2005

Shape based stereovision assistance in rehabilitation robotics

Michael Ulrich Jurczyk
University of South Florida

Shape Based Stereovision Assistance in Rehabilitation Robotics

by

Michael Ulrich Jurczyk

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Mechanical Engineering
Department of Mechanical Engineering
College of Engineering
University of South Florida

Major Professor: Rajiv Dubey, Ph.D.
Shuh-Jing Ying, Ph.D.
Ashok Kumar, Ph.D.

Date of Approval:
March 23, 2005

Keywords: intrinsic camera parameter, extrinsic camera parameter, stereopsis, telerobotics, scaling functions

Dedication

This thesis is dedicated to my parents, Dieter and Ursula Jurczyk.

Diese Diplomarbeit ist meinen Eltern, Dieter und Ursula Jurczyk, gewidmet.

Acknowledgements

I would like to thank Dr. Rajiv Dubey for giving me the opportunity to work on this project for my MSME thesis. Thank you for your support, patience and this wonderful opportunity. It was a pleasure to work on this topic, which really interested me. I would also like to thank Dr. Shuh-Jing Ying for his support and constant corrections and guidance throughout the process. I would like to thank Dr. Ashok Kumar for taking the time to review and give feedback about my work. I also want to thank Sue Britten for her constant support and encouragement; also for all the help she gave me in so many ways that are impossible too numerous to describe. I also have to thank Dr. Ronald Mann for his support especially in the beginning phases of the thesis. Thank you also to all the people in the mechanical engineering department, which I cannot list. I also have to thank my colleagues in Dr. Dubey's group – you know who you are. I would also like to thank Hamza Begdouri, MSME, for his support and his friendship. Also, I would like to thank Christine Woodard for being so understanding and patient. Finally, I really want to thank my parents and family for their endless support and patience. Thank you to everyone!

Table of Contents

List of Tables

List of Figures

Shape Based Stereovision Assistance in Rehabilitation Robotics

Michael Ulrich Jurczyk

ABSTRACT

A graphical user interface program was created along with shape models, which allow persons with disabilities to set up a stereovision system with off-the-shelf hardware and detect objects of interest, which can be picked up using a sensor assisted telerobotic manipulator.

A Hitachi KP-D50 CCD camera and an Imaging Source CCD camera were used along with two Imaging Source DFG/LC1 frame grabbers to set up a stereovision system. In order to use the stereovision system, the two main problems of correspondence and reconstruction are solved using subroutines of the program created for this work.

The user interface allows the user to easily perform the intrinsic and extrinsic camera calibration required for stereovision, by following a few basic steps incorporated into the user interface program, which are described in this thesis. A calibration table required for these tasks can also be easily created using the program.

In order to detect the object of interest, shape models, created by the user interface program, are used to solve the correspondence problem of stereovision. The correspondence problem is that of locating corresponding

points in the left eye and the right eye, which are necessary to perform the calculations to obtain the location of the object of interest with respect to the end-effector. The shape models created for some commonly available items such as a doorknob and a door handle are included in the program and used to test the stereovision system.

As expected, the error of detection decreases as the stereo system is moved closer to the object of interest in the x-, y- and z-position. The user interface created allows the user to detect the object of interest, approach it with a telerobotic manipulator using velocity-scaled assist functions, which scale the velocity of the telerobotic manipulator perpendicular to the desired movement, and finally grasp the object. The user can then move the object to any desired location.

Chapter 1 – Introduction

1.1 Motivation

Performance of tasks ranging from opening doors to retrieving documents from shelves require little thought and effort for most people. To persons with disabilities, however, these every-day tasks oftentimes require a great deal of effort, or are even impossible without the aid of other people or devices.

The Americans with Disabilities Act of 1990 defines disability as a substantial limitation in a major life activity. If this definition is applied to the 1997 Census performed by the National Census Bureau, approximately 19.7 % of the American population is considered to live with a disability [1]. Of this fifth of the population, 12.3 % of the population was considered to have a severe disability, while 3.8 % of the total population requires personal assistance. This means that just over 10 million persons in the United States are considered to be severely disabled and dependent on personal assistance [2]. This number is astronomical, considering that there are no commercially available robotic systems that incorporate sensor assistance to aid persons with disabilities.

1.2 Vision in Robotics

Figure 1-1 shows an altered Manus Manipulator [3] equipped with a single camera attached to the end of its end-effector. Currently, there is no vision

system available on the market that is used for rehabilitation robotic applications. All vision systems that actually are used in industry and research require an expensive software package as well as a person capable of programming. For persons with disabilities, however, this means that it is simply not possible to use vision assistance for control of their manipulator, as this would cost too much. The main function of the vision system is to detect common objects such as doorknobs and, upon confirmation, perform a simple function like turn the knob.

Figure 1-1: RWTH Manus Manipulator

It is extremely interesting to note that, despite extensive research in stereovision itself, there is no commercially available stereovision system for use with robotic manipulators or even for personal use. Hence, this thesis addresses this gap in availability in sensor equipped robotic manipulators.

Since stereovision is not a new concept, a few points should be made. There are many different techniques for detecting objects. These range from applying stickers to objects of interest to detecting motion and using this data to detect an object. However, there is not a program available, which incorporates shape models, as described and developed in this thesis, to detect every-day objects.

1.3 Past Work

There has been some work done in the Rehabilitation and Engineering Program at the University of South Florida in Tampa, Florida. Benjamin E. Fritz [4] has set up a telerobotic system with haptic input to enhance manipulation capabilities of persons with disabilities. In his work, Benjamin Fritz used a robotic manipulator from Robotics Research Corporation, equipped with a single camera and a laser-range finder to detect an object of known circular shape. Once the object was detected, a set of assist functions, described briefly in chapter 6 of this thesis, were employed to scale the velocity in the x-, y-, and z-directions such that the manipulator could close in on the object of interest, while limiting the movement capabilities of the user, such that tremors in input could be limited, providing a smooth trajectory towards the object to be manipulated. While these assist functions help the user in controlling the robotic manipulator, the simple object detection makes the system impractical for daily use, since it is impractical to only be able to manipulate objects with a circular area. Additionally, it is not always possible to determine the area of the object of interest, and it would be

nice to be able to detect objects of similar geometries with varying sizes. This thesis solves these issues, and they are listed in the following section.

Wentao Yu "addresses the development of a telemanipulation system using intelligent mapping from a haptic user interface to a remote manipulator to assist in maximizing the manipulation capabilities of persons with disabilities. This mapping, referred to as assistance function, is determined on the basis of environmental model or real-time sensory data to guide the motion of a telerobotic manipulator while performing a given task. Human input is enhanced rather than superseded by the computer. This is particularly useful when the user has restricted range of movements due to certain disabilities such as muscular dystrophy, a stroke, or any form of pathological tremor" [5]. Wentao Yu's work employed the stereovision system created for this thesis to test a Hidden Markov Model, described in his dissertation, to assist persons with disabilities in enhancing manipulation capabilities when employing telerobotics.

1.4 Thesis Objectives

There are five main objectives of this research work, which are also extensively described herein. These are as follows:

1. Set up a stereovision system from commercially available products at a low cost. In order to do this, off-the-shelf cameras and computer equipment should be used and connected properly. This is important since, as discussed before, there are no commercially available

stereovision systems for use with persons with disabilities. Hence, persons with disabilities should be able to create their own stereovision system.

2. Create a library of shape models for every day items. These shape models should be specific enough to allow for proper detection of the object even in a cluttered environment, but also be broad enough to allow for the object of interest to be detected even if the object is not exactly the same object used to create the model.
3. Create a user-friendly interface to control the acquisition of images, as well as any other stereovision problems. This means that the program should allow the user to obtain images from both cameras, be able to easily obtain the intrinsic as well as the extrinsic camera parameters, and the user should also be able to create their own set of shape models without any programming knowledge.
4. Finally, the user interface should be tested along with the acquired intrinsic and extrinsic camera parameters to determine the accuracy of the software bundle created for this work. The object of interest has to be found with sufficient high accuracy, even if partially covered.
5. Scaling assist functions, as described by Benjamin Fritz [4], should be integrated into the user interface and program, allowing persons with disabilities to approach objects of interest with relative ease, even if the user has tremors in movement.

Chapter 2 – Stereovision

2.1 History of Stereovision

Stereovision systems are nothing new to science and engineering. The principles of stereovision go as far back as ancient Greece when Euclid first explained the principles of stereovision. It was Euclid who showed that the left and right eyes see different images. The brain then uses these two images and especially the discrepancies, or different position of an object relative to each eye, to produce the perception of depth.

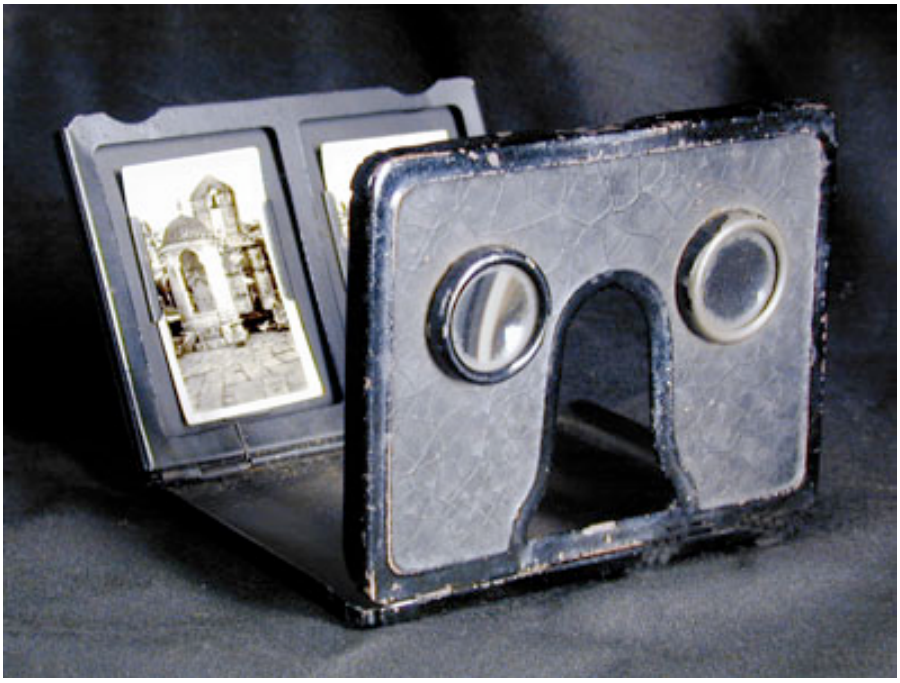

Figure 2-1: Early Stereoscopic Viewing Device [6]

In the first third of the nineteenth century, Sir Charles Wheatstone presented his theories and discoveries on stereo vision by showing two separately drawn images with slight disparities that, when viewed through his stereoscopic viewing apparatus, appeared as a three-dimensional object [7]. Up until roughly the mid twentieth century, stereovision was used mainly as a form of entertainment, where images, or photographs, which were taken with approximately 7 cm spacing, were viewed through a stereoscopic viewing device, such as that in Figure 1-1 above to produce the perception of a three-dimensional scene.

In 1959, Bela Julesz demonstrated that the brain is able to perceive 3-D information without any true perceptual depth cues. He invented the first random dot stereogram in 1959 in attempts to test stereopsis. The experiment he performed was with the random dot image and by shifting it slightly to produce a second image. When viewed in a stereoscope, the pattern appeared as a 3-D image. By surrounding the images with an identical random dot pattern they appear identical when viewed separately, but when viewed stereoscopically, the central square pattern (or whatever shape is created) will appear to float in space [8].

By the early 1980's, Marr and Poggio come up with computational models for describing depth perception in random dot images. In the present, stereovision is emerging in more and more areas such as aerial photography, medical applications, and many more areas.

Stereovision, also known as stereography, has been used in computer vision for such applications as obtaining a three-dimensional profile of a surface. A commonly known example is that of using a satellite, which flies above the earth's surface and obtains information about the earth's topography. While it is simple to set up two cameras to obtain images simultaneously, there are two main problems encountered in stereovision. These are the problems of correspondence and reconstruction. The answer to these two problems is discussed in this chapter. First, this chapter addresses the preliminary steps required to set up a stereovision system.

2.2 Hardware

The hardware used consists of two main component pairs. These components are the Hitachi KP-D50 CCD camera, an Imaging Source CCD camera and two Imaging Source DFG/LC1 frame grabbers. The technical details of these cameras can be found on the manufacturers' websites, included in the references section. The parameters required for stereovision, however, are discussed.

A schematic representation of the set-up of the vision system hardware is shown in Figure 2-2 below. The cameras are attached via a coaxial cable to the frame grabbers and assigned different port numbers so as to not cause any interference when grabbing images.

Both cameras are able to obtain images continuously in the standard NTSC video format, as well as the European PAL format. NTSC is used for this project. The image acquisition speed, however, is limited by the program, which grabs the images and then analyzes these. More about this will be discussed later on in the thesis.

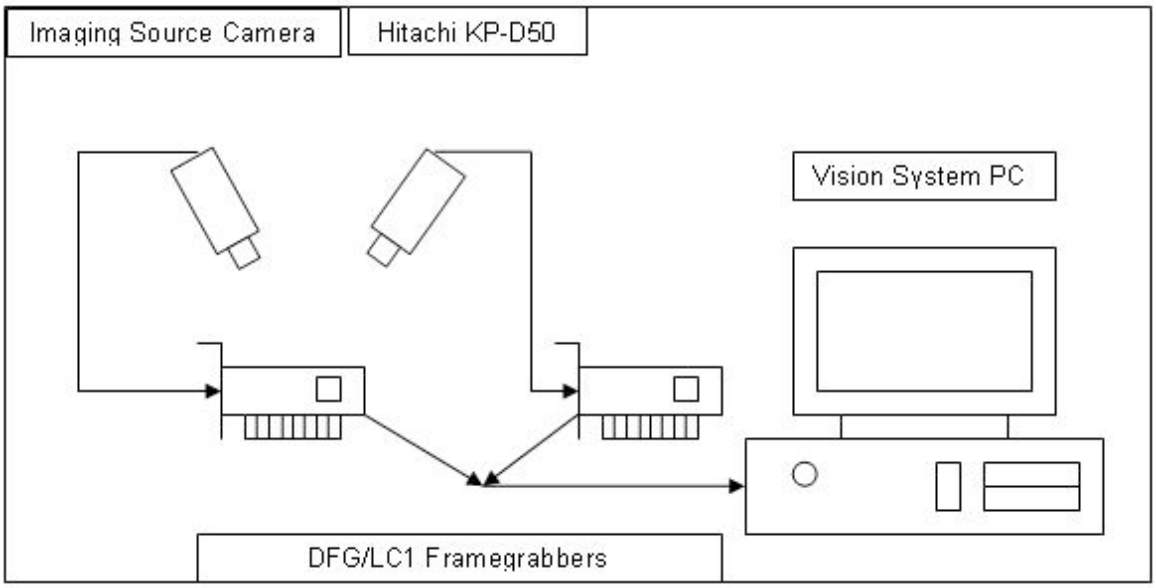

Figure 2-2: Vision System Schematic

2.3 Software

In order to create the shape models, which are used to locate the object of interest in both camera views, Halcon from the company MVTec is used. “MVTec is specialized in software solutions for machine vision applications using standard hardware,” as the company’s website states their mission [8]. The software package employed in this research work is called Halcon. While some of the tasks used in this project can be created using Halcon, it is important to

realize that this is very labor-intensive and as such not a viable option for the average user. Hence, the Shape Detection Program created as part of this project incorporates all of the required tasks in order to enable any person to use stereovision for assistance in robotic manipulation.

Halcon contains a library of close to 1000 vision operators covering applications such as factory automation, quality control, remote sensing, aerial image interpretation, medical image analysis, surveillance tasks, and many more areas. It also is able to match patterns, and perform blob analysis. These two functions are used in this research work to find objects of interest that are to be manipulated by the robotic manipulator. The various vision system algorithms created are discussed in the next chapter in much more detail.

2.4 Camera Model and Theory

In order to understand the intrinsic and extrinsic camera parameter calibration discussed in the next section, it is essential to first describe some basic equations related to optics and more specifically to cameras. The most common model of a camera is the perspective model, also known as the pinhole model, which is shown in Figure 2-3. Point O lies at the center of the camera frame and is known as the center or focus of projection. The image plane, π, contains the pixel coordinates x and y, which represent the projection of the point of interest, P, onto the image plane, giving the point p. The distance between the camera frame and the image plane is defined as the focal length, f.

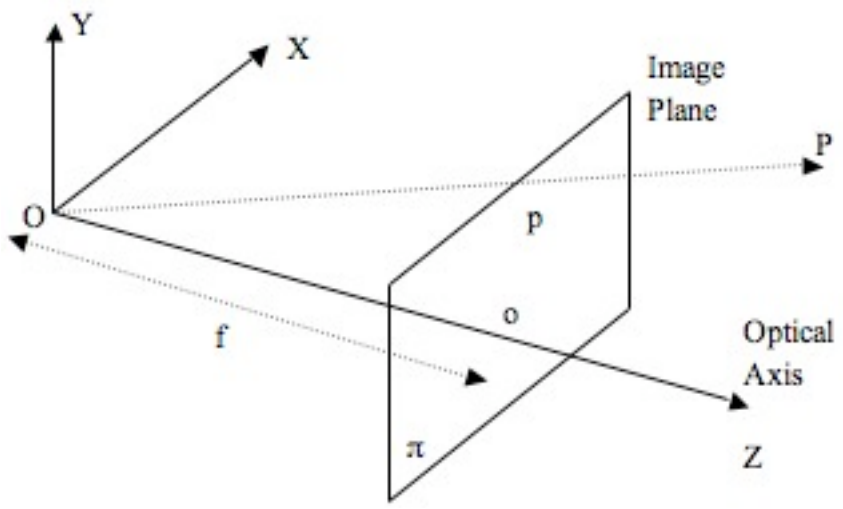

Figure 2-3: The Perspective Camera Model

From basic geometry, it can be shown that a point p=(x,y) in the image plane is related to a point P=(X,Y) in the camera plane by equation (2.1):

$$x = f\frac{X}{Z} \quad (2.1)$$
$$y = f\frac{Y}{Z}$$

In order to translate a point P_w in the world frame to a point P_c in the camera frame, a rotation matrix, denoted by R, and a translation matrix, denoted by T, is required. Using the rotation and translation matrix, one can show that the relation between the two points is given by equation (2.2):

$$P_c = R(P_w - T) \quad (2.2)$$

where the rotation matrix is defined by equation (2.3) as:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.3)$$

Neglecting geometric distortions derived from optics, the coordinates of an image point in pixel units (x_{im}, y_{im}) can be related to the same point (x, y) in the camera reference frame by equation (2.4):

$$\begin{aligned} x &= -(x_{im} - o_x)s_x \\ y &= -(y_{im} - o_y)s_y \end{aligned} \quad (2.4)$$

where (o_x, o_y) are the coordinates in pixels of the image center, and (s_x, s_y) is the effective size of the pixel in the horizontal and vertical direction respectively.

By combining equation (2.2) and equation (2.4) with equation (2.1) it can be shown that the linear version of the perspective projection equation can be written as equation (2.5):

$$\begin{aligned} -(x_{im} - o_x)s_x &= f\frac{R_1^T(P_w - T)}{R_3^T(P_w - T)} \\ -(y_{im} - o_y)s_y &= f\frac{R_2^T(P_w - T)}{R_3^T(P_w - T)} \end{aligned} \quad (2.5)$$

where R_1, R_2, and R_3 denote a 3-D vector formed by the 1st, 2nd, and 3rd row, respectively, of equation (2.3). Equation (2.5) directly links the pixel coordinates of an image point with the world coordinates of the corresponding 3-D point, without explicit reference to the camera reference frame needed. However, the intrinsic and extrinsic camera parameters are not independent of each other. To facilitate calculation of intrinsic and extrinsic camera parameters, equation (2.5) can be written as a matrix product of the intrinsic and extrinsic camera parameters, yielding equations (2.6) and (2.7), which denote the intrinsic and extrinsic camera parameters, respectively.

$$M_{int} = \begin{bmatrix} -\frac{f}{s_x} & 0 & o_x \\ 0 & -\frac{f}{s_y} & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.6)$$

$$M_{ext} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \quad (2.7)$$

Now that the intrinsic and extrinsic matrices and parameters are defined, it is necessary to accurately obtain these values. In addition to the extrinsic matrix denoted by equation (2.7) above, there is also a translation matrix, which defines the translation between the cameras and the end-effector. The intrinsic camera parameters are obtained by performing the intrinsic camera calibration, while the extrinsic camera parameters are obtained via the extrinsic camera calibration. Both of the calibration processes, along with the results, are described in the following sections.

2.5 Intrinsic Camera Parameters

In order to use stereovision, the intrinsic camera parameters need to be determined. The intrinsic camera parameters characterize the transformation mapping of an image point from camera to pixel coordinates for each individual camera. While the manufacturer of a camera generally gives a standardized list of these parameters, it is found that these only resemble an idealized average for these values. Hence, it is important to determine the intrinsic characteristics of

each camera accurately, as these parameters are required to obtain the three-dimensional location of a point in terms of the camera coordinate system, from which the point can then be transformed into world coordinates. The coordinate systems assigned to the setup are described at this point in Figure 2-4 below. This assignment is consistent throughout this project and should be referred to when in doubt of an equation at any point. The coordinate systems should also be assigned as such to any robotic manipulator used with this project's program. While it seems basic to assign coordinate systems, it actually is essential to assign them in the correct manner, as this can make or break the use of stereovision.

The intrinsic camera parameters are the focal length, the scale factors, the center coordinates of the image, and the distortion coefficient. The scale factors give information on the distance of a pixel on the CCD-chip in the x- and y-direction. The center coordinates of the image are the pixel coordinates of the center of projection, which does not necessarily have to be the theoretical center location, while the distortion coefficient is used to correct the image from warping, which is an unpleasant side effect of optics, in order to obtain accurate data. Finally, the image height and width are also useful to know, though not essential.

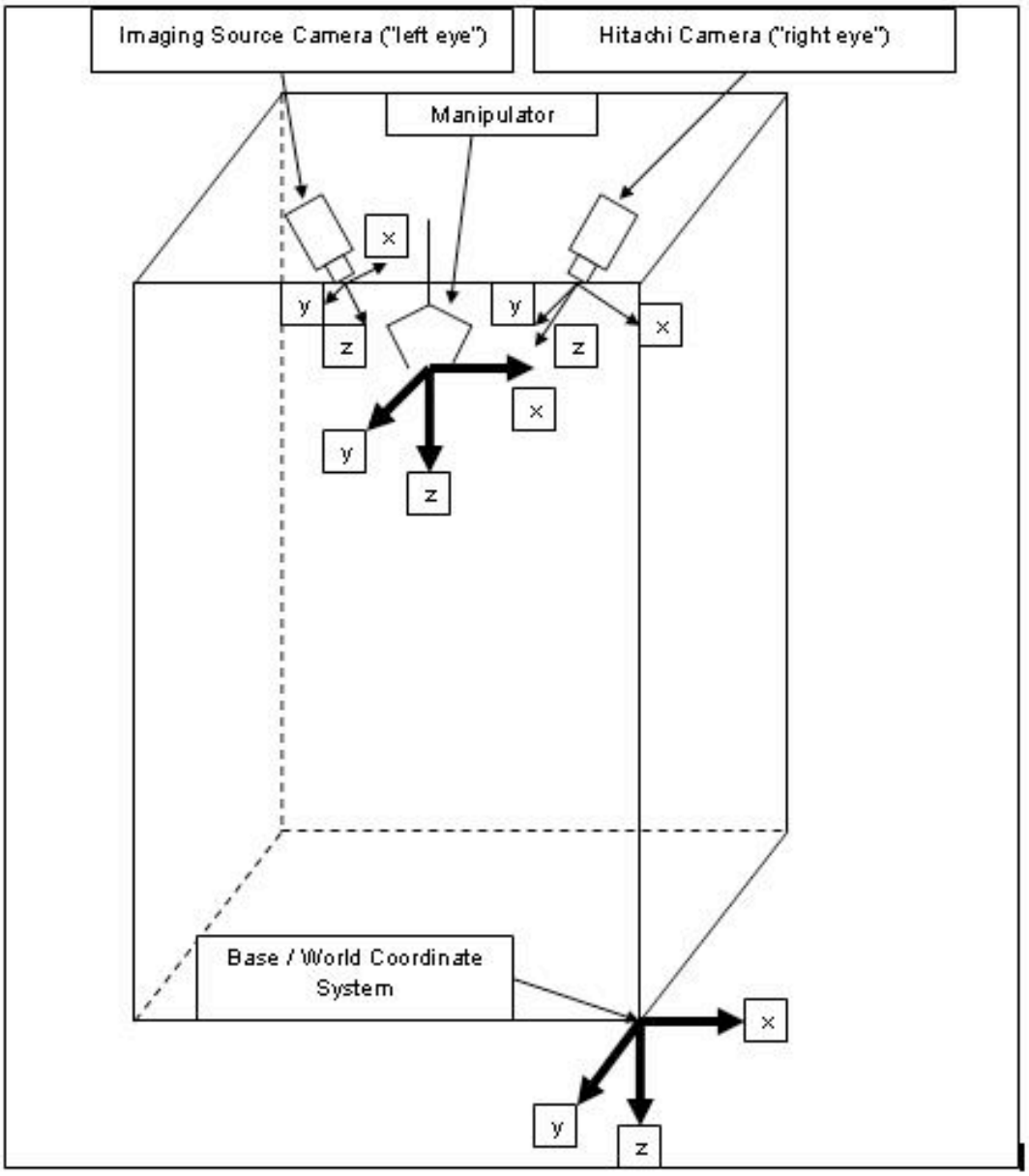

Figure 2-4: Coordinate System Assignment

The internal camera calibration is performed using a calibration table, shown in Figure 2-4, with known coordinates. It can be created using the user interface created as part of this project, which is further discussed in chapter 5.

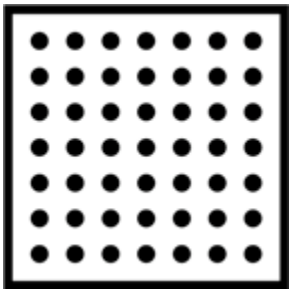

Figure 2-5: Calibration Table

The calibration table consists of 49 circles separated by equal distances. The radius of each circle is constant and known. The border dimensions of the table are also known. Since all of these parameters are known, while the distance to the calibration table is unknown, several images of the calibration table have to be taken by each camera in order to properly and accurately estimate the intrinsic parameters of the cameras used. This is called a multi-image calibration process.

For this multi-image calibration, the known 3D points of the circles and the border of the calibration table are projected into the image plane. The sum of the squared distances between the projections and the corresponding image points (x_{im}, y_{im}, f) is minimized until the minimization converges. The equations described in the previous section are used for this procedure. From this, the intrinsic camera parameters can be obtained.

For the intrinsic parameter calibration, 13 images at greatly varying locations were taken. It is important to ensure that the images of the calibration table are shown at different angles and sizes, so that all of the intrinsic parameters can be assessed with sufficient accuracy. All features of the calibration table have to be

visible, including the border. Additionally, initial values of the intrinsic parameters should be known. After performing the internal camera calibration using the subroutine of the user interface, described in chapter 5, and running iterations using the calculated values for the intrinsic parameters, the final intrinsic camera parameters are summarized in Table 2-1 for the Imaging Source Camera, and Table 2-2 for the Hitachi camera below.

Table 2-1: Intrinsic Camera Parameters for the Imaging Source Camera

Parameter	Value	Unit
Focal Length of Lens	0.00872367	[m]
Width of CCD chip	$1.0249*10^{-5}$	[m]
Height of CCD chip	$1.1*10^{-5}$	[m]
Image Center (x)	311.828	[pixel]
Image Center (y)	253.998	[pixel]
Radial Distortion Coefficient	-4436.43	$[m^{-2}]$
Image Width	640	[pixel]
Image Height	480	[pixel]

Table 2-2: Intrinsic Camera Parameters for the Hitachi Camera

Parameter	Value	Unit
Focal Length of Lens	0.0076443	[m]
Width of CCD chip	$1.07002*10^{-5}$	[m]
Height of CCD chip	$1.1*10^{-5}$	[m]
Image Center (x)	317.915	[pixel]
Image Center (y)	224.494	[pixel]
Radial Distortion Coefficient	-4339.86	$[m^{-2}]$
Image Width	640	[pixel]
Image Height	480	[pixel]

2.6 Extrinsic Camera Parameters

In order to successfully obtain three-dimensional coordinates from stereovision, the position of the cameras with respect to each other, as well as with respect to the manipulator must be known accurately. In order to obtain these values, a so-called hand-eye calibration must be performed. The goal of a hand-eye calibration is to accurately obtain the rotation and translation matrix of the camera with respect to the manipulator.

Analogously to the intrinsic camera calibration, the same calibration table is used for simplicity. Again, thirteen images were obtained for each camera. This time, however, it is essential to accurately know the position of the manipulator with respect to the base coordinate system for each image obtained. This can easily be done when using a robotic manipulator whose joint angles are known. The position of the calibration table should not be changed during this process, as it is also essential to know its position with respect to the base coordinate system. The hand-eye calibration can be performed by using the subroutine built into the user interface program, discussed in further detail in chapter 5. Additionally, the rotation and location of the calibration table with respect to the camera are useful as initial starting guesses. As part of the intrinsic camera calibration, the pose, namely the rotation and translation vector from the camera to the calibration table are measured, as sort of a by-product of the calibration procedure. This is possible, since the dimensions and location of the circles is known. Given a certain area of an object, there is a relation between the actual

area and the projected area, given in square pixels. These values are built into the extrinsic camera calibration subroutine.

As an initial guess for the Hitachi camera, the rotation and translation from manipulator to camera were determined as:

$$R = \begin{bmatrix} 0.819 & 0.00 & 0.574 \\ 0.00 & 1.00 & 0.00 \\ -0.547 & 0.00 & 0.819 \end{bmatrix} \quad (2.8)$$

which corresponds to a rotation of 35^{o} around the y-axis.

$$T = \begin{bmatrix} -0.130 \\ 0.010 \\ 0.046 \end{bmatrix} m \quad (2.9)$$

where R denotes the rotation matrix and T denotes the translation vector from manipulator to camera coordinate system. Similarly, for the Imaging Source camera, the initial guesses for rotation and translation are:

$$R = \begin{bmatrix} 0.819 & 0.00 & -0.574 \\ 0.00 & 1.00 & 0.00 \\ 0.547 & 0.00 & 0.819 \end{bmatrix} \quad (2.10)$$

which corresponds to a rotation of 325^{o} around the y-axis.

$$T = \begin{bmatrix} 0.130 \\ 0.010 \\ 0.050 \end{bmatrix} m \quad (2.11)$$

After performing the hand eye calibration, the following rotation and translation matrices were obtained for the Hitachi camera:

$$R = \begin{bmatrix} 0.8419 & 0.1186 & 0.5263 \\ -0.0886 & 0.99268 & -0.08202 \\ -0.5322 & 0.0224 & 0.8463 \end{bmatrix} \quad (2.12)$$

which corresponds to a rotation of 5.54°, 31.76°, and 351.98° around the x-, y-, and z-axis, respectively.

$$T = \begin{bmatrix} -0.1685 \\ 0.0439 \\ -0.0187 \end{bmatrix} \text{m} \quad (2.13)$$

and for the Imaging Source camera the following was obtained:

$$R = \begin{bmatrix} 0.8419 & 0.0423 & -0.5874 \\ 0.03142 & 0.9929 & 0.11473 \\ 0.5881 & -0.1111 & 0.8011 \end{bmatrix} \quad (2.14)$$

which corresponds to a rotation of the camera of 351.85°, 325.097°, and 359.71° around the x-, y-, and z-axis, respectively.

$$T = \begin{bmatrix} 0.1261 \\ 0.0747 \\ 0.05689 \end{bmatrix} \text{m} \quad (2.15)$$

As already mentioned, these values are very important in accurately determining the three-dimensional position of the object of interest in the manipulator coordinate system and any small change in either the position or the translation of the camera with respect to the end-effector will be detrimental to the accuracy of the stereo vision reconstruction process. Table 2-3 below gives a summary of the extrinsic camera parameters. It shows both the initial guesses, as well as the converged values.

Table 2-3: Extrinsic Camera Parameters

	Hitachi Initial	Hitachi Final	Imaging Source Initial	Imaging Source Final
X	-0.13m	-0.169m	0.13m	0.126m
Y	0.01m	0.044m	0.01m	0.075m
Z	0.046m	-0.019m	0.05m	0.057m
Rot(X)	0^o	5.54^o	0^o	351.85^o
Rot(Y)	35^o	31.76^o	325^o	325.1^o
Rot(Z)	0^o	351.98^o	0^o	359.71^o

Chapter 3 - Shape Models

3.1 Introduction

The purpose of this chapter is to describe the vision assist functions created for use in this thesis. Since a large number of these so-called shape models was created, this chapter only describes the creation of one of these models in detail, as otherwise this chapter would exceed an acceptable amount of space. The other shape models created, however, are summarized at the end of the chapter with the main specifications required to reproduce the shape model effectively.

A shape model is a file, which contains the information required by the control system to locate the object of interest. The shape model also provides the user with important information on the location and orientation of the object in the field of vision in form of output parameters. All of this information is described in great detail within this chapter.

3.2 Background of Vision

Theoretically, a vision algorithm can be created for any object found in this world. However, this would prove impractical, as the amount of memory required for this task would exceed any computing power known to date. The purpose of the vision algorithms is to provide the user with a means of locating a desired

object in the field of view. Since there are practically no two objects that are exactly alike in this world, the vision algorithm should be flexible enough to locate the object of interest even if it does not match the object that was used to create the vision shape model in every detail.

If one looks at a round doorknob, for example, one will notice that there might be signs of wear and tear from touching the knob. The doorknob might also be discolored due to dirt or the paint being chipped off the surface. However, when a human being looks at the door, that person still recognizes the doorknob as a doorknob. This is because humans remember and recognize objects based on many different factors. These factors include, but are not limited to the following:

1. Shape of an object
2. Size of an object
3. Color of an object
4. Location and rotation of an object within the field of vision
5. Other distinguishing features

Some remarks should be made to the four basic factors that contribute to the detection of objects. The shape of an object is the most basic way humans compare what they see with their memory and used to determine the function of the object. A person who sees a round doorknob, for example, knows immediately that this object is used to open a door. The action required after grasping the object is to turn the knob in a certain direction followed by a pulling or pushing action in order to be able to open the door.

The size of an object also helps distinguish it from other items that might be in the field of vision. If an object is larger than another object located next to the first, the person observing this knows that the larger object will most likely be heavier than the smaller object. Hence, when manipulating this specific item, the person will know to employ more force to successfully complete the desired task.

While the shape and size of an object are considered primary distinguishing features, the color of an object is considered secondary. A plain round doorknob, for example, can be found in many different colors ranging from a matt bronze finish to a glossy golden finish. Since color does not affect the purpose of the object or desired action, it is not used for the creation of vision shape models.

The location of an object in the field of vision is an important factor used by humans. In order to be able to manipulate any object, one has to know where the object is located with respect to the hand. This problem of hand-eye calibration, which is required for the vision shape models to be functional, is discussed at the end of the chapter in detail. The vision algorithms are required to locate the object within the field of vision and relate this position to the actual position with respect to “the hand,” which is the gripper mounted at the end of the end-effector.

The last factor listed above encompasses a wide variety of features. The orientation of an object might be of importance when trying to distinguish an object. If one looks at a round doorknob, for example, its appearance changes whether it is viewed from the side or from up-front. A doorknob might have a

keyhole, or it might not. This feature can change the action the person is required to do. Should there be a keyhole, it is very likely that a key is required to be able to turn the doorknob. The absence of a keyhole indicates that the knob can be turned at any time. These secondary distinguishing features are not considered for the purposes of this research work, as the gripper mounted at the end of the end-effector does not have the capability to use the information collected to perform the required function. Also, the library of vision algorithms would increase dramatically in size.

3.3 Creation of a Shape Model

The goal of a shape model is to provide the stereovision with a means of automatic detection of the object of interest, as well as obtaining the location of the object in the field of vision. Additionally, the shape model is required to have enough distinguishing features such as to minimize the occurrence of erroneous object detections, while still keeping the shape model general enough to be able to detect similar objects that only possess minor differences. All of this has to be taken into account during the creation of a shape model and the details are now discussed.

3.3.1 The Original Image of the Object of Interest

In order to create the vision shape models, the subroutine Shape Model Creation, part of the User Interface, was employed. The most important

functions and variables required for the creation of a shape model are described. The first objective of the user is to obtain an image of the object for which the shape model is to be created. Ideally, this image should be acquired with the same camera that will be used later on in order to avoid anything that might alter the image taken. Things that can affect the acquisition of an image can include such things as the focus or lighting of the camera, as well as any built-in distortions due to manufacturing errors. While these factors are not necessarily "life-threatening" to the image or the creation of the shape model, an image with excellent resolution and detail should be captured, such that the distinguishing features required for the shape model are shown. One of the original images used can be seen in Figure 3-1 below. The shape model to be created is for the object with the cross. The desired output coordinates for this object is the center of the circle.

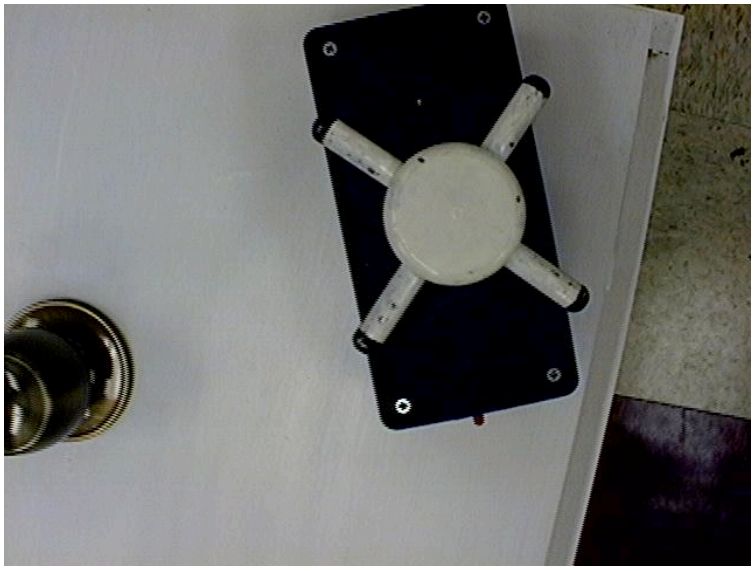

Figure 3-1: Original Image of Cross Object

3.3.2 Region of Interest

In order to create the shape model for this object, or any other object, a region of interest including the object of interest must be created. This is done so that the object, for which the shape model is to be created, can be identified much more easily. Hence, the first step, after acquiring the image, is to determine a suitable area of interest. If one were to overlay the region of interest over the original image, it would look like Figure 3-2 below.

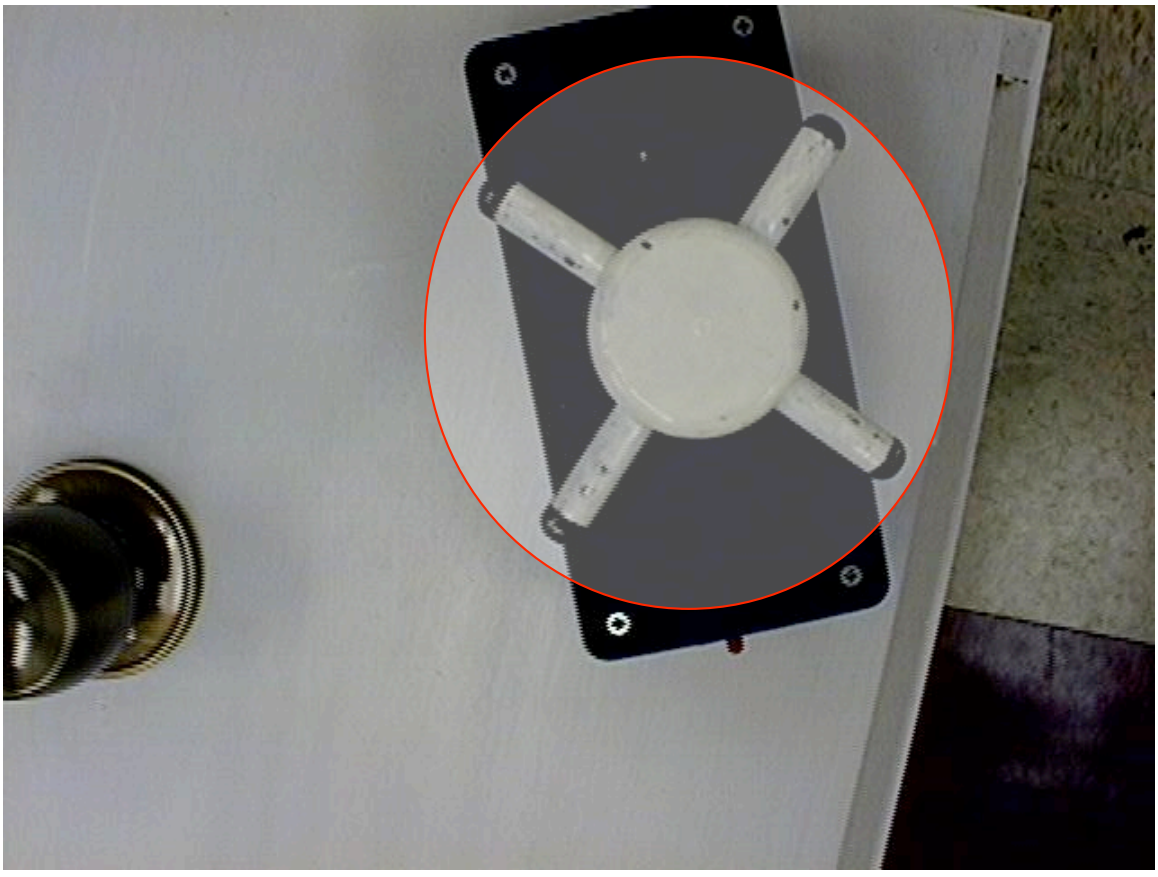

Figure 3-2: Region of Interest

In the case of Figure 3-1, it was decided that a circular region around the approximate center coordinates be created. In order to do this, the center

coordinates along with the desired radius in terms of pixels must be determined. This is done with any available photo software, such as Photoshop. The pixel coordinates of the center are then recorded in terms of row and column, as well as the number of pixels required for the radius. As an alternative, depending on the object, the region of interest can also be a rectangle, or any other geometric shape. These details are listed in the end of the chapter summarizing the creation of the various shape models.

3.3.3 Contrast Threshold and Pyramids

The easiest way to determine the characteristics of an object is to use a certain contrast threshold value. By choosing just the right value of contrast, it is possible to eliminate any undesired features, such as reflections or dirt, while using only the most important distinguishing features to create the shape model. This way, it is ensured that the object of interest will be detected in the future, even though the lighting conditions might be altered, or even if part of the object is occluded. The effect of choosing a different number for the contrast can be seen in Figure 3-3. This figure shows a contrast of 10, 80 and 100, from top to bottom. If the contrast chosen is too low, one can see that too many points are included, as even natural discolorations are interpreted as a distinguishing feature of the object of interest. However, when the contrast chosen is too high, hardly any points are considered to be of any significance. Hence, just the contrast has to be chosen such that the most important features are

distinguished. The values used for the various shape models are listed at the end of this chapter, along with other important information required to recreate these shape models.

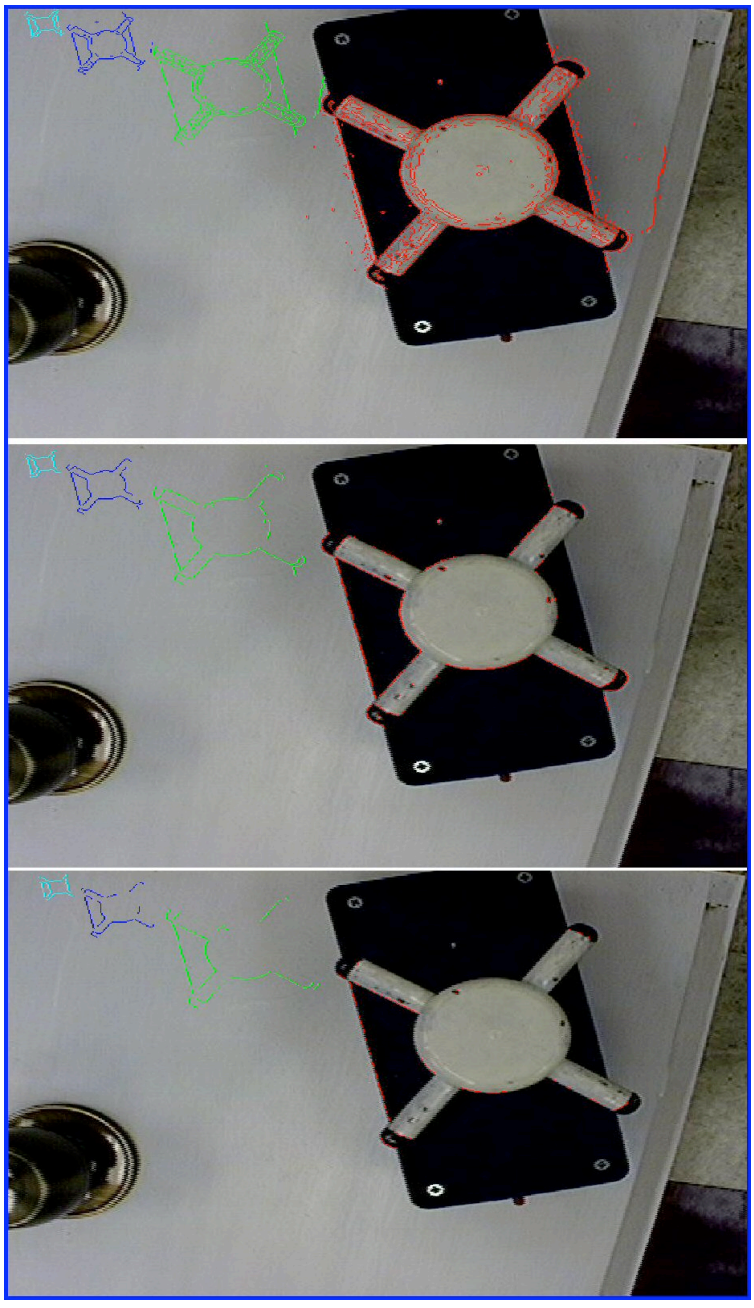

Figure 3-3: Threshold of 10, 80, and 100, Respectively (Top to Bottom)

Figure 3-3 not only shows the effect of choosing different contrast values, but also shows the effect that four different so-called pyramids have. A pyramid, in this case, is simply a scaled down model of the original image. By using different pyramid levels, the process of locating the object of interest in the future is simplified significantly. The shape model created not only stores the most important features of the object for the original size used to create the shape model, but also stores information for various scaled down models. The first pyramid is the original size of the object, while the second pyramid level is one half, the third one third, and the fourth one fourth of the original image. Theoretically, it is possible to store information on ten different pyramid levels, but this is impractical as it takes up too much memory and the anything beyond the fourth pyramid level is too small anyway. Hence, four pyramid levels are used for the creation of all shape models.

3.3.4 Other Parameters Required for the Shape Model

While the region of interest and the contrast of the desired object of interest is of great importance, other parameters have to be specified for inclusion in the shape model file. The inclusion of these parameters is not necessary, but it will speed up the process of locating the object of interest within the field of vision. It is nearly impossible that the object of interest will be rotated exactly as it was in the original image used to create the shape model. Additionally, it is equally unlikely that the object will have the exact same apparent dimensions.

3.4 Detection of Shape Models

Once a shape model has been created for an object, or a class of objects with similar geometrical characteristics, the shape model can be called from the user interface program to locate said object in the field of vision of the stereo cameras. When the shape model has successfully been matched with the object of interest, the pixel coordinates of the center of gravity are determined and stored in memory. These coordinates are then used to calculate the three-dimensional position of the object of interest with respect to the end-effector, by using the reconstruction technique discussed in chapter 4.

Chapter 4 - Three Dimensional Reconstruction

4.1 Overview

Once the object of interest has been found, the first of the two stereovision problems is solved – that of correlation. The next and final step in three-dimensional object location is three-dimensional reconstruction.

4.2 Stereo Geometry

In order to illustrate how the position of a point is determined from two sets of pixel coordinates, it is essential to explore the geometry of a stereovision system. A simplified model of the geometry of a stereo system is shown in Figure 4-1 below and it is essential to understand this geometry before exploring the sets of equations needed for three-dimensional reconstruction of the point location.

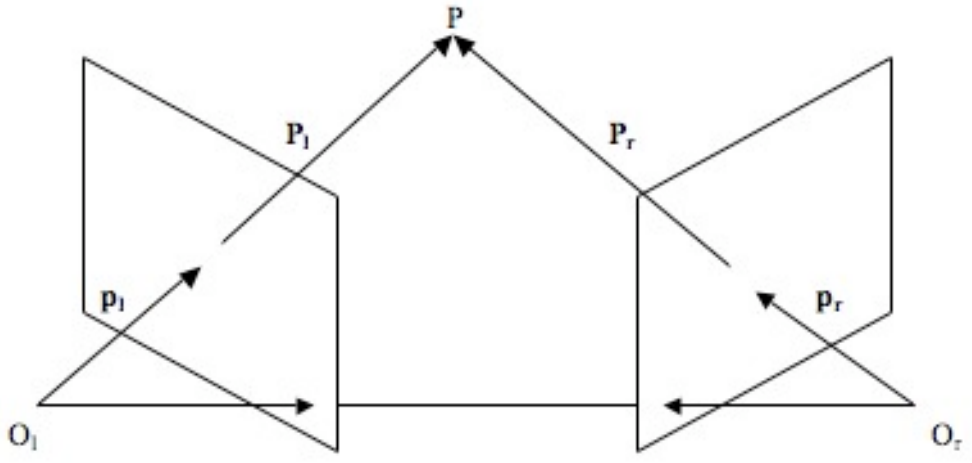

Figure 4-1: Stereo Vision Geometry

Before going any further, the nomenclature used throughout this thesis, as well as Figure 4-1 has to be clarified. The subscripts “l” and “r” denote left and right camera, respectively. The points denoted by “O” are the center coordinates of the camera. The point “P” denotes the position of the object of interest, of which the three-dimensional reconstruction is desired. The vector “p” is the unit vector of “P”, which denotes the vector from the center coordinate of the camera to the point of interest. If T is defined as the translation vector between O_l and O_r, and R denotes a rotation matrix, it can be shown, that given a point P in space, the relation between P_l and P_r is given by equation (4.1) as:

$$P_r = R(P_l - T) \quad (4.1)$$

It should be noted at this point, that the theory for the visionsystem is superbly described by Trucco [9]. For detailed and extensive theory, this reference should be consulted.

4.3 Stereo Reconstruction Calculations

In order to determine the position of the point P, it is useful to do this by using one of the cameras’ coordinate systems as a reference coordinate system. The left eye camera, namely the Imaging Source camera, is chosen to be the reference frame. Hence, the right eye, or Hitachi camera, is related to the left eye by:

$$R = R_R R_L^T \quad (4.2)$$

where the subscripts R and L denote the right Eye, in this case the Hitachi camera, and the left eye, or the Imaging Source camera, respectively. Once this is done, the translation vector between the two cameras needs to be calculated. Applying the following relationship does this:

$$T = T_L - R^T T_R \quad (4.3)$$

The next step in reconstruction is to determine the unit vectors p_L and p_R by applying, for each camera, the respective values obtained from the intrinsic camera parameter calibration:

$$p_a = M_{int,a}^{-1} \begin{bmatrix} x_a \\ y_a \\ f_a \end{bmatrix} \quad (4.4)$$

where x_L and y_L are the pixel coordinates obtained from the shape model detection process and “a” is a dummy variable, which can either be “r” or “l”, depending on whether equation (4.4) is used for the right or the left eye, respectively. These values represent the pixel coordinates of the center of gravity of the object of interest. The constant f_L is the focal length obtained from the intrinsic camera calibration. $M_{int,a}$ is the intrinsic matrix is defined as follows:

$$M_{int,a} = \begin{bmatrix} -\frac{f_a}{s_{xa}} & 0 & o_{xa} \\ 0 & -\frac{f_a}{s_{ya}} & o_{ya} \\ 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

where all values are obtained from the intrinsic camera calibration. The derivation of equation (4.5) is given in chapter 2. Similarly, equations (4.4) and (4.5) can be used for the right camera to obtain the unit vector.

Point P in Figure 4-1 lies at the intersection of the two rays from O_l through p_l, and from O_r through p_r. The point O_a is known and defined by the intrinsic camera parameters o_x and o_y, with the z-poistion equal to zero, for each camera.

The ray **l** through O_l and p_l and the ray **r** through O_r and p_r can be defined as follows bye equations (4.6) and (4.7), respectively.

$$l = ap_l \quad (4.6)$$

$$r = T + bR^T p_r \quad (4.7)$$

where a and b are simply constant scalars, which are to be determined. Finally, define a vector w as an orthogonal vector to both r and l as:

$$w = c(p_l \times R^T p_r) \quad (4.8)$$

where c is again a scalar constant to be determined.

By summing the vectors equations (4.6) through (4.8), equation (4.9) can be solved for the constants a, b, and c, since equation (4.9) is a set of three equations and three unknown – namely the scalars.

$$T = ap_L - bR^T p_R + cp_L \times \left(R^T p_R\right) \quad (4.9)$$

Once the scalars are known, it is easy to determine the actual position of point P by applying one last equation. Point P lies at the intersection of rays r and l. In order to obtain the point in the manipulator coordinate system, the point,

defined by the vectors ap_l and $T + bR^T p_r$ must be transformed. This leads to the final equation (4.10):

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = ap_l + T + bR^T p_r \quad (4.10)$$

where X, Y, and Z denote the coordinates for the point P in the manipulator coordinate system. This equation will then give the final absolute coordinates of the point of interest in terms of the left eye camera system. All that needs to be done now is to transform the position of the point into any desired coordinate system, for further manipulation.

Chapter 5 - User Interface

5.1 Overview

In rehabilitation robotics, it is essential to tailor everything to the person with disabilities. As persons with disabilities often do not have the options as persons without disabilities, the user interface used for the control of the robotic manipulator should be easy to use and straightforward. Therefore, small objects were avoided if possible and everything was made to be used with default setting that were determined to work for most cases involving calibration or shape model creation.

5.2 Main User Interface

Figure 5-1 below shows the main user interface. The picture of the left eye as well as that of the right eye is shown simultaneously. Basic features such as continuous grabbing of images, saving individual images as well as a sequence of images is incorporated for any desired, non-specified function.

A drop-down menu allows the user to select from the shape models stored within the program and when done so, the absolute, as well as the pixel coordinates of the object of interest, if present, are shown to the user as long as the stop button is not employed.

More advanced features, required for the set-up of a stereovision system, are accessed via the program's menu. These include intrinsic camera parameter calibration, hand-eye, or extrinsic parameter calibration, calibration table creation, and of course shape model creation. These features were designed and used throughout this project, and optimized such that the user should be able to perform these tasks rather easily. Should there be the need, however, the user can choose custom parameters such that these functions are optimized for the environment in which the stereovision system is to be used. One case in which this might be of use is if the stereo system is to be employed in an environment in which there is either excessive or not enough lighting available.

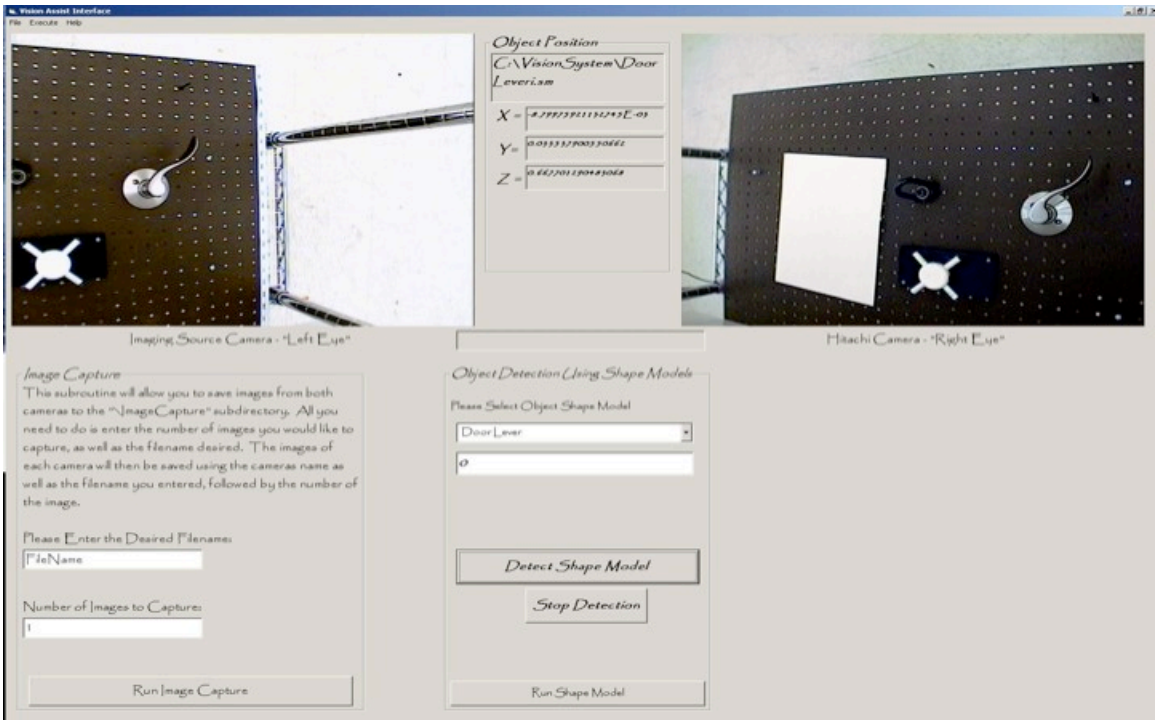

Figure 5-1: Main Graphical User Interface

5.3 Calibration Table Creation

In order to perform the intrinsic and extrinsic camera parameter calibration, a calibration table is needed. For convenience, this calibration table can be created using the “Calibration Table“ subroutine, which can be accessed from the main user interface. The window that is opened is shown in Figure 5-2.

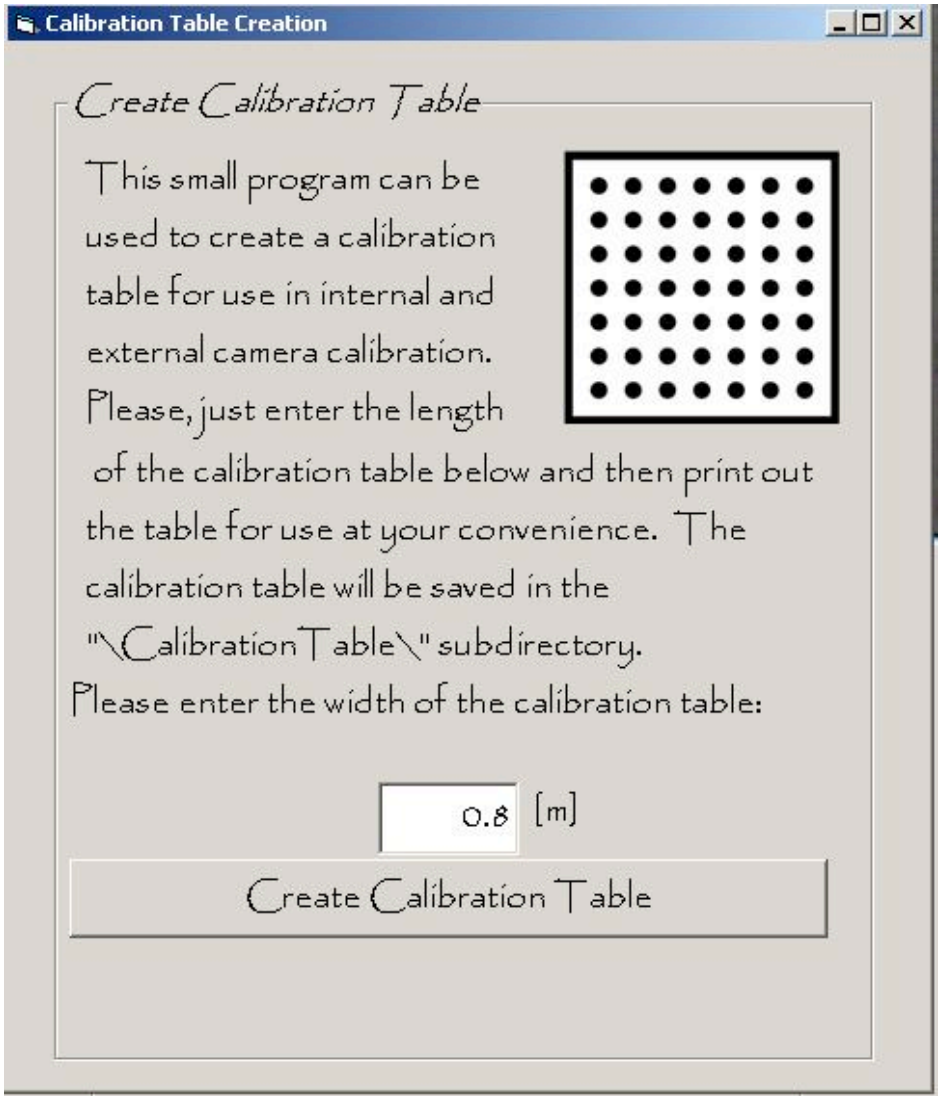

Figure 5-2: Calibration Table Creation Interface

The only input the user needs to give the subroutine is the outside dimension of the calibration table in meters. This value can be of any value, and only depends on the printer available. It is, however, recommended that the calibration table be larger than 0.2 meters and that a good printer with high precision be used, as the calibration marks are essential in obtaining the intrinsic and extrinsic camera parameters. The calibration table should also be The calibration table is saved in “Calibration Table” subdirectory of the user interface program. Its dimensions and location of the calibration marks are also stored in this subdirectory, which is later automatically called by the intrinsic and extrinsic calibration procedure.

5.4 Intrinsic Camera Parameter Calibration Subroutine

As already mentioned, obtaining accurate values for the intrinsic camera parameters is essential to successfully use stereo vision to determine the position of an object. This subroutine only has to be used when either a new camera is to be used, or if the lens of the camera has been changed, thus changing the focal properties. Usually, however, the intrinsic camera calibration needs to be done only once for a stereo system. However, it should also be performed if the accuracy of the object of interest’s location is diminished. Figure 5-3 below shows the window that is opened when this subroutine is accessed through the main user interface.

Since starting values for the intrinsic parameter calibration are needed, a set of typical intrinsic parameters is used. These values represent the wide variety of cameras available on the market and can be found in the appendix.

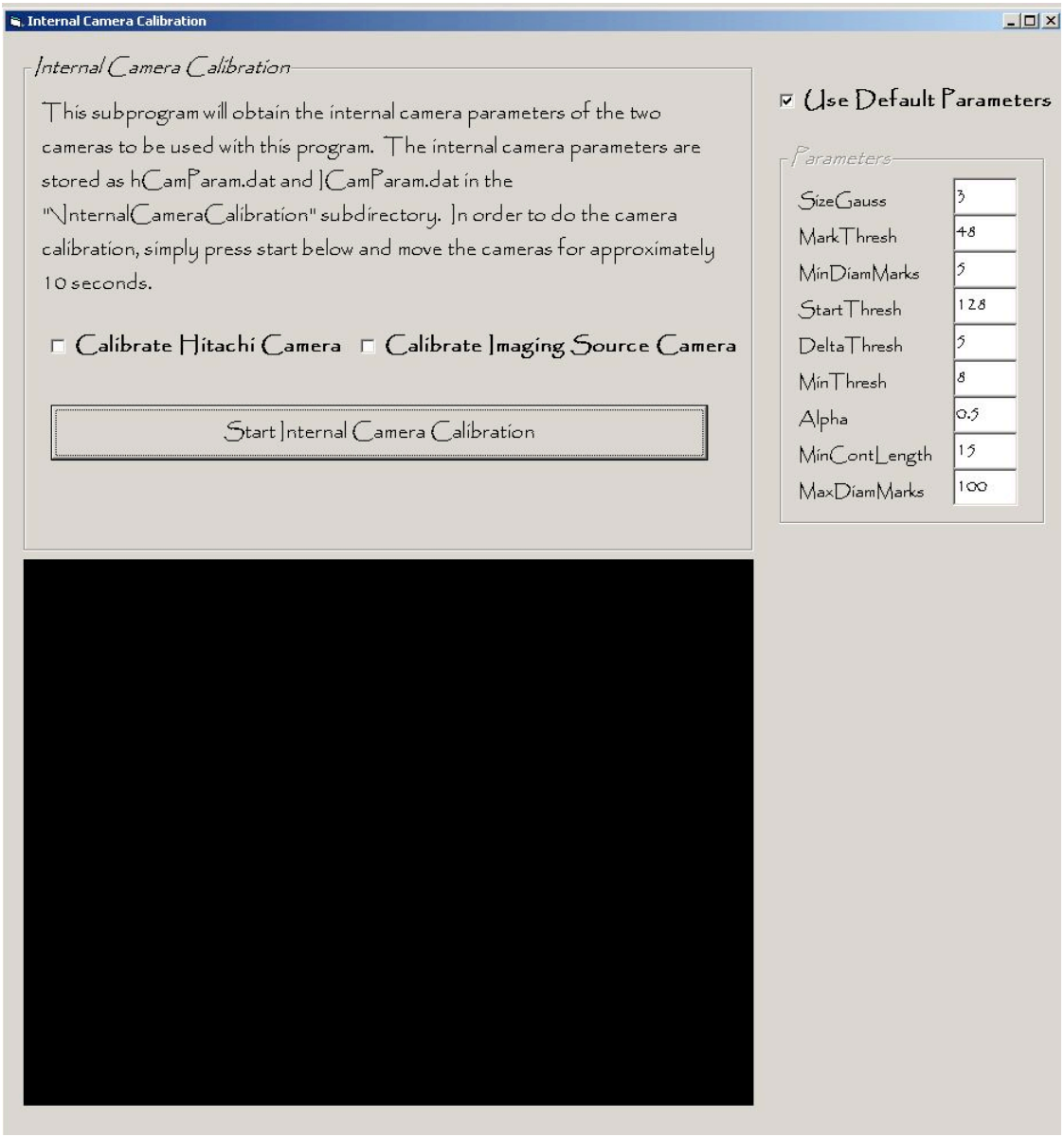

Figure 5-3: Intrinsic Parameter Calibration Interface

The camera calibration can be performed either for an individual camera or for both cameras at the same time. The general procedure of performing the calibration is as follows:

1. Select the camera for which the intrinsic parameter calibration is to be performed by checking the appropriate box.
2. Uncheck the box for the default values, if custom values for the camera calibration are to be used and enter the desired values.
3. Click the button “Obtain an Image” to obtain the first image. The subroutine automatically locates the calibration table, if present, and obtains the required information. If the calibration table is not found, an error box pops up and the image is discarded.
4. Move the camera to another position, sufficiently different from the previous position.
5. Repeat steps 3 and 4 until at least 10 images with a valid calibration table are obtained.
6. Press the “Calculate Intrinsic Parameters” button to obtain the intrinsic parameters for the camera used. The values are stored in a text file in the “InternalCameraCalibration” subdirectory of the main program.

If there is continuously a problem with finding the calibration table in the images, it is recommended to change the default settings until the calibration table is located in each image.

5.5 Extrinsic Camera Parameter Calibration Subroutine

Similar to the intrinsic camera parameter calibration, there is a subroutine integrated into the main user interface of the program. From the main menu, the user simply has to click on “Extrinsic Camera Parameter Calibration” to open the window shown in Figure 5-4. This time, however, it is recommended to perform the calibration for both cameras at the same time, not only to save time, but also in order to more accurately obtain the “hand-eye” calibration.

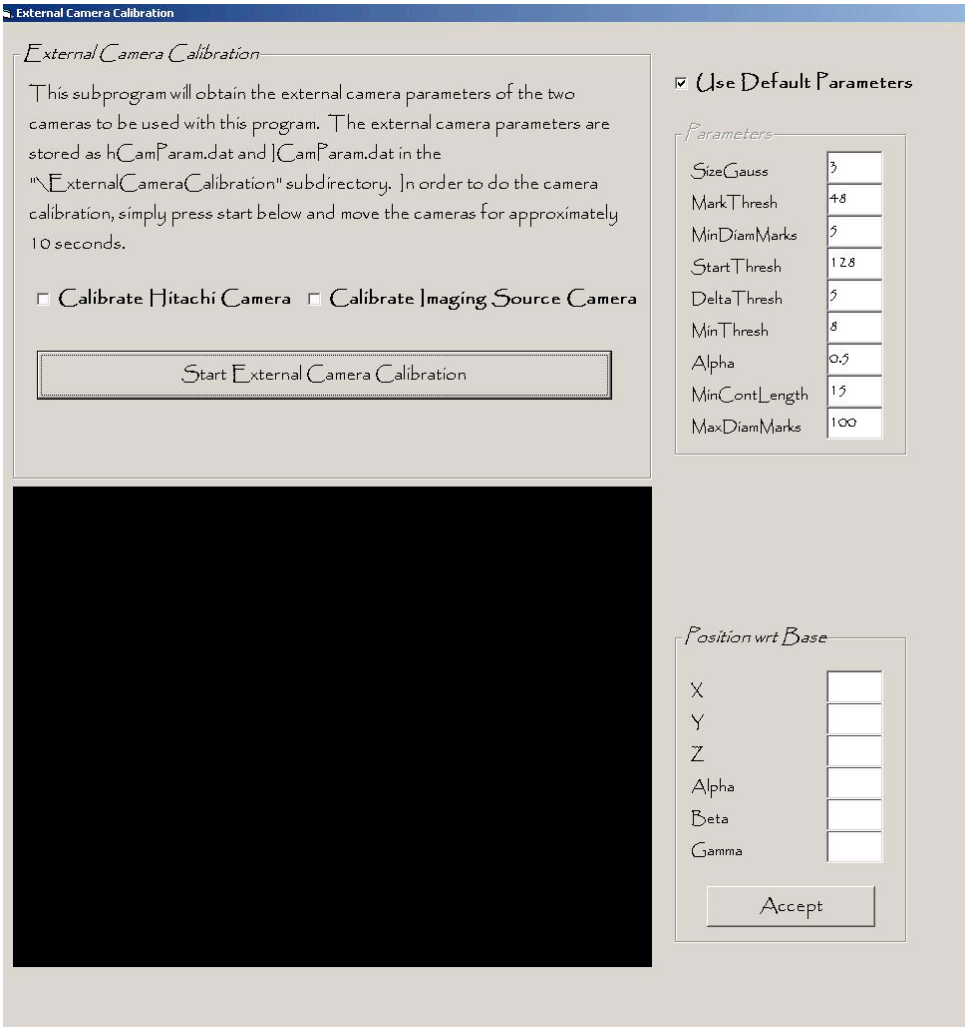

Figure 5-4: External Camera Parameter Calibration Subroutine

While performing the extrinsic camera parameter calibration, some items have to be considered. Again, the calibration table used should be visible in sufficiently different positions and has to be entirely visible in each camera. At least 10 different images should be taken, if possible 20. The position of the manipulator with respect to the base of the robotic manipulator, or set-up has to be known with high accuracy.

The general procedure for performing a successful calibration of the extrinsic parameters using this subroutine is as follows:

1. Click on “Obtain Image” in order to obtain the first image.
2. If the images of both cameras contain the calibration table, enter the measured coordinates of the manipulator with respect to the base of the robotic manipulator according to the label. 6 different values are required. These are the position of the manipulator in Cartesian coordinates, as well as the three main angles, row, pitch and yaw. It is absolutely acceptable to use 0 as a value should the manipulator not possess any rotation, but simply translation.
3. Repeat step 2 until at least 10 successful images have been obtained.
4. Click on the “Calculate Extrinsic Parameters” button to complete the extrinsic calibration.

The extrinsic calibration results are saved as a text file in the “ExternalCameraCalibration” subdirectory of the main program. The file contains six values – the three rotation angles, as well as the translational Cartesian

values of each camera with respect to the end-effector. These values are used for the reconstruction.

5.6 Shape Model Creation

Shape Models can be created using the shape model creation subroutine, which can be accessed from the Execute Menu within the Main User Interface. This will open a new window, as seen in Figure 5-5 below.

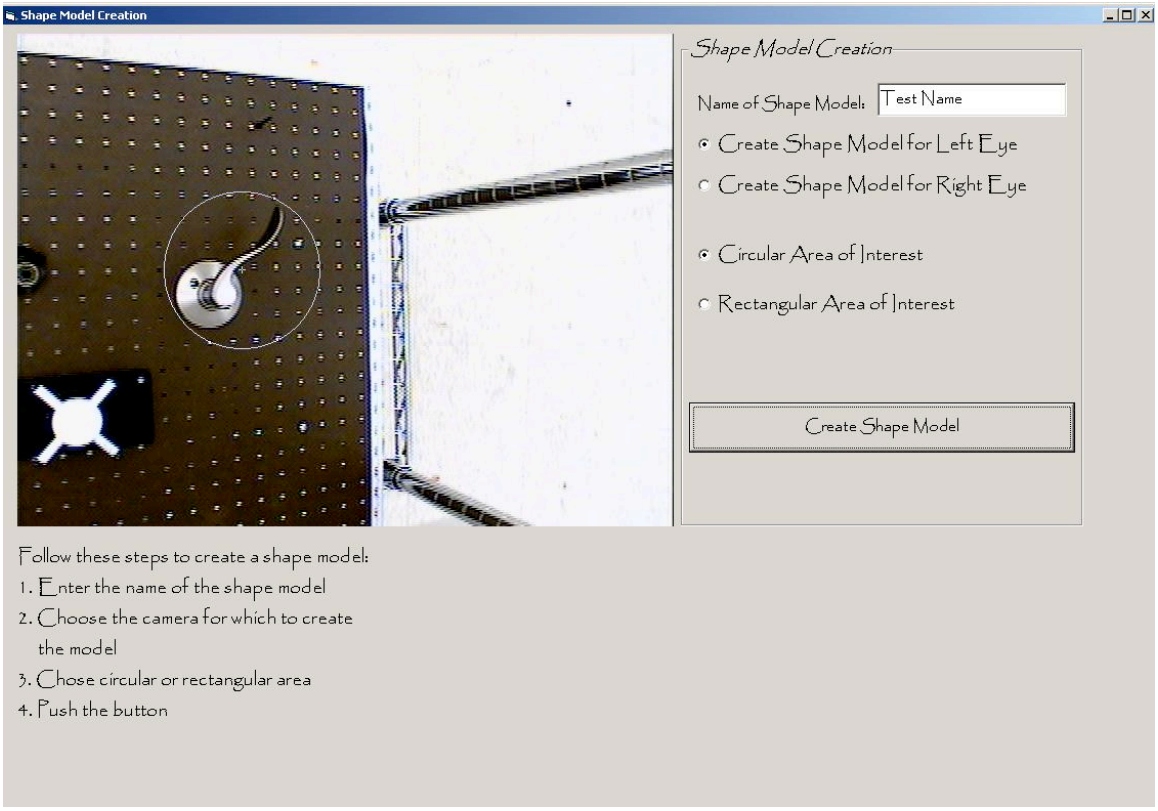

Figure 5-5: Shape Model Creation GUI

In order to create a shape model, the following steps should be followed:

1. The name of the Shape Model should be entered. This can really be anything the user wants. However, it is recommended that it be something that be related to the object.
2. The camera for which the shape model is to be created has is selected. While in theory, only one shape model is necessary, it is recommended to create a shape model for each camera, as this ensures more accurate detection.
3. The user should choose the type of region of interest. This can be circular or rectangular. For the circular region of interest, the user clicks on the location of the circle and then drags the mouse to the radius of the circle. For the rectangular region, the first click defines the first corner of the rectangle and the second click defines the second corner of the rectangle. By rightclicking, the region is accepted.
4. Finally, the button “Create Shape Model” is clicked, which results in the shape model being saved with an “r” or and “l” as an identifier for right and left camera, respectively.

5.7 About Menu

In addition to the previously discussed subroutines, the main program also has an about menu, which contains information on the program in general, such

as the version number, in this case, version 1.0.1, as presented in this thesis. Figure 5-6 shows this about menu.

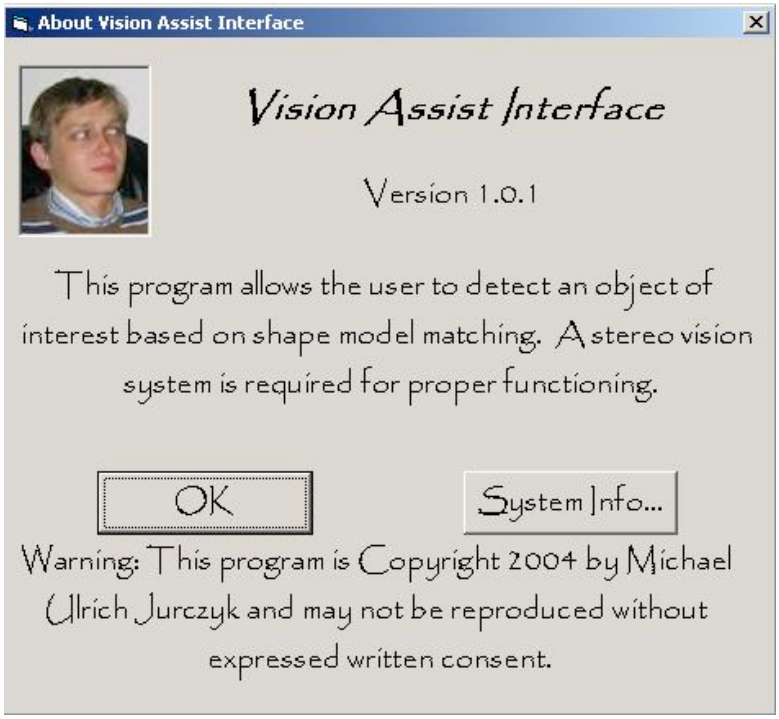

Figure 5-6: About Vision Assist Interface

5.8 Serial Data Transfer

In order to effectively use the object recognition, a serial interface connection was incorporated into the main user program. Should the position of the object be needed to control a robotic manipulator, for example, the data can be sent to any computer, which has a serial interface. Figure 5-7 shows the pin assignment of a typical RS-232 serial connector.

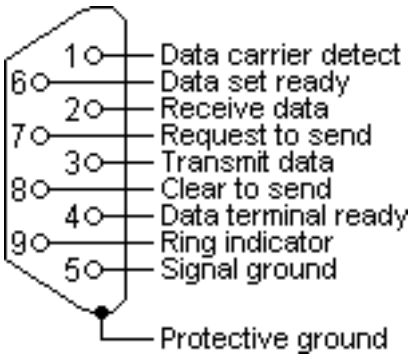

Figure 5-7: RS 232 Serial Connector Pin Assignment

The serial interface protocol is included in Appendix A, which includes all source code for the user interface. As long as there is a serial port protocol available, it is possible to read the data sent from the user interface and use it as desired, such as controlling the Cartesian position of the robotic manipulator.

All that is required for a successful serial communication are two computers, a serial cable with two 9-pin female connectors, where pins 2 and 3, as defined by Figure 5-7 are cross-connected. For the serial communication, the MSComm control incorporated into Visual Basic 6 was used. The port settings required for a successful serial communication, the settings listed in Table 5-1 below should be used on both sides of the communication. The serial connection has to be activated from the Execute Menu of the Main GUI.

Table 5-1: Serial Port Settings

Baud	9600
Parity	N
Data	8
Stop Bit	1

Chapter 6 – Assist Functions

6.1 Overview

The user of a robotic manipulator controls its movement via an input device. This input device could be a joystick, a haptic device, such as the PHANToM, jog control, or any other imaginable input device. Regardless of the input device, a user with a disability might find it difficult to control a robotic manipulator directly, especially if the person has tremors, as caused by, for example, Parkinson's disease.

As part of Benjamin Fritz's [4] thesis, he incorporated three different assist functions into the control of a robotic manipulator. These are described in this chapter, followed by their application to this work.

6.2 Linear Assist Function

The linear assist function restricts the movement of the robotic manipulator to a linear movement along a line connecting the end-effector and the goal position. The goal position is defined as the location of the object of interest. By using the theory described in chapter 4, the position of the object of interest is known with respect to the end-effector. While Fritz [4] employed only one camera along with a laser range finder, this thesis obtains the three-dimensional location of the object of interest directly. Hence, several calculations described by Fritz need

not be employed, and the scaling can be simplified. The image frame of the vision system can be seen in figure 6-1 below.

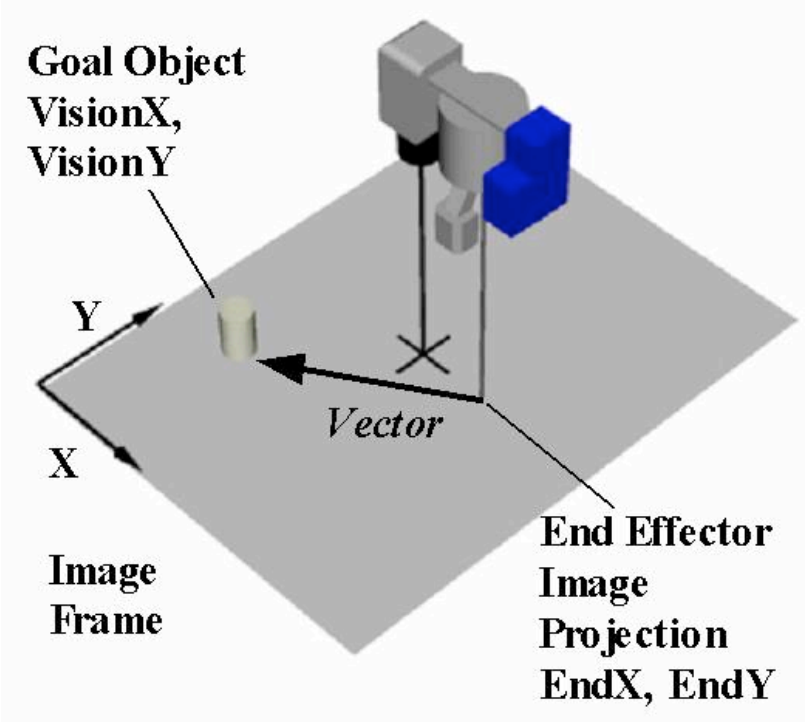

Figure 6-1: Image Frame of Vision System [4]

The vector from the end-effector projection, as defined in figure 6-1, is simply

$$\text{Vector} = \begin{bmatrix} X \\ Y \\ 0 \end{bmatrix} \quad (6.1)$$

where X and Y are the values obtained from the reconstruction calculations. Taking *Z-axis* as a unit vector directly upwards, Y can be defined as *B*=*Z*x*X*. Now, assuming that A = Vector, and B, and Z are the vectors defined and calculated previously, one can define its elements as:

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \end{bmatrix}^T \quad (6.2)$$

$$B = \begin{bmatrix} a_2 & b_2 & c_2 \end{bmatrix}^T \quad (6.3)$$

$$Z = \begin{bmatrix} a_3 & b_3 & c_3 \end{bmatrix}^T \quad (6.4)$$

Now, defining the direction cosines as

$$sum_i = \sqrt{a_i^2 + b_i^2 + c_i^2} \quad (6.5)$$

$$a_{i1} = \frac{a_i}{sum_i} \quad (6.6)$$

$$b_{i1} = \frac{b_i}{sum_i} \quad (6.7)$$

$$c_{i1} = \frac{c_i}{sum_i} \quad (6.8)$$

the transformation matrix between the contraint and the end-effector frame is:

$$R_c^o = \begin{bmatrix} a_{11} & b_{11} & c_{11} \\ a_{21} & b_{21} & c_{21} \\ a_{31} & b_{31} & c_{31} \end{bmatrix} \quad (6.9)$$

Finally, if the input velocity of the control device, such as the PHANToM, is given, the velocity of the manipulator can be calculated as:

$$V_{slave} = V_{input}\left(R_c^o\right)^T \quad (6.10)$$

If the velocity scale matrix is defined as

$$\text{Scale} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix} \quad (6.11)$$

then the modified velocity of the slave can be calculated as

$$V_{modified} = R_c^o(\text{Scale})V_{slave} \quad (6.12)$$

Which is the new velocity command for the robotic manipulator and can be sent the controller of the manipulator used.

6.3 Planar Assist Function

Similar to the linear assist function just described, the planar assist function restricts the movement to a plane. Three points – the position of the end-effector, the position of the object of interest, and finally an arbitrary point along the z-axis of the end-effector define this plane. This produces a vertical plane connecting the end-effector with the object of interest. The calculations needed for the planar assist function are exactly the same as those described by equations (6.1) through (6.10), except that the scale matrix is defined as:

$$\text{Scale} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (6.13)$$

Hence, the velocity of the manipulator is scaled such that the undesired motion is perpendicular to the constraint plane and is described by equation (6.14).

$$V_{modified} = R_c^o(\text{Scale})V_{slave} \quad (6.14)$$

6.4 Velocity Assist Function

The velocity assist function controls the velocity of the robotic manipulator as a function of the vertical distance from the end-effector to the object of interest. If the distance between the two is large, the velocity is scaled such that the it increases. As the distance gets smaller, the velocity is scaled down, allowing for a safe approach to the object of interest. Fritz [4] defined the velocity scale factors as shown in Figure 6-2 below.

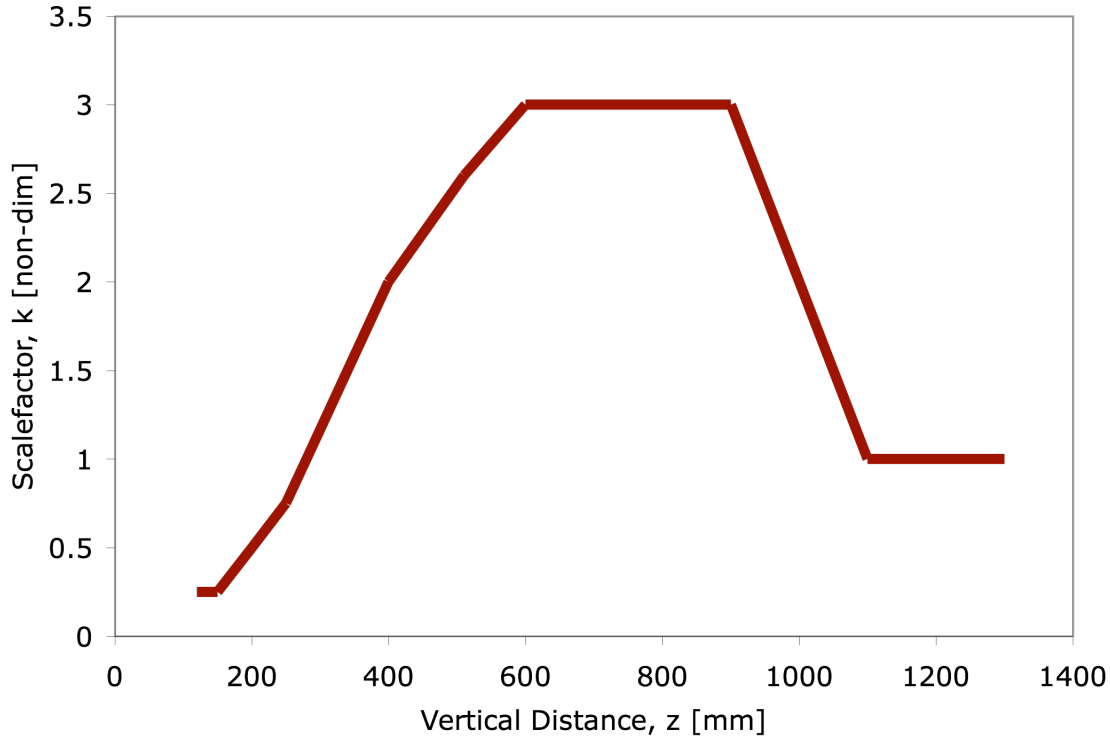

Figure 6-2: Velocity Assist Function Scalefactor vs. Vertical Distance

Once again, the calculations required are described by equations (6.1) through (6.10), while the Scale matrix is defined as:

$$\text{Scale} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \text{ScaleFactor} \end{bmatrix} \quad (6.15)$$

Finally, the modified velocity that is sent to robotic manipulator is defined by:

$$V_{modified} = (\text{Scale})V_{input} \quad (6.16)$$

Unfortunately, it was not possible to implement these assist functions at the time of thesis completion due to technical difficulties with the robotic manipulator, but they have been shown to work nicely and data was collected by Fritz [4] and described in his thesis.

6.5 Assist Function Integration into the User Interface

In order to properly use the assist functions described within this chapter, a subroutine, which can be accessed by the Execute menu of the user interface, was created. Again, the serial port is used to send and receive the required data. For detailed information on the serial, the reader should consult Chapter 5.

The user interface contains all the required equations described within this chapter and calls the appropriate assist function selected from the Execute menu. The data required from the input device is a vector of velocities, as defined by equation (6.17):

$$V_{input} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \quad (6.17)$$

where the elements of the matrix denote the velocity of the input device in the x-, y- and z- direction, respectively. By using equations (6.12), (6.14) or (6.16) for linear, planar, and velocity assist functions, respectively, and of course the appropriate scaling matrix, the input velocities are then transformed to modified velocities. These velocities are again a matrix with three elements and are defined by equation (6.18) as:

$$V_{modified} = \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \quad (6.18)$$

The modified velocities are then transmitted via the serial connection described in chapter 5 to the control computer of the robotic manipulator. These

velocities can then be used to scale the velocity of the input device such that the robotic manipulator is constrained as desired.

It should be noted, however, that it is assumed that the input device provides and collects information on the velocity of the input. Such an input device is the PHANToM [10] haptic interface. This input device has been used in the work of Fritz [4] and has been shown to be excellent for the use with scaling assist function.

The scaling factors used are generally one tenth of the input velocity, but it should also be noted that it is possible, and at times advantageous, to modify the scaling factors to a particular application. The smaller the scaling value, the slower the output velocity will be, and of course the output velocity is increased for larger scaling values.

Fritz [4] has also shown such tasks as obstacle avoidance, which is a combination of assist function. Again, the reader should refer to Fritz's work for detailed information on the assist functions.

Chapter 7 - Experimental Data

7.1 Physical Experimental Setup

The user interface was tested on the set-up shown in Figure 7-1 below. A simple metal shelf was chosen as the skeleton for the setup. The cameras were rigidly mounted to a platform, which can be translated in the x- and y-directions of the set-up. Two metal rulers are mounted to this platform, so that the position of the imaginary end-effector can be measured when obtaining data. Since the rulers tend to bend downward, a metal wire, on which the rulers rest, is wrapped around the skeleton of the set-up. The zero-mark of the rulers is taken to be the center of the end-effector. Then end-effector is taken to be 20 centimeters below this point.

Several objects of interest, for which data is presented later in this chapter, are rigidly mounted to an intermediate shelf or platform, which can be translated in the set-up z-direction. The purpose for mounting these objects is simply to allow for some time saving, as the position of the objects does not need to be measured for each image taken.

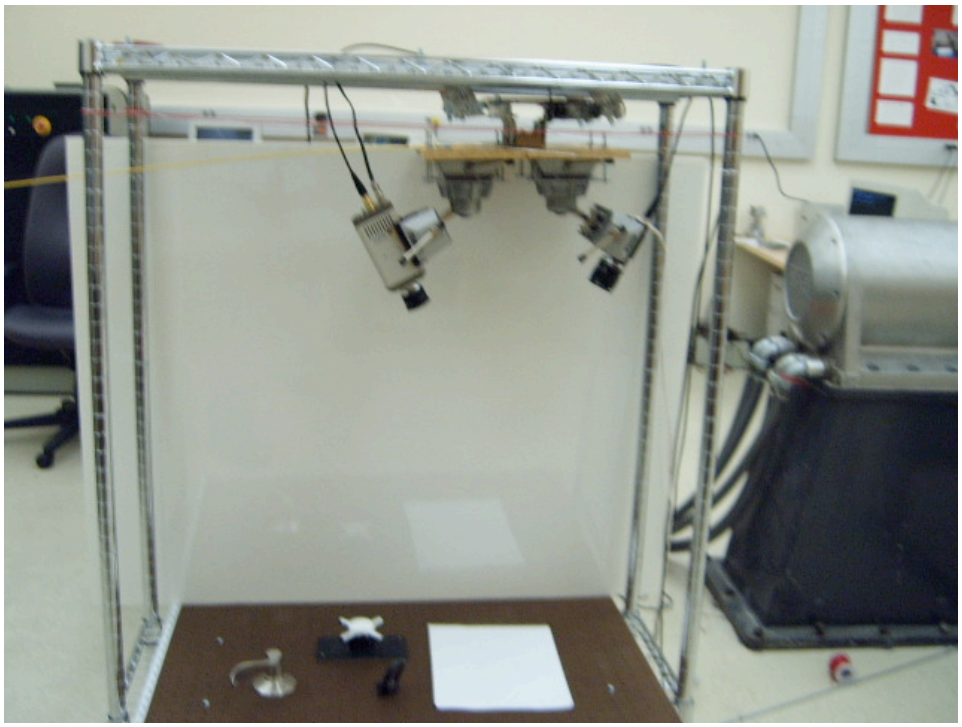

Figure 7-1: Experimental Set-Up

7.2 Experimental Procedure

After having obtained the intrinsic and extrinsic camera parameters, the user-interface is opened. An initial image is acquired to make sure that the cameras are functioning properly. A shape model for the desired object of interest is selected from the drop-down menu of the main interface and activated by clicking on the “Accept Shape Model” button. After this is completed, the data collection can be initialized; clicking on the “Locate Object” button does this.

Now, the positions obtained from the reconstruction algorithm along with the measured positions for each image are recorded for future analysis. Should an object not be detected in an image, this position is still recorded, so as to obtain

information about whether there are certain locations of the camera where the image cannot be detected or whether this is simply a fluke. This information can then be used to revise the shape model if needed.

Five sets of experiments were conducted – each set for a different shape model and object of interest. The data is presented in the following section showing the accuracy and errors involved.

7.3 Experimental Data

The first set of data is for the shape model "Door Handle." A description of the door handle shape model, as well as the other shape models discussed here are included in the appendix. Following is a graph showing a planar measurement of a door handle. Figures 7-2 through 7-5 show the x- and y-comparison of the measured, ie actual position of the door handle, versus the reconstructed, or assumed position of the door handle. Each graph shows a set of data for a different, discreet height, as allowed for by the test bed.

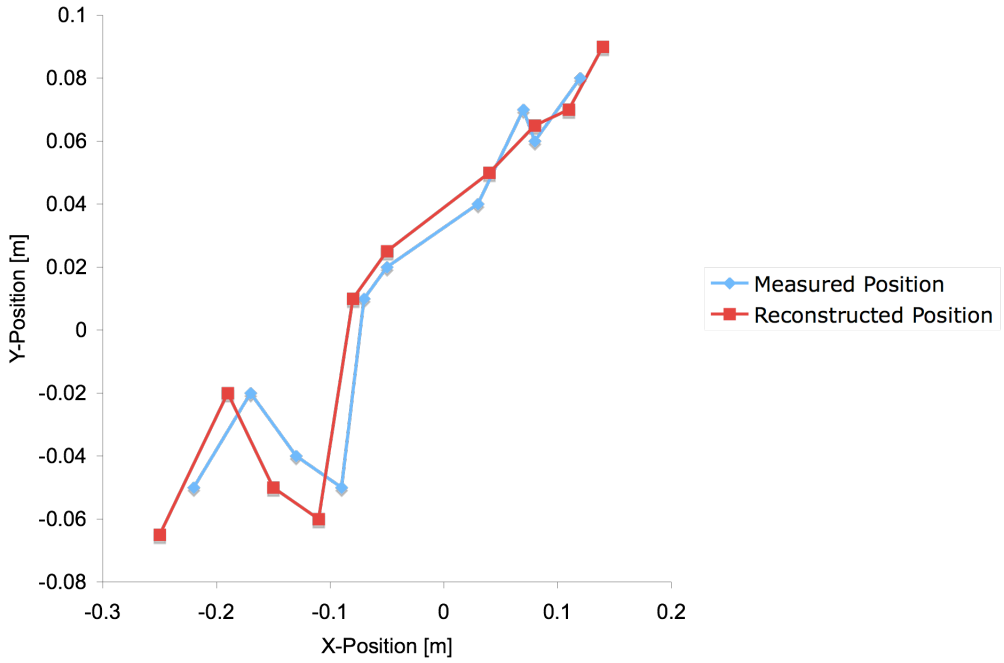

Figure 7-2: Door Handle Detection from Z = 0.54 m

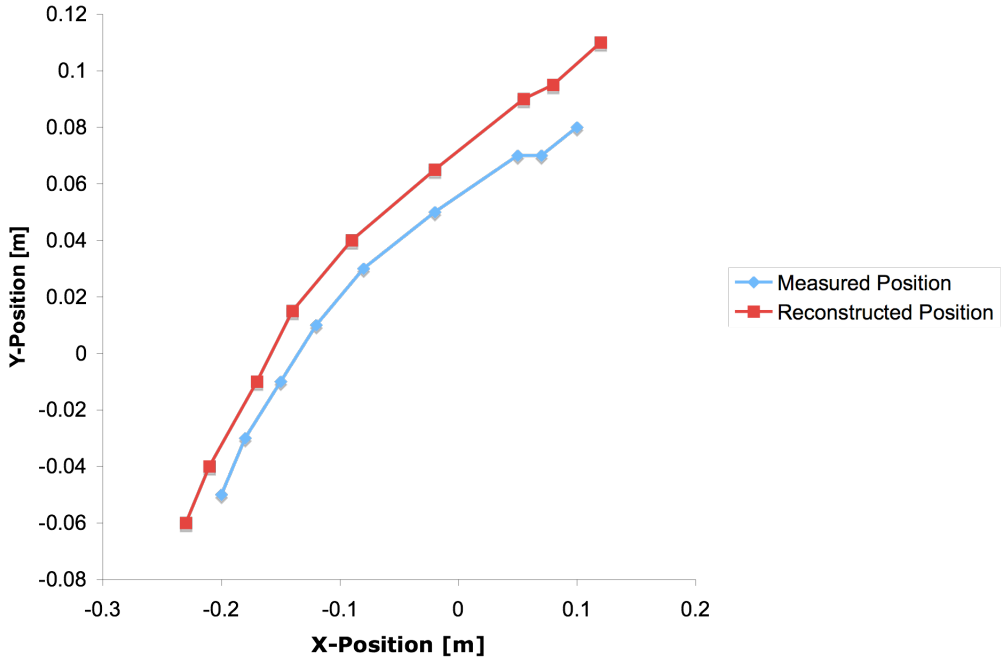

Figure 7-3: Door Handle Detection from Z = 0.44 m

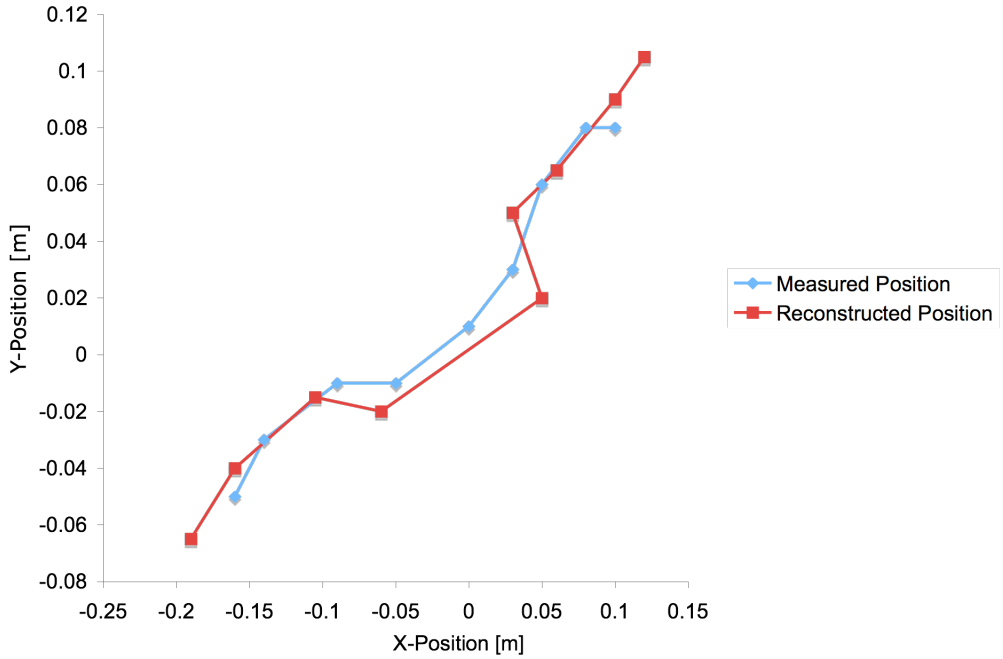

Figure 7-4: Door Handle Detection from Z = 0.34 m

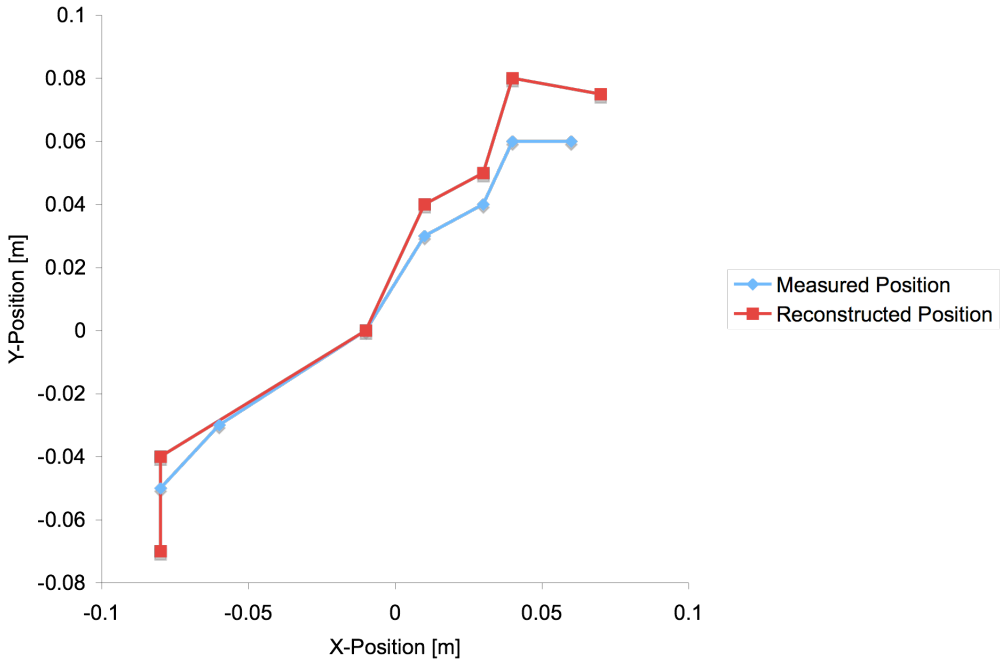

Figure 7-5: Door Handle Detection from Z = 0.14 m

Figures 7-2 through 7-5 show that the error, or difference between the actual position of the object with respect to the virtual manipulator decreases both with a decreased height, as well as with the object of interest being close to the center position of the manipulator. This can be explained by the fact that the shape model was created close to the center point of the manipulator. Due to geometry, the object detected appears to change its shape as the position is changed. This is simply due to the skewing of geometry from different perspectives. The error in x, y, and z, as a function of x, y, and z are shown below in Figure 7-6 through Figure 7-8 below.

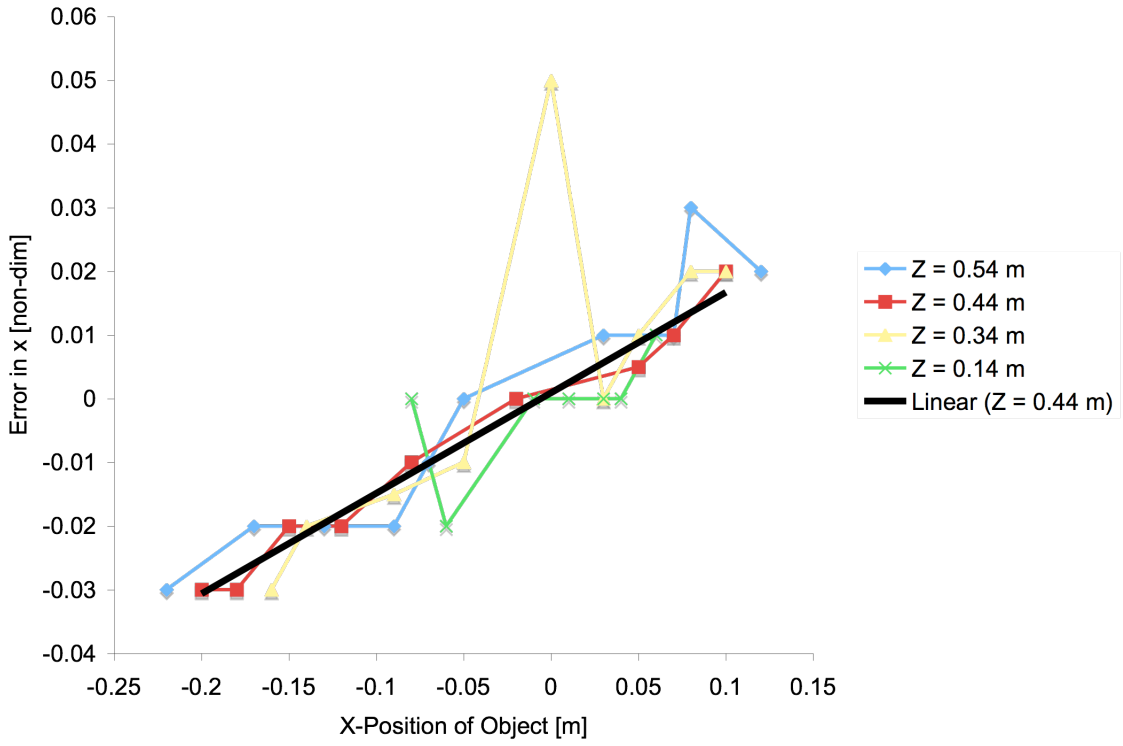

Figure 7-6: Error in X as a Function of X and for Different Z-Positions

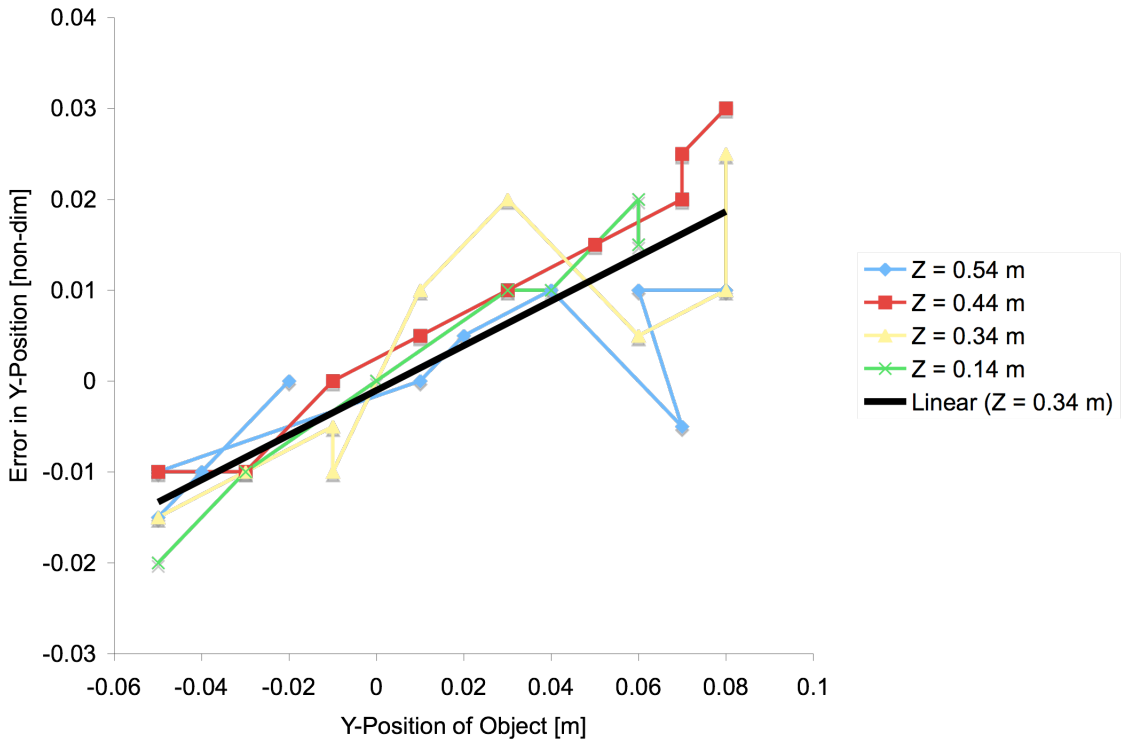

Figure 7-7: Error in Y-Position as a Function of Y and for Different Z-Values

Figures 7-6 and 7-7 show a trend towards decreasing the error in the x- and y-direction as the object to be detected approaches the center point of the manipulator. The linear trend lines in the graphs portray this nicely. Additionally, as the height from the object of interest decreases, the error also decreases slightly.

Chapter 8 - Conclusions and Future Work

8.1 Conclusions

A stereovision system has been created using off-the-shelf hardware and software. A user interface to perform all required tasks of stereovision has been created which addresses the two main problems of stereovision – the correspondence and the reconstruction problem. Scaling assist functions have been incorporated into the user interface, which can be used to assist in the telerobotic control of a robotic manipulator.

The user interface is built such that persons with disabilities can use the program without any extensive training, thus allowing them to use the program with rehabilitation robotic manipulators to manipulate objects. When a user uses the program, it is possible for this person to detect any object of interest, after the user creates a shape model for the object.

Especially persons with disabilities can benefit from this program and the stereovision system, as it is inexpensive and easy to set up with virtually any robotic manipulator. This is the first time, to the knowledge of the author, that a user interface has been created for use with telerobotic manipulation of objects. By using shape based object detection, it is possible to quickly and easily locate objects of interest and approach the objects using scaling assist functions.

The user interface also includes a subroutine, which allows the user to obtain the intrinsic camera parameters. This can easily be performed using the calibration table, which can be created from the calibration table subroutine. Similar to the intrinsic camera calibration subroutine, a subroutine for the extrinsic camera calibration was included. By following the procedures described in this thesis, any person should be able to successfully set-up their own stereovision system.

The program created for this work allows the user to select from several shape models which were created using the user interface and also gives the user the power to create and expand the shape model library as desired for practically any object imaginable.

The detection of objects was tested using the software and the shape models incorporated into the program. The errors in detection lie within acceptable range and decrease, as the manipulator moves towards the object of interest, therefore making the errors acceptable.

It has been shown that the three-dimensional position of the object found by the shape models can be used with the assist functions developed by Fritz. This means that the stereovision system is ready for use in telerobotics and rehabilitation applications. The only procedures that need to be completed lie on the robotic side. This means that if the robotic manipulator can be controlled with almost any type of input device, it is possible to simply equip the robotic manipulator with a stereovision system along with the visual assist program

created for this work. After connecting the two computers with a standard serial cable, as described in chapter 5, it is then possible to use the output of the vision system to control and scale the movement of the robotic manipulator.

8.2 Future Work and Recommendations

The shape model library should be vastly expanded to include many different items commonly encountered in every day life. With an extensive shape model library, the stereo system along with the user interface could be marketed in cooperation with MVTec [9], as this thesis relies partly on their software Halcon. More data should be collected as to the effects of lighting differences, and to fix this problem, a light source could be incorporated into the stereo system, allowing for constant lighting, and hence proper detection of objects of interest. Additionally, the assist functions described in chapter 6 should be properly evaluated and used in experiments to further modify these assist functions.

Once the system is set up with a commonly available robotic manipulator, such as the PUMA or the Manus Manipulator, the ease of setting up the stereo vision system should be tested, and trial runs with persons with disabilities should be performed. Using their feedback, the program and subroutines should then be optimized to best fit the users' requirements.

References

[1] U.S. Census Bureau, “Americans with Disabilities – Household Economic Studies,” February 2001, pp. 70-73.

[2] Håkan Eftring, “The Useworthiness of Robots for People with Physical Disabilities,” Doctoral Dissertation, ISBN 91-628-3711-7, Lund, Sweden, 1999.

[3] Gunnar Bolmsjo, Hakan Neveryd, and Hakan Eftring. “Robotics in Rehabilitation,” IEEE Transactions on Rehabilitation Engineering, Vol. 3, No. 1, March 1995.

[4] Benjamin E. Fritz, “Development and Testing of a Telerobotic System for Enhancing Manipulation Capabilities of Persons with Disabilities”, MSME Thesis, University of South Florida, 2002.

[5] Wentao Yu, “Intelligent Telerobotic Assistance For Enhancing Manipulation Capabilities Of Persons With Disabilities,” Ph.D. Dissertation, University of South Florida, 2004.

[6] Ramesh Jain, “Machine Vision,” McGraw-Hill, Inc., New York, NY, 1995.

[7] Nello Zuech, “Understanding and Applying Machine Vision,” Marcel Dekker, Inc., New York, NY, 2nd edition, 2000.

[8] Emanuele Trucco and Alessandro Verri, “Introductory Techniques for 3-D Computer Vision,” Prentice Hall, Inc., Upper Saddle River, NJ, 1998.

[9] MVTec AG, http://www.mvtec.com/halcon.

[10] SensAble Technologies, http://www.sensable.com/products/phantom.htm.

Bibliography

[1] Luc D. Joly and Claude Andriot, “Imposing motion constraints to a force reflecting telerobot through real-time simulation of a virtual mechanism,” Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, May 1994, pp. 357-362.

[2] Kazuhiro Kosuge, Koji Takeo, and Toshio Pukuda, “Unified approach for teleoperation of virtual and real environment manipulation based on reference dynamics,” Proceedings of the 1995 IEEE International Conference on Robotics and Automation, Nagoya, Japan, May 1995, pp. 938-943.

[3] Thomas B. Sheridan, “Telerobotics, Automation, and Human Supervisory Control,” The MIT Press, London, England, 1992.

[4] “Human Machine Cooperative Telerobotics,” Topical Report, Federal Energy Technology Center, U. S. Department of Energy, Morgantown, WV. November 1998.

[5] Karan A. Manocha. “Development and Implementation of Teleoperation Assist Functions,” Master of Science in Mechanical Engineering, University of South Florida, August 2000.

[6] Norali Pernalete, “Development Of A Robotic Haptic Interface To Perform Vocational Tasks By People With Disabilities,” Doctor of Philosophy, Electrical Engineering, University of South Florida, December 2001.

[7] J. Schuyler, R. Mahoney, Assesing Human-Robotic Performance for Vocational Placement,” IEEE Transactions on Rehabilitation Engineering, Vol. 8, No 3, September 2000.

[8] W. Harwin, T. Rahman, R. Foulds, “A Review of Design Issues in Rehabilitation Robotics with Reference to North American Research,” IEEE Transactions on Rehabilitation Engineering. Vol 3, No 1, March 1995.

[9] R. Erlandson, ”Applications of Robotic/Mechatronic Systems in Special Education, Rehabilitation Therapy and Vocational Training: A Paradigm Shift,” IEEE Transactions on Rehabilitation Engineering. Vol 3, No 1, March 1995.

[10] C. Stanger, C. Angling, W. Harwin, D. Romilly, "Devices for Assisting Manipulation: A Summary of User Task Priorities," IEEE Transactions on Rehabilitation Engineering. Vol 2, No 4, December 1994.

[11] Steven E. Everett, Rajiv V. Dubey, Y. Isoda, and C. Dumont, "Vision-Based End-Effector Alignment Assistance for Teleoperation," Proceedings of the IEEE International Conference on Robotics and Automation, Detroit, MI.. May 1999, pp. 543-549.

[12] Steven E. Everett, Rajiv V. Dubey, "Model-based variable position mapping for Telerobotic Assistance in a Cylindrical Environment," Proceedings of the IEEE International Conference on Robotics and Automation, Detroit, MI.. May 1999, pp. 2197-2202.

[13] R. Dubey, K. Manocha, N. Pernalete, "Variable Position Mapping Based Assistance in Teleoperation for Nuclear Clean Up," The International Conference on Robotics and Automation (ICRA) 2001. Seoul, Korea. May 21-26, 2001.

[14] R. Dubey, S. Everett, N. Pernalete, K. Manocha. "Teleoperation Assistance Through Variable Velocity Mapping," IEEE Transactions on Robotics and Automation, October 2001.

[15] SensAble Technologies, http://www.sensable.com/products/phantom.htm.

Appendices

Appendix A: Graphical User Interface Source Code

frmAbout - 1

```
Option Explicit
' Reg Key Security Options...
Const READ_CONTROL = &H20000
Const KEY_QUERY_VALUE = &H1
Const KEY_SET_VALUE = &H2
Const KEY_CREATE_SUB_KEY = &H4
Const KEY_ENUMERATE_SUB_KEYS = &H8
Const KEY_NOTIFY = &H10
Const KEY_CREATE_LINK = &H20
Const KEY_ALL_ACCESS = KEY_QUERY_VALUE + KEY_SET_VALUE + _
KEY_CREATE_SUB_KEY + KEY_ENUMERATE_SUB_KEYS + _
KEY_NOTIFY + KEY_CREATE_LINK + READ_CONTROL
' Reg Key ROOT Types...
Const HKEY_LOCAL_MACHINE = &H80000002
Const ERROR_SUCCESS = 0
Const REG_SZ = 1 ' Unicode nul terminated string
Const REG_DWORD = 4 ' 32-bit number
Const gREGKEYSYSINFOLOC = "SOFTWARE\Microsoft\Shared Tools Location"
Const gREGVALSYSINFOLOC = "MSINFO"
Const gREGKEYSYSINFO = "SOFTWARE\Microsoft\Shared Tools\MSINFO"
Const gREGVALSYSINFO = "PATH"
Private Declare Function RegOpenKeyEx Lib "advapi32" Alias
"RegOpenKeyExA" (ByVal hKey As Long, ByVal
lpSubKey As String, ByVal ulOptions As Long, ByVal samDesired As Long,
ByRef phkResult As Long) As Long
Private Declare Function RegQueryValueEx Lib "advapi32" Alias
"RegQueryValueExA" (ByVal hKey As Long, ByVal lpValueName As String,
ByVal lpReserved As Long, ByRef lpType As Long, ByVal lpData As String,
ByRef lpcbData As Long) As Long
Private Declare Function RegCloseKey Lib "advapi32" (ByVal hKey As
Long) As Long
Private Sub cmdSysInfo_Click()
Call StartSysInfo
End Sub
Private Sub cmdOK_Click()
Unload Me
End Sub
Private Sub Form_Load()
Me.Caption = "About " & App.Title
lblVersion.Caption = "Version " & App.Major & "." & App.Minor & "." &
App.Revision
lblTitle.Caption = App.Title
End Sub
Public Sub StartSysInfo()
On Error GoTo SysInfoErr
Dim rc As Long
Dim SysInfoPath As String
' Try To Get System Info Program Path\Name From Registry...
```

Appendix A (Continued)

```
If GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFO, gREGVALSYSINFO,
SysInfoPath) Then
' Try To Get System Info Program Path Only From Registry...
ElseIf GetKeyValue(HKEY_LOCAL_MACHINE, gREGKEYSYSINFOLOC,
gREGVALSYSINFOLOC, SysInfoPath) Then
' Validate Existance Of Known 32 Bit File Version
If (Dir(SysInfoPath & "\MSINFO32.EXE") <> "") Then
SysInfoPath = SysInfoPath & "\MSINFO32.EXE"
' Error - File Can Not Be Found...
Else
GoTo SysInfoErr
End If
' Error - Registry Entry Can Not Be Found...
Else
GoTo SysInfoErr
End If
frmAbout - 2
Call Shell(SysInfoPath, vbNormalFocus)
Exit Sub
SysInfoErr:
MsgBox "System Information Is Unavailable At This Time", vbOKOnly
End Sub
Public Function GetKeyValue(KeyRoot As Long, KeyName As String,
SubKeyRef As String, ByRef KeyVal As S
tring) As Boolean
Dim i As Long ' Loop Counter
Dim rc As Long ' Return Code
Dim hKey As Long ' Handle To An Open Registry Key
Dim hDepth As Long '
Dim KeyValType As Long ' Data Type Of A Registry Key
Dim tmpVal As String ' Tempory Storage For A Registry Key Value
Dim KeyValSize As Long ' Size Of Registry Key Variable
'------------------------------------------------------------
' Open RegKey Under KeyRoot {HKEY_LOCAL_MACHINE...}
'------------------------------------------------------------
rc = RegOpenKeyEx(KeyRoot, KeyName, 0, KEY_ALL_ACCESS, hKey) ' Open
Registry Key
If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Error...
tmpVal = String$(1024, 0) ' Allocate Variable Space
KeyValSize = 1024 ' Mark Variable Size
'------------------------------------------------------------
' Retrieve Registry Key Value...
'------------------------------------------------------------
rc = RegQueryValueEx(hKey, SubKeyRef, 0, _
KeyValType, tmpVal, KeyValSize) ' Get/Create Key Value
If (rc <> ERROR_SUCCESS) Then GoTo GetKeyError ' Handle Errors
If (Asc(Mid(tmpVal, KeyValSize, 1)) = 0) Then ' Win95 Adds Null
Terminated String...
tmpVal = Left(tmpVal, KeyValSize - 1) ' Null Found, Extract From String
Else ' WinNT Does NOT Null Terminate String...
tmpVal = Left(tmpVal, KeyValSize) ' Null Not Found, Extract String Only
End If
```

Appendix A (Continued)

```
'------------------------------------------------------------
' Determine Key Value Type For Conversion...
'------------------------------------------------------------
Select Case KeyValType ' Search Data Types...
Case REG_SZ ' String Registry Key Data Type
KeyVal = tmpVal ' Copy String Value
Case REG_DWORD ' Double Word Registry Key Data Type
For i = Len(tmpVal) To 1 Step -1 ' Convert Each Bit
KeyVal = KeyVal + Hex(Asc(Mid(tmpVal, i, 1))) ' Build Value Char. By
Char.
Next
KeyVal = Format$("&h" + KeyVal) ' Convert Double Word To String
End Select
GetKeyValue = True ' Return Success
rc = RegCloseKey(hKey) ' Close Registry Key
Exit Function ' Exit
GetKeyError: ' Cleanup After An Error Has Occured...
KeyVal = "" ' Set Return Val To Empty String
GetKeyValue = False ' Return Failure
rc = RegCloseKey(hKey) ' Close Registry Key
End Function
frmcaltab - 1
Private Sub cmdcreatecaltab_Click()
Dim Op As New HOperatorSetX
' Dim width As String
Dim Tuple As New HTupleX
Dim hx_BasDefWinHandle As Variant
Dim Width As Double
lblcaltabstatus = "Running"
Width = txtcaltabwidth.Text
Call Op.CreateCaltab(Width,
"C:\VisionSystem\CalibrationTable\caltab.descr", "C:\VisionSystem\Cali
brationTable\caltab.ps")
lblcaltabstatus = "Calibration Table Created"
End Sub
frminternalcameracalibration - 1
Private Sub chkdefault_Click()
frameparameters.Enabled = True
End Sub
Private Sub cmdinternalcameracalibration_Click()
Dim checkbutton As Integer
Dim Window As HWindowX
Dim valsizegauss, valmarkthresh, valmindiammarks, valstartthresh As
Integer
Dim valdeltathresh, valminthresh, valmincontlength, valmaxdiammarks As
Integer
Dim valalpha As Long
Set Window = HWindowXCtrli.HalconWindow
valsizegauss = valsizegauss.Text
valmarkthresh = valmarkthresh.Text
valmindiammarks = valmindiammarks.Text
valstartthresh = valstartthresh.Text
```

Appendix A (Continued)

```
valdeltathresh = valdeltathresh.Text
valminthresh = valminthresh.Text
valalpha = valalpha.Text
valmincontlength = valmincontlength.Text
valmaxdiammarks = valmaxdiammarks.Text
If chkhitachi.Value = 1 Then
checkbutton = 1
lblinternalcameracalibrationstatus = "Running Hitachi Calibration"
End If
If chkimaging.Value = 1 Then
checkbutton = 2
lblinternalcameracalibrationstatus = "Running Imaging Source
Calibration"
End If
Call InternalCameraCalibration(Window, checkbutton, valsizegauss,
valmarkthresh _
, valmindiammarks, valstartthresh, valdeltathresh, valminthresh,
valalpha _
, valmincontlength, valmaxdiammarks)
lblinternalcameracalibrationstatus = "Finished"
End Sub
frmshapemodelcreation - 1
Private Sub cmdshapemodelcreation_Click()
' draw_rectangle1 (WindowHandle, R1, C1, R2, C2)
' Please note: In Visual Basic the zooming of images will not be
adjusted automatically (like in H
Develop).
' Therefore you have to call 'op.SetPart()' with the parameters of the
image you want to display.
' gen_rectangle1 (HROI, R1, C1, R2, C2)
' gen_rectangle1 (IROI, 139, 130, 260, 258)
'Call Op.ReadImage(hx_H1, "c:/VisionSystem/ImageCapture/H.jpg")
'Call Op.ReadImage(hx_I1, "c:/VisionSystem/ImageCapture/I.jpg")
'Call Op.FindShapeModel(hx_H1, hx_HModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row1,
hx_Column1, hx_Angle1, hx_Score1)
'Call Op.FindShapeModel(hx_I1, hx_IModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row, h
x_Column, hx_Angle, hx_Score)
End Sub
Private Sub cmdshapemodelgrab_Click()
Dim Op As New HOperatorSetX
' Sub RunHalcon(ByRef Window As HWindowX)
Dim Tuple As New HTupleX
Dim Window As HWindowX
Dim hx_WindowHandle As Variant, hx_R1 As Variant
Dim hx_C1 As Variant, hx_R2 As Variant
Dim hx_C2 As Variant, hx_HModel As Variant
Dim hx_IModel As Variant, hx_Row1 As Variant
Dim hx_Column1 As Variant, hx_Angle1 As Variant
Dim hx_Score1 As Variant, hx_Row As Variant
Dim hx_Column As Variant, hx_Angle As Variant
```

Appendix A (Continued)

```
Dim hx_Score As Variant, hx_FGHandle As Variant
Dim hx_FGHandle1 As Variant, hx_Row2 As Variant
Dim hx_Column2 As Variant, hx_Radius As Variant
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_hrow As Variant, hx_hcolumn As Variant
Dim hx_hradius As Variant, hx_irow As Variant
Dim hx_icolumn As Variant, hx_iradius As Variant
Dim hx_icolumn2, hx_irow2, hx_hrow2, hx_hcolumn2 As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_H As HUntypedObjectX, hx_I As HUntypedObjectX
Dim hx_hroi As HUntypedObjectX, hx_iroi As HUntypedObjectX
Dim hx_HIR As HUntypedObjectX, hx_IIR As HUntypedObjectX
Dim hx_HMI As HUntypedObjectX, hx_HMR As HUntypedObjectX
Dim hx_IMI As HUntypedObjectX, hx_IMR As HUntypedObjectX
Dim hx_H1 As HUntypedObjectX, hx_I1 As HUntypedObjectX
Dim hx_iImage As HUntypedObjectX, hx_himage As HUntypedObjectX
'Dim hx_ROI As HUntypedObjectX, hx_hroi As HUntypedObjectX
'Dim hx_iroi As HUntypedObjectX, hx_himage As HUntypedObjectX
Set Window = HWindowXCtrl.HalconWindow
hx_BasDefWinHandle = Window.HalconID
Call Op.CloseAllFramegrabbers
Dim Dateiname As String
Dateiname = txtdateiname.Text
' For Left Eye
If optleft.Value = True Then
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default",
"ntsc", "2", 0, -1, hx_fgimagingsource)
For i = 1 To 2
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Next i
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
If optcircle.Value = True Then
frmshapemodelcreation - 2
Call Op.DrawCircle(hx_BasDefWinHandle, hx_irow, hx_icolumn, hx_iradius)
Call Op.GenCircle(hx_iroi, hx_irow, hx_icolumn, hx_iradius)
End If
If optrectangle.Value = True Then
Call Op.DrawRectangle1(hx_BasDefWinHandle, hx_irow, hx_icolumn,
hx_irow2, hx_icolumn2)
Call Op.GenRectangle1(hx_iroi, hx_irow, hx_icolumn, hx_irow2,
hx_icolumn2)
End If
Call Op.ReduceDomain(hx_iImage, hx_iroi, hx_IIR)
Call Op.InspectShapeModel(hx_IIR, hx_IMI, hx_IMR, 4, 120)
Call Op.CreateShapeModel(hx_IIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_IMode
l)
Call Op.WriteShapeModel(hx_IModel, "c:/VisionSystem/" & Dateiname &
"i.sm")
End If
' For Right Eye
```

Appendix A (Continued)

```
If optright.Value = True Then
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default",
"ntsc", "0", 0, -1, hx_fghitachi)
For i = 1 To 2
Call Op.GrabImage(hx_himage, hx_fghitachi)
Next i
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
If optcircle.Value = True Then
Call Op.DrawCircle(hx_BasDefWinHandle, hx_hrow, hx_hcolumn, hx_hradius)
Call Op.GenCircle(hx_hroi, hx_hrow, hx_hcolumn, hx_hradius)
End If
If optrectangle.Value = True Then
Call Op.DrawRectangle1(hx_BasDefWinHandle, hx_hrow, hx_hcolumn,
hx_hrow2, hx_hcolumn2)
Call Op.GenRectangle1(hx_hroi, hx_hrow, hx_hcolumn, hx_hrow2,
hx_hcolumn2)
End If
Call Op.ReduceDomain(hx_himage, hx_hroi, hx_HIR)
Call Op.InspectShapeModel(hx_HIR, hx_HMI, hx_HMR, 10, 120)
Call Op.CreateShapeModel(hx_HIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_HMode
l)
Call Op.WriteShapeModel(hx_HModel, "c:/VisionSystem/" & Dateiname &
"h.sm")
End If
End Sub
Sub InitHalcon()
' Default settings used in HDevelop
Call Op.SetSystem("do_low_error", "false")
End Sub
HalconForm - 1
' This is a template VB project to be used with HALCON programs
exported
' from HDevelop. To use exported HDevelop code, load it with the menu
' Project -> Add Module -> Existing. Note that you will usually have to
' adapt the size of the HALCON ActiveX window and/or add additional
HALCON
' ActiveX windows, depending on how many HDevelop windows your program
' uses. If your HDevelop program does its image processing in an
infinite
' loop, you will have to modify the code to allow the program to
process
' the events generated by interaction from the user. Examine the other
' VB examples that come with HALCON to see how to do this.
Option Explicit
Private Sub cmdshapemodelgrab_Click()
Dim Window1 As HWindowX
Dim Window2 As HWindowX
Dim Dateiname As String
Set Window1 = HWindowXCtrl.HalconWindow
Set Window2 = HWindowXCtrl1.HalconWindow
```

Appendix A (Continued)

```
Dateiname = txtdateiname.Text
HalconStatus = "Grabbing Image for Shape Model Creation"
HalconStatus.Refresh
Call ShapeModelGrabImage(Window1, Window2, Dateiname)
HalconStatus = "Finished Image Capture"
End Sub
Private Sub cmdshapemodelcreation_Click()
Dim Window1 As HWindowX
Dim Window2 As HWindowX
Dim Hr1, Hc1, Hr2, Hc2, Ir1, Ic1, Ir2, Ic2 As Integer
Set Window1 = HWindowXCtrl.HalconWindow
Set Window2 = HWindowXCtrl1.HalconWindow
HalconStatus = "Please Wait - Shape Model is Being Created"
HalconStatus.Refresh
'Hr1 = txtHr1.Text
'Hc1 = txtHc1.Text
'Hr2 = txtHr2.Text
'Hc2 = txtHc2.Text
' Ir1 = txtIr1.Text
' Ic1 = txtIc1.Text
' Ir2 = txtIr2.Text
' Ic2 = txtIc2.Text
' Call ShapeModelCreation(Window1, Window2, Hr1, Hr2, Hc1, Hc2, Ir1,
Ir2, Ic1, Ic2)
HalconStatus = "Finished Shape Model Creation"
End Sub
Private Sub cmdshapemodelconfirm_Click()
Dim Op As New HOperatorSetX
Dim Window1 As HWindowX
Dim Window2 As HWindowX
Dim i As Integer
Set Window1 = HWindowXCtrl.HalconWindow
Set Window2 = HWindowXCtrl1.HalconWindow
Dim R(1 To 9) As Double
Dim T(1 To 3) As Double
Dim RT(1 To 9) As Double
Dim MI(1 To 9) As Double
Dim MH(1 To 9) As Double
HalconForm - 2
Dim sxi, syi, fi, oxi, oyi As Double
Dim sxh, syh, fh, oxh, oyh As Double
' Imaging Source Parameters
' Intrinsic Parameters
'Open "C:\Vision System\iintrin.txt" For Input As #1
'Input #1, sxi, syi, oxi, oyi, fi
'Close #1
sxi = 1.0249 * 10 ^ -5
syi = 1.1 * 10 ^ -5
oxi = 311.828
oyi = 253.998
fi = 0.00872367
MI(1) = -sxi / fi
```

Appendix A (Continued)

```
MI(2) = 0
MI(3) = oxi * sxi / fi
MI(4) = 0
MI(5) = -syi / fi
MI(6) = oyi * syi / fi
MI(7) = 0
MI(8) = 0
MI(9) = 1
' Hitachi Parameters
' Intrinsic Parameters
'Open "C:\Vision System\hintrin.txt" For Input As #2
'Input #2, sxh, syh, oxh, oyh, fh
'Close #2
sxh = 1.07002 * 10 ^ -5
syh = 1.1 * 10 ^ -5
oxh = 317.915
oyh = 224.494
fh = 0.0076443
MH(1) = -sxh / fi
MH(2) = 0
MH(3) = oxh * sxh / fh
MH(4) = 0
MH(5) = -syh / fh
MH(6) = oyh * syh / fh
MH(7) = 0
MH(8) = 0
MH(9) = 1
' Rotation Parameters - THIS HAS TO BE REPLACED WITH R =
R(H)*TRANSPOSE(R(I))
'Open "C:\Vision System\rotation.txt" For Input As #3
'For i = 1 To 9
'Input #3, R(i)
'Next i
'Close #3
R(1) = 0.376
R(2) = 0.205
R(3) = 0.904
R(4) = 0.019
R(5) = 0.973
R(6) = -0.229
R(7) = -0.926
R(8) = 0.103
R(9) = 0.362
' Transpose of Rotation Matrix
RT(1) = R(1)
RT(2) = R(4)
RT(3) = R(7)
RT(4) = R(2)
RT(5) = R(5)
RT(6) = R(8)
HalconForm - 3
RT(7) = R(3)
```

Appendix A (Continued)

```
RT(8) = R(6)
RT(9) = R(9)
' Grab Image and Find Shapemodel
Dim Tuple As New HTupleX
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_whhitachi As Variant, hx_whimagingsource As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_BasDefWinHandle2 As Variant
Dim hx_himage As HUntypedObjectX, hx_iImage As HUntypedObjectX
Dim hx_ModelID1 As Variant
Dim hx_ModelID2 As Variant
Set Window1 = HWindowXCtrl.HalconWindow
Set Window2 = HWindowXCtrl1.HalconWindow
Dim YH, XH, YI, XI As Variant
Dim hx_hAngle, hx_hScale, hx_hScore As Variant
Dim hx_iAngle, hx_iScale, hx_iScore As Variant
hx_BasDefWinHandle = Window1.HalconID
hx_BasDefWinHandle2 = Window2.HalconID
Dim shapemodelfilei, shapemodelfileh As String
Dim MPI(1 To 3) As Double
Dim MPH(1 To 3) As Double
Dim RTPH(1 To 3) As Double
Dim X, Y, Z As Double
Dim c As Integer
If Not txtname.Text = "0" Then
shapemodelfilei = "C:\VisionSystem\" & txtname.Text & "i.sm"
shapemodelfileh = "C:\VisionSystem\" & txtname.Text & "h.sm"
End If
' First, make sure all framegrabbers are closed
Call Op.CloseAllFramegrabbers
' Open both Framegrabbers
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
If txtname.Text = "0" Then
shapemodelfilei = "C:\VisionSystem\" & cmboshapemodel.Text & "i.sm"
shapemodelfileh = "C:\VisionSystem\" & cmboshapemodel.Text & "h.sm"
End If
lblshapemodelname.Caption = shapemodelfilei
' Detect object using shape model until Stop Button is pressed
Call Op.ReadShapeModel(shapemodelfilei, hx_ModelID1)
Call Op.ReadShapeModel(shapemodelfileh, hx_ModelID2)
' Do While cmdstop.Value = False
' For c = 1 To 1000
For i = 1 To 2
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Next i
Call Op.DispObj(hx_himage, hx_BasDefWinHandle2)
```

Appendix A (Continued)

```
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Call Op.FindScaledShapeModel(hx_himage, hx_ModelID2, 0, 3.14, 0.5, 1.1,
0.3, 1, 0.9, "interpol
ation", 0, 0.2, YH, XH, hx_hAngle, hx_hScale, hx_hScore)
Call Op.FindScaledShapeModel(hx_iImage, hx_ModelID1, 0, 3.14, 0.5, 1.1,
0.3, 1, 0.9, "interpol
ation", 0, 0.2, YI, XI, hx_iAngle, hx_iScale, hx_iScore)
' Calculate the Position of the Object
MPI(1) = MI(1) * XI + MI(1) * fi
MPI(2) = MI(5) * YI + MI(6) * fh
MPI(3) = fi
HalconForm - 4
MPH(1) = MH(1) * XH + MH(3) * fh
MPH(2) = MH(5) * YH + MH(6) * fh
MPH(3) = fh
RTPH(1) = RT(1) * MPH(1) + RT(2) * MPH(2) + RT(3) * MPH(3)
RTPH(2) = RT(4) * MPH(1) + RT(5) * MPH(2) + RT(6) * MPH(3)
RTPH(3) = RT(7) * MPH(1) + RT(8) * MPH(2) + RT(9) * MPH(3)
X = MPI(1) - RTPH(1) + MPI(2) * RTPH(3) - MPI(3) * RTPH(2)
Y = MPI(2) - RTPH(2) + MPI(3) * RTPH(1) - MPI(1) * RTPH(3)
Z = MPI(3) - RTPH(3) + MPI(1) * RTPH(2) - MPI(2) * RTPH(1)
lblxpos.Caption = X - 0.15
lblypos.Caption = Y - 0.15
lblzpos.Caption = Z + 0.15
'Next c
' Loop
Call Op.CloseAllFramegrabbers
End Sub
Private Sub cmdshapemodelrun_Click()
lblshapemodel = cmboshapemodel.Text
End Sub
Private Sub Form_Load()
' Call InitHalcon
End Sub
Private Sub cmdimagecapture_Click()
Dim i As Integer
Dim Dateiname As String
i = txtimagecapture.Text
Dateiname = txtdateiname.Text
Dim Window1 As HWindowX
Dim Window2 As HWindowX
Set Window1 = HWindowXCtrl.HalconWindow
Set Window2 = HWindowXCtrl1.HalconWindow
HalconStatus = "Running Image Capture"
HalconStatus.Refresh
Call ImageCapture(Window1, Window2, i, Dateiname)
HalconStatus = "Finished."
End Sub
Private Sub mnuabout_Click()
frmAbout.Show
End Sub
Private Sub mnucaltab_Click()
```

Appendix A (Continued)

```
frmcaltab.Show
End Sub
Private Sub mnuexit_Click()
Unload Me
End Sub
Private Sub mnuimagecapture_Click()
' frmimagecapture.Show
End Sub
Private Sub mnuinternalcameracalibration_Click()
frminternalcameracalibration.Show
End Sub
HalconForm - 5
Private Sub mnumanual3Dcalculation_Click()
frmshapemodelcreation.Show
End Sub
CreateShapeModel - 1
'
' File generated by HDevelop for HALCON/COM (Visual Basic) Version 6.1
'
' This file is intended to be used with the project HDevelopTemplate
' which can be found in %HALCONROOT%\examples\vb\HDevelopTemplate\
'
Option Explicit
Dim Op As New HOperatorSetX
Sub RunHalcon(ByRef Window As HWindowX)
Dim Tuple As New HTupleX
Dim hx_WindowHandle As Variant, hx_R1 As Variant
Dim hx_C1 As Variant, hx_R2 As Variant
Dim hx_C2 As Variant, hx_HModel As Variant
Dim hx_IModel As Variant, hx_Row1 As Variant
Dim hx_Column1 As Variant, hx_Angle1 As Variant
Dim hx_Score1 As Variant, hx_Row As Variant
Dim hx_Column As Variant, hx_Angle As Variant
Dim hx_Score As Variant, hx_FGHandle As Variant
Dim hx_FGHandle1 As Variant, hx_Row2 As Variant
Dim hx_Column2 As Variant, hx_Radius As Variant
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_hrow As Variant, hx_hcolumn As Variant
Dim hx_hradius As Variant, hx_irow As Variant
Dim hx_icolumn As Variant, hx_iradius As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_H As HUntypedObjectX, hx_I As HUntypedObjectX
Dim hx_hroi As HUntypedObjectX, hx_iroi As HUntypedObjectX
Dim hx_HIR As HUntypedObjectX, hx_IIR As HUntypedObjectX
Dim hx_HMI As HUntypedObjectX, hx_HMR As HUntypedObjectX
Dim hx_IMI As HUntypedObjectX, hx_IMR As HUntypedObjectX
Dim hx_H1 As HUntypedObjectX, hx_I1 As HUntypedObjectX
Dim hx_iImage As HUntypedObjectX, hx_himage As HUntypedObjectX
Dim hx_ROI As HUntypedObjectX, hx_hroi As HUntypedObjectX
Dim hx_iroi As HUntypedObjectX, hx_himage As HUntypedObjectX
hx_BasDefWinHandle = Window.HalconID
```

Appendix A (Continued)

```
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.GrabImage(hx_himage, hx_fghitachi)
' read_image (H, 'c:/VisionSystem/ImageCapture/HHandle.jpg')
' read_image (I, 'c:/VisionSystem/ImageCapture/IHandle.jpg')
' dev_open_window(...)
' Please note: In Visual Basic the zooming of images will not be
adjusted automatically (like in H
Develop).
' Therefore you have to call 'op.SetPart()' with the parameters of the
image you want to display.
Call Op.DispObj(hx_himage _
, hx_BasDefWinHandle)
' draw_rectangle1 (WindowHandle, R1, C1, R2, C2)
Call Op.DrawCircle(hx_BasDefWinHandle, hx_hrow, hx_hcolumn, hx_hradius)
Call Op.GenCircle(hx_hroi, hx_hrow, hx_hcolumn, hx_hradius)
' Please note: In Visual Basic the zooming of images will not be
adjusted automatically (like in H
Develop).
' Therefore you have to call 'op.SetPart()' with the parameters of the
image you want to display.
Call Op.DispObj(hx_iImage _
, hx_BasDefWinHandle)
Call Op.DrawCircle(hx_BasDefWinHandle, hx_irow, hx_icolumn, hx_iradius)
Call Op.GenCircle(hx_iroi, hx_irow, hx_icolumn, hx_iradius)
' gen_rectangle1 (HROI, R1, C1, R2, C2)
' gen_rectangle1 (IROI, 139, 130, 260, 258)
Call Op.ReduceDomain(hx_himage, hx_hroi, hx_HIR)
Call Op.ReduceDomain(hx_iImage, hx_iroi, hx_IIR)
Call Op.InspectShapeModel(hx_HIR, hx_HMI, hx_HMR, 10, 120)
CreateShapeModel - 2
Call Op.InspectShapeModel(hx_IIR, hx_IMI, hx_IMR, 4, 120)
Call Op.CreateShapeModel(hx_HIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_HModel)
Call Op.CreateShapeModel(hx_IIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_IModel)
Call Op.WriteShapeModel(hx_HModel, "c:/VisionSystem/Cross Griph.sm")
Call Op.WriteShapeModel(hx_IModel, "c:/VisionSystem/Cross Gripi.sm")
Call Op.ReadImage(hx_H1, "c:/VisionSystem/ImageCapture/H.jpg")
Call Op.ReadImage(hx_I1, "c:/VisionSystem/ImageCapture/I.jpg")
Call Op.FindShapeModel(hx_H1, hx_HModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row1, h
x_Column1, hx_Angle1, hx_Score1)
Call Op.FindShapeModel(hx_I1, hx_IModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row, hx
_Column, hx_Angle, hx_Score)
End Sub
```

Appendix A (Continued)

```
Sub InitHalcon()
' Default settings used in HDevelop
Call Op.SetSystem("do_low_error", "false")
End Sub
GrabSaveImage - 1
'
' File generated by HDevelop for HALCON/COM (Visual Basic) Version 6.1
'
' This file is intended to be used with the project HDevelopTemplate
' which can be found in %HALCONROOT%\examples\vb\HDevelopTemplate\
'
Option Explicit
Dim Op As New HOperatorSetX
Sub ImageCapture(ByRef Window As HWindowX, Window2 As HWindowX, i As
Integer, Dateiname As String)
Dim Tuple As New HTupleX
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_whhitachi As Variant, hx_whimagingsource As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_BasDefWinHandle2 As Variant
Dim hx_himage As HUntypedObjectX, hx_iImage As HUntypedObjectX
hx_BasDefWinHandle = Window.HalconID
hx_BasDefWinHandle2 = Window2.HalconID
' First, make sure all framegrabbers are closed
Call Op.CloseAllFramegrabbers
' Open both Framegrabbers
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
' Grab an Image "a" Times in Each Framegrabber
Dim a As Integer
For a = 1 To i
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle2)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Call Op.WriteImage(hx_himage, "jpeg", 0,
"c:/VisionSystem/ImageCapture/" & Dateiname & "Hitach
iImage" & a & ".jpg")
Call Op.WriteImage(hx_iImage, "jpeg", 0,
"c:/VisionSystem/ImageCapture/" & Dateiname & "Imagin
gSourceImage" & a & ".jpg")
Next a
Call Op.CloseAllFramegrabbers
End Sub
'Sub InitHalcon()
' Default settings used in HDevelop
' Call Op.SetSystem("do_low_error", "false")
'End Sub
```

Appendix A (Continued)

```
InternalCalibration - 1
'
' File generated by HDevelop for HALCON/COM (Visual Basic) Version 6.1
'
' This file is intended to be used with the project HDevelopTemplate
' which can be found in %HALCONROOT%\examples\vb\HDevelopTemplate\
'
Option Explicit
Dim Op As New HOperatorSetX
Sub InternalCameraCalibration(ByRef Window As HWindowX, check As
Integer, valsizegauss As Integer _
, valmarkthresh As Integer, valmindiammarks As Integer, valstartthresh
As Integer _
, valdeltathresh As Integer, valminthresh As Integer, valalpha As Long,
valmincontlength _
As Integer, valmaxdiammarks As Integer)
Dim hx_BasDefWinHandle As Variant
hx_BasDefWinHandle = Window.HalconID
Dim HTuple As New HTupleX
Dim ITuple As New HTupleX
Dim HRowTuple As New HTupleX
Dim HColTuple As New HTupleX
Dim IRowTuple As New HTupleX
Dim IColTuple As New HTupleX
Dim HSPTuple As New HTupleX
Dim ISPTuple As New HTupleX
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_hRCoord1 As Variant, hx_hCCoord1 As Variant
Dim hx_IRCoord1 As Variant, hx_ICCoord1 As Variant
Dim hx_hStartPose1 As Variant
Dim hx_IStartPose1 As Variant
Dim hx_hRCoord2 As Variant, hx_hCCoord2 As Variant
Dim hx_IRCoord2 As Variant, hx_ICCoord2 As Variant
Dim hx_hStartPose2 As Variant
Dim hx_IStartPose2 As Variant
Dim hx_hRCoord3 As Variant, hx_hCCoord3 As Variant
Dim hx_IRCoord3 As Variant, hx_ICCoord3 As Variant
Dim hx_hStartPose3 As Variant
Dim hx_IStartPose3 As Variant
Dim hx_hRCoord4 As Variant, hx_hCCoord4 As Variant
Dim hx_IRCoord4 As Variant, hx_ICCoord4 As Variant
Dim hx_hStartPose4 As Variant
Dim hx_IStartPose4 As Variant
Dim hx_hRCoord5 As Variant, hx_hCCoord5 As Variant
Dim hx_IRCoord5 As Variant, hx_ICCoord5 As Variant
Dim hx_hStartPose5 As Variant
Dim hx_IStartPose5 As Variant
Dim hx_hRCoord6 As Variant, hx_hCCoord6 As Variant
Dim hx_IRCoord6 As Variant, hx_ICCoord6 As Variant
Dim hx_hStartPose6 As Variant
Dim hx_IStartPose6 As Variant
Dim hx_hRCoord7 As Variant, hx_hCCoord7 As Variant
```

Appendix A (Continued)

```
Dim hx_IRCoord7 As Variant, hx_ICCoord7 As Variant
Dim hx_hStartPose7 As Variant
Dim hx_IStartPose7 As Variant
Dim hx_hRCoord8 As Variant, hx_hCCoord8 As Variant
Dim hx_IRCoord8 As Variant, hx_ICCoord8 As Variant
Dim hx_hStartPose8 As Variant
Dim hx_IStartPose8 As Variant
InternalCalibration - 2
Dim hx_hRCoord9 As Variant, hx_hCCoord9 As Variant
Dim hx_IRCoord9 As Variant, hx_ICCoord9 As Variant
Dim hx_hStartPose9 As Variant
Dim hx_IStartPose9 As Variant
' Dim hx_BasDefWinHandle As Variant
Dim hx_iImage As HUntypedObjectX, hx_himage As HUntypedObjectX
Dim hx_Image1 As HUntypedObjectX, hx_HCaltab As HUntypedObjectX
Dim hx_ICaltab As HUntypedObjectX
Dim hx_HCamParam As Variant, hx_ICamParam As Variant
Dim hx_NX As Variant
Dim hx_NY As Variant
Dim hx_NZ As Variant
Dim hx_HHCamParam As Variant, hx_HNFinalPose As Variant
Dim hx_IICamParam As Variant, hx_INFinalPose As Variant
Dim hx_HErrors As Variant
Dim hx_IErrors As Variant
Dim i As Integer
Dim a As Integer
Call Op.CloseAllFramegrabbers
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
' For i = 1 To 20
' valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiammarks
' Grab a set of images to get rid of initial distortion
If check = 1 Then
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480))))))) _
```

Appendix A (Continued)

```
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord1, hx_hCCoord1, hx_hStartPose1)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480)))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord2, hx_hCCoord2, hx_hStartPose2)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
InternalCalibration - 3
TupleConcat(640, 480)))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord3, hx_hCCoord3, hx_hStartPose3)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
```

Appendix A (Continued)

```
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480))))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord4, hx_hCCoord4, hx_hStartPose4)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480))))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord5, hx_hCCoord5, hx_hStartPose5)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480))))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord6, hx_hCCoord6, hx_hStartPose6)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
```

Appendix A (Continued)

```
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord7, hx_hCCoord7, hx_hStartPose7)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480)))))))  _
InternalCalibration - 4
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord8, hx_hCCoord8, hx_hStartPose8)
Call Op.DispObj(hx_himage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_himage, hx_fghitachi)
Call Op.FindCaltab(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_himage, hx_HCaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, HTuple.TupleConcat(0.008, HTuple.TupleConcat(0#,
HTuple.TupleConcat(0.000011 _
, HTuple.TupleConcat(0.000011, HTuple.TupleConcat(320,
HTuple.TupleConcat(240, HTuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_hRCoord9, hx_hCCoord9, hx_hStartPose9)
Call
Op.ReadCamPar("c:/VisionSystem/InternalCameraCalibration/campar.dat",
hx_HCamParam)
Call Op.CaltabPoints("C:/VisionSystem/CalibrationTable/caltab.descr",
hx_NX, hx_NY, hx_NZ)
Call Op.CameraCalibration(hx_NX, hx_NY, hx_NZ _
```

Appendix A (Continued)

```
, HRowTuple.TupleConcat(hx_hRCoord1, HRowTuple.TupleConcat(hx_hRCoord2,
HRowTuple.Tupl
eConcat(hx_hRCoord3, HRowTuple.TupleConcat(hx_hRCoord4 _
, HRowTuple.TupleConcat(hx_hRCoord5, HRowTuple.TupleConcat(hx_hRCoord6,
HRowTuple.Tupl
eConcat(hx_hRCoord7, HRowTuple.TupleConcat(hx_hRCoord8 _
, hx_hRCoord9))))))))) _
, HColTuple.TupleConcat(hx_hCCoord1, HColTuple.TupleConcat(hx_hCCoord2,
HColTuple.Tupl
eConcat(hx_hCCoord3, HColTuple.TupleConcat(hx_hCCoord4 _
, HColTuple.TupleConcat(hx_hCCoord5, HColTuple.TupleConcat(hx_hCCoord6,
HColTuple.Tupl
eConcat(hx_hCCoord7, HColTuple.TupleConcat(hx_hCCoord8 _
, hx_hCCoord9))))))))) _
, hx_HCamParam _
, HSPTuple.TupleConcat(hx_hStartPose1,
HSPTuple.TupleConcat(hx_hStartPose2, HSPTuple.T
upleConcat(hx_hStartPose3, HSPTuple.TupleConcat(hx_hStartPose4 _
, HSPTuple.TupleConcat(hx_hStartPose5,
HSPTuple.TupleConcat(hx_hStartPose6, HSPTuple.T
upleConcat(hx_hStartPose7, HSPTuple.TupleConcat(hx_hStartPose8 _
, hx_hStartPose9))))))))) _
, "all", hx_HHCamParam, hx_HNFinalPose, hx_HErrors)
Call Op.WriteCamPar(hx_HHCamParam,
"c:/VisionSystem/InternalCameraCalibration/hCamParam.da
t")
End If
If check = 2 Then
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480))))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord1, hx_ICCoord1, hx_IStartPose1)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
```

Appendix A (Continued)

```
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord2, hx_ICCoord2, hx_IStartPose2)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
InternalCalibration - 5
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord3, hx_ICCoord3, hx_IStartPose3)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord4, hx_ICCoord4, hx_IStartPose4)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
```

Appendix A (Continued)

```
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord5, hx_ICCoord5, hx_IStartPose5)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord6, hx_ICCoord6, hx_IStartPose6)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))) _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord7, hx_ICCoord7, hx_IStartPose7)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
```

Appendix A (Continued)

```
InternalCalibration - 6
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord8, hx_ICCoord8, hx_IStartPose8)
Call Op.DispObj(hx_iImage, hx_BasDefWinHandle)
Beep
Call Op.WaitSeconds(5)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.FindCaltab(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.DESCR",
valsizegauss, valmarkthresh, valmindiammarks)
Call Op.FindMarksAndPose(hx_iImage, hx_ICaltab,
"c:/VisionSystem/CalibrationTable/caltab.d
escr" _
, ITuple.TupleConcat(0.008, ITuple.TupleConcat(0#,
ITuple.TupleConcat(0.000011 _
, ITuple.TupleConcat(0.000011, ITuple.TupleConcat(320,
ITuple.TupleConcat(240, ITuple.
TupleConcat(640, 480)))))))  _
, valstartthresh, valdeltathresh, valminthresh, valalpha,
valmincontlength, valmaxdiam
marks, hx_IRCoord9, hx_ICCoord9, hx_IStartPose9)
Call
Op.ReadCamPar("c:/VisionSystem/InternalCameraCalibration/campar.dat",
hx_ICamParam)
Call Op.CaltabPoints("C:/VisionSystem/CalibrationTable/caltab.descr",
hx_NX, hx_NY, hx_NZ)
Call Op.CameraCalibration(hx_NX, hx_NY, hx_NZ _
, IRowTuple.TupleConcat(hx_IRCoord1, IRowTuple.TupleConcat(hx_IRCoord2,
IRowTuple.Tupl
eConcat(hx_IRCoord3, IRowTuple.TupleConcat(hx_IRCoord4 _
, IRowTuple.TupleConcat(hx_IRCoord5, IRowTuple.TupleConcat(hx_IRCoord6,
IRowTuple.Tupl
eConcat(hx_IRCoord7, IRowTuple.TupleConcat(hx_IRCoord8 _
, hx_IRCoord9)))))))) _
, IColTuple.TupleConcat(hx_ICCoord1, IColTuple.TupleConcat(hx_ICCoord2,
IColTuple.Tupl
eConcat(hx_ICCoord3, IColTuple.TupleConcat(hx_ICCoord4 _
```

Appendix A (Continued)

```
, IColTuple.TupleConcat(hx_ICCoord5, IColTuple.TupleConcat(hx_ICCoord6,
IColTuple.Tupl
eConcat(hx_ICCoord7, IColTuple.TupleConcat(hx_ICCoord8 _
, hx_ICCoord9))))))) _
, hx_ICamParam _
, ISPTuple.TupleConcat(hx_IStartPose1,
ISPTuple.TupleConcat(hx_IStartPose2, ISPTuple.T
upleConcat(hx_IStartPose3, ISPTuple.TupleConcat(hx_IStartPose4 _
, ISPTuple.TupleConcat(hx_IStartPose5,
ISPTuple.TupleConcat(hx_IStartPose6, ISPTuple.T
upleConcat(hx_IStartPose7, ISPTuple.TupleConcat(hx_IStartPose8 _
, hx_IStartPose9))))))) _
, "all", hx_IICamParam, hx_INFinalPose, hx_IErrors)
Call Op.WriteCamPar(hx_IICamParam,
"c:/VisionSystem/InternalCameraCalibration/ICamParam.da
t")
End If
Call Op.CloseAllFramegrabbers
End Sub
Sub InitHalcon()
' Default settings used in HDevelop
Call Op.SetSystem("do_low_error", "false")
End Sub
Module1 - 1
'
' File generated by HDevelop for HALCON/COM (Visual Basic) Version 6.1
'
' This file is intended to be used with the project HDevelopTemplate
' which can be found in %HALCONROOT%\examples\vb\HDevelopTemplate\
'
Option Explicit
Dim Op As New HOperatorSetX
Sub RunHalcon(ByRef Window As HWindowX)
Dim Tuple As New HTupleX
Dim hx_WindowHandle As Variant, hx_R1 As Variant
Dim hx_C1 As Variant, hx_R2 As Variant
Dim hx_C2 As Variant, hx_HModel As Variant
Dim hx_IModel As Variant, hx_Row1 As Variant
Dim hx_Column1 As Variant, hx_Angle1 As Variant
Dim hx_Score1 As Variant, hx_Row As Variant
Dim hx_Column As Variant, hx_Angle As Variant
Dim hx_Score As Variant, hx_FGHandle As Variant
Dim hx_FGHandle1 As Variant, hx_Row2 As Variant
Dim hx_Column2 As Variant, hx_Radius As Variant
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_hrow As Variant, hx_hcolumn As Variant
Dim hx_hradius As Variant, hx_irow As Variant
Dim hx_icolumn As Variant, hx_iradius As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_H As HUntypedObjectX, hx_I As HUntypedObjectX
Dim hx_hroi As HUntypedObjectX, hx_iroi As HUntypedObjectX
Dim hx_HIR As HUntypedObjectX, hx_IIR As HUntypedObjectX
```

Appendix A (Continued)

```
Dim hx_HMI As HUntypedObjectX, hx_HMR As HUntypedObjectX
Dim hx_IMI As HUntypedObjectX, hx_IMR As HUntypedObjectX
Dim hx_H1 As HUntypedObjectX, hx_I1 As HUntypedObjectX
Dim hx_iImage As HUntypedObjectX, hx_himage As HUntypedObjectX
Dim hx_ROI As HUntypedObjectX, hx_hroi As HUntypedObjectX
Dim hx_iroi As HUntypedObjectX, hx_himage As HUntypedObjectX
hx_BasDefWinHandle = Window.HalconID
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
Call Op.GrabImage(hx_iImage, hx_fgimagingsource)
Call Op.GrabImage(hx_himage, hx_fghitachi)
' read_image (H, 'c:/VisionSystem/ImageCapture/HHandle.jpg')
' read_image (I, 'c:/VisionSystem/ImageCapture/IHandle.jpg')
' dev_open_window(...)
' Please note: In Visual Basic the zooming of images will not be
adjusted automatically (like in H
Develop).
' Therefore you have to call 'op.SetPart()' with the parameters of the
image you want to display.
Call Op.DispObj(hx_himage _
, hx_BasDefWinHandle)
' draw_rectangle1 (WindowHandle, R1, C1, R2, C2)
Call Op.DrawCircle(hx_BasDefWinHandle, hx_hrow, hx_hcolumn, hx_hradius)
Call Op.GenCircle(hx_hroi, hx_hrow, hx_hcolumn, hx_hradius)
' Please note: In Visual Basic the zooming of images will not be
adjusted automatically (like in H
Develop).
' Therefore you have to call 'op.SetPart()' with the parameters of the
image you want to display.
Call Op.DispObj(hx_iImage _
, hx_BasDefWinHandle)
Call Op.DrawCircle(hx_BasDefWinHandle, hx_irow, hx_icolumn, hx_iradius)
Call Op.GenCircle(hx_iroi, hx_irow, hx_icolumn, hx_iradius)
' gen_rectangle1 (HROI, R1, C1, R2, C2)
' gen_rectangle1 (IROI, 139, 130, 260, 258)
Call Op.ReduceDomain(hx_himage, hx_hroi, hx_HIR)
Call Op.ReduceDomain(hx_iImage, hx_iroi, hx_IIR)
Call Op.InspectShapeModel(hx_HIR, hx_HMI, hx_HMR, 10, 120)
Module1 - 2
Call Op.InspectShapeModel(hx_IIR, hx_IMI, hx_IMR, 4, 120)
Call Op.CreateShapeModel(hx_HIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_HModel)
Call Op.CreateShapeModel(hx_IIR, 5, 0, 3.14, 0.0175, "none",
"use_polarity", 120, 80, hx_IModel)
Call Op.WriteShapeModel(hx_HModel, "c:/VisionSystem/Cross Griph.sm")
Call Op.WriteShapeModel(hx_IModel, "c:/VisionSystem/Cross Gripi.sm")
Call Op.ReadImage(hx_H1, "c:/VisionSystem/ImageCapture/H.jpg")
Call Op.ReadImage(hx_I1, "c:/VisionSystem/ImageCapture/I.jpg")
```

Appendix A (Continued)

```
Call Op.FindShapeModel(hx_H1, hx_HModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row1, h
x_Column1, hx_Angle1, hx_Score1)
Call Op.FindShapeModel(hx_I1, hx_IModel, 0, 3.14, 0.3, 1, 0.5,
"interpolation", 0, 0.1, hx_Row, hx
_Column, hx_Angle, hx_Score)
End Sub
Sub InitHalcon()
' Default settings used in HDevelop
Call Op.SetSystem("do_low_error", "false")
End Sub
ObtainShapeModel - 1
Option Explicit
Dim Op As New HOperatorSetX
Sub ObtainShapeModel(ByRef Window1 As HWindowX, Window2 As HWindowX)
Dim Tuple As New HTupleX
Dim hx_fghitachi As Variant, hx_fgimagingsource As Variant
Dim hx_whhitachi As Variant, hx_whimagingsource As Variant
Dim hx_BasDefWinHandle As Variant
Dim hx_BasDefWinHandle2 As Variant
Dim hx_himage As HUntypedObjectX, hx_iImage As HUntypedObjectX
' hx_BasDefWinHandle = Window.HalconID
' hx_BasDefWinHandle2 = Window2.HalconID
' Dim hx_FGHandle As Variant, hx_ModelID1 As Variant
' Dim hx_ModelID2 As Variant, hx_ModelID As Variant
' Dim hx_i As Variant, hx_Row As Variant
' Dim hx_Column As Variant, hx_Angle As Variant
' Dim hx_Scale As Variant, hx_Score As Variant
' Dim hx_Row1 As Variant, hx_Column1 As Variant
' Dim hx_Angle1 As Variant, hx_Scale1 As Variant
' Dim hx_Score1 As Variant, hx_Row2 As Variant
' Dim hx_Column2 As Variant, hx_Angle2 As Variant
' Dim hx_Scale2 As Variant, hx_Score2 As Variant
' Dim hx_Image As HUntypedObjectX
' hx_BasDefWinHandle = Window.HalconID
' First make sure all Framegrabbers are closed
Op.CloseAllFramegrabbers
' Now, open both Framegrabbers
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "0", 0, -1, hx_fghitachi)
Call Op.OpenFramegrabber("DFG-LC", 1, 1, 0, 0, 0, 0, "default", -1,
"default", -1, "default", "nts
c", "2", 0, -1, hx_fgimagingsource)
'Call Op.ReadShapeModel("c:/Michael/Pic/Cross1.smd", hx_ModelID1)
'Call Op.ReadShapeModel("c:/Michael/Pic/Cross2.smd", hx_ModelID2)
'Call Op.ReadShapeModel("c:/Michael/Crossobject.smd", hx_ModelID)
'Dim end_val4 As Variant
'Dim step_val4 As Variant
'end_val4 = 200
'step_val4 = 1
' For hx_i = 1 To end_val4 Step step_val4
```

Appendix A (Continued)

```
' Call Op.GrabImage(hx_Image, hx_FGHandle)
' Call Op.FindScaledShapeModel(hx_Image, hx_ModelID2, 0, 3.14, 0.5,
1.1, 0.3, 1, 0.9, "interpol
ation", 0, 0.2, hx_Row, hx_Column, hx_Angle, _
' hx_Scale, hx_Score)
' If Tuple.TupleGreater(hx_Scale, 1#) Then
' Call Op.FindScaledShapeModel(hx_Image, hx_ModelID1, 0, 3.14, 0.5,
1.5, 0.3, 1, 0.9, "inte
rpolation", 0, 0.2, hx_Row1, hx_Column1, hx_Angle1, _
' hx_Scale1, hx_Score1)
' If Tuple.TupleGreater(hx_Scale1, 1#) Then
' Call Op.FindScaledShapeModel(hx_Image, hx_ModelID, 0, 3.14, 0.5, 1.1,
0.3, 1, 0.9, "i
nterpolation", 0, 0.2, hx_Row2, hx_Column2, _
' hx_Angle2, hx_Scale2, hx_Score2)
' End If
' End If
' Please note: The call of DoEvents() is only a hack to
' enable VB to react on events. Please change the code
' so that it can handle events in a standard way.
DoEvents
' Next
ObtainShapeModel - 2
End Sub
Sub InitHalcon()
' Default settings used in HDevelop
Call Op.SetSystem("do_low_error", "false")
End Sub
```