Development of an End-effector Sensory Suite for a Rehabilitation Robot

by

Stephanie A. Stiber

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

Major Professor: Rajiv Dubey, Ph.D. Craig Lusk, Ph.D. Shuh-Jing Ying, Ph.D.

Date of Approval: July 19, 2006

Keywords: Barrett hand, camera, sensors, MatLab, C++, vision, laser range finder

© Copyright 2006, Stephanie A. Stiber
Dedication

I would like to dedicate this work to the memory of my late mother, Judith Stiber.
Acknowledgments

First and foremost, I would like to thank Dr. Rajiv Dubey for making this wonderful experience possible. Without his gracious financial and intellectual support, this project would have not been possible. I would like to thank Dr. Craig Lusk and Dr. Shuh-Jing Ying for their guidance and recommendations during this project. I would also like to recognize the Mechanical Engineering department, faculty and staff for offering such a wonderful program where young engineers can really grow and learn.

Next, I would like thank Eduardo Veras. Thank you so much for your patience and for teaching me C++, MatLab, and basically everything computer related. Without you, this project would have been near impossible. I would like to thank Norali Pernalete for her assistance in editing this work. Everyone else is the lab, thank you for your helpful suggestions, constant cooperation and guidance. It is amazing to work in a lab like this, where great things are made and developed for the well-being of others.

I want to thank my husband, Ryan McKeon, for his understanding of my late hours and awkward schedule. I would like to thank the rest of my family members for their constant support and understanding of my goals. Thank you.
# Table of Contents

List of Tables iii

List of Figures iv

Abstract vi

Chapter One Introduction 1
  1.1 Motivation 1
  1.2 Thesis Objective 2
  1.3 Contributions 2
  1.4 Thesis Outline 3
  1.5 Importance 3

Chapter Two Background 4
  2.1 Disabilities Worldwide 4
  2.2 Rehabilitation Arms 5
    2.2.1 Manus 6
    2.2.2 Raptor 8
    2.2.3 Kares II 8
    2.2.4 WMRA 9
  2.3 Robot End-effectors 11
  2.4 Vision Applications in Rehabilitation 11

Chapter Three Development of the Sensor Suite 14
  3.1 Selection Criteria 14
  3.2 Experimental Platform 15
    3.2.1 Puma Arm 15
    3.2.2 Camera Specifications 22
    3.2.3 Laser Range Finder 26
    3.2.4 BarrettHand End-effector 29

Chapter Four System Integration 33
  4.1 End-effector Integration 33
  4.2 SICK Laser Range Finder 36
  4.3 Camera Set-up 37
  4.4 Image Processing 43
  4.5 Integration Platform 47
  4.6 Operating Procedure 55
List of Tables

Table 1: Number of Persons Using Assistive Technology Devices  5
Table 2: D-H Parameters of Puma Arm  18
Table 3: Technical Specifications of the Bumblebee Stereovision Camera  23
Table 4: Logitech Camera Specifications  24
Table 5: Technical Specifications of the Sony Camera  25
Table 6: Product Specifications for Creative Lab Camera  26
Table 7: DME Product Features  27
Table 8: Technical Data  27
Table 9: BarrettHand™ Specifications  32
Table 10: Common GCL Commands  34
Table 11: Calibration Data for Sony Camera  40
Table 12: Creative Camera Calibration  42
Table 13: Before and After Image Processing  58
Table 14: Actual vs Calculated Centroid in x,y, z  60
Table 15: Creative Camera Calibration Error  61
Table 16: C++ Code Definitions  73
Table 17: MATLAB Image Acquisition Commands  82
Table 18: Common MATLAB Image Processing Commands  83
List of Figures

Figure 1: Manus Arm 7
Figure 2: Manus End-effector 7
Figure 3: Raptor Arm 8
Figure 4: Kares II Robot System 9
Figure 5: New Wheelchair Arm 10
Figure 6: Two Camera Setup 12
Figure 7: The Puma Arm 15
Figure 8: Simple Link 16
Figure 9: Puma Arm Coordinate Configuration and Link Assignment 17
Figure 10: Bumblebee 22
Figure 11: Web Cam 24
Figure 12: Sony CCD Camera 25
Figure 13: Creative Labs Web Camera for Notebooks 26
Figure 14: DME2000 Dimension Drawing 29
Figure 15: BarrettHand End-effector 30
Figure 16: BarrettHands Grasping Abilities 30
Figure 17: Independent Finger Movement 31
Figure 18: MATLAB Command Flowchart 35
Figure 19: Laser Range Finder 36
Development of an End-effector Sensory Suite for a Rehabilitation Robot

Stephanie A. Stiber

ABSTRACT

This research presents an approach in assisting the control and operation of a rehabilitation robot manipulator to execute simple grasping tasks for persons with severe disabilities. It outlines the development of an end-effector sensory suite that includes the BarrettHand end-effector, laser range finder, and a low cost camera.

The approach taken in this research differs greatly from the currently available rehabilitation robot arms in that it requires minimal user instruction, it is easy to operate and more effective for persons severely disabled. A thorough study of the currently available systems; Manus, Raptor and Kares II arm, is also presented.

In order to test the end-effector sensory suite, experiments were performed to find the centroid of an object of interest to direct the robot end-effector towards it with minimal error. Analyses of centroid location data to ensure accurate results are also presented.
The long term goal of this research is to significantly enhance the ability of severely disabled persons to perform activities of daily living using wheelchair mounted robot arms. The sensory suite developed through this project is expected to be integrated into a seven-degree of freedom wheelchair mounted robot arm currently under development at the Rehabilitation Robots Laboratory at the University of South Florida.
Chapter One
Introduction

1.1 Motivation

According to the Census Bureau in 2002, 2.2 million people in the United States over the age of 15 use a wheelchair. Furthermore, 18 million have had difficulty lifting and carrying a ten pound bag of groceries or grasping small objects [1]. A significant number of these people are severely disabled and unable to manipulate objects to perform activities of daily living. With this arises a need for a user-friendly end-effector that can be used while attached to a wheelchair-mounted robot arm. A user-friendly end-effector is needed in order to grasp everyday items, such as water bottles, pens, or even silverware.

People with disabilities often have difficulties navigating through their surroundings. This can make activities of daily living extremely difficult and frustrating. Some activities of daily living include: bathing, dressing, walking, eating, toilet-use, grooming, and transferring from a bed to a chair [2]. People confined to a wheelchair have a reach limited to a semicircle with the radius being no longer than the length of their arm. The purpose of this research is to extend the workspace and allow people with disabilities to perform simple daily tasks. While there are devices that do this as well, these devices are often too complex to control. By integrating vision recognition and a laser range finder with a robot end-effector, the control of a robot arm becomes manageable.
1.2 Thesis Objectives

- Develop an end-effector sensor suite to assist in the control of a rehabilitation robot. This includes:
  - Programming and integration of BarrettHand end-effector to the Puma 560 robot
  - Selection and integration of a laser range finder with the manipulator
  - Selection and integration of a vision system with the manipulator
- Perform experiments with the sensory suite integrated with the manipulator on the arm. This includes:
  - Using the end-effector sensor suite’s assistance in finding the centroid of an object
  - Analyzing object centroid location data to ensure accurate results are obtained
  - Develop algorithms for moving the robot hand to grasp the object

1.3 Contribution

The contribution of this research is the unique combination of computer vision and laser range finder technology integrated with a state of the art robot hand. This integration will aide in the execution of tasks with minimal user input.

The approach taken uses the BarrettHand end-effector with some sensory assistance. These sensors include a laser range finder and a basic camera.

The BarrettHand is a relatively new robot end-effector, and while it has been integrated with cameras, it has never been integrated within the realm of
study of rehabilitation robots. The end-effector sensor suite will allow the hand assembly to transfer from one robot arm to another with ease.

1.4 Thesis Outline

The thesis is outlined as follows: Chapter Two contains background information on rehabilitation robot arms, end-effectors, and vision applications in robots. Chapter Three serves as an overview of the design procedure; it discusses the criteria for selection, product specifications and manipulator details. Chapter Four provides a description of the integration of the robot arm with the end-effector, laser range finder, BarrettHand, and the camera system. Finally, Chapter Five contains the results, discussion, conclusion, and further recommendations for future research.

1.5 Importance

Similar projects have been conducted worldwide, but most have not been applicable towards rehabilitation engineering or servicing people with disabilities. This project integrates a vision system, laser range finder, and advances robotic gripper. The end result will be the calculation of the centroid of an object of interest in order to allow an end-effector to automatically work with minimal user interaction. The long term goal of this research will be to enable a robot arm to view an object and allow a person with severe disability to grasp and manipulate it.
Chapter Two
Background

2.1 Disabilities Worldwide

At a United Nation’s meeting, it was agreed that disability is multidimensional; thus, they could not ascertain the single true size of the disabled population. Different symptoms are related to different levels of disability [3]. The level of disability that this research will benefit is any permanently wheelchair-bound person. This device should be able to be used by someone with full use of upper limbs as well as someone with limited upper extremity mobility.

The following Table gives the demographics of disabilities taken by the National Center for Health Statistics in 1994 [4]. The Table discusses the devices used, the age of the user, and amount of users.
Table 1: Number of Persons Using Assistive Technology Devices by Age of Person and Device: United States 1994 [4]

<table>
<thead>
<tr>
<th>Assistive Device</th>
<th>All ages</th>
<th>44 years and under</th>
<th>45-64 years</th>
<th>65 years and over</th>
</tr>
</thead>
<tbody>
<tr>
<td>Any artificial limb</td>
<td>199</td>
<td>69</td>
<td>59</td>
<td>70</td>
</tr>
<tr>
<td>Artificial leg or foot</td>
<td>173</td>
<td>58</td>
<td>50</td>
<td>65</td>
</tr>
<tr>
<td>Artificial arm or hand</td>
<td>*21</td>
<td>*9</td>
<td>*6</td>
<td>*6</td>
</tr>
<tr>
<td>Any mobility device**</td>
<td>7,394</td>
<td>1,151</td>
<td>1,699</td>
<td>4,544</td>
</tr>
<tr>
<td>Wheelchair</td>
<td>1,564</td>
<td>335</td>
<td>365</td>
<td>863</td>
</tr>
<tr>
<td>Any vision device**</td>
<td>527</td>
<td>123</td>
<td>135</td>
<td>268</td>
</tr>
<tr>
<td>Braille</td>
<td>59</td>
<td>*28</td>
<td>*23</td>
<td>*8</td>
</tr>
<tr>
<td>Computer equipment</td>
<td>*34</td>
<td>*19</td>
<td>*8</td>
<td>*7</td>
</tr>
</tbody>
</table>

Table 1 demonstrates that there is a large population of mobility challenged individuals who may be helped by this technology. This device has the potential to help the mobility challenged individuals. This research can help people with an artificial arm or hand and also people permanently bound to wheelchairs. This allows for the possibility of helping over one million people.

2.2 Rehabilitation Arms

Rehabilitation engineering originated because of a need for assistive devices for people with severe disabilities. An overview of these technologies is
discussed by Lancioni et al [5]. The conclusion of this overview was that the majority of available resources for someone with disabilities have been aimed at promoting a disabled person’s direct access to or request of environmental stimulation. These resources have also been directed towards supporting and increasing a person’s orientation and mobility while reducing their accidental injury rate. The resources that are most commonly available to people with disabilities are micro switches and speech output devices. Other devices, such as robot limbs, are less accessible because they are difficult to implement.

2.2.1 Manus

The Manus is a fully functional wheelchair-mounted robot manipulator that has been built in the Netherlands. The goal of this wheelchair-mounted robot arm is to provide the disabled with a greater level of personal independence. The Manus has a simple controller and is commercially available through Exact Dynamics. The Manus is a 6 degree of freedom robot arm which is mounted on a rotating and telescoping base unit. This arm can be attached to a variety of electric wheelchairs and is capable of grasping up to 2.2 kg. According to Dallaway [6], the current versions of the Manus have a reach of approximately 850 mm.

A picture of the Manus arm is shown below in Figure 1. This arm has a few preset functions such as a home position and drinking function, allowing it to lift a glass while holding the water level even.
With the arm, a cup can be grasped, held level, and then brought to a person’s face. Once at the face, the arm has a preset drinking command. This command tilts the cup about an axis while adjusting the height to allow for easy drinking. Figure 2 shows a close up view of the Manus end-effector.

This end-effector contains two fingers with a rotating wrist. The commands for these fingers consist of simple open and close commands.
2.2.2 Raptor

The Raptor robot arm is another commercially available wheelchair-mounted robot arm. The Raptor is the first commercially available FDA-approved rehabilitative robot. The Raptor is controlled by a joystick or a sip-in-puff. It has a robot end-effector similar to that of Manus and is capable of lifting items off the floor. Figure 3 shows the Raptor arm attached to a wheelchair.

![Figure 3: Raptor Arm](image)

2.2.3 Kares II

Bien et al [7] created a robot system called Kares II (Korea Advances Institute of Science and Technology--KAIST Rehabilitation Engineering Service System II). This system was designed to out-perform the Manus and Raptor. This system used two experimental platforms, a mobile platform and a wheelchair platform. This device is capable of twelve major tasks; among these tasks is face
washing and retrieving a cup. Kares II was designed using task-oriented designs.

Figure 4 shows the shaving and face cleaning tasks.

Figure 4: Kares II Robot system

Figure 4 just shows one of the two operating platforms: a mobile unit not attached to a wheelchair. The Kares II robot comes with another platform that can be used on a mobile wheelchair. Researchers concluded that “Further study is needed to design a convenient operation methodology of the system on behalf of novice users and long-term handling. More sensitive and wide intention reading capability of various kinds is desirable for human-friendly interaction [7].”

2.2.4 WMRA

The next arm being discussed, WMRA (wheelchair mounted robot arm), currently does not have an end-effector. It is a product designed to out lift and out perform both the Raptor and the Manus wheelchair mounted robot arms. It is pictured in Figure 5. This arm was built in 2005 by Edwards [8].
This arm is a variable controlled arm with six main joints. This wheelchair was created at the University of South Florida in the Rehabilitation department of the College of Engineering.

This arm would be optimal because it has a greater payload capacity than the other available arms. The Manus arm only has a payload capacity of about 2.2 kg and the Raptor’s capacity is only 1.5 kg. This arm can easily support a load of over 10 kg. It can do this because of the motors used and the solid joint connections. This arm also has more degrees of freedom than that of the Manus, Raptor, and Puma arms. This allows the arm to go to one position in multiple ways, thereby imitating the human arm (this is called redundancy). As soon as cartesian control is implemented, this sensory suite will be switched over to this arm instead of the Puma arm.
2.3 Robot End-effectors

The end-effectors are essentially one of the most important aspects of a robot. Many available end effectors have a simple open or close function with minimal force feedback. One example of this is shown above in Figure 2. While this robot end-effector does have some amount of force feedback, the end-effector lacks the ability to grasp an object evenly. In addition, it does not offer any finger dexterity. The fingers do not bend to grasp an object evenly, making it easy for the object to simply fall through the grasp of the robot.

2.4 Vision Applications in Rehabilitation

There have been two vision based systems developed at the University of South Florida. One of the systems, developed by Fritz [9], consisted of a seven degree of freedom robot manipulator, an end-effector, a laser range finder, a vision system, and several other sensors. This work was an important basis for the inclusion of the laser range finder in the present study.

The system developed by Jurczyk [10] consisted of two cameras that were used to create a stereovision camera system programmed and calibrated to find a handle. The hardware utilized was a Hitachi KP-D50 CCD (charged couple display) camera, an Imaging Source CCD, and two Imaging Source DFG/LC1 frame grabber cards. This setup is shown in Figure 6.
In Martin’s research conducted at the Massachusetts Institute of Technology, programming was used to create a vision subsystem for obstacle avoidance [11]. In this research study, proximity sensors were utilized to cut down on the information processing for distance. This research provided a very important four-step outline to programming vision systems. Step one is to record, step two is to learn, step three is to build, and step four is to validate. This project is concerned with obstacle avoidance but provides a good basic outline for recognizing and processing images.

Another very similar project operates in a dynamic, unstructured environment. This research was conducted by Kragjic, et al [12]. The experimental setup included the Puma arm, BarrettHand, sonar sensor, laser sensor, a wrist mounted force torque sensor, a color CCD on hand and two cameras for stereovision. This system is simply too bulky for the purposes of rehabilitation engineering, but the researchers’ approach in programming was
very important for the study presented in this thesis. The study broke the
programming into several smaller programs instead of one massive program.

One study was conducted by Allen et al [13] at Columbia University
integrating vision, force, and tactile sensing for grasping. They used a number of
experiments to show how certain sensors, such as strain gages, vision sensing,
and tactile sensors, are integrated. It was demonstrated that the integration of
sensors dramatically increases the capabilities of a robot hand for grasping. This
conclusion is important, as this investigation can expect to reach the same
conclusion.

There are many end-effectors in existence, but few with advanced
technology for grasping door handles and bottles. There are currently several
common end-effectors, or end effectors, that are used in the field of rehabilitation
robots. Many of the end-effectors made are for industrial use and issues of
strength and precision contribute to the lack of end-effectors available for
rehabilitation applications.
Chapter Three

Development of the End-effector sensor suite

3.1 Selection Criteria

The approach presented in this study differs greatly from the current end-effectors for use on wheelchair-mounted robot arms in existence. This project should be relatively low cost, require minimal user instruction, easy to operate, and effective. None of the end-effectors used by the Manus, Raptor, nor Kares II satisfies these requirements.

Although the Manus arm is capable of relatively easy manipulation after the user grows acquainted with the arm, it can take a few minutes to get to a specified spot and the grasp provided by the end-effector is not very strong. If the end-effector is not perfectly centered on the object, the object can easily fall on the floor and become even more difficult to grasp.

The Raptor is neither easy to operate nor effective. The end-effector has the same exact problems as the Manus in addition to the arm being much harder to operate. It is controlled using a simple joystick or sip-in-puff which provides minimal control. Also, there are not many preset options for operating, such as home positions or assistive features.

The Kares II does not satisfy the third condition of being relatively inexpensive. It does, however, implement cameras, force torque sensors, and other types of sensors to assist in activities of daily living. It is perhaps the most
versatile of all the devices, but its operation is hard to learn. The researcher’s at KAIST even state this fact in their documentation.

3.2 Experimental Platform

The following section provides information on the experimental platform. It gives specifications and product selection guidelines for the Puma 560 arm, the BarrettHand, laser range finder, and four different types of cameras.

3.2.1 Puma Arm

The arm chosen for demonstration of this sensory suite is the Puma 560. This is shown below in Figure 7. A Puma arm was chosen for testing purposes because it is one of the most widely used arms in laboratory and industrial settings; it also has six degrees of freedom, making it possible to locate and orient an object in physical space. During this set-up, the Robotics toolbox for MATLAB created by Peter Corke [14] was used. It includes the complete kinematics for the Puma 560 arm.

Figure 7: The Puma Arm
In order to utilize the Puma arm, the link parameters must be known. Link parameters are used in order to accurately define the direction and distances associated with the orientation of a robot arm. The Denavit-Hartenberg (D-H) convention is used to describe the positions of links and joint parameters unambiguously. This convention is explained using further detail in the next pages. Figure 8 shows an example of a simple link.

Figure 8: Simple Link [15]

In Figure 8, $\alpha_{i-1}$ and $a_{i-1}$ are used to describe the kinematic relationship between the two joint axes. The link length is $a_{i-1}$, and is measured as the mutual perpendicular between the two axes. The link twist angle is defined as $\alpha_{i-1}$, and it is used to describe the angle between the two axes [15].

In Figure 8, $d$ is the link offset and theta is the joint angle. These two parameters are used to describe the connections between adjacent links. The link offset is the distance along a common from one link to the next. Theta is the
joint angle, which defines the rotation about the common link. These four variables are defined below in equation form [15].

\[ a_{i-1} = \text{the distance from } Z_i \text{ to } Z_{i+1} \text{ measured along } X_i \]  
(3.1)

\[ \alpha_{i-1} = \text{the angle from } Z_i \text{ to } Z_{i+1} \text{ measured about } X_i \]  
(3.2)

\[ d_i = \text{the distance from } X_{i-1} \text{ to } X_i \text{ measured along } Z_i \]  
(3.3)

\[ \theta_i = \text{the angle from } X_{i+1} \text{ to } X_i \text{ measured about } Z_i \]  
(3.4)

The coordinate system for the Puma arm is given below in Figure 9.

![Figure 9: Puma Arm Coordinate Configuration and Link Assignment](image)

The link-frame assignment follows a six step procedure which is explained in full detail in Craig's work [15]. The procedure is as follow:

1. Identify the joint axes

2. Identify the common perpendicular between them, or a point of interception. At this point of interception, or at the point where the common perpendicular meets the \( i \)th axis, assign the link frame origin.

3. Assign the Z axis pointing along the \( i \)th axis
4. Assign the X axis pointing along the common perpendicular, or normal to the plane of containing the two axes

5. Assign the Y axis to complete the right hand coordinate system

6. Label as necessary

The D-H parameters are listed below in Table 2. “i” in Table 2 represents the link based on the assignment in Figure 8.

<table>
<thead>
<tr>
<th>i</th>
<th>$\alpha_{i-1}$</th>
<th>$a_{i-1}$</th>
<th>$d_i$</th>
<th>$\theta_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$\theta_1$</td>
</tr>
<tr>
<td>2</td>
<td>-90°</td>
<td>0</td>
<td>0</td>
<td>$\theta_2$</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>$a_2$</td>
<td>$d_3$</td>
<td>$\theta_3$</td>
</tr>
<tr>
<td>4</td>
<td>-90°</td>
<td>$a_3$</td>
<td>$d_4$</td>
<td>$\theta_4$</td>
</tr>
<tr>
<td>5</td>
<td>90°</td>
<td>0</td>
<td>0</td>
<td>$\theta_5$</td>
</tr>
<tr>
<td>6</td>
<td>-90°</td>
<td>0</td>
<td>0</td>
<td>$\theta_6$</td>
</tr>
</tbody>
</table>

The D-H parameters are used to calculate the transformation matrices. Transformation matrices are four by four matrices in which the information on the objects rotation and translation is obtained. A frame that strictly undergoes translation is shown in equation 3.5 [15].

$$
^nT = \begin{bmatrix}
1 & 0 & 0 & x \\
0 & 1 & 0 & y \\
0 & 0 & 1 & z \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

(3.5)
In this equation, the translational values in x, y and z directions would be placed in the fourth column of the matrix. The frame would stay oriented the same, with the x, y and z coordinate frames being parallel to each other.

The rotation portion of a transformation matrix is contained within the first three rows and three columns. When an object is rotated around the x-axis by an angle of $\theta$, the x axis value in the matrix remains constant and the other axes are changed, as shown in equation 3.6 [15].

$$
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} = T_{n,m}^{\theta}
$$

Equation 3.6 shows only the rotation about the x axis, and does not have any translation components. When an object is rotated around the y-axis by an angle of $\theta$, the y values in the second column and second row remain constant and the other axes rotation change, as shown in equation 3.7 [15].

$$
\begin{bmatrix}
\cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} = T_{n,m}^{\theta}
$$

The rotation about the z axis follows the same pattern, where in the z column the value will remain constant when rotated by an angle of $\theta$. This is shown in equation 3.8 [15].

$$
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 & 0 \\
\sin \theta & \cos \theta & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} = T_{n,m}^{\theta}
$$
Equation 3.9 provides the general transformation matrix passed solely on link parameters. This equation uses the values from the Table to quantify the frame’s rotation and translation.

\[
{^i_T}_j = \begin{bmatrix}
\cos \theta_i & -\sin \theta_i & 0 & 0 & a_{i-1} \\
\sin \theta_i \cos \alpha_{i-1} & \cos \theta_i \cos \alpha_{i-1} & -\sin \alpha_{i-1} & -\sin \alpha_{i-1} \cdot d_i \\
\sin \theta_i \sin \alpha_{i-1} & \cos \theta_i \sin \alpha_{i-1} & \cos \alpha_{i-1} & -\cos \alpha_{i-1} \cdot d_i \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (3.9)

Equation 3.9 is now used in combination with the D-H parameters to obtain the six transformation matrices that govern the Puma arm [15].

In equation 3.10, \( ^0_T \) represents the transformation matrix of frame one with respect to the base frame, where \( \theta_i \) is the rotation component about the z axis [15].

\[
^0_T = \begin{bmatrix}
\cos \theta_1 & -\sin \theta_1 & 0 & 0 \\
\sin \theta_1 & \cos \theta_1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (3.10)

In equation 3.11, \( ^1_T \) represents the transformation matrix of frame two with respect to the frame one, where \( \theta_2 \) is the rotation component about the z axis [15].

\[
^1_T = \begin{bmatrix}
\cos \theta_2 & -\sin \theta_2 & 0 & 0 \\
0 & 0 & 1 & 0 \\
-\sin \theta_2 & -\cos \theta_2 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (3.11)
In equation 3.12, $^3_3T$ represents the transformation matrix of frame three with respect to the frame two, where $\theta_3$ is the rotation component about the z axis. There is also translation of this frame in the x and z directions [15].

$$^3_3T = \begin{bmatrix} \cos \theta_3 & -\sin \theta_3 & 0 & a_2 \\ \sin \theta_3 & \cos \theta_3 & 0 & 0 \\ 0 & 0 & 1 & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(3.12)}$$

In equation 3.13, $^4_3T$ represents the transformation matrix of frame four with respect to the frame three, where $\theta_4$ is the rotation component about the z axis. There is also translation of this frame in the x and y directions [15].

$$^4_3T = \begin{bmatrix} \cos \theta_4 & -\sin \theta_4 & 0 & a_3 \\ 0 & 0 & 1 & d_4 \\ -\sin \theta_4 & -\cos \theta_4 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(3.13)}$$

In equation 3.14, $^5_4T$ represents the transformation matrix of frame five with respect to the frame four, where $\theta_5$ is the rotation component about the z axis [15].

$$^5_4T = \begin{bmatrix} \cos \theta_5 & -\sin \theta_5 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin \theta_5 & \cos \theta_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{(3.14)}$$

In equation 3.15, $^5_6T$ represents the transformation matrix of frame five with respect to the frame four, where $\theta_6$ is the rotation component about the z axis [15].
The full forward kinematic parameters are given by $^{6}T_{0}$ which is shown below in equation 3.16, which will be used later in chapter four.

\[
^{6}T_{0} \equiv T_{1}^{-1}T_{2}^{-2}T_{3}^{-3}T_{4}^{-4}T_{5}^{-5}T_{6}^{-6}
\]

(3.16)

### 3.2.2 Camera Specifications

Four cameras were tested in the course of this study. These cameras include a Sony, Creative Labs Web Cam for Notebooks, Logitech QuickCam Orbit MP, and Point Grey’s Bumblebee camera. The implementation of these will be described in chapter four, but the technical information is contained in the charts below.

The first camera tested is Point Grey’s Bumblebee new two-lens stereo vision camera system. It is shown below in Figure 10.
The bumblebee camera comes preprogrammed in C++ and can return a three-dimensional point cloud of its field of vision. Programming is done to help recognize a cylinder in the point cloud. The bumblebee was chosen because of its size, weight, and capabilities. The technical chart is shown below in Table 3.

Table 3: Technical Specifications of the Bumblebee Stereovision Camera

<table>
<thead>
<tr>
<th>Image Sensors</th>
<th>Imaging Device 1/3 “ progressive scan CCDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>640x480-1024x768 VGA format</td>
</tr>
<tr>
<td>Size</td>
<td>16 x 4 x 4 cm</td>
</tr>
<tr>
<td>Signal to noise ratio</td>
<td>TBD</td>
</tr>
<tr>
<td>Consumption</td>
<td>2,1 W</td>
</tr>
<tr>
<td>Power</td>
<td>By IEEE-1994</td>
</tr>
<tr>
<td>Focal Length</td>
<td>Lens focal length High quality 4mm focal length pre-focused micro lenses</td>
</tr>
<tr>
<td>Baseline</td>
<td>120 mm</td>
</tr>
<tr>
<td>HFOV</td>
<td>70° degrees</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Less than 20 µs</td>
</tr>
</tbody>
</table>

The second camera tested is Logitech’s QuickCam Orbit MP. This camera was chosen because it was inexpensive, light-weight, small, and easy to obtain. The specifications for this camera are shown below in Table 4.
Table 4: Logitech Camera Specifications [16]

<table>
<thead>
<tr>
<th>System Requirements</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Windows® 2000, XP</td>
<td>• Logitech® QuickCam® Orbit™ MP Camera with motorized camera head</td>
</tr>
<tr>
<td>• CD-Rom drive</td>
<td>• 9&quot; stand</td>
</tr>
<tr>
<td>• Pentium® P4 1.4 GHz or AMD Athlon® processor 1 GHz (Pentium® P4 2.4 GHz or better recommended*)</td>
<td>• Base</td>
</tr>
<tr>
<td>• 128MB RAM (256MB RAM recommended*)</td>
<td>• QuickCam® Software CD</td>
</tr>
<tr>
<td>• 200MB Free Hard Disk space</td>
<td>• 6-foot USB cable</td>
</tr>
<tr>
<td>• 16-bit color display adaptor</td>
<td>• Stereo headset</td>
</tr>
<tr>
<td>• Windows compatible sound card and speakers</td>
<td>• High-quality 1.3 Mega pixel sensor</td>
</tr>
<tr>
<td>• Available 1.1 or 2.0 USB port (USB 2.0 High Speed port (Required for mega pixel image capture)</td>
<td>• Camera set-up guide</td>
</tr>
</tbody>
</table>

A picture of this web cam is show in Figure 11. This camera can either be mounted directly on the base or with a nine inch stand.

Figure 11: Web Cam
The third camera tested was a Sony camera pictured in Figure 12. It is a digital color charged couple display (CCD) with a cosmicar / pentax lens. The model number is IV-CCAM2, serial number U3000029. The specifications are shown below in Table 5.

![Figure 12: Sony CCD Camera](image)

**Table 5: Technical Specifications of the Sony Camera**

<table>
<thead>
<tr>
<th>Image Sensors</th>
<th>Imaging Device 1/3 &quot; progressive scan CCDs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>640x480-1024x768 VGA format</td>
</tr>
<tr>
<td>Size</td>
<td>16 x 4 x 4cm</td>
</tr>
<tr>
<td>Signal to noise ratio</td>
<td>TBD</td>
</tr>
<tr>
<td>Consumption</td>
<td>2.1 W</td>
</tr>
<tr>
<td>Power</td>
<td>By IEEE-1994</td>
</tr>
<tr>
<td>Focal Length</td>
<td>Lens focal length High quality 4mm focal length pre-focused micro lenses</td>
</tr>
<tr>
<td>Baseline</td>
<td>120 mm</td>
</tr>
<tr>
<td>HFOV</td>
<td>70° degrees</td>
</tr>
<tr>
<td>Synchronization</td>
<td>Less than 20 µs</td>
</tr>
</tbody>
</table>

The last camera tested is Creative Lab’s Web Cam for notebooks. This camera was chosen because it is small in size, weighs just ounces, and was the
least expensive of all the cameras tested. The camera is pictured in Figure 13 and the specifications are shown in Table 6.

Table 6: Product Specifications for Creative Lab Camera [17]

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 year warranty</td>
</tr>
<tr>
<td>USB Port connection</td>
</tr>
<tr>
<td>VGA CMOS sensor</td>
</tr>
<tr>
<td>Portable</td>
</tr>
<tr>
<td>640 x 480 video resolution</td>
</tr>
<tr>
<td>Requires Notebook PC with Intel® Pentium® II or AMD Athlon™ processor 350MHz; Windows 98, 98 SE, 2000, ME or XP; 128MB RAM; available USB port; CD-ROM drive</td>
</tr>
<tr>
<td>Additional Requirements Networking hardware and Internet connection (dial-up, LAN or wireless)</td>
</tr>
</tbody>
</table>

Figure 13: Creative Labs Web Camera for Notebooks

3.2.3 Laser Range Finder

A SICK laser range finder was used in this experiment as well. Table 7 from source [18] shows the DME 2000 laser range finder’s product features. The
laser range finder operates on “time of flight technology.” This means that the laser beam is pulsated and sent out toward an object in a narrow line. The time that it takes for the laser beam to hit the object and reflect back to the camera is used to determine the distance to an object based on the speed of light constant and time of flight. This range finder is accurate up to millimeters; however, accuracy in sub millimeters is substandard.

Table 7: DME Product Features

| Sensing range min ... max (reflector mode): | 0,1 ... 130 m |
| Sensing range min ... max (proximity mode): | 100 ... 2,047 mm |
| Light source: | Laser diode |
| Type of light: | Laser, red light |
| Laser protection class: | 2 (IEC 60825-1/EN 60825-1) |

Table 8 shows the technical specifications for the DME 2000 range finder.

Table 8: Technical Data [18]

| Dimensions (W x H x D): | 54 x 105 x 138 mm |
| Supply voltage min ... max: | DC 18 ... 30 V |
| Output current: | <= 100 Ma |
| Light spot diameter: | Reflector mode: Approx. 250 mm / 130 m, Proximity mode: Approx. 3 mm/2 m |
| Data interface: | RS-232 |
| Resolution: | Reflector / Proximity mode: 1 mm |
| Accuracy: | +- 11 mm (>18% remission), +- 5 mm (=90% remission), +- 65 mm (6% remission), Proximity mode: |
A dimensioned drawing of the DME2000 is shown in Figure 14. The dimensions are given in mm. This particular laser range finder has a mass of approximately 2 pounds.
3.2.4 BarrettHand End-effector

The BarrettHand offers a viable solution to the problems of common end-effectors. The BarrettHand is a four degree of freedom robot end-effector with advanced gripping capabilities. This end-effector is an adaptive end-effector, meaning the force of the end-effector can be adjusted. The BarrettHand robot end-effector is pictured in Figure 15.
Figure 15: BarrettHand End-effector

Figure 16 shows that the three fingers on this end-effector curl at a joint to allow for optimal gripping of complex shaped objects. The end-effector is able to grasp balls, handles, door knobs, and even cell phones. The hand’s three fingers have the ability to spread, step open, and step close.

Figure 16: BarrettHand's Grasping Abilities

The BarrettHand grasps objects by closing the fingers around an object. Once a certain force is reached in the bottom portion of the finger, the top portion of the finger curls around to allow for proper grasping of the object of interest.
The fingers can also be controlled independently of each other. This makes it more difficult for an object to slip out of grasp and gives the end-effector even more versatility. With the control of individual fingers, it allows the user to draw an object in close without the use of all three fingers. This is extremely useful when trying to center or close in on an object quickly and accurately. This is shown in Figure 17.

![Independent Finger Movement](image)

**Figure 17: Independent Finger Movement**

The robot hand used in this project is the BarrettHand by Barrett Technologies Inc. The specifications are shown in Table 9.

The BarrettHand is used because it is the most effective end-effector, both in terms of price and ease of operation. The BarrettHand costs approximately $20,000, which is cheaper than a single year of dependent care taking. The BarrettHand comes with pre-loaded GCL commands. It is also equipped with an ADR interface card to allow for port communication.
<table>
<thead>
<tr>
<th>Specification</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total fingers</td>
<td>3</td>
</tr>
<tr>
<td>Fingers that spread</td>
<td>Fingers 1 and 2</td>
</tr>
<tr>
<td>Joints per finger</td>
<td>2</td>
</tr>
<tr>
<td>Motors per finger</td>
<td>1</td>
</tr>
<tr>
<td>Motors for spread</td>
<td>1</td>
</tr>
<tr>
<td>Total number of motors</td>
<td>4</td>
</tr>
<tr>
<td>Range of Motion for base joint</td>
<td>140°</td>
</tr>
<tr>
<td>Range of Motion for fingertip</td>
<td>45°</td>
</tr>
<tr>
<td>Range of Motion for finger spread</td>
<td>180°</td>
</tr>
<tr>
<td>Time for finger to move from fully open to fully closed</td>
<td>1.0 sec</td>
</tr>
<tr>
<td>Time for Full 180° finger spread</td>
<td>0.5 sec</td>
</tr>
<tr>
<td>Optical Incremental encoder for position sensing</td>
<td>0.008° at the finger base joint 17,500 encoder counts full finger open to full close</td>
</tr>
<tr>
<td>Hand Weight</td>
<td>1.18kg (2.60lbs)</td>
</tr>
<tr>
<td>Payload</td>
<td>6.0 kg (13.2 lbs), 2.0 kg (4.4 lb) per finger</td>
</tr>
<tr>
<td>Active Finger Forces (at tip)</td>
<td>15 N (3.3 lb)</td>
</tr>
<tr>
<td>Passive Finger Forces (at tip)</td>
<td>20 N (4.4 lb)</td>
</tr>
<tr>
<td>Motor Type</td>
<td>Samarium-Cobalt, brushless, DC, servo motors</td>
</tr>
<tr>
<td>Mechanisms</td>
<td>Worm drives integrated with patented cable drive and breakaway clutch</td>
</tr>
<tr>
<td>Cycles per Month</td>
<td>10,000</td>
</tr>
<tr>
<td>Power Requirements</td>
<td>Typical AC electrical outlet</td>
</tr>
<tr>
<td>Load</td>
<td>600 W</td>
</tr>
<tr>
<td>Phases</td>
<td>Single</td>
</tr>
<tr>
<td>Voltage</td>
<td>120/240 ±10% VAC</td>
</tr>
<tr>
<td>Frequency</td>
<td>50/60 Hz</td>
</tr>
<tr>
<td>Power Supply Size H,W,D:</td>
<td>200 x 200 x 300 mm (7.5 x 7.5 x 12 in)</td>
</tr>
<tr>
<td>Power Supply Size Weight</td>
<td>5 kg (11 lb)</td>
</tr>
<tr>
<td>Single Cable to Hand</td>
<td>3m continuous-flex cable, 8mm diameter</td>
</tr>
</tbody>
</table>
Chapter Four

System Integration

4.1 End-effector Integration

When running the BarrettHand program through the provided interface, the interface is capable of producing C++ code. The problem with this automatically generated code, shown in Appendix A section A.1, is that it calls to a C-function library that handles the port communication.

In order to bypass this C-function library, which was an additional $2,000, the appropriate port communication was needed to be established. The first program experimented with was one written by Ontrack.com and modified to work with the BarrettHand. This code was very long, slow, and was proven to be unnecessary.

The next step was to write the code in Appendix A, section A.2. The code’s sequence opens the port of communication, sets the baud rates, time out, and other parameters. The BarrettHand has on-board microprocessors and is preprogrammed to recognize End-effector Control Language (GCL). The common GCL commands are listed below in Table 10.
Table 10: Common GCL Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Name</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>“C”</td>
<td>Close</td>
<td>Commands the motor to the closed position</td>
</tr>
<tr>
<td>“HI”</td>
<td>Hand Initialize</td>
<td>This initializes the GCL commands in the hand</td>
</tr>
<tr>
<td>“HOME”</td>
<td>Home</td>
<td>Moves the motors to the home position</td>
</tr>
<tr>
<td>“IO”</td>
<td>Incremental Open</td>
<td>Steps the motors open by a user specified increment</td>
</tr>
<tr>
<td>“IC”</td>
<td>Incremental Close</td>
<td>Steps the motors close by a user specified increment</td>
</tr>
<tr>
<td>“LOOP”</td>
<td>Loop</td>
<td>Enters real-time mode</td>
</tr>
<tr>
<td>“M”</td>
<td>Move</td>
<td>Moves the to a user specified position</td>
</tr>
<tr>
<td>“O”</td>
<td>Open</td>
<td>Opens the selected motor</td>
</tr>
<tr>
<td>“T”</td>
<td>Terminates Power</td>
<td>Turns selected motors off</td>
</tr>
<tr>
<td>“HSG”</td>
<td>Highest Strain Gage Value</td>
<td>Sets the highest strain gage value</td>
</tr>
</tbody>
</table>

The “HI” is always run first, prior to all other commands, in order to initialize the GCL commands.

Since there is no safety triggers built into the C++ code, the programmed commands are first run in the BarrettHand software package prior to writing the code. This is done to test the sequence of commands and to make sure that the BarrettHand is not forced into an unstable position or a position that would cause irreparable damage.

An example of a series of commands created this way is shown in Appendix A, section A.3. To insert the GCL commands into the code, the user should edit the port write command. It looks like: port.write ("123C\r",5); where the 123C refers to the GCL command and the number five following the comma corresponds to number of characters the program should read in the command.
The BarrettHand will be mounted on the Puma robot arm. The versatility of this end-effector allows it to easily grasp most objects required for daily living.

Using the BarrettHand in MATLAB is very simple because the hand can still be sent commands using the GCL language. The commands are sent as follows in Figure 18.

![MATLAB Command Flowchart]

Figure 18: MATLAB Command Flowchart

The first two steps handle port communication, where “s1” is assigned to port one, and the command serial (‘COM1’) opens port one. The Baud Rate is being set to 9600 bits per second, the parity is none, and the remaining parameters are set in accordance to the operating manual.

The third command creates a temporary file, where the “%5s” indicates the length of data to be sent to the hand’s interface. The “s1” command is the port that is assigned to the BarrettHand and the “fprintf” is actually writing the command to the port.
With these commands, it is possible to open and operate the BarrettHand in MATLAB or C++, depending on the platform that the robot manipulator operates on. The MATLAB program used to control the BarrettHand is shown in Appendix B, section in B.3.

4.2 SICK Laser Range Finder

A SICK laser was also used in this experiment. The purpose for using this is to detect distance, the z coordinate. This is used for vision applications as well as sending a signal to close the end-effector. This laser was used because it was already available; however, others may be a better choice in order to avoid eye damage and perhaps downgrade the size of the unit. The picture of the laser range finder is shown in Figure 19.

![Figure 19: Laser Range Finder](image)

The range finder was programmed using a similar port writing C ++ program to that of the BarrettHand. This code is shown in Appendix A, section 4. When running this program, the numbers from the laser range finder are simply sent to the computer and displayed on a separate screen.
Appendix A, section 5 shows the integration of the BarrettHand commands with the laser range finder. This program runs, and once an object is within five inches of the BarrettHand, the hand closes. This value can be adjusted as necessary and will be experimentally tested, as the mounting of the laser on the arm can change.

The laser range finder’s programming in MATLAB was a challenge because, although the port opening is similar to that of the BarrettHand (a Baud rate of 9600 bits per second), data was determined to be lost. The baud rate was slowed down to 1200 to reduce the frequency of data being lost. In the MATLAB program, the values read by the device had to be put through an if-then loop to ensure that an actual value was received rather than a null value.

Calculating the value output was different in MATLAB because the data is displayed in bits. The bits must be converted as follows. Bits 0 though 9 represent values 45 through 57 and the characters A through F represent 65 through 70. Once the value of each digit is known, multiply each by the appropriate value (1, 10, 100) or (1, 16), and add the results. This is shown in Appendix B section 1. The integration of the BarrettHand with the laser range finder is shown Appendix B, section 4.

4.3 Camera Systems

Four cameras were tested and used at different stages of the experimental process. The bumblebee camera was eliminated the quickest, followed by the Logitech camera, Sony camera, and the Creative web camera. The vision aspect of this project is perhaps the hardest to understand. The initial
tests were run with the bumblebee stereovision camera because it is an inexpensive system that would be able to return three dimensional coordinates if needed. The stereovision camera chosen for this application is the bumblebee camera because of its light weight.

The general procedure taken was to collect many bottles and start with the camera very close to obtain the point cloud of the bottle. Then it was necessary to write a simple algorithm for recognizing the object in the point clouds when the bottles are further away. Once that phase is accomplished, the camera can be mounted to a robot arm where it will obtain the three dimensional coordinates and send the coordinates to a robot arm. Once the arm moves to a position, a command is sent to the BarrettHand to grasp the object and then return to the wheelchair, or some fixed reference point, to finally being released.

Although this approach contains only one camera, was determined to be expensive and labor intensive. When trying to retrieve three dimensional coordinates of an object, the camera had limited capabilities in regards to adjusting the light and seeing clear objects and noise. This prompted the need to set up a different vision system.

The second vision setup was less expensive and has two components: a range finder and a web camera. It was necessary to determine a transformation matrix for the laser range finder mounted on the robot arm.

The range finder will be pointed at an object of reference, in this case a red object, and will return the distance from the range finder to the object of interest. The range finder will be stationary with respect to the robot arm and will
be part of the robot arm’s workspace. It will need a constant transformation from the range finder to the robot arm so the proper final position can be obtained. This would work in a similar manner if the range finder was mounted on a wheelchair with a robot arm attached to it.

The camera will be attached above the BarrettHand on the arm and will take pictures before the z coordinates are reached. Using MATLAB’s image acquisition and image processing toolboxes, the x and y coordinates will be obtained from some known images such as a bottle or a door knob. The cameras used for these applications are a Logitech, Sony, and Creative cameras. Any windows based wed camera can be used for this application.

The Logitech camera was used for the early stages of the vision algorithms develop with the MATLAB image processing toolbox. This camera was utilized because it was the easiest to use and the resolution was perfect for this application. The Logitech also came with automatic light adjusting capabilities. This camera was later replaced because of the face tracking capabilities.

The face tracking made it impossible to hold the camera steady. The face tracking feature would capture a large image and then would follow it regardless of its movement. The camera would not stay focused in one direction, and the coding that was responsible for the motion of the camera is highly proprietary.

The next camera to use was the Sony camera. This camera is a high-quality camera with excellent resolution. This camera required a frame grabber card which could not be located, so a video card was sufficient for the
experimental purpose. The experiments included the entire processing plan as well as the calibration which is shown in Table 11.

Table 11: Calibration Data for Sony Camera

<table>
<thead>
<tr>
<th>Distance</th>
<th>Height</th>
<th>Width</th>
<th>Area</th>
<th>Width / Height</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.18</td>
<td>2</td>
<td>1.3</td>
<td>2.6</td>
<td>0.65</td>
</tr>
<tr>
<td>1.96</td>
<td>3</td>
<td>2</td>
<td>6</td>
<td>0.666667</td>
</tr>
<tr>
<td>2.75</td>
<td>4</td>
<td>2.7</td>
<td>10.8</td>
<td>0.675</td>
</tr>
<tr>
<td>3.54</td>
<td>4.5</td>
<td>3.1</td>
<td>13.95</td>
<td>0.688889</td>
</tr>
<tr>
<td>4.33</td>
<td>5.5</td>
<td>4</td>
<td>22</td>
<td>0.727273</td>
</tr>
<tr>
<td>5.11</td>
<td>6.3</td>
<td>4.5</td>
<td>28.35</td>
<td>0.714286</td>
</tr>
<tr>
<td>5.9</td>
<td>7.2</td>
<td>5.2</td>
<td>37.44</td>
<td>0.722222</td>
</tr>
<tr>
<td>6.69</td>
<td>8</td>
<td>5.5</td>
<td>44</td>
<td>0.6875</td>
</tr>
<tr>
<td>44.68504</td>
<td>54</td>
<td>41</td>
<td>2214</td>
<td>0.759259</td>
</tr>
<tr>
<td>42.87402</td>
<td>51</td>
<td>36</td>
<td>1836</td>
<td>0.705882</td>
</tr>
<tr>
<td>40.3937</td>
<td>47</td>
<td>34</td>
<td>1598</td>
<td>0.723404</td>
</tr>
<tr>
<td>37.87402</td>
<td>45</td>
<td>33</td>
<td>1485</td>
<td>0.733333</td>
</tr>
<tr>
<td>35.74803</td>
<td>43</td>
<td>32</td>
<td>1376</td>
<td>0.744186</td>
</tr>
<tr>
<td>33.34646</td>
<td>40</td>
<td>30</td>
<td>1200</td>
<td>0.75</td>
</tr>
<tr>
<td>30.15748</td>
<td>38</td>
<td>28</td>
<td>1064</td>
<td>0.736842</td>
</tr>
<tr>
<td>29.44882</td>
<td>36</td>
<td>27</td>
<td>972</td>
<td>0.75</td>
</tr>
<tr>
<td>27.12598</td>
<td>33</td>
<td>24</td>
<td>792</td>
<td>0.727273</td>
</tr>
<tr>
<td>24.84252</td>
<td>31</td>
<td>23</td>
<td>713</td>
<td>0.741935</td>
</tr>
<tr>
<td>22.83465</td>
<td>29</td>
<td>22</td>
<td>638</td>
<td>0.758621</td>
</tr>
<tr>
<td>20.90551</td>
<td>27</td>
<td>20</td>
<td>540</td>
<td>0.740741</td>
</tr>
<tr>
<td>19.68504</td>
<td>25</td>
<td>18</td>
<td>450</td>
<td>0.72</td>
</tr>
<tr>
<td>18.38583</td>
<td>23</td>
<td>16</td>
<td>368</td>
<td>0.695652</td>
</tr>
<tr>
<td>16.10236</td>
<td>21</td>
<td>14</td>
<td>294</td>
<td>0.666667</td>
</tr>
<tr>
<td>14.92126</td>
<td>18.5</td>
<td>13</td>
<td>240.5</td>
<td>0.702703</td>
</tr>
</tbody>
</table>

The calibration was done by adjusting the distance from a plain white wall to the camera. This distance was recorded, and the height and width of the displayed screen on the computer was measured on the white wall using a tape measure. This was done for the anticipated range of use for the camera, as all cameras have different characteristics which govern their field of view. Figures
20 and 21 show the calibration charts with a trend line which predicts the width and height of the camera view based on the range finder values.

\[
y = 1.1277x + 2.634 \\
y = 0.852x + 1.1639
\]

![Sony Camera Distance Verse Height and Width](image)

Figure 20: Range Finder Distance Verse Height for Sony

This camera was later scrapped because the resolution made it difficult for the MATLAB image processing toolbox to differentiate between colors and often provided inconsistent results. This camera may have performed better with an adequate frame grabber card, as the card used for this was a simple video card with no special functions. This camera also had to be mounted over the BarrettHand, and the fingers of the BarrettHand would often obstruct the view of the camera.
The final camera chosen for the application is the Creative Web Camera for Notebooks. This camera combines the best features of the Sony camera (size and versatility) with the Logitech camera (ease of use and dependability). The calibration is as shown below in Table 12 and was conducted in exactly the same manner as the Sony camera. The calibration equations are displayed below in Figure 21.

Table 12: Creative Camera Calibration

<table>
<thead>
<tr>
<th>Distance (in)</th>
<th>Range Finder Distance (in)</th>
<th>Width</th>
<th>Height</th>
<th>Height / Width</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12.5</td>
<td>1.0625</td>
<td>0.75</td>
<td>0.705882</td>
</tr>
<tr>
<td>2</td>
<td>13.5</td>
<td>2</td>
<td>1.5</td>
<td>0.75</td>
</tr>
<tr>
<td>3</td>
<td>14.5</td>
<td>2.625</td>
<td>2</td>
<td>0.761905</td>
</tr>
<tr>
<td>4</td>
<td>15.5</td>
<td>3.375</td>
<td>2.375</td>
<td>0.703704</td>
</tr>
<tr>
<td>6</td>
<td>17.5</td>
<td>5</td>
<td>3.6</td>
<td>0.72</td>
</tr>
<tr>
<td>8</td>
<td>19.5</td>
<td>6.375</td>
<td>4.625</td>
<td>0.72549</td>
</tr>
<tr>
<td>10</td>
<td>21.5</td>
<td>8.875</td>
<td>6.5</td>
<td>0.732394</td>
</tr>
<tr>
<td>12</td>
<td>23.5</td>
<td>9.625</td>
<td>7</td>
<td>0.727273</td>
</tr>
<tr>
<td>15</td>
<td>26.5</td>
<td>12.5</td>
<td>9</td>
<td>0.72</td>
</tr>
<tr>
<td>18</td>
<td>29.5</td>
<td>15.125</td>
<td>11.625</td>
<td>0.768595</td>
</tr>
<tr>
<td>21</td>
<td>32.5</td>
<td>17.625</td>
<td>13</td>
<td>0.737589</td>
</tr>
<tr>
<td>24</td>
<td>35.5</td>
<td>21</td>
<td>15</td>
<td>0.714286</td>
</tr>
<tr>
<td>30</td>
<td>41.5</td>
<td>26</td>
<td>19</td>
<td>0.730769</td>
</tr>
<tr>
<td>36</td>
<td>47.5</td>
<td>29.5</td>
<td>22.5</td>
<td>0.762712</td>
</tr>
<tr>
<td>42</td>
<td>53.5</td>
<td>36</td>
<td>28.5</td>
<td>0.791667</td>
</tr>
<tr>
<td>48</td>
<td>59.5</td>
<td>40</td>
<td>31</td>
<td>0.775</td>
</tr>
</tbody>
</table>
4.4 Image Processing

In order to locate a specific object of interest, the object must first be defined. In this study, objects of interest were defined simply as objects colored red. This was done because red is not a common household color and allows for the greatest amount of contrast with other colors. In order to locate red objects, the image acquisition and processing toolboxes from MATLAB version 7.1 were used during these experiments.

Once the object was obtained using the image acquisition toolbox, the image processing toolbox was used. MATLAB has a built-in application called L*A*B color space. This color space is pre-programmed with a pixel library for blue, green, red, magenta, yellow, and cyan. It distinguishes these colors by...
comparing the pixel spaces at the boundaries of an object. By utilizing L*A*B color space, the program can distinguish which colors are present by comparing the edge pixels distance. Based on this comparison, it is able to determine which colors are present and separate them from other colors. This MATLAB code is shown in Appendix B section 2.

The general procedure is as follows for the image recognition: the colors are segmented and then filtered only for red. Once the red is located, all other colors turn black and the picture is converted to grey scale. From here, the holes are filled by the color next to it. A hole was defined as an object less than thirty pixels in diameter.

The next step is the boundary trace function in MATLAB, in which the boundaries are traced along the white to black threshold. Once the boundaries are determined, the centroids are calculated through the same toolbox. The program calculates the area, in pixels, that the object is occupying, then places the centroid in the center of the object. While this does not necessarily represent the mass center, it provides a good basis for the grasping application. When running the program, the camera should be positioned so that only one red object is present.

The following four pictures are going to outline the image processing steps. Step 1, shown in Figure 22, shows the original image. This is the step where the image is acquired.
As can be seen, there are many colors present Figure 22. The object of interest is going to be the red ruler. Figure 23 shows the image once it has been filtered for red and then converted to grey scale.

The image is converted to grey scale to allow the holes to be filled in the next step without a color bias. In the next step, the holes in the image will be filled. The black will be completely black with no white marks and the ruler will look like a solid rectangle. This is shown in Figure 24.
The next step is to use MATLAB’s trace function. With this function, the edges are traced and the centroid is then calculated based on the pixel area. This is shown in the following picture with the green mark in Figure 25 representing the calculated centroid of the ruler.

As can be seen, this process is completed with relatively little error. The closer to the object the camera gets, the more accurate the calculations become.

4.5 Integration Platform
The integration of these programs is done in MATLAB. The programs follow the flow of Figure 26.

![Program Flow Chart](image)

Figure 26: Program Flow Chart

The main program is broken into three sub-programs, the Centroid, Laser with Barret Hand and BarrettHand Open. The Centroid operates by calling to the vision algorithm and laser range finder programs shown in appendix B section 2. The vision program runs and finds the centroid’s location, after which it calls to the laser range finder and inputs its value into the calibration equation. This returns the x, y and z coordinate of the centroid’s location with respect to the camera’s center.

The integration of the BarrettHand, camera, and laser range finder is being applied to a Puma arm as shown in Figure 27.
Figure 27: Puma Arm Kinematic Setup
The transformation matrices were shown in full detail in chapter three. These transformations are necessary for calculating the inverse kinematics of the Puma arm. This suite did not utilize this information, but the groundwork is being laid for future testing of the output results from the MATLAB programming.

All of the following matrices are necessary in order to obtain the angles of rotation necessary for the robot arm to move to. From chapter three, equation 3.16 states:

\[
0 T_6 = 0 T_1 T_2 T_3 T_4 T_5 T_6
\]  

(3.16)

In order to obtain the joint angles, the following paradigm has to be followed.

First, a nonmoving base frame must be defined. Equation 4.1 shows the transformation matrix of the Puma arm base frame with respect to the frame zero. This is illustrated in Figure 28.

\[
0 T_b = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & -0.672 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\]  

(4.1)

Figure 28: Translation of O Frame to Base
Equation 4.1 shows only a translation component in the z direction during this transformation.

Secondly, the workstation frame must be known with respect to the base frame. There is only translation in this transformation matrix shown in equation 4.2.

\[
{s_B}^T = \begin{bmatrix}
1 & 0 & 0 & 0.68 \\
0 & 1 & 0 & 0.23 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix}
\] (4.2)

This is also illustrated in Figure 29. The orange lines in Figure 29 represent the translation in the x and y directions in the transformation matrix, since the workstation was at the same height as the base frame, no translation in the z axis occur.

![Figure 29: Translation from Base Frame to Workstation Frame](image)

The next step is to find the transformation matrix of the camera with respect to the wrist frame, which was assigned to be frame number six in chapter.
three. This transformation matrix is shown in equation 4.3 and only shows translation along the y and z axes. Frame 6 is shown in Figure 32.

\[
T_6 = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & -0.02 \\
0 & 0 & 1 & 0.2 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \tag{4.3}
\]

The goal point, called frame P in this case, was described with respect to the workstation frame. This is demonstrated in equation 4.4 and illustrated in Figure 30.

\[
T_{SP} = \begin{bmatrix}
1 & 0 & 0 & 0.038 \\
0 & 1 & 0 & -0.41 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 \\
\end{bmatrix} \tag{4.4}
\]

Figure 30 shows the translation portion of the transformation matrix in yellow; again, there is no rotation to take into consideration.
Equation 4.5 shows the transformation matrix of the part frame with respect to the view of the camera. In this equation, the fourth column represents the results of the centroid computation. This is shown in Figure 31.

\[
\begin{bmatrix}
1 & 0 & 0 & X_{\text{centroid}} \\
0 & -1 & 0 & Y_{\text{centroid}} \\
0 & 0 & -1 & Z_{\text{centroid}} \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (4.5)

Figure 31 shows that the y and z axis changes in rotation and that there is translation along the x, y and z coordinates, shown in green.

The last frame that must be defined is the end-effector frame. The end-effector frame is defined with respect to joint 6. This is shown in equation 4.6 and illustrated in Figure 32.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0.33 \\
0 & 0 & 0 & 1
\end{bmatrix}
\] (4.6)
In Figure 32, the red error represents translation along the z axis, and there is no rotation in this transformation. In order to obtain the necessary joint angles for the Puma arm configuration, equation 4.7 must be satisfied. This provides a check point to ensure everything is operational and accurate. Figure 33 shows the overall transformation of the Puma arm.
Figure 33: Transformations of the Puma Arm

In Figure 33, the burgundy line represents the transformation of frame 6 with respect to frame 0. The dark green line represents the transformation of the part with respect to frame 6. The product of these two transformations is equal to the products of the following transformations:

- The transformation of the base with respect to the origin, shown in blue.
- The transformation of the workstation with respect to the base, shown in light green.
- The transformation of the part with respect to the base, shown in teal.
- The inverse transformation of the part with respect the camera, shown in army green.
• The transformation of the camera with respect to frame 6, shown in yellow.

These transformations are shown in equation 4.7.

$$
0^d_T \cdot 6^c_T = 0^G_T \cdot 6^S_T \cdot 5^p_T \cdot 4^C_T^{-1}
$$

Equation 4.7 is true in the current position. This allows for the calculation of the joint angles for $^d_O$ in the current position. These results can be used as input for the inverse kinematics solver in future applications. For the arm to have the joint angles of the desired location, the end-effector frame and the part frame should have zero translation, meaning they should be coincident. Equations 4.8 and 4.9 shows this, where $^d_O T$ represents the transformation matrix of the desired end-effector location with respect to the origin.

$$
0^d_T \cdot 6^d_T = 0^G_T \cdot 6^C_T \cdot 5^p_T
$$

This is also equal to equation 4.9.

$$
0^G_T \cdot 6^C_T \cdot 5^p_T = 0^d_T \cdot 6^d_T
$$

Equation 4.9 serves as a check for equation 4.8. These last two equations are used to determine what angles are necessary for the Puma arm to move to in order to have the end-effector at the part’s location.

### 4.6 Operating Procedures

The sensory suite was shown to be very user friendly and is suitable to be integrated into any robot arm capable of Cartesian movement, with the exception of one change. The transformation matrix from the camera to the laser range finder must be changed every time the camera or the laser range finder is moved to a different location. However, as long as the displacement between the two
objects remains in one coordinate, in this case the z-coordinate, the change should be a matter of subtracting the offset value in the MATLAB program seen in Appendix B.

Once on a wheelchair the camera must be aimed at an object, which means that some user input is necessary. Once an object of interest is within the camera’s viewpoint, the image processing program must be run. Once a positive centroid is calculated, the results need to be sent to the arm. This step is dependant on the robot arm and must be integrated accordingly.

Once the results are sent to the robot arm, it starts to move. When this starts, the automatic gripping program should be run. Once the robot arm is within a certain distance from the object of interest, the hand will automatically close. Once closed, the arm can be moved back to the user’s workplace.
Chapter Five

Analysis of Results and Conclusions

5.1 Results

The final set-up appears as Figure 34.

![Figure 34: Final Puma Arm Configuration](image)

This Figure shows the workstation, the arm, the BarrettHand, the camera in the center of the hand, and the laser range finder. The camera in the center of the hand is shown below in Figure 35.
In order to properly determine how the integration works, several experiments were run. The first experiment was placing objects of interest in different locations with different backgrounds and seeing how closely the program calculated the centroid as opposed to the actual centroid, shown below in Table 13.

Table 13: Before and After Image Processing

<table>
<thead>
<tr>
<th>Original Image</th>
<th>Centroid Location</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Original Image" /></td>
<td><img src="image2.png" alt="Centroid Location" /></td>
</tr>
</tbody>
</table>
In the next test, 4 different objects were tested. The image processing program was run and the values of the centroid were displayed. Ten readings were taken for each object. These readings are then averaged together, and their results are compared.

The four objects include a ratcheting tie down, ball, ruler, and box of staples; all of which are red. These results are quantified in Table 14.
Table 14: Actual vs. Calculated Centroid in x, y, z

<table>
<thead>
<tr>
<th>Direction</th>
<th>Computer Value Average (m)</th>
<th>Actual Measured Distance (m)</th>
<th>Difference (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-0.005</td>
<td>-0.006</td>
<td>0.001</td>
</tr>
<tr>
<td>Y</td>
<td>-0.0079</td>
<td>-0.009</td>
<td>0.0011</td>
</tr>
<tr>
<td>Z</td>
<td>0.2018</td>
<td>0.205</td>
<td>0.0032</td>
</tr>
<tr>
<td>Object B</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-0.009</td>
<td>-0.007</td>
<td>0.002</td>
</tr>
<tr>
<td>Y</td>
<td>-0.008</td>
<td>-0.007</td>
<td>0.001</td>
</tr>
<tr>
<td>Z</td>
<td>0.267</td>
<td>0.268</td>
<td>0.001</td>
</tr>
<tr>
<td>Object C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-0.009</td>
<td>-0.010</td>
<td>0.001</td>
</tr>
<tr>
<td>Y</td>
<td>-0.008</td>
<td>-0.008</td>
<td>0.000</td>
</tr>
<tr>
<td>Z</td>
<td>0.2345</td>
<td>0.234</td>
<td>0.0005</td>
</tr>
<tr>
<td>Object D</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>-0.007</td>
<td>-0.005</td>
<td>0.001</td>
</tr>
<tr>
<td>Y</td>
<td>-0.009</td>
<td>-0.008</td>
<td>0.001</td>
</tr>
<tr>
<td>Z</td>
<td>0.212</td>
<td>0.210</td>
<td>0.002</td>
</tr>
</tbody>
</table>

As can be seen from Table 14, the greatest error that occurs is about 3 millimeters in the z direction, 2 millimeters in the x, and 1 millimeter in the y. This means the x, y, and z coordinates calculated is equal to their actual location. Even though there is a difference, this difference falls within the margin of error when physically measuring the actual centroid.

There was also some error found in the camera calibration of the Creative web camera, which is shown in Table 15.
The expected ratio should be the same as the measured ratio, and the maximum error seen in this calibration was about 6.2%. This can contribute to the error in the calculations of the x and y directions.

The final results are shown in the following sequence of pictures, with each step being illustrated in the next four images.

Step 1: Acquire Original Image
Step 2: Image Centroid Location

The location of the centroid is x coordinate = -0.008, y coordinate = -0.005, z coordinate = 0.218 with all dimensions being in meters.

Step 3: Arm moving into place, shown in Figure 38.
Figure 38: Puma Arm Moving

Step 4: BarrettHand automatically closing around center of object shown in Figure 39.

Figure 39: Grasping Test
Step 5: Object returning to person’s workspace, in this case the blue cup, shown in Figure 40.

![Figure 40: Returning to Workspace](image)

5.2 Discussion

There are several possible sources of error. The greatest source of error in the results can be as a result of camera calibration readout. When calibrating the camera, a simple ruler was used. This is not nearly as accurate as using the laser range finder, so there is a certain amount of acceptable loss in this process. Generally, the error was a maximum of 6%, which is still acceptable for the tasks preformed.

The BarrettHand compensates for this error because of the way the fingers curl around an object. As the object is being grasped, it is forced to the center of the palm of the hand. This allows for even grasping of the object even with some error present.
Another source of error was encountered with the image processing. If the room is too dark, the program will not work. It is also essential that there only be one object of interest in the camera field of view. If this does not happen, the arm receives two output coordinates and cannot distinguish between the two coordinates; therefore, the arm does not know where to go. This type of error is a fatal error and will result in the program malfunctioning.

5.3 Conclusion

In conclusion, the end-effector sensory suite meets all the objectives as defined in chapter one. An end-effector sensor suite to assist in the control of a rehabilitation robot end-effector was developed. This included:

- Programming and integration of the BarrettHand end effector to the Puma 560 robot
- Selection and integration of a laser range finder with the manipulator
- Selection and integration of a vision system with the manipulator

Experiments were also performed with the sensory suite integrated with the manipulator on the arm. This included:

- Using the end-effector sensor suite’s assistance in finding the centroid of an object
- Analyzing object centroid location data to ensure accurate results were obtained
- Developed algorithms for moving the robot hand to grasp the object
5.4 Recommendations

The operating platform for this experiment would be greatly improved by complete cartesian control of the Puma arm. This would allow for a fully functional robot system. A better arm would also provide a better platform for the sensory suite. The wheelchair arm created by Kevin Edwards, shown in Figure 32, would provide the perfect mobile platform.

Other improvements can be made to the vision algorithms. If object recognition was utilized instead of color recognition, there would be less interference with the background. For object recognition, high resolution is necessary in order to properly identify the object. This would work best with the Sony Camera tested. In the future, combining the Sony Camera with a proper frame grabber card could lead to more stable and accurate results.
References


Bibliography


Appendices
Appendix A: C++ Code

C++ is used throughout this project to control a robot end-effector, the BarrettHand and a laser range finder. The C code used in this thesis was designed to control Acquisition Data Report (ADR) interfaces. When connecting to a serial port, the ADR interface boards allows control of analog and digital input and outputs using American standard code for information interchange (ASCII) control commands. ADR interfaces are easy to use with Visual Basic, Basic, C or other high level languages that allow access to a serial port. The C++ language protocol and definitions are set by the American National Standards Institute (ANSI).

The common codes and definition used for this project are listed on the next page in Table 16.
### Table 16: C++ Code Definitions

<table>
<thead>
<tr>
<th>Command</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iostream.h</code></td>
<td>Library that provides functionality to perform input and output operations with a stream.</td>
</tr>
<tr>
<td><code>BUF_SIZE 80</code></td>
<td>Sets the buffer size for 80</td>
</tr>
<tr>
<td><code>COMPort port(&quot;COM1&quot;)</code></td>
<td>Calls to ADR interface port 1</td>
</tr>
<tr>
<td><code>comport.h</code></td>
<td>Library that provides the functionality to perform port communication</td>
</tr>
<tr>
<td><code>conio.h</code></td>
<td>Loads DOS specific commands, it is not a standard ANSI Commands</td>
</tr>
<tr>
<td><code>Define</code></td>
<td>Defines a variable in the program</td>
</tr>
<tr>
<td><code>If</code></td>
<td>The beginning of a common if then statement</td>
</tr>
<tr>
<td><code>Include</code></td>
<td>Initiates the library in the program</td>
</tr>
<tr>
<td><code>Int</code></td>
<td>Initiates a command</td>
</tr>
<tr>
<td><code>port.setBitRate</code></td>
<td>Sets the bit rate data rate expressed in bits per second. This is a similar to baud but the latter is more applicable to channels with more than two states.</td>
</tr>
<tr>
<td><code>port.setDataBits</code></td>
<td>Sets the data bit rate through the port</td>
</tr>
<tr>
<td><code>port.setParity</code></td>
<td>Sets the parity which is used in the error detection procedure</td>
</tr>
<tr>
<td><code>port.setStopBits</code></td>
<td>Sets the stop bits, which are extra &quot;1&quot; bits which follow the data and any parity bit. They mark the end of a unit of transmission</td>
</tr>
<tr>
<td><code>Port.write</code></td>
<td>Writes commands to the device connected to the port</td>
</tr>
<tr>
<td><code>Printf</code></td>
<td>Prints display in a command window</td>
</tr>
<tr>
<td><code>Sleep</code></td>
<td>Delays the program a specified amount of time</td>
</tr>
<tr>
<td><code>stdio.h</code></td>
<td>Library that provides ANSI I/O function library that allow reading and writing to files and devices</td>
</tr>
<tr>
<td><code>time.h</code></td>
<td>Library that provides time data</td>
</tr>
</tbody>
</table>
Appendix A: (Continued)

A.1 Automatic C++ Code

The following code is the automatically generated C++ code generated by the BarrettHand program. This code is useful if the C function library for the hand is installed. In this research, that library was not available, but an example of the code is as follows.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include "BHand.h"

BHand bh;   // Handles all hand communication
int value;  // Hand parameter obtained with Get
int result; // Return value (error) of all BHand calls

void Error(void)
{
    printf( "ERROR: %d\n%s\n", result, bh.ErrorMessage(result) );
    exit(0);
}

void Initialize(void)
{
    if( result=bh.InitSoftware(1,THREAD_PRIORITY_TIME_CRITICAL) )
        Error();
    if( result=bh.ComSetTimeouts(0,100,15000,100,5000) )
        Error();
    if( result=bh.Baud(9600) )
        Error();
    if( result=bh.InitHand("") )
        Error();
}

// Execute commands, return 1 if interrupted with a key
int Execute(void)
{
    printf( "Press Any Key to Abort..." );
    if( result=bh.GoToHome() )
        Error();
    if( _kbhit() )
    {
        _getch(); return 1; }
    return 0;
}
```

74
Appendix A: (Continued)

if( result=bh.Delay( 10000 ) )
Error();
if( _kbhit() )
{ _getch(); return 1; }
if( result=bh.GoToHome() )
Error();
if( _kbhit() )
{ _getch(); return 1; }
return 0;

// Main function - initialize, execute
void main(void)
{
    printf( "Initialization..." );
    Initialize();
    printf( " Done\n" );
    printf( "Executing - " );
    Execute();
    printf( " Done\n" );
}

A.2: BarrettHand Initialization C++ Code
This code was created for this project to open communication with the BarrettHand Port and to initialize the hand. Whenever using the BarrettHand in C++, this code must be run prior to any other code or else the GCL library will not be activated.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "comport.h"
#include <iostream.h>
#include <time.h>
#define BUF_SIZE 80
//global variables
int mStartFlagG = 0;
// prototypes
void sleep( clock_t wait );
int main()
{
    COMPort port("COM1");
    //char buffer[BUF_SIZE];
    //int bytesRead;
}
Appendix A: (Continued)

```c
int i = 0;
port.setBitRate(COMPort::br9600);
port.setParity(COMPort::None);
port.setDataBits(COMPort::db8);
port.setStopBits(COMPort::sb1);
//Command lines being sent to the hand
/*******************************************************************************
HI only needs to be run when first powering up the BHand, otherwise it can be
commented out, also is HSG is set to low, the fingers will not close. The
timer also needs to be places in-between each command to allow, otherwise the
command is bypassed
*******************************************************************************/
if (mStartFlagG == 0) {
printf("Hi Starting\n");
port.write("HI\r",3);
sleep( (clock_t)5 * CLOCKS_PER_SEC );
port.write("123FSET HSG 350\r",16);
}
sleep( (clock_t)1 * CLOCKS_PER_SEC );
sleep( (clock_t)7 * CLOCKS_PER_SEC );
return 0;
}
/* Pauses for a specified number of milliseconds. */
void sleep( clock_t wait )
{
  clock_t goal;
goal = wait + clock();
while( goal > clock() );
}

A.3: BarrettHand Demo C++ Code
This code was created to serve as a template for writing different
commands to the BarrettHand. Basically every closed loop GCL command is
tested in this program. This code is great for demonstrating the capabilities of the
BarrettHand.

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "comport.h"
#include <iostream.h>
#include <time.h>
define BUF_SIZE 80
```
Appendix A: (Continued)

    int mStartFlagG = 0;
    // prototypes
    void sleep( clock_t wait);
    int main()
    {
      COMPort port("COM1");
      //char buffer[BUF_SIZE];
      //int bytesRead;
      int i = 0;
      //int MtimeClose, MtimeOpen;
      //port.setBitRate(COMPort::br19200);
      port.setBitRate(COMPort::br9600);
      port.setParity(COMPort::None);
      port.setDataBits(COMPort::db8);
      port.setStopBits(COMPort::sb1);
      /*printf("Enter seconds to stay close: ");
       scanf("%i", &MtimeClose);
      printf("\nEnter seconds to stay open: ");
      scanf("%i", &MtimeOpen);
      while(port.read() != '+');
      while(1)
      {
        bytesRead = port.read(buffer, 5);
        //sleep(5000);
        sleep( (clock_t)MtimeClose * CLOCKS_PER_SEC );
        //getchar();
        buffer[bytesRead-1] = '0';
        cout << "Read " << bytesRead << " bytes. Message was:" << endl;
        cout << buffer << endl;
      }*/
      /*if (mStartFlagG == 0) {
        printf("Hi Starting\n");
        port.write("HI\r",3);
        sleep( (clock_t)5 * CLOCKS_PER_SEC );
        port.write("123FSET HSG 350\r",16);
      }
      sleep( (clock_t)5 * CLOCKS_PER_SEC );*/
      sleep( (clock_t)7 * CLOCKS_PER_SEC );
      port.write("123O\r",5);
      sleep( (clock_t)2 * CLOCKS_PER_SEC );
      port.write("123C\r",5);
      sleep( (clock_t)1 * CLOCKS_PER_SEC );
      port.write("123O\r",5);
Appendix A: (Continued)

```c
sleep( (clock_t)1 * CLOCKS_PER_SEC );
port.write("SC\r",3);
sleep( (clock_t)1 * CLOCKS_PER_SEC );
port.write("SO\r",3);
sleep( (clock_t)1 * CLOCKS_PER_SEC );
port.write("123C\r",5);
sleep( (clock_t)1 * CLOCKS_PER_SEC );
port.write("123O\r",5);
sleep( (clock_t)5 * CLOCKS_PER_SEC );
port.write("12C\r",4);
sleep( (clock_t)6 * CLOCKS_PER_SEC );
port.write("12O\r",4);
sleep( (clock_t)2 * CLOCKS_PER_SEC );
port.write("123SC\r",6);
sleep( (clock_t)1 * CLOCKS_PER_SEC );
port.write("123O\r",5);
sleep( (clock_t)3 * CLOCKS_PER_SEC );
port.write("123SO\r",6);
sleep( (clock_t)14 * CLOCKS_PER_SEC );
port.write("123C\r",5);
sleep( (clock_t)22 * CLOCKS_PER_SEC );
port.write("123O\r",5);
printf("T starting\n");
port.write("T\r",2);
mStartFlagG = 1;
return 0;
}
/* Pauses for a specified number of milliseconds. */
void sleep( clock_t wait )
{clock_t goal;
goal = wait + clock();
while( goal > clock() ) ;
}
```

A.4: Laser Range Finder C++ Code

The following code operates the laser range finder. The port is opened in the same way as the BarrettHand, but only values are read. The port automatically converts the bites the actual values, so no parsing is needed in C.

```c
#include "comport.h"
#include <iostream.h>
#include <math.h>
#include <stdio.h>
```
Appendix A: (Continued)

#include <stdlib.h>
#define BUF_SIZE 80
int main()
{COMPort port("COM2");
 char buffer[BUF_SIZE];
 int bytesRead, i;
 int NReadings = 5;
 port.setBitRate(COMPort::br9600);
 port.setParity(COMPort::Even);
 port.setDataBits(COMPort::db7);
 port.setStopBits(COMPort::sb1);
 float avg = 0.0;
 while(port.read() != '+');
 i = 0;
 while(i <= NReadings)
 {
  bytesRead = port.read(buffer, 5);
  buffer[bytesRead-1] = '0';
  //cout << "Read " << bytesRead << " bytes. Message was:" << endl;
  cout << buffer << endl;
  i++;
  avg = avg + atof(buffer);
 }
 avg = avg/(NReadings+1)*0.0393700787;
 printf("distance in inches is = %4.6f\n",avg);
 return 0;
}

A.5: Laser Range Finder with BarrettHand C++ Code

This code combines the BarrettHand with the laser range finder. It does this with an if statement that can be adjusted as needed. Basically, when a specific reading of the laser range finder occurs, the hand will close.

/*#include "comport.h"
#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <time.h>
#define BUF_SIZE 80
int main()
{COMPort port("COM2");
 char buffer[BUF_SIZE];
 int bytesRead;
Appendix A: (Continued)

```c
int n;
int avg;
int NReadings;
int i;
port.setBitRate(COMPort::br9600);
port.setParity(COMPort::Even);
port.setDataBits(COMPort::db7);
port.setStopBits(COMPort::sb1);
while(port.read() != '+');
{
  bytesRead = port.read(buffer, 5);
  buffer[bytesRead-1] = '0';
  //cout << "Read " << bytesRead << " bytes. Message was:" << endl;
  cout << buffer << endl;
  n = atof(buffer);
}
return 0;
*/
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include "comport.h"
#include <iostream.h>
#include <math.h>
#include <time.h>
#define BUF_SIZE 80
// prototypes
void sleep( clock_t wait );
int main()
{
  int NReadings = 10;
  char buffer[BUF_SIZE];
  int bytesRead, i;
  //create port1: will be assigned to BHand
  COMPort port1("COM1");
  port1.setBitRate(COMPort::br9600);
  port1.setParity(COMPort::None);
  port1.setDataBits(COMPort::db8);
  port1.setStopBits(COMPort::sb1);
  //create port2: will be assigned to Laser
  COMPort port2("COM2");
  port2.setBitRate(COMPort::br9600);
  port2.setParity(COMPort::Even);
  port2.setDataBits(COMPort::db7);
```
Appendix A: (Continued)

port2.setStopBits(COMPort::sb1);
float avg = 0.0;
while(port2.read() != '+');
i= 0;
//while (i <= NReadings)
for(;;) {
    i=0;
i++;
bytesRead = port2.read(buffer, 5);
buffer[bytesRead-1] = '\0';
//cout << "Read " << bytesRead << " bytes. Message was:" << endl;
cout << buffer << endl;
avg = 0.0;
avg = atof(buffer);
//avg = avg + atof(buffer);
avg = avg * 0.0393700787;
if(avg <= 5.0){ port1.write("123C\r",5);
    //sleep( (clock_t)1 * CLOCKS_PER_SEC );
    port1.write("123c\r",5);
sleep( (clock_t)100 * CLOCKS_PER_SEC);
    //port1.write("123T\r",5);
    port1.write("123o\r",5);
    avg = 100.0;}
}
return 0;
}
/* Pauses for a specified number of milliseconds. */
void sleep( clock_t wait )
{
clock_t goal;
goal = wait + clock();while( goal > clock() ) ;
}
Appendix B: MATLAB Code

MATLAB’s vision acquisition and processing toolboxes were utilized during this project. MATLAB was chosen because it can be easily integrated with the robot arm manipulation calculations. The toolboxes are able to be integrated with over the counter cameras such as ones from Logitech, Sony and other basic type web cameras.

The vision acquisition toolbox provides functions for acquiring and displaying images from a camera. It interfaces with Windows-compatible video-capture devices, such as USB and FireWire (IEEE-1394) scientific video cameras, as well as Web cameras, capture boards, and DV camcorders. There are only a few main commands used from this toolbox, which is shown below in Table 17.

<table>
<thead>
<tr>
<th>Command</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>data = getsnapshot(vid);</td>
<td>Gets an instant snapshot from the video device</td>
</tr>
<tr>
<td>Delete(vid);</td>
<td>Deletes the video to free extra memory</td>
</tr>
<tr>
<td>preview(vid);</td>
<td>Previews the video using a frame rate of 30 frames per second</td>
</tr>
<tr>
<td>set(vid.source,'Brightness',100);</td>
<td>Set the brightness for the previewed video</td>
</tr>
<tr>
<td>vid = videoinput('winvideo', 1);</td>
<td>Accesses a windows ready image device, such as a web camera</td>
</tr>
</tbody>
</table>

Once the image has been acquired, it is processed using MATLABs image processing toolbox. Table 18, located on the next page, shows the common commands and definitions for the MATLAB image processing toolbox [19].
Appendix B: (Continued)

Table 18: Common MATLAB Image Processing Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>rgb2gray</td>
<td>Convert RGB image or colormap to grayscale</td>
</tr>
<tr>
<td>Bwareaopen</td>
<td>Removes all objects underneath a specified pixel requirement</td>
</tr>
<tr>
<td>imfill</td>
<td>Fills in holes underneath a specified pixel requirement</td>
</tr>
<tr>
<td>bwboundaries</td>
<td>Traces the boundaries</td>
</tr>
<tr>
<td>centroid</td>
<td>Calculates the centroid</td>
</tr>
</tbody>
</table>

B.1 Laser Range Finder MATLAB Code

This code is used for the laser range finder. It first opens the port, then it obtains the data. Afterwards the data is parsed, meaning that the bits read are changed into values. When converting bits to numerical values the ASCII code for 0 -> 9 are 48 -> 57, so the value of a numeric character equals (ASCII code - 48). In Hex. (characters A -> F, 65 -> 70) would be (ASCII code - 55). Once the value of each digit is found, multiply each by the appropriate value (1, 10, 100) or (1, 16), and add the results.

```matlab
function avg1 = test_serial()
    clear
    clear all
    s1 = serial('COM1', 'BaudRate', 9600);
    fopen(s1)
    A = fread(s1,7)
    fclose(s1);
    A1(1) = 53;
    A1(2) = 48;
    A1(3) = 48;
    R = 0.0;
    R = (char(A(3)) - 48)*100+(char(A(4)) - 48)*10+(char(A(5)) - 48)
    % Parsing of Sick data set % 1
```
Appendix B: (Continued)

if A(2) == 177
A(2) = 48+1;
end;
if A(3) == 177
A(3) = 48+1;
end;
if A(4) == 177
A(4) = 48+1;
end;
if A(5) == 177
A(5) = 48+1;
end;

% 2
if A(2) == 178
A(2) = 48+2;
end;
if A(3) == 178
A(3) = 48+2;
end;
if A(4) == 178
A(4) = 48+2;
end;
if A(5) == 178
A(5) = 48+2;
end;
if A(2) == 179
A(2) = 48+3;
end;
if A(3) == 179
A(3) = 48+3;
end;
if A(4) == 179
A(4) = 48+3;
end;
if A(5) == 179
A(5) = 48+3;
end;
if A(2) == 180
A(2) = 48+4;
end;
if A(3) == 180
A(3) = 48+4;
end;
if A(4) == 180
A(4) = 48+4;
end;
if A(5) == 180
A(5) = 48+4;
end;
Appendix B: (Continued)

A(4) = 48+4;
end;
if A(5) == 180
A(5) = 48+4;
end;
% 5
if A(2) == 181
A(2) = 48+5;
end;
if A(3) == 181
A(3) = 48+5;
end;
if A(4) == 181
A(4) = 48+5;
end;
if A(5) == 181
A(5) = 48+5;
end;
% 6
if A(2) == 182
A(2) = 48+6;
end;
if A(3) == 182
A(3) = 48+6;
end;
if A(4) == 182
A(4) = 48+6;
end;
if A(5) == 182
A(5) = 48+6;
end;
% 7
if A(2) == 183
A(2) = 48+7;
end;
if A(3) == 183
A(3) = 48+7;
end;
if A(4) == 183
A(4) = 48+7;
end;
if A(5) == 183
A(5) = 48+7;
end;
Appendix B: (Continued)

% 8
if A(2) == 184
A(2) = 48+8;
end;
if A(3) == 184
A(3) = 48+8;
end;
if A(4) == 184
A(4) = 48+8;
end;
if A(5) == 184
A(5) = 48+8;
end;
% 9
if A(2) == 185
A(2) = 48+9;
end;
if A(3) == 185
A(3) = 48+9;
end;
if A(4) == 185
A(4) = 48+9;
end;
if A(5) == 185
A(5) = 48+9;
end;
R = [A(1) A(2) A(3) A(4) A(5)];
% current measurement is contained in A (elements 2, 3,4, and 5)
Rx = char(R);
avg1 = Rx;
display(avg1);

B.2: Centroid MATLAB Code
This code handles the main image processing in MATLAB. It is explained in full detail in Chapter 4.

function [xcen, ycen] = mycentroid()
vidobj = videoinput('winvideo', 2);
preview(vidobj);
pause(15);
fabric = getsnapshot(vidobj);
imwrite(fabric,'fabric2.png','png');
delete(vidobj)
clear vidobj;
Appendix B: (Continued)

fabric = imread('fabric2.png');
load regioncoordinates;
nColors = 6;
sample_regions = false([size(fabric,1) size(fabric,2) nColors]);
for count = 1:nColors
    sample_regions(:,:,count) = roipoly(fabric,region_coordinates(:,1,count),...
                                      region_coordinates(:,2,count));
end
%imshow(sample_regions(:,:,2)),title('sample region for red');
cform = makecform('srgb2lab');
lab_fabric = applycform(fabric,cform);
a = lab_fabric(:,:,2);
b = lab_fabric(:,:,3);
color_markers = repmat(0, [nColors, 2]);
for count = 1:nColors
    color_markers(count,1) = mean2(a(sample_regions(:,:,count)));
    color_markers(count,2) = mean2(b(sample_regions(:,:,count)));
end
disp(sprintf('%0.3f,%0.3f',color_markers(2,1),color_markers(2,2)));
color_labels = 0:nColors-1;
a = double(a);
b = double(b);
distance = repmat(0,[size(a), nColors]);
for count = 1:nColors
    distance(:,:,count) = ( (a - color_markers(count,1)).^2 + ... 
                           (b - color_markers(count,2)).^2 ).^0.5;
end
[value, label] = min(distance,[],3);
[rgb_label, idx] = false([1 1 3]);
segmented_images = repmat(uint8(0),[size(fabric), nColors]);
for count = 1:nColors
    color = fabric;
    color(rgb_label ~= color_labels(count)) = 0;
    segmented_images(:,:,count) = color;
end
%imshow(segmented_images(:,:,2)), title('red objects');
red1=segmented_images(:,:,2);
I = rgb2gray(red1);
threshold = graythresh(I);
bw = im2bw(I,threshold);
%imshow(bw)
Appendix B: (Continued)

%% Step 3: Remove the noise
% Using morphology functions, remove pixels which do not belong to the
% objects of interest.
% remove all object containing fewer than 30 pixels
bw = bwareaopen(bw,30);
% fill a gap in the pen's cap
se = strel('disk',2);
bw = imclose(bw,se);
% fill any holes, so that regionprops can be used to estimate
% the area enclosed by each of the boundaries
bw = imfill(bw,'holes');
%imshow(bw)

%% Step 4: Find the boundaries
% Concentrate only on the exterior boundaries. Option 'noholes' will
% accelerate the processing by preventing [bwboundaries] from searching
% for inner contours.
[B,L] = bwboundaries(bw,'noholes');
% Display the label matrix and draw each boundary
%imshow(label2rgb(L, @jet, [.5 .5 .5]))
%pause
a=label2rgb(L, @jet, [.5 .5 .5]);
%imshow(a);
L = rgb2gray(a);
threshold = graythresh(L);
bw = im2bw(L,threshold);
%imshow(bw)
%pause
bw2 = imfill(bw,'holes');
L = bwlabel(bw2);
s = regionprops(L, 'centroid');
centroids = cat(1, s.Centroid);
%Display original image I and superimpose centroids.
imtool(I)
hold(imgca,'on')
plot(imgca,centroids(:,1), centroids(:,2), 'g*
); plot(imgca, 160, 120, 'r*
);
hold(imgca,'off')
xcen = centroids(1,1);
ycen = centroids(1,2);

B.3: BarrettHand Demo

This code opens the port for the BarrettHand and demonstrates many of
the functions capable by the hand.
Appendix B: (Continued)

close all;
try
  fclose(instrfind); %close serial comm, if any
end;
%create serial communication with the hand
s1 = serial('COM1');
set(s1,'BaudRate',9600,'Parity','none', 'StopBits', 1, 'DataBits',8,...
  'terminator',13);
out1 = get(s1);
out2 = get(s1,{'BaudRate','DataBits'});
get(s1,'Parity');
fopen(s1);
%initialize hand once
s1.FlowControl = 'hardware';
temp = sprintf('%3s','HI');
fprintf(s1,temp);
pause(5);
temp = sprintf('%3s','1C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','2C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','3C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','1O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','2O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','3O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','SC');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','SO');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','SC');
fprintf(s1,temp);
pause(1.0);
89
Appendix B: (Continued)

```c
temp = sprintf('%3s','1C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','1O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','2C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','2O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','3C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','3O');
fprintf(s1,temp);
pause(1.0)
; temp = sprintf('%5s','123C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%5s','123O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%3s','SO');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','12IC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','13IC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','23IC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SOC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','12OC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','13OC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','13OC 5000');
fprintf(s1,temp);
pause(1.0);
```
Appendix B: (Continued)

```
pause(1.0);
temp = sprintf('%10s','23OC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SIC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SOC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','12IC 1000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','13IC 1000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%10s','23IC 1000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SIC 1000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SIO 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%5s','123C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%5s','123O');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SIC 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%9s','SIO 5000');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%5s','123C');
fprintf(s1,temp);
pause(1.0);
temp = sprintf('%5s','123O');
```
fprintf(s1,temp);
pause(1.0);
temp = sprintf("%5s","HOME");
fprintf(s1,temp);
pause(1.0);
temp = sprintf("%9s","SIC 300");
fprintf(s1,temp);
pause(1.0);
temp = sprintf("%9s","SIC 300");
fprintf(s1,temp);
pause(1.0);
temp = sprintf("%9s","SIC 300");
fprintf(s1,temp);
pause(1.0);
temp = sprintf("%9s","SOC 1000");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","1C");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","1O");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","2C");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","2O");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","3C");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%3s","3O");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%5s","123C");
fprintf(s1,temp);
pause(1.1);
temp = sprintf("%5s","123O");
fprintf(s1,temp);
pause(1.1);
Appendix B: (Continued)

B.4 BarrettHand and Laser Code

This code combines the BarrettHand software with the laser range finder. It calls up the laser range finder as a function, so all the parsing does not show up. This program handles port communication for both devices.

function testHand()
try
fclose(instrfind); %close serial comm, if any
end;
%create serial communication with the hand
s1 = serial('COM1');
set(s1,'BaudRate',9600,'Parity','none', 'StopBits', 1, 'DataBits', 8,...
    'terminator',13);
out1 = get(s1);
out2 = get(s1,{'BaudRate','DataBits'});
get(s1,'Parity');
fopen(s1);
temp = sprintf('%5s','123o');     %123o = open 3 fingers
fprintf(s1,temp);
%initialize hand once
s1.FlowControl = 'hardware';
%if mdone == 1
%  temp = sprintf('%3s','HI')
%  fprintf(s1,temp);
%  pause(20);
%end;
%mdone = 0;
%create serial communication with Laser Ranger
s5 = serial('COM5', 'BaudRate', 1200);
%connect the serial port object to the serial port (COM5)
fopen(s5);
temp = sprintf('%5s','home');     %123o = open 3 fingers
fprintf(s1,temp);
%simulation
dt = 0.03; %sampling period in seconds
t=0;tm=0;nt=0;done=0;tic;
timeFinal = 2.1500;
avg1 = 300;
%while (t < timeFinal)
while(1)
    while tm<nt;tm=toc;end;t=nt;nt=t+dt;
    A = fread(s5,7);
    fprintf('i am here %d',i);
Appendix B: (Continued)

A = ParsingLaser(A);
R = [A(1) A(2) A(3) A(4) A(5)];
% current measurement is contained in A (elements 2, 3, 4, and 5)
Rx = char(R);
Rx = str2num(Rx);
avg1 = double(Rx);
%end;
if (isempty(avg1) == 1)
%do nothing
%fprintf('wrong reading');
end;
if ((isempty(avg1) == 0) && (avg1 >= 199))
mok = 0;
if (avg1 <=405)
fprintf('%d',avg1);
fprintf('Close the hand');
%close hand
temp = sprintf('%5s','123c'); %123c = close 3 fingers
fprintf(s1,temp);
% pause (45)
% temp = sprintf('%5s','123o'); %123o = open 3 fingers
%fprintf(s1,temp);
break;
fclose(s1);delete(s1);
fclose(s5);delete(s5)
end;
end;

B.5 Final Integration

This is the final code used for running all the programs. It calls to the laser range finder code and the BarrettHand code with laser range finder to ease computation time.

function [finalx,finaly,avg1] = newcamera()
clear all;
close all;
try
fclose(instrfind); %close serial comm, if any
end;
[xcen, ycen] = mycentroid();
%create serial communication with Laser Ranger
s5 = serial('COM5', 'BaudRate', 1200);
%connect the serial port object to the serial port (COM5)
Appendix B: (Continued)

```matlab
fopen(s5);
% simulation
dt = 0.1; % sampling period in seconds
t=0;tm=0;nt=0;done=0;tic;
timeFinal = 2.1500;
avg1 = 300;
% while (t < timeFinal)
while tm<nt;tm=toc;end;t=nt;nt=t+dt;
% for j=1:3
A = fread(s5,7);
% fprintf('i am here %d',i);
A = ParsingLaser(A);
R = [A(1) A(2) A(3) A(4) A(5)];
% current measurement is contained in A (elements 2, 3, 4, and 5)
Rx = char(R);
Rx = str2num(Rx);
avg1 = double(Rx);
% end;
avg1=avg1/1000;
%x=centroids(:,1);
%y=centroids(:,2);
x = xcen; y = ycen;
yy = (0.6521*avg1-0.1978)/10;
xx = (0.8407*avg1 -0.2435)/10;
xxx=xx/320;
yyy=yy/240;
%xx1=xxx*centroids(:,1)
xx1 = xxx*xcen;
yy1 = yyy*ycen;
fprintf('x coord = %4.3f, y coord = %4.3f
',xx1,yy1);
% now to get results from center)
centx=xxx*.5;
centy=yyy*.5;
finalx=(centx-xx1)*(1)
finaly=(centy-yy1)*(1)
d_C2LR = .28;
avg1=avg1-d_C2LR; % subtract distance between camera and LR
fprintf('x coord = %4.3f, y coord = %4.3f, z coord = %4.3f
',finalx,finaly,avg1);
% hold(imgca,'off')
% disp(centroids);
if (avg1 ~= 0)
testHand()
end;
```
Appendix B: (Continued)

%%avg 1= z coordinate and xx1 and yy1 are the x and y coordinates of a part
%%from the left side of the screen (this can be changed). Once this is
%%completed the arm will be given some time to respond by being sent a
%%command to move incrementally closer to the location. Another picture
%%will be taken.