Haptic Control with a Robotic Gripper

Morgan Rody
Technology
Haptic Control with a Robotic Gripper
Morgan Rody

Haptic Control with a Robotic Gripper

Supervisor: Prof. Ivan Kalaykov
Examiners: Dr. Boyko Iliev
MSc. Robert Krug
Haptic Control with a Robotic Gripper

Morgan Rody
Technology
Haptic Control with a Robotic Gripper
Abstract

The Novint Falcon is a low cost, 3-axis, haptic device primarily designed and built for the gaming industry. Meant to replace the conventional mouse, the Novint Falcon has sub-millimeter accuracy and is capable of real time updates. The device itself has the potential to be used in telerobotics applications when coupled with a robotic gripper for example. Recently, the Intelligent Control Lab at Örebro University in Sweden built such a robotic gripper. The robotic gripper has three fingers and an additional w-axis which is able to move two of the fingers into alternate positions. Four motors control the movement of each respective axis.

The purpose of this thesis is to integrate the afore mentioned devices in such a way that the Novint Falcon will control the robotic gripper movements. As the user moves the Novint Falcon handle in, the robotic gripper fingers will match this movement and close. Grasp and Home Position sub-routines will be implemented to showcase the buttons found on the Novint Falcon handle.

The entire control system consists of a user friendly graphical user interface (GUI), an initialization procedure for both devices and workspace matching functions to ensure that any movement in the Novint Falcon is mirrored in the robotic gripper. The robotic gripper is controlled by a Galil motion controller.

Experiments with haptic feedback have been carried out with Squirrel scripting and in C++. The GUI has been written using OpenGL and OpenGLUT resulting in a simple, yet informative user GUI. The robotic gripper experiments have been carried out by using Galil Tools Lite, C++ and the Galil programming language.

All experiments have led to a robust and stable control system which allows the Novint Falcon to control all movements available on the robotic gripper, paving the way for future telerobotic applications.
Acknowledgements

There are many people I would like to thank for their support during the development of this Masters Thesis. Firstly, I would like to thank my supervisor, Professor Ivan Kalaykov for his support, assistance and wise feedback. I’d like to also thank Associate Professor Anani Ananiev for designing and building the Robotic Gripper that was used.

I also want to thank the classmates that I have worked with, during this Masters Thesis and during the numerous projects that we have worked on together during the Robotics and Intelligent Systems program. Your humor and friendship has helped make all of this possible, thank you!

Last but not least, I would like to thank my sambo, E, for her infinite patience and support, especially during the writing phase.

I hope you enjoy reading this Masters Thesis...
## Contents

### Appendix

1 Introduction ........................................... 19  
   1.1 Haptic Feedback ................................... 19  
   1.2 The Novint Falcon .................................. 20  
   1.3 Robotic Gripper ..................................... 20  
   1.4 Control System ..................................... 21  
   1.5 Goal .................................................. 21  
   1.6 Workflow ............................................. 22  
   1.7 Practical Applications ............................... 22  
   1.8 Contributions ....................................... 23  
   1.9 Previous Work ....................................... 23  
   1.9.1 Novint Falcon .................................... 23  
   1.9.2 Robotic Gripper ................................. 24  
   1.9.3 Galil Motion Controller ......................... 24  
   1.10 Thesis Structure .................................... 24  
   1.10.1 Chapter 1: Introduction ......................... 24  
   1.10.2 Chapter 2: Equipment ........................... 24  
   1.10.3 Chapter 3: Developed System .................... 24  
   1.10.4 Chapter 4: Experiments ......................... 25  
   1.10.5 Chapter 5: Conclusion ......................... 25  
   1.10.6 Appendices ....................................... 25

2 Equipment .............................................. 27  
   2.1 Hardware ............................................. 27  
       The Novint Falcon .................................... 27  
       Robotic Gripper ..................................... 27  
       Galil Motion Controller Unit ........................ 31  
   2.2 Software ............................................. 33  
       Novint Falcon Bundled Software .................... 33  
       Squirrel ............................................. 35

11
Novint Falcon Software Development Kit (SDK) .................................. 36
Galil Motion Controller ................................................. 37
Communication .......................................................... 38

3 Developed System .................................................... 39
3.1 Overview ............................................................ 39
3.2 GUI ................................................................. 39
3.3 Novint Falcon Handle Buttons ................................... 41
3.4 Novint Falcon, GUI and Robotic Gripper Workspace Matching 42
   Novint Falcon Workspace ........................................... 42
   GUI Workspace ..................................................... 43
   Robotic Gripper ..................................................... 43
3.5 Initialization ........................................................ 43
3.6 Haptic Forces ....................................................... 44

4 Experiments .......................................................... 47
4.1 Overview ............................................................ 47
4.2 Novint Falcon ....................................................... 47
   Squirrel ............................................................. 47
   GUI ................................................................. 49
   SDK (HDAL) ........................................................ 49
4.3 Robotic Gripper ..................................................... 50
   Galil Motion Controller .......................................... 50
   Force Sensitive Resistor (FSR) Sensors ....................... 51
   Motors ............................................................. 51
4.4 Summary ........................................................... 51
   Software ............................................................ 52
   Hardware ........................................................... 52

5 Conclusion ............................................................ 55
5.1 Summary ............................................................ 55
5.2 Future Work ........................................................ 56
   Software ............................................................ 56
   Hardware ........................................................... 56

A Novint Falcon ........................................................ 59

B General Novint Falcon Software Setup ................................ 61

C General Squirrel Software Setup ...................................... 63

D Novint Falcon SDK Setup ............................................... 65

E OpenGLUT ............................................................ 69
<table>
<thead>
<tr>
<th>CONTENTS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>F Galil Tools Lite</td>
<td>71</td>
</tr>
<tr>
<td>G Galil Commands in C++</td>
<td>73</td>
</tr>
<tr>
<td>H Source Code - Squirrel</td>
<td>75</td>
</tr>
<tr>
<td>I Source Code - Falcon Control</td>
<td>83</td>
</tr>
<tr>
<td>J Source Code - Galil Motion Controller</td>
<td>95</td>
</tr>
</tbody>
</table>
# List of Figures

1.1 Leading Haptic Devices................................................. 19
1.2 Novint Falcon and Pistol Grip........................................ 20
1.3 Robotic Gripper v2.0 and Galil Motion Controller Unit .......... 21
1.4 General Workflow ....................................................... 22

2.1 Robotic Gripper .......................................................... 28
2.2 Robotic Gripper Motor Movement Reference ...................... 29
2.3 Force Sensitive Resistor (FSR) Sensor ............................... 30
2.4 Inductive Proximity Limit Switches .................................. 31
2.5 Alternate Gripper Finger Positions (w-axis) ....................... 32
2.6 Galil Motion Controllers ............................................... 33
2.7 System Communications ................................................. 33
2.8 NVeNT Software .......................................................... 34
2.9 F-Gen ........................................................................ 35

3.1 GUI ............................................................................. 40
3.2 Novint Falcon Grip (Handle) Buttons ................................. 41
3.3 Spring Mass Damper Model .............................................. 45

4.1 How Timings are Handled with F-Gen ................................. 48

B.1 NDSSetter Path .............................................................. 62
D.1 NDSSetter SDK Path ...................................................... 65
D.2 "Include" Path ............................................................... 66
D.3 "Lib" Path ..................................................................... 66
D.4 "Include" Path ............................................................... 67
D.5 "Lib" Path ..................................................................... 67
D.6 Library File "hdl.lib" ..................................................... 68

E.1 Confirmation of IP Connection to Galil motion controller ......... 72
List of Tables

3.1 Actual Limits and Range (inches) for Novint Falcon handle . . . 42
3.2 Chosen Limits and Range (inches) for Novint Falcon handle . . 42
3.3 Actual and Safe ROM (quadrature units) . . . . . . . . . . . 43
Chapter 1
Introduction

1.1 Haptic Feedback

Haptic devices or force feedback devices, add a third sense (sight, sound and touch) to what a user experiences when controlling an electronic device. This "touch" feedback can assist an operator by giving touch feedback when an object is grasped by a gripper for example, or further immerse a gamer into a computer generated world by providing touch feedback when their player is injured. This touch feedback can come in the form of vibrations or pushing / pulling actions supplied by a built in motor in the controller. The first force feedback systems were built in the 1950’s to handle radioactive waste through teleoperation, while the first gaming application was developed by SEGA in 1975 for a motorcycle driving game. "Moto-Cross" implemented haptic feedback by vibrating the handle bars if a vehicle collision occurred. Car driving games later incorporated haptic feedback into the steering wheel to simulate a collision or hard turn which introduced a type of "push" haptic feedback.

(a) Delta 6 DOF  (b) Omega 3 DOF  (c) Omni 6 DOF

Figure 1.1: Leading Haptic Devices

In 1997, the "Rumble Pak" was used with the "Nintendo 64" and allowed haptic feedback controllers to be used in the average home. These smaller con-
controllers were purely hand held, therefore limited to a "vibration" type of haptic feedback, generally when your player took damage or bumped into an object. In addition to a strong gaming interest, research institutions used haptic feedback devices in teleoperation applications such as material handling and medical surgery. Unfortunately, these devices cost between $10,000 and $25,000 US! Three of the most popular devices used in research are shown in Figure 1.1.

1.2 The Novint Falcon

In 2007, the Novint Falcon (Figure 1.2a) was released at the Consumer Electronics Show (CES) primarily for the gaming industry, making it the cheapest and most versatile haptic feedback device offered on the consumer market. The optional "pistol grip" attachment (Figure 1.2b) makes the Novint Falcon an excellent addition to any of the supported FPS (First Person Shooter) games, allowing gamers to immerse themselves even further into the gaming experience. Due to its low cost and simplicity, the Novint Falcon has been well received by gamers, enthusiasts and researchers alike.

(a) Novint Falcon with Standard Grip  (b) Optional Pistol Grip

Figure 1.2: Novint Falcon and Pistol Grip

1.3 Robotic Gripper

The Novint Falcon is an interesting device with numerous research directions, one of which is to control another device using the Novint Falcon and its haptic capabilities. The device to be controlled by the Novint Falcon is a Robotic Gripper (Figure 1.3a) which has been built by the AASS Intelligent Control

---

1 Novint Falcon refers to the object a user grasps to control the entire unit as a "Grip" but this terminology will be replaced with "Handle" to avoid confusion with the frequently used term, "Gripper".

2 Applied Autonomous Sensor Systems
1.4 Control System

In order to control the robotic gripper, a Galil motion controller (Figure 1.3b) is used. This allows complete control of all motors, encoders, sensors and limit switches that are located on the robotic gripper. Accessing these motors, sensors etc, is done with Galil’s own communications utility, Galil Tools-Lite which is a free download from their website. This offers a quick and easy interface, including a program editor, watch list and a terminal.

1.5 Goal

The purpose of this thesis is to create an integration between the Novint Falcon gaming device and a robotic gripper which has been developed in the AASS Intelligent Control Lab. Moving the Novint Falcon handle will result in a corresponding movement on the robotic gripper. Haptic (touch) feedback will make the user aware when any limits are reached with the robotic gripper. Three sensors (one on each finger) will offer haptic feedback once an object is firmly grasped. In addition to an integration of movement, pressing the button(s) on the Novint Falcon handle will move the gripper fingers to predetermined positions. If time is available, a separate control scheme will be written for the ABB
industrial robot located in the AASS Lab. This will allow the Novint Falcon to control two devices, the robotic gripper and the ABB industrial robot. Pressing the chosen button on the Novint Falcon handle will toggle control between the two devices.

1.6 Workflow

The general workflow (Figure 1.4) of this project will begin with movement in the Novint Falcon by the user. A corresponding movement will be seen in the robotic gripper via the Galil motion controller. All communications, control schemes and haptic functions will be carried out in the main program which will rely on the Novint Falcon SDK.

![Workflow Diagram](image)

Figure 1.4: General Workflow

1.7 Practical Applications

Some of the advantages with using the Novint Falcon include a fast refresh rate, sub millimeter accuracy and real-time control possibilities. Teleoperation is fast becoming a popular branch of robotics, primarily in the medical, industrial and military areas of development. Most recently, teleoperated robotics have been used in the damaged Japanese Nuclear Plant[1] and years earlier during the search for survivors at the World Trade Center disaster. The Novint Falcon was not used during these unfortunate events but this does highlight the potential of such a device. Teleoperated Medical Surgery is another area in development and the Novint Falcons real-time control, accuracy and ability to offer haptic feedback to the user opens many possibilities.

---

[1] Fukushima Nuclear Facility was hit with a tsunami in 2011
1.8 Contributions

A successful integration of the Novint Falcon and robotic gripper has been achieved. The haptic functionality was tested using the Squirrel programming language and later implemented using the Novint Falcon SDK to alert the user when a physical limit has been reached by the robotic gripper fingers. Warning messages are also displayed in the GUI when physical limits are reached. Separate "Home Position" and "Grasp" programs have been written in the Galil motion controller language (DMC) and each can be activated by pressing the "Lightning" and "Plus" buttons on the Novint Falcon handle respectively. Control between all three fingers and the "alternate finger positions" is toggled by pressing the "Triangle" button. The fingers will stop grasping an object once sensed via the FSR sensors.

A basic GUI was built using OpenGL and OpenGLUT showing the position of each finger and which state of control is currently being used. The position of each finger is represented by a solid yellow ball whose position is updated when the Novint Falcon handle is moving along the z-axis for three finger movement or the x-axis for alternate finger position movement.

Integrating control of the ABB IRB-140 industrial robot with the Novint Falcon was mentioned at the start of this thesis and would be implemented if time remained. Unfortunately, this ambitious project did not see fruition although very basic functionality has been implemented. Pressing the "Logo" button will toggle control between the Novint Falcon and the ABB industrial robot in text at least, so only the code needs to be inserted.

1.9 Previous Work

Novint Falcon

This is the first project at the AASS Lab to deal with the Novint Falcon, therefore much of the information in this thesis has been presented in a manner which allows future projects a head start in their development. All information to install, setup and operate the Novint Falcon has been included in a thorough guide with screenshots. Information on the internet is sparse at best but developers can ask for advice at Novint Falcons own forums, "The Falcon Army". Keep in mind that the "Falcon Army" is a forum designed to primarily assist gamers with their purchased Novint Falcon and as such, little help from a researcher or developers aspect is available (but there is no harm in asking).

---

1 www.falconarmy.com
Robotic Gripper

The robotic gripper used (v2.0) was only completed in May 2011 and is still in evolution as improvements can be made. The previous version was used in at least one masters thesis during 2010\footnote{"Vision Based Grasp Planning for Robot Assembly" by Naresh Marturi, Örebro University}. The 2010 thesis used a vision system to determine which work piece will be grasped and then arranged the robotic gripper accordingly to firmly grasp the work piece for assembly. Due to its weight, the first robotic gripper has been replaced by v2.0 which offers reduced weight and a smaller platform as two of the main benefits.

Galil Motion Controller

The Galil motion controller has been used on many projects in the AASS Lab and is an industrial leader in providing Motion Control Solutions to nearly every industry imaginable. Full installation procedures, examples and a small forum can be found on their homepage\footnote{www.galilmc.com}. Instructions have been included in Appendix G (pg 73) on how to send commands to the Galil motion controller from within C++. The command set used with Galil motion controllers is quite straightforward with manuals available from their website to suite the chosen controller.

1.10 Thesis Structure

Chapter 1: Introduction

A brief background of how Haptics have developed is covered along with an introduction of the Novint Falcon, robotic gripper, Galil motion controller and the General Workflow. A small section on the practical applications is discussed.

Chapter 2: Equipment

Any hardware or software used in this project is described in this chapter.

Chapter 3: Developed System

The finished GUI interface and system is described in full here.
Chapter 4: Experiments

Any experiments carried out are discussed which includes the Squirrel programming language, Novint Falcon SDK, the robotic gripper and the Galil motion controller.

Chapter 5: Conclusion

A summary of the tasks completed and suggestions for future work can be found in chapter five.

Appendices

Any setup and installation instructions have been included in the appropriate Appendix so that this project can be quickly re-created and built upon in the future. All Squirrel script, C++ and Galil code has been included in this section for reference.

Note:
For ease of reading, any references used have been included as footnotes.
Chapter 2
Equipment

2.1 Hardware

The Novint Falcon

The Novint Falcon (Figure 1.2a pg 20) requires a power supply, USB port and has three arms which join a removable handle that replaces the conventional mouse. The base has extra weight in order to provide necessary stability when larger haptic forces are triggered. The user is able to move the handle in three dimensions which follows the Right-Hand Rule; left-right (x-axis), forwards-backwards (z-axis) and up-down (y-axis). Three motors inside the Novint Falcon produce haptic feedback when triggered by the software while three optical encoders track the handle position with sub-millimeter accuracy at a rate of 1000 times per second. It is important to note that the handle position is represented by (x, y, z) which is a combined calculation of the three encoder positions. The top motor / arm, for example, does not represent the z-axis respectively as one might assume when first looking at the Novint Falcon. Instead the z-axis is represented by moving the handle in and out. The default handle is a sphere with 4 buttons on the top while the optional "Pistol Grip" (Figure 1.2b pg 20) has three buttons on the right side and replaces one button with a gun trigger, making it the handle of choice when playing FPS games. See Appendix A (pg 59) for the Novint Falcon Technical Specifications and the Recommended System Requirements.

Robotic Gripper

This second iteration of the AASS robotic gripper (Figure 2.1) has 3 single-joint "fingers"\(^1\) 4 DC motors (1 brushed and 3 brushless), 4 incremental encoders, 3 force sensitive resistor (FSR) sensors, 5 inductive proximity limit switches, a

\(^1\) refers to the robotic gripper
standard definition web camera and 12 LED lights to illuminate grasped objects. The fingers move along a straight horizontal plane when they are opened or closed while their grasping surface remains completely vertical. Each finger has a total horizontal travel distance of 22 cm and does not travel vertically when being moved.

Figure 2.1: Robotic Gripper
2.1. HARDWARE

Motors

All motors are manufactured by Maxon Motors. Each finger is opened and closed by one 12V DC Brushless Motor. A fourth 12V DC Brushed Motor is used to move two of the fingers (x-axis and y-axis) into alternate finger positions, ranging from 0° to 90°, allowing for a wider variety of shapes to be grasped. It should be noted that the motor controlling the x-axis finger is mounted in an opposite manner compared to the y-axis and z-axis.

Figure 2.2: Robotic Gripper Motor Movement Reference

1 http://www.maxonmotor.com
2 this is also referred to as the "w-axis"
Force Sensitive Resistor (FSR) Sensors

Each finger has a force sensitive resistor sensor (Figure 2.3) on its vertical surface. These sensors are used to sense when an object or work piece is firmly grasped by the robotic gripper finger(s) which results in a no-contact / contact state only.

Inductive Proximity Limit Switches

There are five inductive proximity limit switches (Figure 2.4) which prevent the fingers from traveling beyond their physical limits. Three of these switches are used at the home position, one for each finger. The remaining two switches are used to stop the alternate finger positions at their extreme limits, 0° to 90° (Figure 2.5).
2.1. HARDWARE

Galil Motion Controller Unit

The Galil motion controller unit is able to control all motors, limit switches, sensors and consists of two items, a motion controller (DMC-2143) (Figure 2.6a) and an amplifier (AMP-20540) (Figure 2.6b). The DMC-2143 has room for 4 axis inputs and a large number of Auxiliary Input/Outputs. Communication can be done via serial port or ethernet. In this case, ethernet is used via a switch as seen in Figure 2.7.
CHAPTER 2. EQUIPMENT

(a) Normal Gripper Finger Position (90°)

(b) Alternate Gripper Finger Position (0°)

Figure 2.5: Alternate Gripper Finger Positions (w-axis)
2.2 Software

Novint Falcon Bundled Software

Novint has bundled a variety of games with the Novint Falcon in a "Limited Edition Novint Falcon Bundle" which also includes a very useful tutorial. Running the tutorial is highly recommended as it highlights the best aspects of the Novint Falcon, its haptic feedback abilities. Various surfaces are simulated during the tutorial, ranging from sandpaper to ice to honey, each of which simulates the actual "feel" as the user is moving the handle over its surface. Moving the handle over sandpaper results in a grainy, rough haptic sensation, while the honey will produce a very sticky sensation, making it difficult to move the handle away. Furthermore, the tutorial highlights another haptic aspect with a solid cube which stops the handle from penetrating the cube once you contact the surface. Through these simple examples, it is easy to see how haptic feedback can drastically alter a user's digital world perception when using the Novint Falcon.

The bundled games are "Newton’s Monkey Business" and "The Feelin’ It Sports Pack" which are accessed via a Steam-like interface called NVeNT (Fig-
Figure 2.8: NVeNT Software

which is primarily aimed at a younger audience. Hundreds of games can be purchased and accessed through the NVeNT interface. Once purchased, customers can use the bundled haptic responses or create, customize and use their own. Novint Falcon support exists for numerous top end gaming titles which includes Half-Life 2, Call of Duty, F.E.A.R., Left 4 Dead and most recently, Crysis 2. The user is able to customize the amount and type of feedback using Novint’s own customization tool, "F-Gen" which will produce a set of F-Gen Drivers (Figure 2.9).

This mainly centers around the "feel" given for a particular attack or defense while playing the game of choice and despite Novint including a bundled haptics package, there is strong demand for customization among users. "The Falcon Army", Novint’s own forum group, provides a common ground where users can come together, share ideas, customizations and educate one another on haptics in general.

To use the Novint Falcon and complete the recommended tutorial, please see Appendix B (pg. 61) for the General Setup instructions.
2.2. SOFTWARE

The simplest introduction to programming the Novint Falcon is to use a scripting language called "Squirrel". "Squirrel is a high level imperative, object-oriented programming language, designed to be a light-weight scripting language that fits in the size, memory bandwidth, and real-time requirements of applications like video games." Developed by Alberto Demichelis in 2004, Squirrel has a syntax similar to C, C++, Java and has progressed to v3.0 with further developments planned for future releases. When using Squirrel, very little prior programming experience is required to successfully execute haptic commands with the Novint Falcon. Setting up Squirrel is a straightforward process and a brief setup guide is in Appendix C (pg 63). Button states and keyboard mapping can be easily programmed to provide various levels of "feedback" to the user. Squirrel scripting is an excellent way to become familiar with the Novint Falcon and was used to determine what is haptically possible with the Novint Falcon, thereby giving fresh ideas for the main project. Specifically, the keyboard was mapped to moving the Novint Falcon handle up, down, left, right, forward and backwards. The buttons on the handle were mapped to activate different levels of haptic feedback ranging from no feedback to continuous feedback. The remaining buttons on the handle were mapped to output the current position of each axis found on the Novint Falcon. The script used for these tasks can be found in Appendix H (pg 75).

1 http://www.squirrel-lang.org
CHAPTER 2. EQUIPMENT

Novint Falcon Software Development Kit (SDK)

Once familiar with Squirrel, the next step was to delve into the Haptics Distribution Abstraction Layer (HDAL), which is provided with the Novint Falcon SDK. The HDAL allows near total control of the Novint Falcon but does require a solid background in C++ programming skills.

The Novint Falcon was released in 2007 on the Windows XP platform and is written using C++ in the Visual Studio 2005 (VS 2005) IDE. Windows XP has since been replaced by Windows 7 which works just as well. Although newer versions of VS are available, issues when trying to convert from VS 2005 to VS 2008 or VS 2011 do exist so it best to avoid newer versions of VS altogether. The HDAL uses OpenGL for graphical representations of the Novint Falcon cursor and for any objects it interacts with on the screen. Learning the basics of OpenGL is recommended but any advanced understanding is not necessary as OpenGLUT can be used due to its simplicity. OpenGLUT, an extension of the original GLUT project, making it easier to learn and explore OpenGL programming by offering a portable API so one can write a single OpenGL program that works across all OS platforms. GLUT is owned and copyrighted by Mark Kilgard who stopped working on the project in 1999. Since this time, alternatives to GLUT have surfaced with OpenGLUT being one of the better choices, in large part for its open source licensing.

It should be added that using OpenGL (OpenGLUT) is preferred over DirectX and 3D Game Studio due to its simplistic implementation. If the preference is to use DirectX or 3D Game Studio then please read the "HDAL Programmers Guide" located in the SDK documentation (HDAL_SDK_2.1.3/doc) folder.

Follow the steps in Appendix D (pg 63) to install the Novint Falcon SDK (HDAL). Appendix E (pg 69) describes what is needed in order to use OpenGLUT.

Novint Falcon SDK Basic Example

To ensure that the Novint Falcon SDK is installed correctly, it is recommended to run the included "Basic" example but first ensure that the Novint Falcon is connected and powered on.

- Navigate to "C:/Program Files/Novint/HDAL_SDK_2.1.3/examples /Basic/vs2005/basic_opengl".

1 http://www.openglut.sourceforge.net/index.html
2 http://www.opengl.org/resources/libraries/glut
• Open the VS 2005 project file 'basic_opengl.vcproj' and run the project.

A new window will appear with a 3-Dimensional Cube in its center and the cursor will be represented by a sphere which is controlled by the Novint Falcon. The main purpose of this example is to show the haptic capabilities of the Novint Falcon where the sphere (cursor) is unable to penetrate any side of the solid cube. This is conveyed to the user by the abrupt stop of Novint Falcon movement once the sphere comes in contract with the cube. This holds true for each side of the solid cube.

**Galil Motion Controller**

**Galil Tools Lite**

Sending commands to the Galil motion controller is the final piece needed to control the robotic gripper. A free software tool is available from the Galil Motion Control website[^1], "Galil Tools Lite", which allows a user to control any devices connected, view their current state and create larger programs which can be later downloaded to the controller itself for automatic execution. Galil Tools Lite is available on Windows (x86 and x64), Linux and Mac platforms. There are two methods to send commands with Galil Tools Lite. By using the Terminal (Tools, Terminal), a user is able to send single line commands (one command or multiple commands separated by a ";") to the motion controller. All commands are case sensitive and sent by using upper case letters. It is recommended that the user open the "Watch" window (Tools, Watch) and display the appropriate device states as this is useful when starting to use or troubleshoot an unfamiliar device. The second method is to write your own program (File, New, Program) and once complete, click on the "Execute" arrow to run your program.

Refer to Appendix F (pg 71) on how to send your first commands to the robotic gripper using Galil Tools Lite.

**Galil Commands in C++**

Galil Tools Lite has allowed a certain level of familiarity and exploration of what is possible with the robotic gripper, but our purposes require Galil commands to be sent from within the Falcon C++ code. The setup is quite straightforward and similar to that of the Novint Falcon SDK. Follow the steps in Appendix G (pg 73) to start sending commands from within C++.

[^1]: [www.galilmc.com](http://www.galilmc.com)
Communication

The hardware and software is connected either via USB or Ethernet port and all communication protocols have been included in the respective SDK or installed software. This has made the setup of each device and accompanying software a pleasant experience.
Chapter 3
Developed System

3.1 Overview

This thesis has set out to use the Novint Falcon as a control device for the robotic gripper where the use of a keyboard is not considered. The main motion to be controlled is the opening and closing of the robotic gripper, meaning that as the handle of the Novint Falcon moves in, the fingers on the robotic gripper will close. Inversely, if the handle moves out, the fingers will open. Two functions have been developed, "Home Position" and "Grasp" which are activated by pressing the respective buttons on the Novint Falcon handle. Keeping in mind that this has the potential to be a telerobotics system, it is important to have a visual interface for the user. Therefore a simple GUI has been built using OpenGL and OpenGLUT.

3.2 GUI

Although haptic feedback is used to inform the user when limits are reached or work pieces are grasped, it is important to provide visual feedback when developing any complex system. Users can forget which mode they are currently operating in or perhaps cannot imagine when the device they are controlling is near a physical limit (in terms of telerobotics). These issues have been addressed by providing all necessary information in a straight-forward-easy-to-understand GUI (Figure 3.1).

Each gripper finger is represented by a solid yellow circle and travels in a linear path with the limit of travel being represented by solid white lines. Moving the Novint Falcon (in default mode) along the z-axis (in / out) will move all three circles towards the center. Any movement along the x-axis or y-axis is not registered. When in "Alternate Finger Position" mode, the handle is able to control the position of the w-axis which is represented by two solid white lines behind
CHAPTER 3. DEVELOPED SYSTEM

(a) GUI Default Mode

(b) GUI w-axis Mode

Figure 3.1: GUI

the x and y-axis solid yellow circles in the GUI\textsuperscript{1}. Moving the handle along the x-axis (left / right) will change the position of the w-axis in the GUI and on the robotic gripper. Any movement along the z-axis or y-axis is not registered. If the user goes beyond any preset limits, haptic feedback is triggered along with a warning in the middle of the GUI. The current position of the x, y and z-axis is printed on the screen respectively in real time and these represent the location of the handle, not how far out a respective arm is from the Novint Falcon.

It is worth mentioning that the GUI was controlled by a variety of keyboard keys in the beginning stages of development. The keyboard was first used due to its simplicity but as the project evolved, all control of the GUI was transferred to the Novint Falcon handle and its four buttons.

\textsuperscript{1} It was originally thought to move the solid yellow circles along the w-axis white lines but this was not realized.
3.3 Novint Falcon Handle Buttons

The Novint Falcon handle (Figure 3.2) has four buttons and each button has been assigned a specific function.

- Pressing the "Logo" button will alternate control of the robotic gripper (default) to the ABB industrial robot and vice versa.
- Pressing the "Triangle" button will toggle between controlling all three fingers (default) of the robotic gripper or the w-axis (x-axis and y-axis fingers) to set a new alternate finger position.
- Pressing the "Lightning" button will activate the home position function.
- Pressing the "Plus" button will force a grasp movement until contact with an object or work piece is registered via the FSR sensors. An object must be in place before activating this function otherwise a collision between the fingers is possible in certain configurations.

Figure 3.2: Novint Falcon Grip (Handle) Buttons
3.4 Novint Falcon, GUI and Robotic Gripper

Workspace Matching

One of the challenges in this project was to match the workspace of the Novint Falcon with the workspace of the robotic gripper fingers. Specifically this means that as the Novint Falcon handle is moved \textit{in}, the gripper fingers should match this movement and simultaneously move \textit{in}. These movements should be accurately reflected in the GUI and on the robotic gripper. Real-time updates are a preference.

Novint Falcon Workspace

Access to the position of each respective axis on the Novint Falcon is handled by an array within the accompanying SDK. This makes it possible to find the actual range of movement of each axis (Table 3.1) and assign an arbitrary respective safe range of motion (ROM) (Table 3.2). This is done to emphasize the inclusion of haptic feedback to the user (when limits are reached) so that the haptic feedback is not mistaken for physical limits.

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-axis</td>
<td>2.10 (right)</td>
<td>-1.98 (left)</td>
<td>4.08</td>
</tr>
<tr>
<td>y-axis</td>
<td>2.20 (up)</td>
<td>-1.92 (down)</td>
<td>4.12</td>
</tr>
<tr>
<td>z-axis</td>
<td>1.60 (out)</td>
<td>-1.52 (in)</td>
<td>3.12</td>
</tr>
</tbody>
</table>

Table 3.1: Actual Limits and Range (inches) for Novint Falcon handle

<table>
<thead>
<tr>
<th></th>
<th>Max</th>
<th>Min</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>x-axis</td>
<td>2.00 (right)</td>
<td>-1.90 (left)</td>
<td>3.90</td>
</tr>
<tr>
<td>y-axis</td>
<td>1.80 (up)</td>
<td>-1.80 (down)</td>
<td>3.60</td>
</tr>
<tr>
<td>z-axis</td>
<td>1.40 (out)</td>
<td>-1.40 (in)</td>
<td>2.80</td>
</tr>
</tbody>
</table>

Table 3.2: Chosen Limits and Range (inches) for Novint Falcon handle

To achieve this, the handles range of motion is limited for the appropriate axis (3.1) and subsequently the position of the handle is calculated (3.2) as a resulting percentage of the allowed (safe) movement.

\[
\text{handle pos (\%)} = \frac{(\text{safe ROM} / 2) - \text{axis position(SDK)})}{\text{safe ROM}} 
\]  

(3.2)

((\text{min} < \text{axis position(SDK)}) \&\& (\text{axis position(SDK)} < \text{max}))  

(3.1)
GUI Workspace

The workspace of the GUI has been set up in an arbitrary way but remains compact and conveys all important information to the user. Using the "position of the handle as a percentage" it is possible to change the coordinate of each solid yellow circle so that it accurately represents the physical position of the Novint Falcon handle.

\[
\text{yellow circle coord} = \text{start coord} - \text{handle pos (\%)} \times \text{yellow circle ROM}
\]  

(3.3)

Robotic Gripper

The workspace of each robotic gripper axis is based on their physical limits and is measured in quadrature counts which are taken directly from the respective encoders. For safety (and to prevent unnecessary collisions), the total range of motion has been measured and an arbitrary safe range of motion (Table 3.3) has been chosen.

<table>
<thead>
<tr>
<th></th>
<th>Actual ROM</th>
<th>Safe ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (x-axis)</td>
<td>0 - 91678</td>
<td>0 - 70000</td>
</tr>
<tr>
<td>B (y-axis)</td>
<td>0 - 92762</td>
<td>0 - 70000</td>
</tr>
<tr>
<td>C (z-axis)</td>
<td>0 - 91374</td>
<td>0 - 70000</td>
</tr>
<tr>
<td>D (w-axis)</td>
<td>0 - 21742</td>
<td>0 - 20000</td>
</tr>
</tbody>
</table>

Table 3.3: Actual and Safe ROM (quadrature units)

The "position of the handle as a percentage" value is used to determine the corresponding position of the respective robotic gripper axis. This value is then sent to the Galil motion controller which in turn moves the respective axis to a corresponding position that is matched with the Novint Falcon handle.

\[
\text{move axis to (abs pos)} = \text{handle pos (\%)} \times \text{safe ROM}
\]  

(3.4)

3.5 Initialization

Once the system is started an initialization of the Novint Falcon and the robotic gripper is begun. The Novint Falcon has its own initialization functions which have been supplied by the SDK. The main functions checked are the servos for each motor and encoder pair (3 in total) and that the device is "homed" If x, y and z-axis are fully open and the w-axis is at 90°
the device is not "homed" then the LED lights at its center will be red and a dialogue appears, asking the user to move the handle in and out until the LED lights turn blue. This takes no more than a few seconds and clicking on OK will allow the remaining functions to be properly initialized.

The first step needed when using the robotic gripper is to connect to the Galil motion controller by entering the corresponding IP address of the unit. This is a simple command in C++ which is taken from the Galil header file. The robotic gripper does not have an inbuilt initialization function so one was created and included in the main DMC program code. This starts all motors on the robotic gripper and moves each finger outwards while the w-axis is moved into a $90^\circ$ position. Once the respective inductive proximity limit switches are triggered, each encoder is reset to zero and the main part of the code begins after a 500 ms delay. This delay is used to ensure all communications and axis movements have been completed and once finished, the gripper fingers will match their workspace with the workspace of the Novint Falcon.

This type of initialization and calculation of the respective workspaces allows for a fluid start and the user can immediately begin using the devices as intended.

### 3.6 Haptic Forces

Haptic force feedback is used to give the user an additional level of interaction and in this thesis haptics are used to inform the user:

- When a limit is reached.
- When a part has been grasped.

The haptic forces used can be customized to meet a users preference and are built upon a simple "Spring Mass Damper" Model as shown in Figure (3.3). For the haptic forces used in this thesis, a simple call to the Novint Falcon SDK was used as complicated haptic forces are not required for the desired effect.
3.6. HAPTIC FORCES

Figure 3.3: Spring Mass Damper Model
Chapter 4
Experiments

4.1 Overview

Numerous experiments were completed before the final integration of the Novint Falcon and the robotic gripper could be completed. These have been highlighted in the text that follows.

4.2 Novint Falcon

There are two methods to program the Novint Falcon, Squirrel scripting (beginner) and the SDK (advanced). The following experiments were realized during this phase.

Squirrel

Squirrel scripting is an excellent way for any beginner to become familiar with the Novint Falcon and its haptic capabilities. Novint uses Squirrel as the base programming language for its F-Gen interface which allows gamers to customize their Novint Falcon according to which game they are playing and what level of haptic feedback they wish to experience. Many functions are available to the user in addition to a large amount of control to how haptic forces are generated. The main disadvantage with Squirrel is that many functions that are available in C++ cannot be realized when purely using Squirrel.

- Create keyboard integration.
  - Move the Novint Falcon handle left, right, up, down, in and out using keyboard keys.
- Create any haptic force.
• Map buttons to specific functions, ie, pressing the "Plus" button outputs the handle position in (x,y,z).

• Use different button states for different functions.
  – States include, "button-was-just-pressed", "button-is-held-down", "button-was-just-released".

• Create haptic forces and map them to different keyboard and / or button states.
  – Single haptic feedback.
  – Constant haptic feedback.
  – Variable strength of haptic feedback.

• Integrate key presses with moving the solid yellow circles into various positions which simulate robotic gripper axes movement.

• Experiment with various timings (ms) when the haptic forces are executed (Figure 4.1).
  – A - delay between input and force action
  – B - ramp up force
  – C - sustain force
  – D - ramp down force

Figure 4.1: How Timings are Handled with F-Gen
4.2. NOVINT FALCON

GUI

The goal with a GUI is to create an area that conveys all important information to the user which can include important messages, different modes of operation, the current mode of operation and most importantly, a visual representation of the target device, ie, the robotic gripper. A simple GUI has been created using OpenGL and OpenGLUT which visually brings together the integration of the Novint Falcon and the robotic gripper along with the following advantages:

- Relay information to the user, ie, which mode is active.
- Telerobotics possibility.
- Simplicity when future work is implemented.

In order to create the GUI, a number of experiments were realized.

- Decide on an appropriate layout.
- Create all items and text of the GUI.
- Organize information in an easy to understand way.
- Connect movable GUI items with their appropriate keyboard press to simulate gripper movement.
- Integrate the Novint Falcon workspace with the GUI workspace, ie, moving handle in moves the solid yellow circles in at the same time.

SDK (HDAL)

The Novint Falcon SDK (HDAL) is an integral part of this thesis and requires a solid understanding of C++ to be fully understood. Implementing the SDK, adding haptic forces and integrating the GUI required a large part of the available thesis time frame. The SDK offers many advantages over Squirrel, one being the ability to communicate with the Galil motion controller from within the main program. This would not have been possible when purely using Squirrel. The gains made in flexibility come at the cost of a required higher level of C++ understanding. The experiments completed during this phase include:

- Get x, y and z-axis position.
- Integrate z-axis movement with the GUI.
- Add haptic feedback when limits are reached.
- Integrate x-axis movement with the w-axis GUI movement.
- Add button functionality for the four modes of operation.
  - Novint Falcon / ABB
  - All Fingers / w-axis
4.3 Robotic Gripper

Galil Motion Controller

Using the Galil motion controller is simplified by the free software download of "Galil Tools Lite" which is a recommended starting point when experimenting with sending commands to the robotic gripper. Numerous experiments were carried out when using the Galil motion controller as some time was needed to become familiar with the programming language.

- Load the Watch window and monitor the state of each sensor found on the robotic gripper.
- Send single commands to the robotic gripper from the terminal found in Galil Tools Lite.
- A short program was made in the program editor which opened and closed the gripper fingers on the x, y and z-axis.
- A Home Position routine was created which is used during initialization.
  - Can be started by pressing the "Lightning" button on the Novint Falcon handle.
  - The x, y and z-axis are fully opened until the inductive proximity switches are activated.
  - The w-axis is placed at 90°.
- The Grasp program is activated by pressing the "Plus" button on the Novint Falcon handle.
  - The x, y and z-axis move forward (close) until a work piece is detected by the FSR sensors.
  - Once a work piece is detected, the motors close 1000 quadrature units to increase force on the work piece to ensure a firm grasp.
- Send simple commands from within C++.
- Download a program to the Galil motion controller from within C++ and execute.
- Port this method into the existing Novint Falcon / GUI C++ code and execute.
- Connect the control of z-axis (Novint Falcon) with the control of the gripper fingers.
- Connect the control of the x-axis (Novint Falcon) with the control of the w-axis.
• Test various speeds to reduce any delay in workspace matching.

• Create a exiting program which turns off all motors when the main program is quit. This is to prevent overheating of the motors.

Force Sensitive Resistor (FSR) Sensors

The FSR sensors (one on each gripper finger) are a simple low-cost solution for detecting when a work piece has been grasped (logical "0") by the gripper fingers. Calling the FSR sensors in the Galil Tools Lite watcher window allowed some basic testing to be done when grasping different sized work pieces. Although "contact" can be detected by the FSR sensors, the pressure required to register a logical "0" is quite high. A common example of the required force can be compared to that of opening the tab on a aluminum pop can. Both the y-axis and z-axis on the robotic gripper have a large amount of slippage which makes testing of a successful grasp not possible. This minor part of the thesis was unfortunately not completed.

Motors

There are four identical motors used on the robotic gripper which generate movement for the x, y, z and w-axis. After one day of testing it was apparent that the y-axis and z-axis motors had different backlashes and speeds. This made it very difficult to match the workspace as the movements were sporadic and inconsistent. In addition, there is a large amount of mechanical free-play in the motors when the power is off. The x-axis and w-axis by comparison have minimal to zero mechanical free-play and their movement is consistent. Therefore focus was shifted to the x-axis and w-axis for all further testing. Since all gripper fingers get the same position, working with only one finger is acceptable. A common test movement involved opening and closing the gripper fingers repeatedly which resulted in the motors becoming extremely hot and as such, testing in periods of 10 minutes is recommended to prevent over heating or other damage. The w-axis becomes the hottest of all the motors and extra care should be observed here.

4.4 Summary

The experiments carried out have successfully tested the software and hardware used in this thesis. When testing the final unit, robust and crash free operation was observed. Control of the robotic gripper by the Novint Falcon occurs in real time in the GUI while a minimal delay in finger movement can be observed when large sporadic movements are made by the user. Increasing the speed of each motor on the robotic gripper helped to reduce this delay but some delay
is inevitable when sporadic movements are made. If normal, fluid movements are used, the delay in finger movement is negligible. In short, I am very pleased with these results.

Software

The Squirrel programming language is a recommended starting point for anyone wanting to explore the Novint Falcon but if the need to create a GUI or connect to other devices arises, then the Novint Falcon SDK will need to be used.

Although all aspects of the SDK have been explored, more focus could have been spent on the haptic aspect as there are many possibilities here. For example, instead of creating haptic feedback when limits are reached, it is possible to create an invisible wall which simulates an actual physical limit.

The GUI created serves its purpose in relaying important information to the user although visually, operation of the w-axis could have been handled differently. It was originally planned to transfer the solid yellow circles to the w-axis white lines, thereby creating a complete fluid GUI for the user when controlling the w-axis on the robotic gripper.

Various sounds could have been included, when limits are reached for example, to add a fourth sense to the Novint Falcon experience.

Hardware

The Novint Falcon and galil motion controller are purchased items which functioned without fault for the duration of this thesis and as such, no suggestions for improvement can be offered. The Robotic Gripper (v2.0) has been built by the AASS Intelligent Control Lab and is an enjoyable experience, especially considering that this is not a professionally mass produced product. A wish list of improvements has been included.

- The motors used are discontinued models with some requiring replacement or maintenance. New fully functioning models will perhaps eliminate overheating.

- Longer connection cables for the motors will allow for improved cable management and a 100% secure fit.

- The mechanical design for finger movement is advanced and although each finger axis is identical, the operation is not. There is a large amount of mechanical free-play in the y-axis (B) and z-axis (C) while the x-axis (A) and w-axis (D) is very stable.
• Fine tuning of the limit switch placement is needed to ensure consistent results.

• The inclusion of Matrix Touch Sensors on each finger will allow for multiple touch points to be registered and have a lower force requirement for activation.
Chapter 5
Conclusion

5.1 Summary

This thesis has set out to create an integration between the Novint Falcon and the AASS Intelligent Control Lab robotic gripper which is controlled by a Galil motion controller.

A successful implementation has been executed resulting in a near real time workspace matching of the Novint Falcon, GUI and the robotic gripper. Upon startup, initialization of both devices occurs and the workspaces of the Novint Falcon and robotic gripper are automatically matched, allowing the user to fully control each finger axis on the robotic gripper (default mode) by using the handle on the Novint Falcon. Pressing the Triangle button will allow control of all fingers and the w-axis simultaneously. A GUI provides the user with any important messages, the available modes of operation, which mode is currently active, how to activate each mode and a overhead visual representation of the robotic gripper axes.

During testing of the final code, the system remained stable and there were no crashes. The same cannot be said for the hardware of the robotic gripper which proved to be frustrating when testing all axes. As a result, testing concentrated on the x-axis and w-axis which proved to be adequate for achieving the goals of this thesis. Given some time and adequate funding, the provided short list of improvements can be made, completing a prospective telerobotics systems for development of the suggested future works.

\[1\] It is important to note that the y-axis and z-axis are not being used due to mechanical faults.
5.2 Future Work

As with any project, there is always something the researcher wishes to add, build upon or improve for future iterations and this project is no exception. A collection of future work for the hardware and software components has been summarized below.

Software

- Add an audio chime when a limit is reached.
- Use haptics to simulate the weight of a grasped work piece in the Novint Falcon handle. This can be achieved by using the web cam on the robotic gripper in conjunction with the matrix touch sensors (mentioned in hardware) to identify the grasped object. Once identified, the appropriate weight (including gravity) can be simulated on the handle by using haptics.
- Create a haptic wall which will represent the physical limits of the robotic gripper. At the moment haptic feedback in the form of a vibration is used. It is possible to remove this vibration as in the basic cube example included in the SDK.
- Currently, there is a 2 second delay between the actual handle position and current position of the gripper fingers. Real-time updating is desired.
- Improve the current GUI so that the solid yellow circles move onto the w-axis white lines when alternate finger position mode is enabled.
- Add the code needed to control the ABB industrial robot via the Novint Falcon. An appropriate change to the GUI will be needed to ensure a fluid experience for the user. The ABB industrial robot has 6-axis, therefore controlling 3-axis at one time with the Novint Falcon is recommended. This can be achieved by using the handle buttons to switch between different modes.

Hardware

Other than a single addition below, any future work (improvements) for the robotic gripper have been discussed in section 4.4.

- It was originally planned to install Matrix Touch Sensors \[1\] onto each gripper finger but these were replaced by FSR sensors. Using the matrix

\[1\]www.weiss-robotics.de/
sensors can have some advantages if automatic work piece identification is to be used with the web cam. The matrix sensors can detect multiple contact points from the work piece and theoretically, this can be used to assist in work piece identification when coupled with installed the web cam. A further advantage is the detection of slippage when more contact points are registered on each finger. This would be especially useful when moving large, heavy, dangerous materials. Any slippage detected could be conveyed to the user through haptic feedback.
Appendix A
Novint Falcon

Technical Specifications Summary

- 3D Touch Workspace: 4" x 4" x 4" (4" = 10.2 cm)
- Force Capabilities: > 2 lbs (0.91 kg)
- Position Resolution: > 400 dpi
- Communication Interface: USB 2.0
- Size: 9" x 9" x 9" (9" = 22.86 cm)
- Weight: 6 lbs (2.7 kg)
- Power: 30 watts, 100 V - 240 V, 50 Hz - 60 Hz

Recommended System Requirements

- Processor: 2.4 GHz Processor
- OS: Windows XP Service Pack 2, Windows Vista
- Graphic card: 256Mb 3D hardware accelerated graphics card
- DirectX Version: DirectX 9.0c
- Hard Drive: 1.5 GB free disk space
- Memory: 1 GB RAM
- Broadband Internet Connection

http://www.novint.com
Appendix B
General Novint Falcon
Software Setup

To begin using the Novint Falcon, the included tutorial and bundled games, it is recommended that you check the Novint website for the latest software and drivers before following the instructions below.

- Novint Website
- Novint Community
- Download and install Novint Falcon driver (v4.0.28)
- Download and install F-Gen V1
- Ensure that NDSSetter (Novint Device Support Setter) is set as: C:/Program Files/Novint/Falcon/HDAL (Figure B.1).

*** NOTE *** Novint Falcon drivers MUST be installed BEFORE F-Gen.

Once completed, you will be able to run the Novint FALCON DEMO (highly recommended) and play the included "Limited Edition Novint Falcon Bundle" through the NVeNT interface.

---

1 http://www.novint.com
2 http://www.falconarmy.com
3 https://backup.filesanywhere.com/fs/v.aspx?v=896a6886606576b0a368
4 https://backup.filesanywhere.com/fs/v.aspx?v=896a68865a6373afa59f
If using the Novint Falcon in the AASS Lab, please use the following login details:

Login: aassLAB
Password: aassorebro
ScreenName: funLAB

In addition to the vast support for games, the Novint Falcon is also supported by two different development platforms, Squirrel Scripting and the Novint Falcon SDK (HDAL). These platforms are the main focus of this thesis.
Appendix C
General Squirrel Software Setup

With the Novint Falcon drivers and F-Gen already installed there are only two components needed in order to begin experimenting with the Novint Falcon and its haptic capabilities; Squirrel and an IDE.

If you don’t already have a preferred IDE, "Geany" is recommended and can be found in the Ubuntu Software Center.

- Download the latest version of Squirrel\[^1\] and install.
- It is recommended you read Part 1, 2, 3, 4, 10 and 12 of the Basic Novint Falcon\[^2\] using Squirrel Scripting to get started.

---

\[^1\] http://www.squirrel-lang.org
\[^2\] http://www.opusgames.com/games/novint/FGen/tutorial01.html
Appendix D
Novint Falcon SDK Setup

Follow these steps to install the Novint Falcon SDK (HDAL).

- Ensure you have installed the drivers as discussed in section B (pg 61).
- Connect the power and USB cables.
- Download the Novint Falcon SDK. (HDAL) and install.
- Set the path for the NDSSetter (Novint Device Support Setter) as: C:/Program Files/Novint/HDAL_SDK_2.1.3/config (Figure D.1).

GLOBAL SEARCH SETUP

This procedure tells VS 2005 where to find additional directories which have needed include and library files. These settings apply to ALL development projects.

- Select "Tools" then "Options".

1 https://backup.filesanywhere.com/fs/v.aspx?v=896d658b5c6272b2a1aa
APPENDIX D. NOVINT FALCON SDK SETUP

- In the new window, on the left hand side, select "Projects and Solutions", "VC++ Directories".

- On the right hand side under "Show Directories For" choose "Include Files" from the drop down menu.

- Double click on an empty slot then use the "...." icon to browse to the "C:/Program Files/Novint/HDAL_SDK_2.1.3/include" directory (Figure D.2).

- In the same window, select "Library Files" in the drop down menu and repeat the process, this time navigating to the "C:/Program Files/Novint/HDAL_SDK_2.1.3/lib" folder (Figure D.3).

- Click "OK"

![Figure D.2: "Include" Path](image1)

![Figure D.3: "Lib" Path](image2)

PROJECT SEARCH SETUP

This procedure tells VS 2005 where to find the necessary files in the Novint Falcon SDK which must be setup for each project using the Novint Falcon SDK (HDAL).
• Select "Project" then "Properties".

• In the new window, on the left hand side, select "Configuration Properties", "C/C++", "General".

• On the right hand side, choose "Additional Include Directories" and add "$(NOVINT_DEVICE_SUPPORT)/include" (Figure D.4).

• In the same window on the left hand side, select "Configuration Properties", "Linker", "General".

• On the right hand side, choose "Additional Library Directories" and add "$(NOVINT_DEVICE_SUPPORT)/lib" (Figure D.5).

• Click "OK"

![Figure D.4: "Include" Path](image1)

![Figure D.5: "Lib" Path](image2)

**Library File**

• Navigate to "Project", "Properties", "Configuration Properties", "Linker", "Input" and add the "hdl.lib" file (copy it from the SDK location to your project "/lib" folder) to the "Additional Dependencies" window. (Figure D.6).
Figure D.6: Library File "hdl.lib"
Appendix E
OpenGLUT

At this point, it is possible to run the included SDK "Basic" example which is described in the next section (pg 73), however, a successful interaction with the Novint Falcon requires some form of visual representation and to achieve this, OpenGLUT was used to draw the graphical interface for this project. Installing OpenGLUT is a relatively simple process.

- Download and unpack [OpenGLUT](http://www.sourceforge.net/projects/openglut/files/).
- Copy the *.h files to the VS 2005 compiler OpenGL "include" directory "C:/Program Files/Microsoft Visual Studio 8/VC/include/GL".
  1. openglut.h
  2. openglut_exp.h
  3. openglut_ext.h
  4. openglut_std.h
- Copy the *.lib files to the VS 2005 compiler "library" directory "C:/Program Files/Microsoft Visual Studio 8/VC/lib".
  1. OpenGLUT.lib
  2. OpenGLUT_static.lib
- The #include used in the code is — #include <GL/openglut.h>

Appendix F  
Galil Tools Lite

Follow these steps to install Galil Tools Lite and to begin sending commands to the robotic gripper.

• Download and install the latest version of Galil Tools Lite from their website[1].

• Connect an ethernet cable between the DMC-2143 controller and a Desktop Ethernet Switch.

• Connect another ethernet cable between the Desktop Ethernet Switch and your computer.

• Open Galil Tools Lite and you will notice the "OFFLINE" notice.

• Select "Controller" then "Connect...".

• The IP Address of the controller will be available from the options, choose it.

• Watch the "Terminal" window for a status update as in Figure F.1.

• Open the "Watch" window (Tools, Watch) and call up any sensors or motors you want to see states of.

• Become familiar with the available commands in the controller user manual[2] and begin sending commands using the "Terminal" window.

Figure F.1: Confirmation of IP Connection to Galil motion controller
Appendix G
Galil Commands in C++

The main purpose of this thesis is to control the robotic gripper using the Novint Falcon and as such, any Galil commands used, must be sent from the program in C++. Follow these steps to begin sending Galil commands from within C++.

- Download the appropriate version of the Galil Libraries:

- Copy the "Galil.h", "Galil1.dll" and "Galil1.lib" files into your main code directory.

- In VS 2005 set your Project Preferences by choosing "Project", "Properties" and then on the left of the new window, "Configuration Properties", "Linker", "General".

- Add the directory where you have the "Galil1.lib" to the "Additional Library Directories" field.

- In the same window, add the filename of the library you just added under "Configuration Properites", "Linker", "Input".

- Add "Galil1.lib" to the "Additional Dependencies" input field.

- You should notice that "hdl.lib" is in this field as well.

- Now the Global Preferences must be set.

- Navigate to "Tools", "Properites", "Projects and Solutions", "VC++ Directories".

- In the top right of this window, "Show the Directories for:" the "Include Files" and add the directory where the "Galil.h" and "Galil1.dll" files are.

1 http://www.galilmc.com/support/downloads/software/galiltools/windows/
• Repeat the previous step for the "Library Files".
• In your C++ code, add "#include "galil.h"."

In order to connect with the Galil motion controller, add "Galil g("IP Address Goes Here");" before your Main(). To send commands to the robotic gripper, use the notation "g.command("Any Galil Command");".
Appendix H
Source Code - Squirrel

1 // F-GEN Squirrel Script
2 // This script demonstrates various types of haptic movement
3 // that the falcon is capable of. Forces include 'none',
4 // 'simple' and 'recoil'. Movements include 'left', 'right',
5 // 'up', 'down', 'forward' and 'backward'. Positions of each
6 // axis are retrievable with the appropriate key press.
7 //
8 // F-Gen is meant purely to get the user familiarized with
9 // what type of haptics the falcon is capable of. There is not
10 // much freedom or room for creativity and once you have a
11 // feel for what is possible, it is time to move onto the SDK.
12 //
13 // creating default Falcon effect stack
14 gEffectsStack <- effectstack("effects", 2.0);
15 // registering envelope effect.
16 gEnvelopeEffectID <- registereffect("NoiseEnvelope8X");
17
18 // variable to store which type of recoil effect to use
19 gEffectType <- 1; // 0 = no effect
20 // 1 = simple recoil
21 // 2 = looping recoil
22 // init to 1
23
24 // setup event timing
25 gScriptTime <- timekeeper(); // setup time checker
26 gLoopRecoilTimer <- 0.3; // timer to use for looping a
27 // recoil effect (initially set to zero)
28
29 // create a simple recoil effect and set its parameters.
30 gSimpleRecoil <- effectparameters(gEnvelopeEffectID, gEffectsStack);
31 // create the effect
32 gSimpleRecoil.setvarelement("force", 0, 0); // 0 newtons rightwards
33 gSimpleRecoil.setvarelement("force", 1, 0); // 0 newtons upwards

75
// 20 newtons backwards
35 gSimpleRecoil.setvarelement("force", 2, 20); // ramp up to the force
36 gSimpleRecoil.setvar("force", 20); // hold time once at maximum force
37 gSimpleRecoil.setvar("force", 0); // fall off time at end of effect

// create a separate effects stack for 'movement' forces
40 gMovementStack <- effectstack("movement", 1.0);

// create a constant effect type
42 gConstantEffectID <- registereffect("Constant");

// create the parameters for a constant effect for use as a movement force
45 gMovementEffectParameters <- effectparameters(gConstantEffectID, gMovementStack);
46 gMovementForce <- null;

// setup keyboard settings
48 gSendE <- inputsender(KEY_E); // setup keyboard settings
49 gSendX <- inputsender(KEY_X);
50 gSendI <- inputsender(KEY_I);
51 gSendT <- inputsender(KEY_T);

// activate or deactivate the effects stacks
// unlocking the device list on stacks
59 gEffectsStack.setdevicelock(false);
60 gControlBoxStack.setdevicelock(false);

// add the device to the stacks, input slot 0
64 if (bConnect == true) {
65 gEffectsStack.adddevice(deviceHandle, 0);
66 gMovementStack.adddevice(deviceHandle, 0);
67 }
68 else {
69 // remove the device from the stacks, input slot 0
70 gEffectsStack.removedevice(deviceHandle, 0);
71 gMovementStack.removedevice(deviceHandle, 0);
72 }

// lock the device list on the stacks
75 gEffectsStack.setdevicelock(true);
76 gMovementStack.setdevicelock(true);

if (bConnect == true) {
// connect the stacks to the device for output
79 deviceconnectstack(deviceHandle, gEffectsStack);
80 deviceconnectstack(deviceHandle, gMovementStack);
81 }
else {
    // disconnect the stacks from the device
    devicedisconnectstack(deviceHandle, gEffectsStack);
    devicedisconnectstack(deviceHandle, gMovementStack);
}

// Initialize this script
// Called once when this script is first loaded
function HapticsInitialize(registryConfigHandle) {
    print("Initialize Script\n");
}

// Activate this script
// Called whenever this script becomes active (such as after
// returning to the application this script is attached to
// after switching to another window)
function HapticsActivated(deviceHandle) {
    // activate the effects stacks
    ConnectOrDisconnectStacks(deviceHandle, true);
    // launch the movement force
    gMovementForce = constantforce(gMovementEffectParameters, gMovementStack)
    print("Script Activated\n");
    print(" Press 'c' at any time for a list of commands\n");
    print(" Press 'x' at any time to EXIT\n");
    print(" Press '1' = no recoil\n");
    print(" Press '2' = simple recoil\n");
    print(" Press '3' = looping recoil\n");
    print(" Press 'w' = creates a force in front of you\n");
    print(" Press 's' = creates a force behind you\n");
    print(" Press 'a' = creates a force from your left\n");
    print(" Press 'd' = creates a force from your right\n");
    print(" Press 'q' = creates a force from above\n");
    print(" Press 'e' = creates a force from below\n");
    print(" Press grip 'TRIANGLE' = x-axis position\n");
    print(" Press grip 'PLUS' = y-axis position\n");
    print(" Press grip 'LIGHTNING' = Z-axis position\n");
}

// Run this script
// Called repeatedly while this script is active

function HapticsThink (deviceHandle) {
  gScriptTime.update();  // update the time checker
  // grab the time (in seconds) since the last call to HapticsThink
  local elapsedTime = gScriptTime.elapsedseconds();

  local falcon_X = deviceaxis(deviceHandle, 0);  // 0 = X axis
    (left/right)
  local falcon_Y = deviceaxis(deviceHandle, 1);  // 1 = Y axis
    (up/down)
  local falcon_Z = deviceaxis(deviceHandle, 2);  // 2 = Z axis
    (in/out)

  if (devicewasbuttonjustpressed(deviceHandle, FALCON_LOGO)) {
    print("LOGO button was pressed, 1 = no force\n");
    print("2 = simple force\n");
    print("3 = recoil force\n");
    print("DEFAULT = 2\n");
    if (gEffectType == 1) { // firing a single recoil
      gSimpleRecoil.fire();  // launch the simple recoil effect
    }
  }

  else if (devicebuttonisdown(deviceHandle, FALCON_LOGO)) {
    if (gEffectType == 0) {
      print("LOGO button is being held down but NO FORCE is chosen so you feel nothing\n");
    }
    else if (gEffectType == 1) {
      print("LOGO button is being held down and a SIMPLE FORCE should be felt\n");
    }
    else if (gEffectType == 2) { // firing a looping recoil
      print("LOGO button is being held down and a Recoil FORCE should be felt\n");
      gLoopRecoilTimer = elapsedTime;  // subtract elapsed time from the timer
      if (gLoopRecoilTimer <= 0.2) { // time to fire off another recoil
        gSimpleRecoil.fire();
      }
  }
// reset timer... fire off a recoil every tenth of a second

gLoopRecoilTimer += 0.1;  // increase timer by a
tenth of a second

else if (devicewasbuttonjustpressed(deviceHandle, FALCON_TRIANGLE)) {
    print("TRIANGLE button was pressed = x–axis pos: " + falcon_X + 
    "\n\n");
}

else if (devicewasbuttonjustpressed(deviceHandle, FALCON_PLUS)) {
    print("PLUS button is pressed = y–axis pos: " + falcon_Y + 
    "\n\n");
}

else if (devicewasbuttonjustpressed(deviceHandle, FALCON_LIGHTNING )) {
    print("LIGHTNING BOLT button is pressed = z–axis pos: " + 
    falcon_Z + "\n\n");
}

// update the keyboard listener before checking keyboard values

gKeyInput <- inputlistener();  // keyboard listener

if (gKeyInput.wasinputjustpressed(KEY_C)) {
    print("Script Activated\n");
    print("Press 'c' at any time for a list of commands\n");
    print("Press 'x' at any time to EXIT\n");
    print("Press '1' = no recoil\n");
    print("Press '2' = simple recoil\n");
    print("Press '3' = looping recoil\n");
    print("Press 'w' = creates a force in front of you\n");
    print("Press 's' = creates a force behind you\n");
    print("Press 'a' = creates a force from your left\n");
    print("Press 'd' = creates a force from your right\n");
    print("Press 'q' = creates a force from above\n");
    print("Press 'e' = creates a force from below\n");
    print("Press grip 'TRIANGLE' = x–axis position\n");
    print("Press grip 'PLUS' = y–axis position\n");
    print("Press grip 'LIGHTNING' = Z–axis position\n\n");
}

else if (gKeyInput.wasinputjustpressed(KEY_1)) {
    print("Number '1' was pressed = 'no recoil' selected\n");
}
APPENDIX H. SOURCE CODE - SQUIRREL

```cpp
230     gEffectType = 0;  // set effect type to single recoil
231 }
232
233 } else if (gKeyInput.wasInputJustPressed(KEY_2)) {
234     print("Number '2' was pressed = 'simple recoil' selected\n\n")
235     gEffectType = 1;  // set effect type to no recoil
236 }
237
238 } else if (gKeyInput.wasInputJustPressed(KEY_3)) {
239     print("Number '3' was pressed = 'looping recoil' selected\n"");
240     gEffectType = 2;  // set effect type to looping recoil
241 }
242
243 } else if (gKeyInput.wasInputJustPressed(KEY_X)) {
244     print("\n\n\n\n\n−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−
\n")
245     print(" press ENTER twice to EXIT \n")
246     print("−−−−−−−−−−−−−−−−−−−−−−−−−−−−
\n")
247     exitNUT();
248 }
249
250 } else if (gKeyInput.wasInputJustPressed(KEY_W)) {
251     print("Key 'W' was pressed = 'forward force' selected\n\n")
252     gMovementForce.setForce(0, 0, -6);
253 }
254
255 } else if (gKeyInput.wasInputJustPressed(KEY_S)) {
256     print("Key 'S' was pressed = 'backward force' selected\n\n")
257     gMovementForce.setForce(0, 0, 6);
258 }
259
260 } else if (gKeyInput.wasInputJustPressed(KEY_A)) {
261     print("Key 'A' was pressed = 'left force' selected\n\n")
262     gMovementForce.setForce(-4, 0, 0);
263 }
264
265 } else if (gKeyInput.wasInputJustPressed(KEY_D)) {
266     print("Key 'D' was pressed = 'right force' selected\n\n")
267     gMovementForce.setForce(4, 0, 0);
268 }
269
270 } else if (gKeyInput.wasInputJustPressed(KEY_Q)) {
271     print("Key 'Q' was pressed = 'up force' selected\n\n")
272     gMovementForce.setForce(0, 5, 0);
273 }
274
275 } else if (gKeyInput.wasInputJustPressed(KEY_E)) {
276     print("Key 'E' was pressed = 'down force' selected\n\n")
277     gMovementForce.setForce(0, -2, 0);
278 }
279 else {
280     gMovementForce.setForce(0, 0, 0);
281 }
282 ```
function exitNUT () {
    gSendE.press();
gSendX.press();
gSendI.press();
gSendT.press();
gSendE.release();
gSendX.release();
gSendI.release();
gSendT.release();
}

// Deactivate this script
// called whenever this script stops being active (such as if
// you switch to another window)
function HapticsDeactivated (deviceHandle) {
    ConnectOrDisconnectStacks(deviceHandle, false); // deactivate
    the effects stacks
    gMovementForce.dispose(); // remove the
    movement force
    gMovementForce = null;
    print("Script Deactivated\n");
}

// Shutdown this script
// called when this script is cancelled and unloaded
function HapticsShutdown () {
    print("Shutdown Script\n");
}
Appendix I
Source Code - Falcon Control

// ***************************************************
// ***************************************************
// *** Haptic Control with a Robotic Gripper ***
// ***************************************************
// *** Morgan Rody ***
// ***************************************************
// *** University of Orebro, Sweden ***
// ***************************************************
// *** Completed June 2011 ***
// ***************************************************
// ***************************************************

#include "galil.h"
#include <stdlib.h>
#include <math.h>
#include <windows.h>
#include <GL/openglut.h>
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include "haptics.h"
#include <string.h>

// creates a GUI of the robotic gripper
// instructions are listed
// the user can control the robotic gripper using
// the Novint Falcon
// different modes are available for control
// ***************************************************

#include <iostream>
#include <windows.h>
#include <GL/openglut.h>
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <sstream>
#include <cstdlib>
#include "haptics.h"
#include <string.h>

using namespace std;
```c
38 int w, h;              // width and height of main window
39 int posGrip = 0;       // position of the two moveable gripper fingers
40 // xyz position values of each sphere
41 GLfloat x1, y1, z1, x2, y2, z2, x3, y3, z3;
42 double posXYZ[3];      // array where all pos values are stored
43 int modeA = 0;         // modeA = 0 move all three grippers in/out
44 // modeA = 1 move two bottom grippers to alternate position
45 int modeB = 0;         // modeB = 0 control falcon
46 // modeB = 1 control abb robot
47
48 // spring stiffness
49 const double gStiffness = 100.0;
50
51 // The haptics object, with which we must interact
52 HapticsClass gHaptics;
53 Galil g("192.168.0.102"); // initializes contact with the Galil Controller
54
55 void initScene();
56 void glutReshape(int w, int h);
57 void renderBitmapString(float x, float y, float z, void *font, char *string);
58 void restorePerspectiveProjection();
59 void setOrthographicProjection();
60 void drawGraphics();
61 void drawLabels();
62 void glutDisplay();
63 void posFalcon();
64 void home();
65 void grasp();
66 bool wait(int seconds);
67 void glutIdle(void);
68 void glutKeyboard(unsigned char key, int x, int y);
69 void exitHandler(void);
70
71 int main(int argc, char **argv) {
72
73 // initialize OpenGL, GLUT, glutKeyboard and create Window
74 glutInit(&argc, argv);
75 glutInitDisplayMode(GLUT_DEPTH | GLUT_DOUBLE | GLUT_RGBA);
76 glutInitWindowSize(800, 650);
77 glutCreateWindow("GUI");
78 glutKeyboardFunc(glutKeyboard);
79 glutSetCursor(GLUT_CURSOR_NONE); // hides the cursor
80
81 // register callbacks
82 glutDisplayFunc(glutDisplay);
83 glutReshapeFunc(glutReshape);
84 glutIdleFunc(glutIdle);
85```
// sets up the exit properly
atexit(exitHandler);

// download the galil program to the controller ONCE
programDownloadFile("galilCODE/mainPROG.dmc");

// execute the program ONCE
g.command("XQ");

// initialize opengl, draw spheres
initScene();

// enter GLUT event processing cycle

// initialize the Sphere variables
void initScene() {

// Call the haptics initialization function
gHaptics.init(gStiffness);

// Some time is required between init() and checking status,
// for the device to initialize and stabilize. In a complex
// application, this time can be consumed in the initGL()
// function. Here, it is simulated with Sleep().
Sleep(100);

// Tell the user what to do if the device is not calibrated
if (!gHaptics.isDeviceCalibrated())
    MessageBoxA(NULL,
    "The next two lines are one long string" "Please home the device by extending\n" "then pushing the arms all the way in.", "Not Homed",
    MB_OK);

// initial position of the spheres
x1 = 0.0f;     // top sphere
y1 = 1.0f;
z1 = -5.0f;
x2 = -1.0f;    // left sphere
y2 = -0.5f;
z2 = -5.0f;
x3 = 1.0f;     // right sphere
y3 = -0.5f;
z3 = -5.0f;
}

// creates three solid circles and text
void glutDisplay() {

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // clear screen

// must sync
gHaptics.synchFromServo();
drawGraphics();
drawLabels();
posFalcon();  // must be placed here to get real time updates!!!
glutSwapBuffers();
}

void posFalcon() {

double per;  // percentage of change when moving the falcon in/out along z-axis
double diff = 0.7;  // difference between sphere positions (initial open and fully closed)
double diffAlt = 0.5;
double realTIME;  // used for z axis movement
string moveZX;  // moves the z-axis on the falcon but the x-axis on the gripper

gHaptics.synchFromServo();

// gets the position of each axis and puts into array posXYZ
ghaptics.getPosition(posXYZ);

// prints out the position of each axis to the screen in real time
char bufferX[10]={'\0'};
printf_s(bufferX, "%f", posXYZ[0]);
renderBitmapString(-1.25f, -0.2f, -5.0f, GLUT_BITMAP_HELVETICA_18, bufferX);

char bufferY[10]={'\0'};
printf_s(bufferY, "%f", posXYZ[1]);
renderBitmapString(0.75f, -0.2f, -5.0f, GLUT_BITMAP_HELVETICA_18, bufferY);

char bufferZ[10]={'\0'};
printf_s(bufferZ, "%f", posXYZ[2]);
renderBitmapString(-0.25f, 1.3f, -5.0f, GLUT_BITMAP_HELVETICA_18, bufferZ);

// limits travel of the falcon to safe boundaries
// draws yellow spheres to match the movement of the falcon
if ((-1.40 < posXYZ[2]) && (posXYZ[2] < 1.40)) {
    per = (1.4 - posXYZ[2]) / 2.8;  // 2.8 is full distance of travel for falcon travel
    yy1 = 1.0 - per * diff;  // 1.0 is the initial position value, var 'yy1' is taken for some reason
x2 = -1.0 - per * -diff;  // left sphere
x3 = 1.0 - per * diff;   // right sphere

g.command("mode=0");    // sets mode in the galil program
realTIME = per*70000;    // max allowed travel for
    // finger is 70000 units

std::ostringstream s;    // takes a double and puts it
    // into a string
s << "dist=" << realTIME;
moveZX = s.str();
cout << "falcon z-axis: " << moveZX << endl;   // send to
    // console to see values

g.command(moveZX);        // sends the string (galil
    // command) to the galil
 glutPostRedisplay();
}

// change between controlling the Falcon or the ABB robot
if (gHaptics.getBUTTON() == 1) {    // LOGO button
    wait(1);    // pauses system to get only one state of the
                // button
    if (modeB == 0) {
        modeB = 1;
        gHaptics.setMode(modeA, modeB);
    } else if (modeB == 1) {
        modeB = 0;
        gHaptics.setMode(modeA, modeB);
    }
}

// change between controlling all three grippers OR moving the two
    // grippers into alternate positions
if (gHaptics.getBUTTON() == 2) {    // TRIANGLE button
    wait(1);    // pauses system to get only
                // one state of the button
    if (modeA == 0) {
        modeA = 1;
        g.command("mode=1");    // sets the mode in galil for
            // w-axis control
        cout << "MODE ONE" << endl;
        gHaptics.setMode(modeA, modeB);    // alternate
            // position
    } else if (modeA == 1) {    // TRIANGLE button
        modeA = 0;    // all grippers
        g.command("mode=0");    // sets the mode in galil for
            // abc-axis control
        cout << "MODE two" << endl;
        gHaptics.setMode(modeA, modeB);
    }
}
// takes the gripper fingers to their home position (all the way open)
if (gHaptics.getBUTTON() == 4) {   // LIGHTNING button
    //wait(1);
    home();
}
// closed the gripper UNTIL an object is grasped.
if (gHaptics.getBUTTON() == 8) {   // PLUS button
    //wait(1);
    grasp();
}
    renderBitmapString(-0.47f, -0.1f, -5.0f, GLUT_BITMAP_TIMES_ROMAN_24, "Limit Reached!!");
}

// used during development to grasp the objects
void grasp() {
    renderBitmapString(-2.1f, 1.0f, -5.0f, GLUT_BITMAP_TIMES_ROMAN_24, "Grasping Object");
    g.programDownloadFile("galilCODE/grasp.dmc");
    g.command("X Q");
    cout << "used during testing phase ONLY" << endl;
}

// can be used at any time to 'home' the gripper
// used mostly during development but can be useful if the gripper limit
// switches cause the fingers to get stuck
void home() {
    renderBitmapString(-2.1f, 1.0f, -5.0f, GLUT_BITMAP_TIMES_ROMAN_24, "HOME POSITION");
    g.programDownloadFile("galilCODE/home.dmc");
    g.command("XQ");
    cout << "used during testing phase ONLY" << endl;
    g.programDownloadFile("galilCODE/mainPROG.dmc");
    g.command("XQ");
}

bool wait (int seconds) {
    clock_t endwait;
    endwait = clock() + seconds * CLOCKS_PER_SEC;
    while (clock() < endwait) {
        if (!gHaptics.isButtonDown()) {
            return false;
        }
    }
    return true;
void drawGraphics() {
    float altPosX, altPosY;
    double perAlt;
    double diff = -0.7;  // difference between line travels
    double realTIMEx;   // used for x-axis movement
    string moveXW;      // moves the x-axis on the falcon but the
                        // w-axis on the gripper

    // declares for sphere
    GLdouble radius = 0.2;
    GLint slices = 30;
    GLint stacks = 30;
    GLfloat r = 1.0f;
    GLfloat g1 = 1.0f;
    GLfloat b = 0.0f;

    // draw three spheres
    glLoadIdentity();
    glTranslatef(x1, yy1, z1);
    glColor3f(r, g1, b);
    glutSolidSphere(radius, slices, stacks);  // top sphere

    glLoadIdentity();
    glTranslatef(x2, y2, z2);
    glColor3f(r, g1, b);
    glutSolidSphere(radius, slices, stacks);  // top sphere

    glLoadIdentity();
    glTranslatef(x3, y3, z3);
    glColor3f(r, g1, b);
    glutSolidSphere(radius, slices, stacks);  // top sphere

    glColor3f(1.0f, 1.0f, 1.0f);

    // draws all 'limit lines'
    glLoadIdentity();
    glBegin(GL_LINES);
    glVertex3f(-0.21f, 1.22f, -5.0f);  // top sphere, iniPOS, TOP line
    glVertex3f(0.21f, 1.22f, -5.0f);
    glVertex3f(-0.21f, 1.22f, -5.0f);  // top sphere, iniPOS, LEFT line
    glVertex3f(-0.21f, 0.8f, -5.0f);
    glVertex3f(0.21f, 1.22f, -5.0f);  // top sphere, iniPOS, RIGHT line
    glVertex3f(0.21f, 0.8f, -5.0f);
    glVertex3f(-1.22f, -0.29f, -5.0f);  // left sphere, iniPOS, TOP line
}
APPENDIX I. SOURCE CODE - FALCON CONTROL

```cpp
332    glVertex3f(-0.8f, -0.29f, -5.0f);
333    glVertex3f(-1.22f, -0.29f, -5.0f); // left sphere, iniPOS, LEFT line
335    glVertex3f(-1.22f, -0.7f, -5.0f);
336    glVertex3f(-0.8f, -0.7f, -5.0f);
337    glVertex3f(-1.22f, -0.7f, -5.0f); // left sphere, iniPOS, BOTTOM line
338    glVertex3f(0.8f, -0.7f, -5.0f);
339    glVertex3f(-1.22f, -0.7f, -5.0f); // left sphere, iniPOS, BOTTOM line
340    glVertex3f(1.22f, -0.29f, -5.0f); // right sphere, iniPOS, TOP line
342    glVertex3f(0.8f, -0.29f, -5.0f);
343    glVertex3f(-1.22f, -0.29f, -5.0f); // right sphere, iniPOS, RIGHT line
344    glVertex3f(-1.22f, -0.7f, -5.0f);
345    glVertex3f(1.22f, -0.7f, -5.0f);
346    glVertex3f(0.8f, -0.7f, -5.0f); // right sphere, iniPOS, BOTTOM line
347    glVertex3f(1.22f, -0.7f, -5.0f);
349    glVertex3f(-0.21f, 0.09f, -5.0f); // top sphere, gripCLOSE, BOTTOM line
351    glVertex3f(0.21f, 0.09f, -5.0f);
352    glVertex3f(-0.3f, -0.2f, -5.0f); // left sphere, gripCLOSE, DIAGONAL line
353    glVertex3f(0.0f, -0.5f, -5.0f);
354    glVertex3f(0.3f, -0.2f, -5.0f); // right sphere, gripCLOSE, DIAGONAL line
356    glVertex3f(0.0f, -0.5f, -5.0f);
357    glEnd();

359    // if in gripper mode, keep alternate line horizontal
360    if (modeA == 0) {
361        altPosX = -1.0f;
362        altPosY = -0.5f;
363    }

364    // moves alternate position line to new position using x-axis
365    else if (modeA == 1) {
366        perAlt = (1.8 - posXYZ[0]) / 3.6;
367        altPosX = -1.0 - perAlt * diff;
368        altPosY = -0.5 + perAlt * diff;
369    }
370    g.command("mode=1"); // sets the mode in galil for
371    realTIMEx = perAlt*22000;
373    // stringstream ss;
374    std::ostringstream s2; // takes a double and puts it
375    moveXW = s2.str();
```
cout << "falcon x-axis: " << moveXW << endl;
g.command(moveXW); // sends the string (galil command) to the galil
// Sleep(3000); // needed so that the gripper can complete the movement
glutPostRedisplay();
}

// drawing the two alternate gripper position lines
glLoadIdentity();
 glBegin(GL_LINES);
 glVertex3f(-0.3f, -0.5f, -5.0f); // left
 glVertex3f(altPosX, altPosY, -5.0f);
 glVertex3f(0.3f, -0.5f, -5.0f); // right
 glVertex3f(-altPosX, altPosY, -5.0f);
 glEnd();

// any text is drawn on the screen here
void drawLabels() {
  GLfloat r = 0.0f;
  GLfloat g = 0.0f;
  GLfloat b = 0.0f;
  GLfloat r1 = 1.0f;
  GLfloat g1 = 1.0f;
  GLfloat b1 = 1.0f;

  renderBitmapString(-1.3f, -1.5f, -5.0f,
                    GLUT_BITMAP_TIMES_ROMAN_24,
                    "Gripper Mode");
  renderBitmapString(0.3f, -1.5f, -5.0f, GLUT_BITMAP_TIMES_ROMAN_24,
                    "Device Mode");

  if (modeA == 0) {
    glColor3f(r1, g1, b);
    renderBitmapString(-1.1f, -1.3f, -5.0f,
                       GLUT_BITMAP_HELVETICA_18,
                       "All Fingers");
  }
  else if (modeA == 1) {
    glColor3f(r1, g, b);
    renderBitmapString(-1.3f, -1.1f, -5.0f,
                       GLUT_BITMAP_HELVETICA_18,
                       "Alternate Position (w-axis)");  
  }
  if (modeB == 0) {
    glColor3f(r1, g1, b);
    renderBitmapString(0.3f, -1.3f, -5.0f,
                       GLUT_BITMAP_HELVETICA_18,
                       "Falcon Control");
APPENDIX I. SOURCE CODE - FALCON CONTROL

425 }  
426 else if (modeB == 1) {
427     glColor3f(r1, g, b);
428     renderBitmapString(0.3f, -1.1f, -5.0f,
429                         GLUT_BITMAP_HELVETICA_18,
430                         "ABB Control");
431 }
432 glColor3f(1.0f, 1.0f, 1.0f);
433 
434 renderBitmapString(-1.5f, -0.5f, -5.0f, GLUT_BITMAP_HELVETICA_18,
435                     "'X'");
436 renderBitmapString(1.4f, -0.5f, -5.0f, GLUT_BITMAP_HELVETICA_18,
437                     "'Y'");
438 renderBitmapString(-0.08f, 1.5f, -5.0f, GLUT_BITMAP_HELVETICA_18,
439                     "'Z'");
440 renderBitmapString(-1.0f, 1.9f, -5.0f, GLUT_BITMAP_HELVETICA_18,
441                     "Simulation of Falcon / Gripper Movements");
442 
443 // instructions
444 renderBitmapString(0.5f, 1.5f, -5.0f, GLUT_BITMAP_HELVETICA_12,
445                     "'esc' = EXIT");
446 renderBitmapString(0.5f, 1.35f, -5.0f, GLUT_BITMAP_HELVETICA_12,
447                     "'Press 'Logo' = Falcon or ABB Control");
448 renderBitmapString(0.5f, 1.2f, -5.0f, GLUT_BITMAP_HELVETICA_12,
449                     "'Press 'Triangle' = 'W-axis Position Control'");
450 renderBitmapString(0.5f, 1.05f, -5.0f, GLUT_BITMAP_HELVETICA_12,
451                     "'Press 'Lightning' = Grippers to HOME position'");
452 renderBitmapString(0.5f, 0.9f, -5.0f, GLUT_BITMAP_HELVETICA_12,
453                     "'Press 'PLUS' = Grippers to Grasp Object");
454 glColor3f(1.0f, 1.0f, 1.0f);
455 }
456 
457 // What to do when doing nothing else
458 void glutIdle ()
459 {
460     glutPostRedisplay();
461 }
462 
463 // checking keyboard for key presses
464 void glutKeyboard(unsigned char key, int x, int y) {
465     static bool initiated = true;
466 
467     if (key == 27) {  // esc key
468         // calls the 'exit' galil program
469         // does not always work perfectly, depends on state of gripper when run
470         g.programDownloadFile("galilCODE\exit.dmc");  // exits code cleanly turning off all motors
471         g.command("XQ");
472     }
Sleep(2000);
exit(0);
}
glutPostRedisplay();
}

// if the window size is changed, the objects inside will be resized as well
void glutReshape(int w, int h) {
    // Prevent a divide by zero, when window is too short
    // (you can't make a window of zero width).
    if (h == 0)
        h = 1;
    double ratio = w * 1.0 / h;  // changed 'float' to 'double'
    due to warning
    glMatrixMode(GL_PROJECTION);  // Use the Projection Matrix
    glLoadIdentity();  // Reset Matrix
    glViewport(0, 0, w, h);  // Set the viewport to be the entire window
    gluPerspective(45, ratio, 0.1, 100);  // Set the correct perspective.
    glMatrixMode(GL_MODELVIEW);  // Get Back to the Modelview
}

// make sure we exit cleanly
void exitHandler() {
    gHaptics.uninit();
}

// renders any text used
void renderBitmapString(float x,
                        float y,
                        float z,
                        void *font,
                        char *string) {
    char *c;
    glLoadIdentity();  // reset matrix
    glRasterPos3f(x, y, z);
    for (c = string; *c != '\0'; c++) {
        glutBitmapCharacter(font, *c);
    }
}

void restorePerspectiveProjection() {
    glMatrixMode(GL_PROJECTION);
glPopMatrix(); // restore previous projection matrix

glMatrixMode(GL_MODELVIEW); // get back to modelview mode

void setOrthographicProjection() {

glMatrixMode(GL_PROJECTION); // switch to projection mode

// save previous matrix which contains the
// settings for the

glPushMatrix(); // perspective projection.

glLoadIdentity(); // reset matrix

gluOrtho2D(0, w, h, 0); // set a 2D orthographic projection

glMatrixMode(GL_MODELVIEW); // switch back to modelview mode

}
Appendix J
Source Code - Galil Motion Controller

------------------- Main Program -------------------

1 '------------------------------------------------
2 ' Main Program
3 '------------------------------------------------
4 #AUTO
5 '------------------------------------------------
6 ' Variables
7 '------------------------------------------------
8 spd=30000;
9 spdD=3000;
10 inAD=-1;
11 outAD=1;
12 inBC=1;
13 outBC=-1;
14
15 #MAIN
16 JP#HOME;
17 '------------------------------------------------
18 ' Home Position
19 '------------------------------------------------
20 #HOME
21 SHAD
22 JG spd, -spd, -spd, spdD
23 IF (_LFA=1); BGA; ENDIF;
24 IF (_LFD=1); BGD; ENDIF;
25 '------------------------------------------------
26 AMAD
27 'set encoders to ZERO
28 DPA=0;
29 DPD=0;
30 WT 500
31 JP#CHECK
32
33
APPENDIX J. SOURCE CODE - GALIL MOTION CONTROLLER

...
'------------------
' Grasp Program
'------------------
'------------------
' Variables
'------------------
#VAR
spd = 5000;
spdD = 1500;
gripA = 1000;
inAD = −1;
outAD = 1;
inBC = 1;
outBC = −1;

#MAIN
CN , , 1
MG "grasping"
WT1000
JP #GRASP

'------------------
' Grasping
'------------------
#GRASP
JGA = −spd;
SHA;
MG "start"
BGA;

#LOOP
JP #LOOP, (@IN[01] = 1)
IF (@IN[01] = 0)
MG "Stop A..."
STA
MO
JP #MAIN
ENDIF
EN