = mkdir -p

####### Output directory

OBJECTS_DIR   = ./
```

```
####### Files
```

```
SOURCES       = main.cc \
              stackexample.cc \
              VideoWidget.cc \
              calibrateWidget.cc moc_stackexample.cpp \
              moc_VideoWidget.cpp \
              moc_calibrateWidget.cpp
OBJECTS       = main.o \
              stackexample.o \
              VideoWidget.o \
              calibrateWidget.o \
              moc_stackexample.o \
              moc_VideoWidget.o \
              moc_calibrateWidget.o
                        DIST          = /usr/share/qt/mkspecs/qconfig.pri \
              /usr/share/qt/mkspecs/features/qt_config.prf \
              /usr/share/qt/mkspecs/features/exclusive_builds.prf \
              /usr/share/qt/mkspecs/features/default_pre.prf \
              /usr/share/qt/mkspecs/features/release.prf \
              /usr/share/qt/mkspecs/features/default_post.prf \
              /usr/share/qt/mkspecs/features/warn_on.prf \
              /usr/share/qt/mkspecs/features/qt.prf \
              /usr/share/qt/mkspecs/features/unix/opengl.prf \
              /usr/share/qt/mkspecs/features/unix/thread.prf \
              /usr/share/qt/mkspecs/features/moc.prf \
              /usr/share/qt/mkspecs/features/resources.prf \
              /usr/share/qt/mkspecs/features/uic.prf \
              project.pro
QMAKE_TARGET  = project
DESTDIR       = 
TARGET        = project
```

```
first: all
####### Implicit rules
```

```
.SUFFIXES: .c .o .cpp .cc .cxx .C
                              .cpp.o:
       $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.cc.o:
       $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.cxx.o:
       $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.C.o:
       $(CXX) -c $(CXXFLAGS) $(INCPATH) -o $@ $<

.c.o:
       $(CC) -c $(CFLAGS) $(INCPATH) -o $@ $<

####### Build rules

all: Makefile $(TARGET)

$(TARGET):  $(OBJECTS)
       $(LINK) $(LFLAGS) -o $(TARGET) $(OBJECTS) $(OBJCOMP) $(LIBS)
```

Appendix:

```
Makefile: project.pro  /usr/share/qt/mkspecs/default/qmake.conf
/usr/share/qt/mkspecs/qconfig.pri \
              /usr/share/qt/mkspecs/features/qt_config.prf \
              /usr/share/qt/mkspecs/features/exclusive_builds.prf \
		/usr/share/qt/mkspecs/features/default_pre.prf \
		/usr/share/qt/mkspecs/features/release.prf \
		/usr/share/qt/mkspecs/features/default_post.prf \
		/usr/share/qt/mkspecs/features/warn_on.prf \
		/usr/share/qt/mkspecs/features/qt.prf \
		/usr/share/qt/mkspecs/features/unix/opengl.prf \
		/usr/share/qt/mkspecs/features/unix/thread.prf \
		/usr/share/qt/mkspecs/features/moc.prf \
		/usr/share/qt/mkspecs/features/resources.prf \
		/usr/share/qt/mkspecs/features/uic.prf \
		/usr/lib/libQtCore.prl \
		/usr/lib/libQtGui.prl \
		/usr/lib/libQtOpenGL.prl
	$(QMAKE) -unix -o Makefile project.pro
/usr/share/qt/mkspecs/qconfig.pri:
/usr/share/qt/mkspecs/features/qt_config.prf:
/usr/share/qt/mkspecs/features/exclusive_builds.prf:
/usr/share/qt/mkspecs/features/default_pre.prf:
						/usr/share/qt/mkspecs/features/release.prf:
/usr/share/qt/mkspecs/features/default_post.prf:
/usr/share/qt/mkspecs/features/warn_on.prf:
/usr/share/qt/mkspecs/features/qt.prf:
/usr/share/qt/mkspecs/features/unix/opengl.prf:
/usr/share/qt/mkspecs/features/unix/thread.prf:
```

```
/usr/share/qt/mkspecs/features/moc.prf:
/usr/share/qt/mkspecs/features/resources.prf:
/usr/share/qt/mkspecs/features/uic.prf:
/usr/lib/libQtCore.prl:
/usr/lib/libQtGui.prl:
```

```
/usr/lib/libQtOpenGL.prl:
qmake:  FORCE
	@$(QMAKE) -unix -o Makefile project.pro

dist:
	@$(CHK_DIR_EXISTS) ".tmp/project1.0.0" || $(MKDIR) ".tmp/project1.0.0"
	  $(COPY_FILE) --parents $(SOURCES) $(DIST) .tmp/project1.0.0/ &&
$(COPY_FILE) --parents stackexample.hh VideoWidget.hh calibrateWidget.hh
.tmp/project1.0.0/ && $(COPY_FILE) --parents main.cc stackexample.cc
VideoWidget.cc calibrateWidget.cc .tmp/project1.0.0/ && $(COPY_FILE) --parents
stackexample.ui .tmp/project1.0.0/ && (cd `dirname .tmp/project1.0.0` && $(TAR)
project1.0.0.tar project1.0.0 && $(COMPRESS) project1.0.0.tar) && $(MOVE)
`dirname .tmp/project1.0.0`/project1.0.0.tar.gz . && $(DEL_FILE) -r .tmp/project1.0.0


yaccclean:
lexclean:
clean:compiler_clean
	-$(DEL_FILE) $(OBJECTS)
	-$(DEL_FILE) *~ core *.core
```

```
####### Sub-libraries

distclean: clean
	-$(DEL_FILE) $(TARGET)
	-$(DEL_FILE) Makefile

                        /usr/bin/moc:
```

```
	(cd "$(QTDIR)/src/tools/moc" && $(MAKE))

mocclean: compiler_moc_header_clean compiler_moc_source_clean

mocables: compiler_moc_header_make_all compiler_moc_source_make_all

compiler_moc_header_make_all: moc_stackexample.cpp moc_VideoWidget.cpp
moc_calibrateWidget.cpp
compiler_moc_header_clean:
	-$(DEL_FILE) moc_stackexample.cpp moc_VideoWidget.cpp
moc_calibrateWidget.cpp
moc_stackexample.cpp: ui_stackexample.h \
		VideoWidget.hh \
		cameraProjectorCalibration.hh \
		pointerRecognition.hh \
		displayWidget.hh \
		stackexample.hh \
		/usr/bin/moc
	/usr/bin/moc $(DEFINES) $(INCPATH) stackexample.hh -o moc_stackexample.cpp
```

```
moc_VideoWidget.cpp: VideoWidget.hh \
		/usr/bin/moc
	/usr/bin/moc $(DEFINES) $(INCPATH) VideoWidget.hh -o moc_VideoWidget.cpp

moc_calibrateWidget.cpp: calibrateWidget.hh \
		/usr/bin/moc
	/usr/bin/moc $(DEFINES) $(INCPATH) calibrateWidget.hh -o
moc_calibrateWidget.cpp

compiler_rcc_make_all:
compiler_rcc_clean:
compiler_image_collection_make_all:
compiler_image_collection_clean:
	-$(DEL_FILE) qmake_image_collection.cpp
compiler_moc_source_make_all:
compiler_moc_source_clean:
compiler_uic_make_all: ui_stackexample.h
compiler_uic_clean:
	-$(DEL_FILE) ui_stackexample.h
ui_stackexample.h: stackexample.ui \
		VideoWidget.hh
	/usr/bin/uic stackexample.ui -o ui_stackexample.h

compiler_clean: compiler_moc_header_clean compiler_rcc_clean
compiler_image_collection_clean compiler_moc_source_clean compiler_uic_clean

####### Compile
```

```
main.o: main.cc

stackexample.o: stackexample.cc stackexample.hh \
		ui_stackexample.h \
		VideoWidget.hh \
		cameraProjectorCalibration.hh \
		pointerRecognition.hh \
		displayWidget.hh \
		calibrateWidget.hh \
		cameraProjectorCalibration.cc \
		pointerRecognition.cc

VideoWidget.o: VideoWidget.cc VideoWidget.hh

calibrateWidget.o: calibrateWidget.cc calibrateWidget.hh

moc_stackexample.o: moc_stackexample.cpp

moc_VideoWidget.o: moc_VideoWidget.cpp

moc_calibrateWidget.o: moc_calibrateWidget.cpp

####### Install

install:   FORCE

uninstall:   FORCE

FORCE:
```

```
######################################################################
# Automatically generated by qmake (2.00a) Mon Sep 4 12:03:14 2006
######################################################################

TEMPLATE = app
TARGET +=
DEPENDPATH += .
INCLUDEPATH += .
LIBS += -lmimas
```

```
QT += opengl

# Input
HEADERS += stackexample.hh VideoWidget.hh calibrateWidget.hh
FORMS += stackexample.ui
SOURCES += main.cc stackexample.cc VideoWidget.cc calibrateWidget.cc
```