NTNU

Norwegian University of
Science and Technology

Computerised Methods and Device for Intuitive Use of the Human Hand for Touching and Re-shaping of Three-Dimensional Virtual Objects

Vegar Neshaug

Master of Science in Computer Science
Submission date: June 2010
Supervisor: Torbjørn Hallgren, IDI
Co-supervisor: Jo Skjermo, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The task entails the development of a device which measures the position and main movements of the human hand and process these signals, such that they can be realistically visualized. Further, it includes modeling the interaction and deformation of the human hand and mathematical surfaces.

Assignment given: 15. January 2010
Supervisor: Torbjørn Hallgren, IDI

Abstract

Today, the use of the mouse and keyboard input devices to interface with the computer is common almost regardless for what end the computer is used. This is no less true for users who daily work in three-dimensional modeling. Because of this, there is a significant threshold for individuals beginning three-dimensional modeling before being able to form even the most rudimentary of objects. A possible approach to lowering this threshold is to bridge the gap between traditional sculpting and virtual sculpting by utilizing the full range of human hand motion when interfacing with the computer.

This work review the status of Human-Computer Interaction devices and methods, and set out to design and implement a low-cost data glove prototype.

A polygon mesh deformation method is developed which demonstrates the functionality of the glove and system design.

The work constitutes a useful platform for further academic work in this field.

Contents

List of Figures

1 Introduction

Today, the use of the mouse and keyboard input devices to interface with the computer is common almost regardless for what end the computer is used. This is no less true for users who daily work in three-dimensional modeling. Because of this, there is a significant threshold for individuals beginning three-dimensional modeling before being able to form even the most rudimentary of objects. A possible approach to lowering this threshold is to bridge the gap between traditional sculpting and virtual sculpting by utilizing the full range of human hand motion when interfacing with the computer.

There exist specialized HCI devices today called data gloves which capture the essential movements of the human hand. Support for these devices in three-dimensional modeling packages is non-existent or low. Furthermore, they depend on bundled proprietary closed-source software development kits and the cost of obtaining a data glove is not insignificant. Proprietary closed-source software is generally not the ideal starting point for academic research.

One approach to overcome these challenges consists of several tasks. First develop an open design and open source data glove. Then demonstrate how the data glove can interface with an open source three-dimensional modeling application. Surface deformation methods are implemented to show how virtual sculpting using a data glove can be achieved.

1.1 Problem Definition

The task entails the development of a device which measures the position and main movements of the human hand and process these signals, such that they can be realistically visualized. Further, it includes modeling the interaction and deformation of the human hand and mathematical surfaces.

1.2 Goals

The problem is broken down into the following goals.

1. A literature review of:
   (a) State of the art data gloves

(b) Surface deformation methods

(c) Hand Gesture recognition methods

2. Design and develop a prototype data glove
3. Design and develop an electronic circuit to interface the data glove with a personal computer
4. Choose a three-dimensional modeling application and implement support for the data glove
5. Design and develop a model in the selected modeling application for hand and finger movements in 3D
6. Implement a hand gesture recognition method which is suitable in a modeling context
7. Develop and implement surface deformation methods which are suitable for use with a data glove

1.3 Problem Solving

To be able to reach the goals outlined within the time restrictions, some constraints and decisions were made.

Blender will be used as the modeling application and framework for the implementation. The main reasons for this choice are:

1. Blender is open source and therefore allows compiling in external libraries(like VRPN)
2. Blender has a Python API and integrates the Python interpreter, allowing for fast prototyping

The FlexPoint Bend Sensor will be used as the joint flexion sensor. FlexPoint provided samples of their Bend Sensor free of charge.

The Atmega168 microcontroller with on-chip ADC and USART will be used for conversion of analog values and serial transmission. This microcontroller limits the number of sensors which can be sampled to six sensors. It is however selected because the aim is to create proof-of-concept prototype. It represents an easily available and convenient component with sufficient functionality to demonstrate the main goal.

2 Previous Work

In "Virtual Reality and Human Hand Haptic Methods"[23] the use of a data glove and a haptic skeleton was studied. Methods and models for computing forces were implemented and discussed. The focus was on using touch in interacting with virtual objects and how natural this felt. The work proposes permanent deformations as a potential area for further study. Also, abstract use of the data glove is mentioned as a method for rotating and moving an object while being edited using a different tool like the Phantom.

A hand model was developed based on the principles of armature and bone rigging. This is a hierarchical method of orientating a bone in which a given bone is dependent on its own rotation and the transformation of the bones preceding it. Bone orientation was represented using quaternions.

Further, the hand was given a three-dimensional representation using capsule and box collision primitives for the bones of the fingers and palm respectively. This allowed interacting with objects using all parts of the hand, although only the fingertips were given haptic feedback because of the nature of the haptic exoskeleton used. Figure 2 shows a screen capture of a real-time simulation.

Figure 1: Human hand interaction with rigid bodies

Figure 2: Hand of collision primitives simulating haptic feedback from rough surface

Future work was outlined, mentioning permanent deformations and abstract use of the data glove. These ideas are built on and expanded upon here.

3 Literature Review

The literature review focused on collecting works within the fields of 3D animation, deformation methods and motion capture methods and devices.

A large body of literature on deformation methods is based on the application of FEM and NURBS surfaces. FEM methods are used largely for accurate and physically based stress and strain computation, while NURBS are extensively used for 3D Computer-Aided Design applications. The basic form of modeling using NURBS is done by moving control points which are not part of the mesh, and often not even very near a mesh vertex.

The emphasis in this thesis is to deform a mesh without regarding stress and strain. What is important is the interaction between the hand movements and the visual response. Hence, a simpler model will suffice. The idea of using a simpler method was inspired by the example application in the open source haptics software development platform H3DAPI(http://www.h3dapi.org). This simpler method is based on deforming the surface by a gaussian distribution centered on a haptic stylus. A specific literature search on this method did not produce relevant material.

3.1 Human Machine Interaction

The field of Human-Machine Interaction is very broad.

"Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review"[21] includes a list of relevant literature in the field of hand gesture recognition. The focus in this article is on visual interpretation of the human hand. The use of data gloves is dismissed as cumbersome, and methods for visual recognition are suggested.

In the bulk of literature found, the most relevant application of human hand motion capture is based on the use of data gloves. Hence, the literature review was narrowed down to the use of data gloves in hand gesture recognition and telerobotics[21, 20, 13, 10, 8, 4].

3.2 Deformation and Modeling

From the literature review on deformation and modeling, the most relevant works was related to haptic applications[9, 19, 2, 18, 27]. Many of the haptic deformation methods focus on volumetric models, which is often referred to by the term Virtual Clay[2, 19]. Other deformation methods are based on parametric surfaces[9], or by indirectly deforming a polygon mesh by a surrounding lattice "cage"[14]. All of these methods are potentially suitable for this work. However, as stated in the beginning of this chapter, a simpler method of directly deforming mesh vertices is chosen as more suitable.

3.3 Hand Gestures

Hand gesture recognition is a big field of research. There are two dominant forms of approaching the problem of gesture recognition. The difficulty involved in gesture recognition depends on which approach is used. This is discussed in [21].

One form of gesture recognition is based on computer vision and digital image processing[25]. Many methods are used to achieve image based gesture recognition. Some methods attempt to recognize the general shape of the hand by pattern recognition when it is in a certain pose[6]. Other methods try to estimate the internal angles of the hand and assign these to an intermediate hand

model[26]. This intermediate hand model, call it a skeleton, is then used to recognize a gesture.

Another form of gesture recognition is based on the use of a dataglove[16]. Estimating the angles of the joints becomes much simpler, although not trivial. These angles can be assigned to a skeleton hand model which is used as the basis for gesture recognition.

3.4 Data Gloves

No relevant published works on the construction of data gloves were found. To gain some insight on their inner workings, the commercial producers(notably CyberGlove and 5DT) publish data sheets and manuals which give an overview of sensor placement and output.

There are however some published works on the calibration of a data glove[13, 10, 8, 4]. To accurately calibrate a data glove it is necessary to compare the sensor readings to the actual articulation of the hand. In other words, an accurate form of measuring the articulation must already be available for accurate calibration.

4 Overview and Architecture

The Top Level Data Flow Diagram(DFD) shown in Figure 3 gives a perspective on the system design. This perspective called the data flow perspective shows where data is generated and which processes act on the data.

Starting from the top, the data glove and the ADC circuit shows that the analog data is processed into a digital serialized form, which is then processed by the VRPN[22] Glove Device Driver. The VRPN Server uses the data from the VRPN Glove Device Driver and the VRPN Tracker Device Driver for transmission to the VRPN Client. The Flock-Of-Birds is used as the tracker. FoB has limited range, but high accuracy within that range. The range is appropriate for the purpose of operating data gloves in front of a monitor.

VRPN sets the premise for the system architecture in that it is a network transparent client-server architecture[22]. VRPN acts as the device layer for interfacing with the tracker, and was also chosen as the device layer for the data glove. This is because of three major reasons:

1. VRPN provides the same client interface for all supported devices in a device class(e.g a tracker).

2. VRPN is network-transparent allowing decoupling equipment-machine and client machine.

3. VRPN can easily be extended to support new hardware.

The Blender Python API allows for rapid prototyping. VRPN is written in C/C++ and compiled to native libraries. To bridge between native code and Python, it is possible to use utilities which generates a python-accessible "wrapper" to interface with the native code. This is illustrated by the VRPN Python Wrappers interface shown in Figure 3.

The Visual Hand Representation component uses the now python-accessible VRPN data to create a visual hand representation in the Blender Python API.

Surface Deformation, Hand Gesture Recognition and Visual Hand Representation has a complex interconnection. Surface Deformation needs data from the Visual Hand Representation to be able to deform surfaces using the hand. Similarly, Hand Gesture Recognition needs data from the Visual Hand Representation to detect gestures. They all use the Blender Python API to produce desired effects in the Blender Scene. The Blender Scene is rendered onto the computer monitor by Blender.

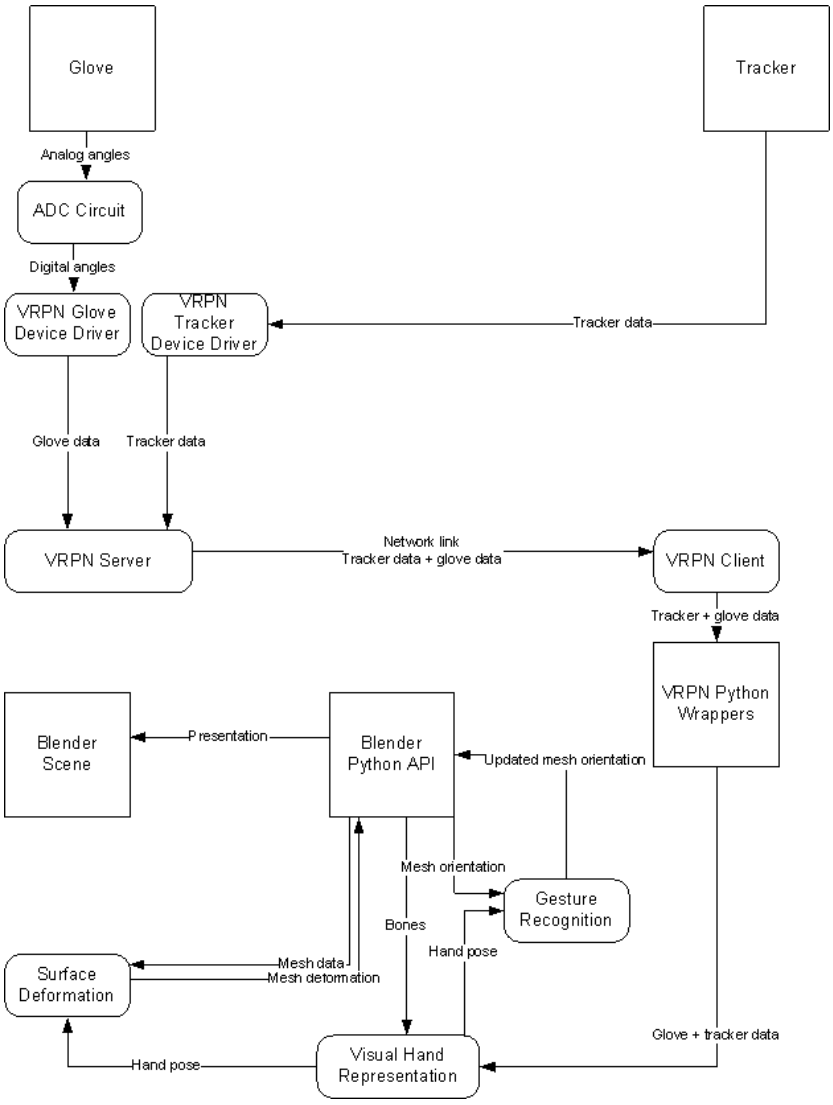

Figure 3: Top Level Data Flow Diagram

5 Data Glove

The design of the data glove is largely determined by the decision to use the FlexPoint Bend Sensor. These sensors will be placed over the finger joints and will thus be similar to the CyberGlove and 5DT Ultra 16. The design diverges from the CyberGlove and 5DT Ultra 16 in that a sensor is placed in the palm of

the hand. This is intended to provide a better sensor value for the movement of the metacarpal phalanx of the thumb than the commercial products. In Figure 4 the data glove with wires attached can be seen. There are two wires connected to each sensors which are led inside a double-layered glove and terminated in a 2-wire connector which is to be connected to a pin header on the circuit board.

Figure 4: Data glove and Printed Circuit Board w/USB

5.1 Sensor Layout

The range of motion of the human hand is complex[11]. The hand consists of bones called phalanges connected by ligaments which form the joints. One should realize that all the joint ligaments allow displacement and rotation in all directions.

Fingers consist of a distal, an intermediate and a proximal phalanx. The thumb does not have an intermediate phalanx.

A sensor layout using bend sensors laid over the hand is inherently limiting.

The bend sensors are placed such as to isolate one axis of rotation. This simplifies the translation of the sensor readings to a 3D skeletal model, discussed in Chapter 7.

5.2 Electronic Circuit

The electronic circuit is designed around the Atmega168 microcontroller. Since the ADC converts a voltage signal, the bend sensor potentiometer variation must be converted to a voltage signal for sampling by the ADC. A voltage divider is used for this purpose. This is one of several conversion circuits suggested by the FlexPoint Bend Sensor Design Considerations[5] document.

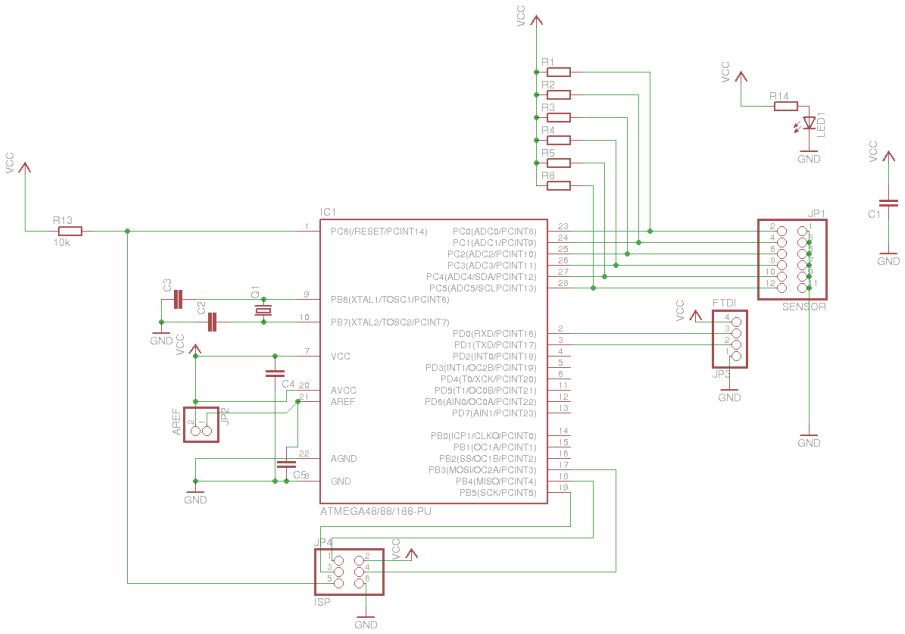

Figure 5: Circuit Schematic

Consider Figure 5. The output of the voltage divider can be easily found by applying Ohm's law.

$V_{in} = I \cdot (R_1 + R_2)$

$V_{out} = I \cdot R_2$

$V_{out} = \frac{V_{in}}{R_1+R_2} \cdot R_2 = V_{in} \cdot \frac{R_2}{R_1+R_2}$

Hence, the voltage signal input to the ADC is a function of the bend sensor resistivity:

$$V_{out}(R_{bend}) = V_{in} \cdot \frac{R_{bend}}{R_1 + R_{bend}} \tag{1}$$

The electronic schematic was made using the Eagle CAD software[3] and can be seen in Figure 5.

The circuit is powered by the USB computer port through the FTDI 3v3 level shifter device cable. This provides a 5v voltage source Vcc to power the circuit and provide the reference voltage V_{ref} to the voltage divider V_{in} and ADC. Hence $Vcc = V_{ref} = V_{in}$. The digital conversion value is related to the bend sensor in the voltage divider by the following equation(cit datasheet):

$$ADC = 1024 \cdot \frac{V_{out}(R_{bend})}{V_{ref}} \tag{2}$$

Thus, it follows that:

$$ADC = 1024 \cdot \frac{R_{bend}}{R_1 + R_{bend}} \tag{3}$$

Equation 3 has an influence on the calibration of the visual representation which is considered later.

To utilize the full resolution of the ADC, the fixed resistor R_1 should be selected with care. Let R_{max} and R_{min} be respectively the maximum and minimum resistivity of the bend sensor. The following equation should be maximized:

$$1024 \cdot \frac{R_{max}}{R_1 + R_{max}} - 1024 \cdot \frac{R_{min}}{R_1 + R_{min}} \tag{4}$$

The resistivity of the bend sensors used in the data glove had an average R_{max} of 250kOhm at maximum joint flexion. When the joint was unflexex R_{min} was measured to an average of 9kOhm. Tabulating equation 4 to some standard resistor values available at the lab reveals the following:

R_{min}(Ohm)	R_{max}(Ohm)	R_1(Ohm)	$1024 \cdot \frac{R_{max}}{R_1+R_{max}} - 1024 \cdot \frac{R_{min}}{R_1+R_{min}}$
9000	250000	24000	655
9000	250000	47000	697
9000	250000	96000	652

Thus, 47kOhm was selected for R_1.

Most of the components are selected according to the Atmega168 datasheet[1] recommendations. The crystal oscillator value of 18.432Mhz however is selected for two reasons. The Atmega168 can run at a maximum clock frequency of

20Mhz. Also, it produces minimum transmission error (0.0%) with a BAUD rate of 115200 symbols/sec. For an in-depth discussion on the relationship between oscillator frequency, transmission error and baud rate, see Appendix D.

It is common to use a MAX232 level shifter to produce a voltage level for connecting to a standard computer serial port. Newer computers are rarely shipped with such serial ports in favor of the USB peripheral ports. The FTDI 3v3 level shifter provides an interface between the USART(Unified Synchronous-Asynchronous Receiver/Transmitter) voltage levels USB.

5.3 Microcontroller Program

The microcontroller was programmed specifically for high resolution and minimum noise ADC conversion. Without going into too much detail, the microcontroller was programmed to halt the clocks for circuitry not involved in ADC conversion. This method of ADC conversion is described in the datasheet.

To transmit the converted sensor values, the microcontroller was programmed to support a serial messaging protocol for communicating with a computer over the USB cable. The serial messaging protocol supports querying the number of sensors available. When the signal to transmit sensor readings is received it will continuously sample the ADC channels in a sequential manner. After each sequential sampling iteration, it transmits all sensor readings to the computer.

For in-depth details on the workings of the microcontroller program, see Appendix D.

5.4 VRPN Device Driver

To use the data glove with a computer, a device driver must be implemented to communicate with the electronic circuit.

The device driver queries the number of sensors available on the glove. Then, it signals the glove to start transmitting continuous sensor readings. It then continuously reads the sensor data from the glove and transmits them to all connected VRPN clients. The specifics of the VRPN device driver can be found in Appendix D.

5.5 Printed Circuit Board

To create a printed circuit board, the schematic must be traced out. This was done using the Eagle CAD software[3]. Figure 6 shows the trace schematic which has been traced out in one plane.

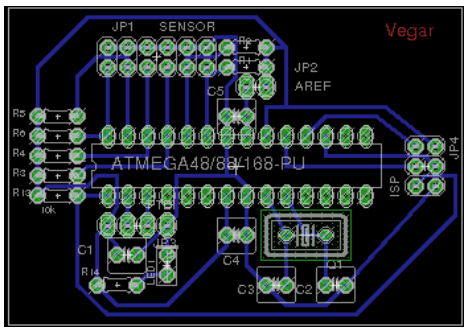

Figure 6: Trace Schematic

6 Laboratory Work

The laboratory work of creating a working data glove is significant. Choosing components, building and testing is an iterative task. Much attention to detail is necessary to arrive at a working prototype. The details of this work is presented in Appendix D and should be helpful for future work in this field.

Much experimentation was conducted to arrive at a near noise-free bend sensor sampling circuit. The microcontroller program and circuit is usable for many designs which samples analog signals.

7 Hand Representation

To create a presence for the hand in a virtual scene a three-dimensional representation of the hand must be created. Chapter 2 discussed previous work on creating this representation based on collision primitives.

There are two important purposes a hand representation serves. One is to render a visual representation on-screen. The other is to use the hand representation for collision detection and object interaction.

7.1 Skeletal animation

Skeletal animation is an animation technique for deforming a polygon mesh, called the skin, according to the movement of rigid objects, called bones. The terms skeletal animation and bones are very fitting for deforming the mesh of a character. Skeletal animation is usually available in modern 3D authoring packages, including Blender.

Thus, the hand representation is achieved by creating a hierarchical bone structure for the human hand. A polygon mesh of the hand will create a visual representation. The bones were carefully positioned using the mesh as a guide.

The displacement of the bones will cause deformation in the mesh. Weber[24] gives a good description of skeletal animation and deformation. The bone deforms the mesh by moving the mesh vertices. A hierarchy consisting of a single bone attached to all vertices will simply move the vertices analogous to a fixed rod. Each vertex is translated and rotated by the bone transform directly. The bone transformation can be a matrix transformation matrix or a rotation quaternion and displacement vector representation.

When the hierarchy consists of several bones, the question becomes which vertices a bone displacement should affect. This is solved partially by using vertex groups. Each bone is assigned a vertex group which the bone transform is applied to. Ambiguity arises if vertex groups overlap, meaning a vertex is assigned to more than one vertex group. Solving this ambiguity is sometimes called blending. A simple blending method is to assign a weight to each relation between a vertex and a bone vertex group, and use the weighted sum as the vertex displacement.

$$\mathbf{p}_{deformed} = \sum_{i=1}^{n} (w_i \mathbf{p}_i \mathbf{M}_i)$$

Where p_i denotes the position of the vertex relative to bone i and M_i is the bone transformation matrix.

This method is called Skeleton-Subspace Deformation and is discussed by Lewis et al[17] and a solution to the infamous collapsing elbow problem is proposed.

The following figures show how skeleton based deformation looks in Blender. Both vertex painting and blending is shown. Note that Blender probably uses a more sophisticated blending method than Skeleton-Subspace Deformation.

Figure 7 shows some of the vertex groups on the mesh skin. Note the smooth overlap between adjacent finger phalanges.

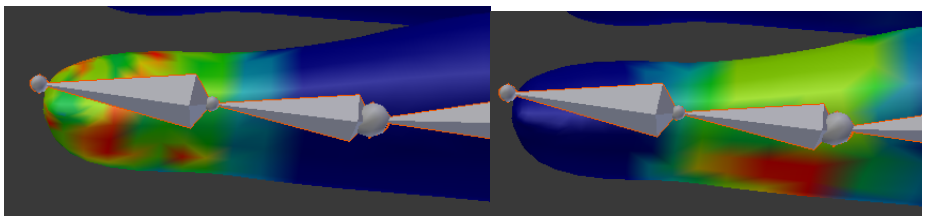

Figure 7: Vertex group painting in Blender

Figure 8 shows the blending between two finger phalanges.

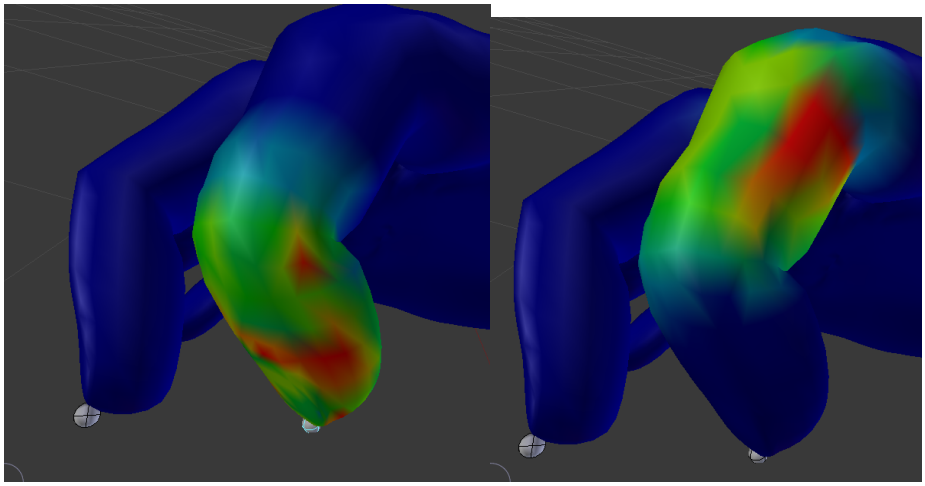

Figure 8: Bone deformation in Blender

Kry et al.[15] discusses a more sophisticated skin deformation technique and illustrates its operation on a human hand mesh.

Such a representation does not only serve the purpose of presenting the hand on a monitor. It also provides the basis for interacting with objects in the scene. In contrast, here the representation is based on a mesh model of a human hand.

7.2 Calibration

The sensor readings from the data glove are 10-bit integer values. Generally, each sensor value is a complex function of joint extension, roll,abduction and displacement. Note that although constrained, every joint ligament can in fact move and rotate in all directions. Further, the sensor does not measure the bending of the joint itself, it measures the curve of the glove surface over the joint. The glove surface over one joint in many cases will deform when other joints extend or abduct. This is referred to as cross-coupling. The degree of cross-coupling in the CyberGlove is evaluated in[13].

In the following, a simplification of the above problem, by reducing the degrees of freedom for a joint, is stated. Define $s_i \in \vec{s}$ as the converted value of sensor i which measures a *single* angular quantity $\theta_i \in \vec{\theta}$. Then, $\vec{s}$ is a function of the angular quantities $\vec{\theta}$:

$$\vec{s} = \vec{f}(\vec{\theta}) \tag{5}$$

Thus, to find the angular quantities, the inverse function $\vec{g} = \vec{f}^{-1}$ must be found, such that:

$$\vec{\theta} = \vec{g}(\vec{s}) \tag{6}$$

In general, this problem is very difficult[10]. An inverse function for the voltage divider can easily be found to obtain the actual bend sensor resistance, however the bend sensors respond differently depending on the type of bending and where on the sensor strip the bending occurs. Also, the cross-coupling is different for every sensor. Efforts to solve this problem has largely consisted of approximation and simplified models.

To be able to assess the effectiveness of an approximation, and to aid in the calibration, an accurate measurement rig must be present. Some use computer vision systems, which the sensor readings are compared to[4]. In the absence of an accurate measurement rig, the hand model can be placed in a known configuration which the user attempts to imitate. The hand model can be placed in several known configurations, where linear regression and the method of least squares can be used to estimate a line which does not necessarily intersect the

calibration points, but instead will minimize the distance to several calibration points[8].

The term Visual-fidelity[13] describes an approach based on plausibility. The essence of this approach is that as long as certain poses are reflected accurately, like when two fingertips meet, the inaccuracies between the specific poses are not perceived by the user. Also discussed by Kahlesz et al.[13] is that linear calibration and an assumption of no significant cross-coupling is appropriate on the sensors over the finger joints which measure distal, intermediate and proximal flexion and extension.

In this project, an accurate measurement rig was not available. Therefore linear calibration is used with the assumption that no significant cross-coupling is present. The calibration mechanism will then consist of the following:

1. Set the hand model in the first known configuration called the rest pose, $\vec{\theta}_{rest}$
2. Store the sensor values when the user attempts to imitate the rest pose, $\vec{s}_{rest}$
3. Set the hand model in the second known configuration called the calibration pose, $\vec{\theta}_{cal}$
4. Store the sensor values when the user attempts to imitate the calibration pose, $\vec{s}_{cal}$

For a single sensor, s_i the hand model angle θ_i will then be given from the equation:

$$\frac{\theta_i - \theta_{i,rest}}{s_i - s_{i,rest}} = \frac{\theta_{i,cal} - \theta_{i,rest}}{s_{i,cal} - s_{i,rest}}$$

Solving for θ_i yields

$$\theta_i = \theta_{i,rest} + (s_i - s_{rest})\frac{\theta_{i,cal} - \theta_{i,rest}}{s_{i,cal} - s_{i,rest}}$$

$$\theta_i = g(s_i) = s_i - s_{i,cal}\frac{(\theta_{i,cal} - \theta_{i,rest})}{(s_{i,cal} - s_{i,rest})} \tag{7}$$

Even if the bend sensor is placed right over a hinge joint like the joint connecting the proximal and metacarpal phalanges the sensor will be affected by other parts of the hand articulation. This has an impact on the accuracy of the sensor measurement. Likewise, any small displacement of the sensors during the session has an effect on the accuracy.

There are methods which reduce the effect of other parts of the hand articulation, like the use of a neural network[4]. They generally require more time to calibrate than linear interpolation. Factors like ease of calibration can be more important than high precision as long as the virtual hand behaves according to some key user expectations. An example is the touching of fingertips. As long as the virtual hand seems to touch fingertips when the actual hand touches fingertips, the user will probably not notice the inaccuracies in between[13]. Hence, as long as the calibration points of the linear interpolation are set at these key points, the results should prove sufficient.

7.3 Rotation and Translation

To articulate the hand according to the data glove sensor readings, the bones are assigned the calibrated angular value corresponding to the sensor over the specific joint. The bone data structure uses quaternions to represent rotation. Thus, the angle must be converted to a quaternion. The mathematical treatment was established in previous work[23].

The translation and rotation of the human hand itself is assigned by a 6DoF tracker.

7.4 Gesture Recognition

The difficulty in gesture recognition is strongly linked to the complexity of hand gestures to be recognized and the hand motion capture method[21]. When a data glove and a skeletal hand representation is used as the motion capture method, the process is greatly simplified. However, complex hand gestures still require advanced methods of recognition.

A simple gesture like a pinching gesture is easy to recognize using the distance between the tip of the distal thumb phalanx and distal index phalanx. When the distance is less than a specific threshold, the hand is said to be in the pinching

gesture. Conversely, when the distance increases beyond the threshold, the hand is no longer in the pinching gesture.

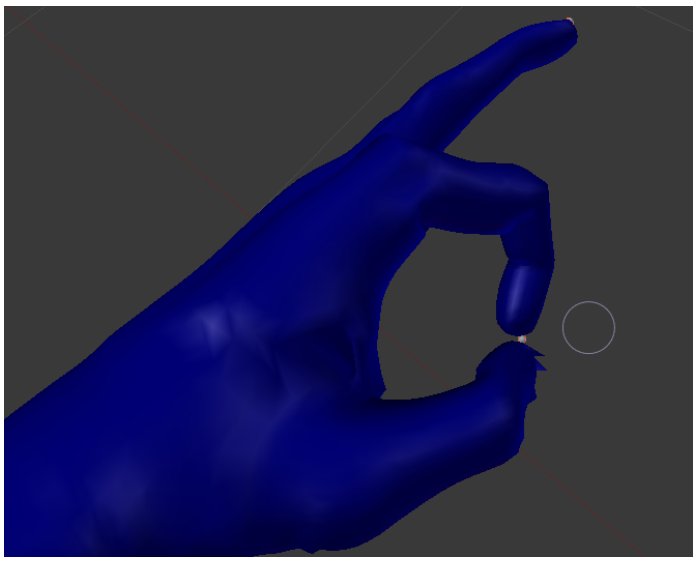

Figure 9: Pinching gesture

This is useful in two-handed operation, where the pinching gesture can be used as a command. The command can be to copy the rotation of the pinching hand to the mesh being edited.

8 Blender Add on

This section describes how the skeletal structure is created in Blender and how the data from the tracker and the data glove is obtained.

8.1 Using VRPN Python Wrappers in Blender

Blender supports scripting through Python. The Blender API for Python scripting has been revised in the development version of Blender and is released in Blender 2.5 Alpha 2. The implementation described here uses the development version of the Blender API for Python scripting, which requires Python version 3.1.

To be able to use the VRPN client library in a Python script, the VRPN distribution contains files for generating Python wrappers under the name "python_vrpn". There is little documentation for python_vrpn. Python_vrpn is based on generating Python wrappers using SWIG(Simplified Wrapper and Interface Generator).

> SWIG is a software development tool that connects programs written in C and C++ with a variety of high-level programming languages. (SWIG.org)

The python_vrpn wrappers were not readily usable. Some minor changes in the source files had to be made in order for it to compile, and a minor change for correct operation of vrpn_Analog_Remote objects.

Placing the compiled python_vrpn wrappers in Blender's python library directory gives access to these wrappers and make them immediately usable through Blender's scripting console.

8.2 Visual Representation

Creating a visual representation of the human hand in Blender is simple. This is in fact one of the main functionalities of Blender. The human hand mesh model is imported into Blender and bones are positioned inside the mesh. This process is called rigging.

First, a palm bone is positioned inside the base of the hand. The finger bones are created by extruding the main bone to the tip of the finger. Then the finger bone is subdivided to create distal, intermediate and proximal phalanges. When translated or rotated, these finger bones deform the mesh such that the mesh vertices follow the bone. For a believable deformation, the mesh must be "weight painted" in relation to each bone.

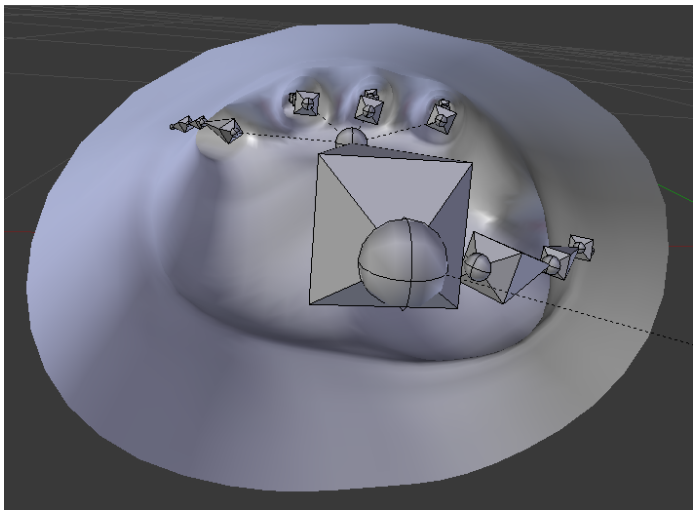

Figure 10: Bone positioning inside mesh

The bones are given names, such as "index.distal". The Blender Python API allows objects in a scene to be accessed using this name. All properties of the object, most importantly its translation vector and rotation quaternion are available through this object. Thus, the bones of the human hand can be manipulated in the python script and updated according to the data from the tracker and the data glove. This provides not only a visual representation, but also a basis for calculating distance and collisions between a fingertip and a mesh in the scene. This basis is used in mesh surface deformations which are explained in section 9.1.

9 Surface Deformation

9.1 Polygon Mesh Deformation

The hand representation can be used to deform a polygon mesh by translating the vertices of the mesh.

It is common to specify the coordinates of objects in a scene relative to a local reference frame. To obtain the world coordinates of the mesh vertices, the vertex position vector is left-multiplied with the mesh transform matrix. Similarly, to

obtain the world position of the tail end of the distal bone, the local position vector is left-multiplied with the skeleton transformation matrix.

$$\vec{v}_{world} = \mathbf{M}_{transform}\vec{v}_{local}$$

Transforming a world coordinate to a local reference frame is done by using the inverse of the matrix transform:

$$\mathbf{M}^{-\mathbf{1}}_{\mathbf{transform}}\vec{v}_{world} = \mathbf{M}^{-\mathbf{1}}_{\mathbf{transform}}\mathbf{M}_{\mathbf{transform}}\vec{v}_{local} = \mathbf{I}\vec{v_{local}} = \vec{v_{local}}$$

A mesh vertex is translated if it is within the radius of a sphere around the fingertip. In other words, the magnitude of the distance vector between the fingertip and the mesh vertex is compared to a scalar.

When the mesh vertices are within the sphere, they should be translated in a visually pleasing manner. When the fingertip touches the mesh, a smooth crater should be formed under the fingertip. Thus, the translation should create a depression in the mesh which appears continuous. Also, to preserve the integrity of the mesh, the translation should only occur in one direction such that vertex positions do not begin to cross. The normal distribution satisfies this in the x,y plane. A line, y=0, along the x-axis in a Cartesian coordinate system can be "deformed" by the normal distribution to create a smooth peak.

In three-dimensional space, the bivariate normal distribution(BND) "deforms" a plane to create a smooth crater. If the mesh is a structured grid in the x,y plane, the BND can be used to translate the depth(z component) of the vertices. Thus, if the vertices are within the fingertip radius, they can be translated by the BND centered on the x,y components of the fingertip position vector. This will create a smooth depression centered on the fingertip. If the hand is moved away from the depression, the amplitude will decrease as the BND will be centered around the new position of the fingertip. To avoid this, the mesh vertex is only translated if the BND evaluates to a value greater than the current depth. The following shows the BND equation when the variables are uncorrelated.

$$f(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}\right)$$

The depth of the depression is determined by the standard deviation when using a pure BND and not the depth of penetration into the mesh. Therefore, a different form of the BND is used, where the amplitude depends on the depth of penetration.

$$f(x,y) = A \exp\left(-\frac{(x-\mu_x)^2}{2\sigma_x^2} - \frac{(y-\mu_y)^2}{2\sigma_y^2}\right)$$

Figure 11 shows the deformation from using this method.

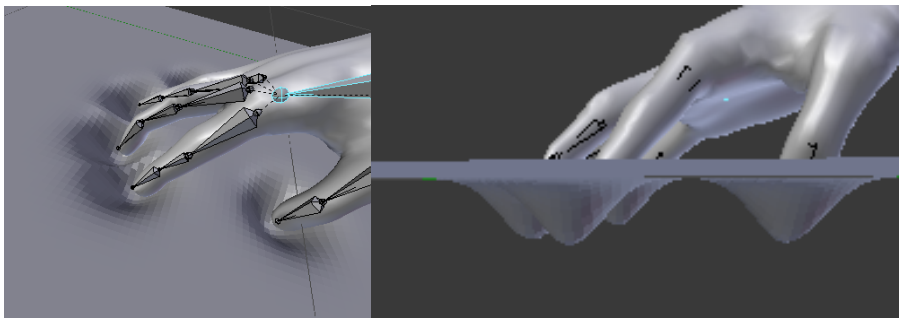

Figure 11: Simple BND deformation

The discussion so far covers the basic idea of the deformation method. It is not very useful if the mesh should be deformed in other directions. Thus, the method should be expanded to translate vertices in an arbitrary direction. However, the direction.

In Gouraud shading, vertex normals are computed by averaging the normals of the surrounding faces[7]. It is conceivable to use the closest vertex normal as the deformation direction. However, the angle of the finger would not have any effect on the deformation.

A different approach is to use the direction of the distal bone as the direction of deformation. This means the angle of the finger relative to the mesh will affect the deformation and therefore gives greater control when modeling.

First, the deformation direction is computed by subtracting the distal tip(or tail) from the distal joint position(head). The direction vector is normalized. To be able to evaluate the BND relative to the deformation direction, an orthonormal basis is constructed with the deformation direction as one of the orthogonal basis vectors. The basis vectors are given below.

$$\hat{e}_3 = \frac{\vec{p}_{tail} - \vec{p}_{head}}{\|\vec{p}_{tail} - \vec{p}_{head}\|}$$

$$\hat{e}_2 = \hat{e}_3^{\perp}$$

$$\hat{e}_1 = \hat{e}_2 \times \hat{e}_3$$

These basis vectors are used to create an orthogonal matrix $\mathbf{M}$

$$\mathbf{M} = \begin{bmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \hat{e}_3 \end{bmatrix}$$

Then, each mesh vertex $\hat{p}_{mesh}$ within the sphere is transformed to the "fingertip space" by translating its origin and left-multiplying the orthogonal matrix, yielding a vector $\vec{p}_{loc}$ in the local fingertip reference frame.

$$\vec{p}_{loc} = \mathbf{M}(\vec{p}_{mesh} - \vec{p}_{tail})$$

Then, $\vec{p}_{loc}$ is translated by the BND. The third component of the local mesh vertex is the translation in the deformation direction, and thus the first and second components are used as the x and y arguments of the BND respectively.

The translated local mesh vertex is then transformed to world coordinates by left-multiplying the inverse of the orthogonal matrix and translating the origin . Note that the inverse of an orthogonal matrix is also its transpose.

$$\mathbf{M^T}\vec{p}_{loc} + \vec{p}_{tail} = \mathbf{I}(\vec{p}_{mesh} - \vec{p}_{tail}) + \vec{p}_{tail} = \vec{p}_{mesh}$$

Figure 12 shows the resulting mesh deformation from applying this method.

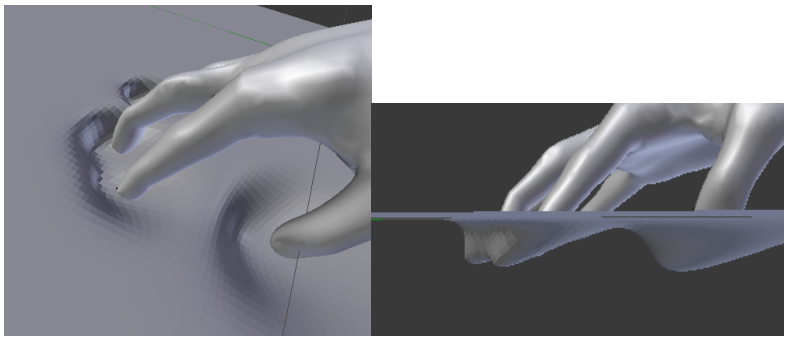

Figure 12: Directional BND deformation

Compare Figure 12 and Figure 11, especially the side views, and notice the difference in directionality of the deformation.

The selection of standard deviations have an impact on the deformation, and should be selected according to the users wishes. For example, a slider panel could be available to select the standard deviation.

10 Results

The main goals as presented in Chapter 1.2 were reached.

The literature review provided the foundation for the methods and models which were built upon and developed. A data glove and the necessary circuitry was designed, built and tested. Furthermore, a software driver for communicating with the data glove was implemented. A modern 3D modeling application was extended to be able to use the data glove for modeling and a mesh deformation method was developed. The architecture developed and shown in Figure 3 demonstrates the system.

The nature of the data glove, hand representation and mesh deformation is motion, and can only be demonstrated fully by observing the running of the system. The following screenshots and photos show snapshots of operation.

The prototype data glove and circuit as well as program code are included in the thesis submission.

10.1 Data Glove and Hand Representation

The prototype glove demonstrates the feasibility of a low cost data glove. The design schematics can be built upon to create a refined data glove with higher sensor density.

The design and implementation of the data glove provides a contribution to the academic research of creating a data glove.

Figure 4 shows that the skeletal hand representation takes on the same articulation as the physical hand and data glove.

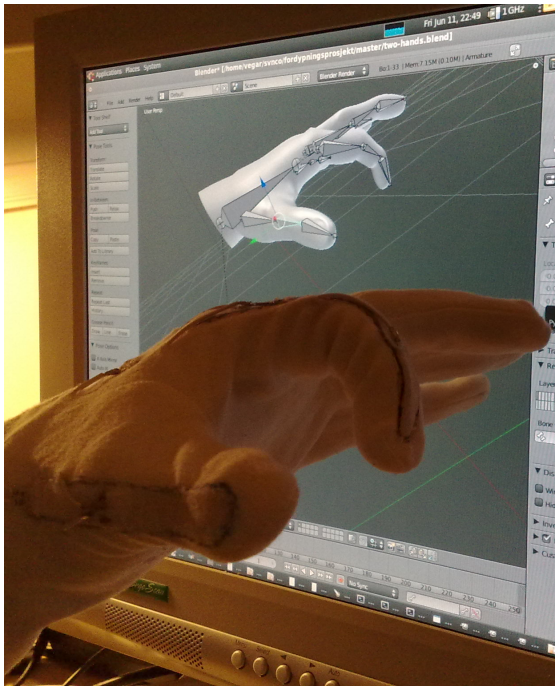

Figure 13: Data Glove and Blender

Figure 14 is an illustration of a user wearing the prototype glove on the left hand and the CyberGlove on the right hand.

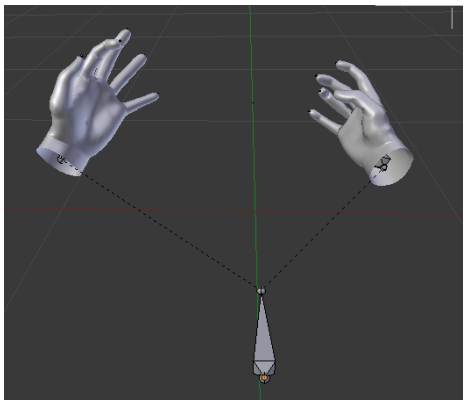

Figure 14: Two-Handed Operation

10.2 Polygon Mesh Deformation

The discussion in 9.1 show screenshots of two forms of mesh deformation using the data glove. Compare Figure 11 and 12.

11 Discussion

11.1 Data Glove

The Data Glove prototype proved functional. It is completely realistic to expand upon the design to create a glove with full sensor coverage. The cost of such a glove will be a fraction of current similar commercially available products.

The fabric used in the glove is cotton. Cotton feels nice to wear, but it has a tendency to stretch over time. Since the accuracy of the sensors depend on tight fit, a different material should be used. For example, cotton with a certain content of a stretchable component, such as Lycra.

To expand the sensor coverage, a simple external multiplexer circuit controlled by the microcontroller easily increases the number of ADC channels.

11.2 Hand Representation

The skeletal hand representation worked very well and deformed the hand mesh in a visually pleasing manner. Some artifacts can be seen however a more

thorough vertex paint would likely solve this. The hand mesh is not textured and looks somewhat like a plastic toy. Since the hand mesh is only meant to provide a guide for the user to see where the hand is, and not provide a photo-realistic hand simulation, this is sufficient.

11.3 Polygon Mesh Deformation

The polygon mesh deformation methods can be said to stretch the surface to cover a larger area. Because the vertex resolution is constant throughout, excessive stretching can cause blocky and displeasing results. This can be resolved by further subdivisioning of the mesh. However, subdivisioning the entire mesh is very inefficient as it introduces much more vertices than necessary. There are some efforts made in adaptive sub divisioning which increase local resolution, however little literature can be found on the subject.

11.4 Further Work

The Python interpreter causes quite a bit of overhead. Reading two datagloves and a tracker continuously at a high sample rate consumes the CPU. Combined with interactive mesh deformation scripted in Python, the Blender UI can become unresponsive at times. It could be possible to run VRPN inside Blenders mainloop, making the data available through Blenders data API without having python explicitly calling the VRPN mainloop or populating lists of data. Taking this idea further, Blender should support the use of data gloves, trackers and haptic devices through a library like HAPI.

12 Conclusion

This work provides a platform for human hand interaction in three-dimensional modeling. This platform consists of a theoretical foundation and methods, as well as a hardware data glove design. Suitable open source software packages are recommended and extended.

References

[1] Atmel. 8-bit Microcontroller with 8K Bytes In-System Programmable Flash Atmega48-88-168. http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf, Accessed 04/04/2010.

[2] G. Dewaele and M.P. Cani. Virtual clay for direct hand manipulation. *Eurographics (short papers)*, 2004.

[3] Eagle CAD. http://www.cadsoft.de/.

[4] M. Fischer, P. Van der Smagt, and G. Hirzinger. Learning techniques in a dataglove based telemanipulation system for the DLR hand. In *IEEE International Conference on Robotics and Automation*, pages 1603–1608. Citeseer, 1998.

[5] FlexPoint. Electronic Design Guide. http://www.flexpoint.com/technicalDataSheets/electronicDesignGuide.pdf.

[6] W.T. Freeman and M. Roth. Orientation histograms for hand gesture recognition. In *International Workshop on Automatic Face and Gesture Recognition*, volume 12. Citeseer, 1995.

[7] H. Gouraud. Continuous shading of curved surfaces. *IEEE transactions on computers*, 20(6):623–628, 1971.

[8] W.B. Griffin, R.P. Findley, M.L. Turner, and M.R. Cutkosky. Calibration and mapping of a human hand for dexterous telemanipulation. In *ASME IMECE 2000 Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems*, pages 1–8, 2000.

[9] Harald Vistnes. A physics based approach to modeling geological structures using haptics. Master's thesis, Norwegian University of Technology and Science, Department of Information and Computer Science, 7 June 2004.

[10] J. Hong and X. Tan. Calibrating a VPL DataGlove for teleoperating the Utah/MIT hand. In *1989 IEEE International Conference on Robotics and Automation, 1989. Proceedings.*, pages 1752–1757, 1989.

[11] MD Ian C Marrero. Hand, Anatomy. 9 December 2007. http://emedicine.medscape.com/article/1285060-overview , accessed 12/09/2009.

[12] J. Eric Townsen. Mattel PowerGlove FAQ. http://mellottsvrpage.com/glove.htm, Accessed 05/28/2010.

[13] F. Kahlesz, G. Zachmann, and R. Klein. Visual-fidelity dataglove calibration. In *CGI*, volume 4, pages 403–410. Citeseer.

[14] Kazuya G. Kobayashi and Katsutoshi Ootsubo. t-ffd: free-form deformation by using triangular mesh. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 226–234, New York, NY, USA, 2003. ACM.

[15] Paul G. Kry, Doug L. James, and Dinesh K. Pai. Eigenskin: real time large deformation character skinning in hardware. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 153–159, New York, NY, USA, 2002. ACM.

[16] C. Lee and Y. Xu. Online, interactive learning of gestures for human/robot interfaces. In *IEEE International Conference on Robotics and Automation*, pages 2982–2987. Citeseer, 1996.

[17] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[18] T. Massie. A tangible goal for 3D modeling. *IEEE Computer Graphics and Applications*, 18(3):62–65, 1998.

[19] Kevin T. McDonnell, Hong Qin, and Robert A. Wlodarczyk. Virtual clay: a real-time sculpting system with haptic toolkits. In *I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 179–190, New York, NY, USA, 2001. ACM.

[20] R. Ott, V. De Perrot, D. Thalmann, and F. Vexo. MHaptic: a Haptic Manipulation Library for Generic Virtual Environments. In *Proceedings of the 2007 International Conference on Cyberworlds*, pages 338–345. Citeseer, 2007.

[21] V.I. Pavlovic, R. Sharma, and T.S. Huang. Visual interpretation of hand gestures for human-computer interaction: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 19(7):677–695, 1997.

[22] Adam Seeger Russell M. Taylor II, Thomas C. Hudson. VRPN: A Device-Independent, Network-Transparent VR Peripheral System. *Proceedings of the ACM Symposium on Virtual Reality Software & Technology*, 15 November 2001.

[23] Vegar Neshaug. Virtual Reality and Human Hand Haptic Methods.

[24] J. Weber. Run-time skin deformation. In *Proceedings of Game Developers Conference*, 2000.

[25] Y. Wu and T. Huang. Vision-based gesture recognition: A review. *Gesture-based communication in human-computer interaction*, pages 103–115, 1999.

[26] Y. Wu, J.Y. Lin, and T.S. Huang. Capturing natural hand articulation. In *International Conference on Computer Vision*, pages 426–432. Citeseer, 2001.

[27] Y. Zhuang and J. Canny. Haptic interaction with global deformations. *University of California, Berkeley*, 94720, 1776.

A Guide for using Data Glove with VRPN

To be able to use the data glove in VRPN, the server must be compiled to support the data glove. Provided in the accompanied CD is the vrpn_CyberGlove driver. The driver supports both the CyberGlove and the open data glove design described in this thesis. If VRPN has not released a newer version than the one bundled on the CD, you can just use the source tree on the CD. If you use the Linux operating system running on an x86 processor architecture, the binaries are already compiled. For compiling VRPN, refer to the VRPN website for the steps involved.

The patched source tree on the CD not only contains the vrpn_CyberGlove driver, it also has a patched vrpn_Generic_Server_Object to be able to use the vrpn_CyberGlove driver by configuring vrpn.cfg. Appending the following line to vrpn.cfg will make vrpn look for a CyberGlove on the ttyUSB0 serial

device, and an Open Data Glove on ttyUSB1. Also, it will look for a FoB tracker on ttyUSB2. If you have two wrist-mounted trackers connected to your FoB, both will be available through the same Tracker0 resource. Open Data Glove supports the same protocol as CyberGlove, so interchanging the serial devices for the two gloves should not be problematic:

vrpn_CyberGlove CyberGlove /dev/ttyUSB0 115200

vrpn_CyberGlove OpenGlove /dev/ttyUSB1 115200

vrpn_Tracker_Flock Tracker0 2 /dev/ttyUSB2 115200 1 n

B Guide for Blender Python Scripting with VRPN

VRPN is a C/C++ framework. To be able to use VRPN and a data glove in Blender Python scripts, there must be generated python wrappers for the C/C++ interfaces. Luckily, there is a python_vrpn project directory bundled with VRPN. However, they do not compile cleanly with newer SWIG distributions and newer Python versions. In addition, the vrpn_Analog is just a conversion hack providing access to four analog channels. Since both the Cyberglove and Open Data Glove requires more several channels this conversion routine must be changed to support several analog channels.

On the accompanied CD, python_vrpn is already patched to compile for SWIG 1.3.40 and Python 3.1. The analog channel conversion routines are also updated.

C Examples of Mesh Deformation using Blender Python API

Using the blender Blender Python API to access mesh data is easy. The following code show mesh deformation for the BND deformation method discussed in the thesis.

```
import math
from mathutils import Vector
from mathutils import Matrix
import bpy
```

```
def run_deform():
    gd = deform.GaussianDeform()
    gd.deform()

def gaussian(x, mu, sigma):
        return (1.0/math.sqrt(2.0*math.pi*sigma**2))*math.exp(-((x-mu)**2)/(2.0*

def gaussian2d(x, y, mux, muy, sigmax, sigmay):
        return (1.0/math.sqrt(2.0*math.pi*sigmax*sigmay))*math.exp(-((x-mux)**2)

def gaussian2dmod(x, y, mux, muy, sigmax, sigmay):
        return math.exp(-((x-mux)**2)/(2.0*sigmax**2) - ((y-muy)**2/(2.0*sigmay*

class GaussianDeform:
    def __init__(self, key="GridMesh"):
        self.key = key
        self.radius = 0.4

    def deform(self):
        mesh = bpy.data.objects[self.key]
        bones = bpy.data.objects["Armature"].pose.bones
        bonematrix = bpy.data.objects["Armature"].matrix
        fingertips = ["Bone_R.thumb.distal", "Bone_R.index.distal", "Bone_R.midd
        fingertips = ["Bone_R.index.distal"]
        for tip in fingertips:
            worldtail = bonematrix*bones[tip].tail
            worldhead = bonematrix*bones[tip].head
            normal = worldtail - worldhead
            normal /= normal.magnitude
            self.simpleDeform(worldtail, normal, mesh)

    def meshVertexLocalToWorld(mesh, i):
        return mesh.matrix * mesh.data.verts[i].co

    def simpleDeform(self, fingertip,normal,mesh):
        for vert in mesh.data.verts:
```

```
            vertworld = mesh.matrix * vert.co
            diff = vertworld - fingertip
            if diff.magnitude < self.radius:
                repel = 0.1*gaussian2dmod(vertworld.x,vertworld.y, fingertip.x,
                if vert.co.z > -repel:
                    vert.co.z = -repel

    def directionalDeform(self, fingertip, normal, mesh):
        perp = Vector((-normal.y, normal.x, 0))
        perp2 = perp.cross(normal)
        tr = Matrix( (perp.x, perp.y, perp.z),
                     (perp2.x, perp2.y, perp2.z),
                     (normal.x, normal.y, normal.z))
        inv = tr.copy()
        inv.invert()
        meshinv = mesh.matrix.copy()
        meshinv.invert()
        for vert in mesh.data.verts:
            vertworld = mesh.matrix * vert.co
            diff = vertworld - fingertip
            if diff.magnitude < self.radius:
                moved = tr*diff
                moved.z += 0.1*gaussian2dmod(moved.x, moved.y, 0.0, 0.0, 0.025,
                b = inv*moved + fingertip
                vert.co = meshinv*b
```

D Data Glove and Computer Interface Circuit

D.1 Glove Design

The FlexPoint bend sensor is the fundamental component of the data glove. These bend sensors are variable resistors, or potentiometers.

> The Bend Sensor® potentiometer is a product consisting of a coated substrate such as plastic that changes in electrical conductivity as it is bent.(Bend Sensor Technology Electronic Interface Guide)

D.2 Circuit Design

Electrical conductivity is an analog signal, meaning it is continuous and variable. To be able to use the bend sensors in a digital computer, this signal must be converted to discrete values. This digitization can be achieved through the use of an electronic device called an Analog-to-Digital Converter(ADC). When the analog signal is sampled it must be transmitted to the computer to be able to use in an application.

The Atmega168 microcontroller has integrated voltage signal ADC and serial transmission circuitry(USART). The Atmega168 is a member of a family of microcontrollers produced by Atmel called AVR. The selection of the Atmega168 was based on the large amount of online documentation, as well as being one of the few PDIP microcontroller available at the hardware lab at the time.

Since the Atmega168 ADC samples voltage signals, the resistivity of the bend sensor must be converted to a voltage signal. This can be done using a circuit known as a voltage divider.

By Ohm's law the following relationships can be found:

$V_{in} = I \cdot (R_1 + R_2)$

$V_{out} = I \cdot R_2$

Solving for I in equation 1 and substituting into equation 2 yields:

$V_{out} = \frac{V_{in}}{R_1+R_2} \cdot R_2 = V_{in} \cdot \frac{R_2}{R_1+R_2}$

Assigning a constant to $R_1$and letting R_2be the bend sensor potentiometer results in a variable voltage V_{out}. This will be the input voltage to the ADC:

$V_{out}(R_{bend}) = V_{in} \cdot \frac{R_{bend}}{R_1+R_{bend}}$

This equation assumes no current flows through the V_{out} node. Some current will flow as the ADC capacitor charges up in the sample-and-hold circuitry.

D.3 Glove and Bend Sensors

The glove itself was made of two pairs of thinly woven gloves. One left handed glove was inserted into another making a layered glove. Pockets were made by stitching the gloves together over the index finger, thumb, palm and wrist. Openings were made for inserting the sensors. Several openings were made over

the joints of the index finger and thumb. The bend sensors were glued into place at the opening. Wires were led inside the layered gloves, coming out slightly above the bend sensor and clamped into the wire clamps on the bend sensors. The two wire endpoints of each sensor was inserted into a wire connector for easily connecting them to the header pins of the circuit.

D.4 Circuit Prototyping

Creating an initial prototype for testing the basic functionality of the data glove circuit was done through the use of a prototyping board. Such a board has holes in which components and wires are pushed in and requires no soldering. The holes on the left and right side rails are vertically connected, allowing convenient access to power and ground. The rest of the board is horizontally connected, broken by the space in the middle. This allows for easy prototyping of designs based on DIP(Dual In-line-Package) devices.

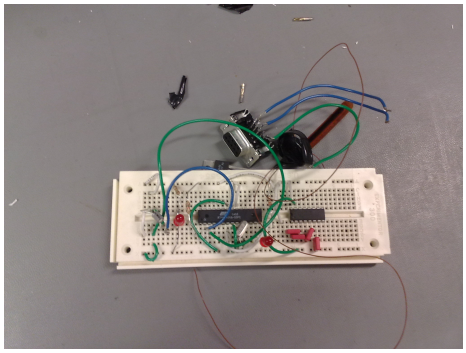

Figure 15: Prototype Board

The components were laid out according to the schematic and power and ground were hooked up to a lab-grade power supply, which gives a very stable voltage and shows the amount of current used.

In the prototype board, the PIN headers and connections for ISP(In-System Programming) were not included. The microcontroller program was therefore written to the Atmega168 while mounted on the STK500, and subsequently inserted it in the prototype board for testing.

D.5 Printed Circuit Board

A prototype board is useful for quickly testing a schematic, however it is not very sturdy. Wires can easily come off since they are not soldered. A printed circuit board is a board where the components are soldered on copper lines or traces. All the wires seen on the prototyping board, except external power and serial transmission wires are replaced by copper traces. In order to create a printed circuit board, the design schematic must be traced out into a trace schematic. This functionality is usually included in the electronics CAD software package.

The printed circuit board was traced out using the Eagle electronics CAD software. First, the design schematic is loaded into the board editor. Each component is then carefully laid out such that the traces are as short as possible. For example, the pin headers for the bend sensors are placed right next to the ADC input pins. The capacitors are placed as close as possible to the pins of the microcontroller as recommended by the datasheet. For a hobbyist it is desirable to use one plane of traces as it simplifies the process of producing the PCB.

When the components are first laid out, all the connections are air-wires, meaning no traces are laid out. It is possible to let Eagle attempt to auto-route the traces, however it did not produce a complete trace. Manually tracing the connections is an intricate process of laying out traces, ripping up traces, repositioning components and repeating.

Note that the ISP header is included in the PCB, which allows for re-programming the microcontroller. This is very useful as extracting and re-inserting the microcontroller in the circuit and the STK500 development board is time-consuming and may bend the pins of the PDIP.

With all components laid out and connections traced, the trace schematic is printed on semi-transparent paper using a high quality laser-printer. A piece of photo-resist PCB is cut out to fit the schematic. The schematic is then placed on top of the PCB and exposed to Ultra-Violet light. It is important that the printed schematic and the PCB are held together tightly. The UV exposure unit had a cushion which pressed the PCB and printed schematic against a glass pane. After exposure, the PCB is placed in a small tank with developer. The developer dissolves the parts of the photo-resist layer exposed to UV, thereby exposing the underlying copper. Then the PCB is quickly rinsed and put in a tank with heated circulating etchant. Slowly, the exposed copper will etch away leaving the traces behind. It is important to observe the process and remove

the PCB before the traces begin to deteriorate. When the etching is complete, the mounting holes are drilled using a drill press. The drill bit is aligned with the mounting hole using a sight on the PCB drill press.

D.6 Microcontroller program

The program for the Atmega168 was written in the C/C++ language(s), although it is possible to do it in assembly. There are several guides, tutorials and examples available online through websites like avrfreaks.net which were very helpful for writing the microcontroller program. Atmel also include several examples with their development boards. Although similar, the range of AVR/Atmega microcontrollers are not identical. The datasheet for the actual must be used to determine which modes of operation and registers apply to the actual microcontroller, in this case the Atmega168.

There are two main tasks which the Atmega168 must perform. It should sample it's ADC(Analog-Digital Converter) input ports as fast and accurate as possible in a sequential manner. When all ADC channels are sampled it should serialize the results for transmission using the USART(Universal Synchronous-Asynchronous Receiver/Transmitter).

ADC The ADC uses a sample-and-hold circuit to sample one of six analog signal, V_{in}, channels selected by a multiplexer. The analog signal is converted to a digital value, denoted ADC, relative to a reference voltage V_{ref} according to the equation

$$ADC = \frac{V_{in}}{V_{ref}} \cdot 1024$$

Using the ADC involves assigning binary values to the ADMUX(ADC Multiplexer Select) and ADCSR(ADC Control and Status Register) registers. Given here are tables showing which settings are chosen for the registers. Rationale and explanation is given below the table. An exhaustive list of register uses can be found in the datasheet.

Bit	Name	Setting	Purpose
0	ADPS0	1	ADC Prescaler Select. Setting the division factor relative to the system clock, called prescaling. This particular setting has a decimal value of 128_{10}
1	ADPS1	1	
2	ADPS2	1	
3	ADIE	1	ADC Interrupt Enable. Enabling/Disabling interrupt signal on the completion of a conversion.
4	ADIF	x	ADC Interrupt Flag. Set by the ADC itself when an ADC conversion is complete. Cleared by hardware when the interrupt routine is executed.
5	ADATE	0	ADC Auto Trigger Enable. Setting this bit to 1 causes the ADC to start a conversion automatically by a trigger source.
6	ADSC	x	ADC Start Conversion. Setting this bit to 1 causes the ADC to start a conversion. It is cleared by hardware when the conversion is complete.
7	ADEN	1	ADC Enable. Enables/disables ADC.

The ADC prescaler is set to a divison factor of 128. The datasheet states that for high resolution(10-bit) conversion, the ADC input should be in the range from 50Khz to 200Khz. Since the system clock crystal in the glove circuit oscillates at a frequency of 18.432Mhz, *a division factor of 128* is used which gives an ADC clock of *144Khz*.

The ADC can be set up in two primary modes of operation, given by the setting of the ADATE(Auto Trigger Enable) register bit in ADCSRA. This determines how an ADC conversion is started(Datasheet figure 23-2).

Starting a single conversion is done by setting the ADSC(ADC Start Conversion) bit. When the conversion is complete, ADSC will be cleared by hardware. The ADC will not start another conversion until the ADSC bit is again set. Depending on different clock rates, this will likey cause it to waste some ADC

clock cycles before the CPU is able to set the ADSC bit.

The alternative to starting a single conversion is for the ADC to auto trigger a conversion on a trigger source, for example on its own interrupt flag. When ADATE is set, a conversion is started by an ADTS(ADC Trigger Source). When the ADC interrupt flag is set as the trigger source, the ADC will trigger a new conversion based on its own interrupt and hence continuously sample the selected ADC channel. The ADC is then said to be in Free-Running mode.

Free-Running mode has the benefit of the ADC starting another conversion immediately, not wasting any ADC clock cycles. To read the ADC channels in a sequential manner in Free-Running mode, some form of timing must be employed by the CPU to update the channel in ADMUX after the current conversion has started but before the ADC completes the conversion. Otherwise, the ADC will start a new conversion using the same channel. This can be prone to errors through race conditions if the CPU also has other tasks or interrupts to execute.

Furthermore, for reduced noise(EMI) while converting, the Atmega168 can perform the ADC conversion during sleep mode. This means the ADC clock is kept alive while the CPU, I/O and flash clocks are stopped. An ADC conversion will start once the CPU has halted. When the conversion is complete, the ADC interrupt will wake up the CPU and begin execution of the ADC interrupt routine. The ADC will start a new conversion upon re-entering sleep mode. To be able to use this feature, the ADC must be in single conversion mode. This noise reduction feature is desirable even if some ADC clock cycles may be wasted. Particularly if the full 10 bits of resolution are used.

In summary, this setting of the ADSCR is suitable for high-resolution conversion. By using a high clock prescaling division factor, and using single conversion with sleep mode, the accuracy of the ADC is maximized.

The ADMUX register is used to

Bit	Name	Setting	Purpose
0	MUX0	x	
1	MUX1	x	Analog Channel Selection bits. Set which analog input pin is is connected to the ADC.
2	MUX2	x	
3	MUX3	x	
4	-		
5	ADLAR	0	ADC Left Adjust Result. Whether the result should be left adjusted in the case that 8-bit resolution is sufficient.
6	REFS0	1	Reference Selection Bits. This assignment sets the reference voltage Vref to AVcc(5v)
7	REFS1	0	

The MUX bits are iterated from 0000_2(decimal 0) to 0101_2(decimal 5) sequentially, starting a conversion on each iteration and storing the values in an array.

The ADLAR bit is not set since full 10-bit resolution is desired.

The circuit schematic is designed for using AVcc(5v) as the reference voltage and the reference voltage bits are set accordingly.

When a conversion is complete, the sampled value is available in the ADCH(ADC High) and ADCL(ADC Low) registers. The result is split over two registers because the Atmega168 is an 8-bit microcontroller. These registers are concatenated in the microcontroller program and stored in an indexed array for later serialization over the USART.

USART The USART(Universal Synchronous and Asynchronous Receiver and Transmitter) is a complex piece of circuitry. When set up correctly, it is quite easy to use. Sending a byte(or word) consists of simply loading a register. However, to set the USART up correctly it is useful to know how the transmission process works(Maxim Clock accuracy paper). Considered here is the USART operating in asynchronous mode(UART).

For a serial link to be set up between two DTE(Data Terminal Equipment) units, they must agree on how a word should be transmitted. The discussion presented here is somewhat simplified.

First, the clocks of both receiver and transmitter must be the same. Otherwise, the receiver would not be able sample the signal at the pace it is being generated. In the case of a too fast receiver it would be sampling the same signal multiple times(e.g two 0's when there is only one), or in the case of a too slow receiver it would skip a signal(bit) entirely. Thus, the receiver and transmitter must agree on the number of bits per second, or baud rate. Note that the term baud rate refers to the media bit rate including the overhead of the start and stop bits, not the data rate delivered to the application.

Second, they must agree on the number of bits(N) in one message. This is because of the binary nature(two voltage levels) of the communication. Since there is no third value to indicate no communication, the receiver must know when a signal starts and when it stops. One of the voltage levels are defined as a stop bit and the other voltage level is defined as a start bit. When nothing is transmitted, the signal is a continuous stop bit. As soon as the receiver detects a transition to a start bit, it resynchronizes its clock so that not only the rate but also the sample time coincides with the transmitter. Then, the receiver samples the signals (at the baud rate) N+1 times, where the last time sample is the stop bit. If it detects a start bit instead of a stop bit on the last sample, a framing error has occurred. The N data bits are loaded into the receive register for the microcontroller program to read. For the transmitter, the operation is much easier. It simply signals the start bit, the N bits of the message, and a stop bit. It is possible to use parity bits for simple error checking. The use of parity bits is not considered here, as the crystal and baud rate is selected for 0.0% error.

Initializing the USART correctly involves setting up the registers with the appropriate baud rate and frame format and enabling the receive and transmit circuitry. The default frame format of the microcontroller is 8 data bits and 1 stop bit, which is a very common setting and is thus left unchanged. The baud rate(and crystal) should be selected such that there is no remainder when the clock is divided. In the circuit schematic, the clock generator(crystal) has the following value

$f_{osc} = 18432000hz.$

In normal asynchronous mode, the signal is actually sampled at 16 times the baud rate to be able to detect the transition from the stop bit to the start bit for receiving a frame. Samples 8, 9 and 10 are used by the synchronization logic

to verify that a valid start bit has been detected and not a noise spike. Thus, the clock divider(d_{clock}) should be selected according to the equation:

$16 \cdot BAUD = \frac{f_{osc}}{d_{clock}}$

In the Atmega168, the clock divider is set by assigning a value to the UBRR(USART Baud Rate Register), which causes the the clock divider to be set according to the equation

$d_{clock} = UBRR + 1$

Substituting and rearranging the baud rate equation gives an UBRR value of:

$UBRR = \frac{f_{osc}}{16 \cdot BAUD} - 1$

The register is used for integer binary counting to generate the UART clock, and this is why there can be no remainder. Consider selecting a baud rate of $BAUD = 100000$. Solving for UBRR would yield a non-integer value:

$UBRR = \frac{18432000}{16(100000)} - 1 \approx 10.52 \approx 11$

Substituting the rounded integer value into the baud rate equation would give an actual baud rate of:

$BAUD = \frac{18432000}{16(11+1)} = 96000$

This is an error of 4.2% and undesirable. Referring to the discussion earlier in this paragraph, this would give the problem of different baud rates between transmitter and receiver. Now, it is possible to select the baud rate of 96000 instead of 100000, since this would give an error of 0.0%. However, a more standard baud rate of 115200 (incidentally the baud rate setting of the CyberGlove) also gives an error of 0.0%. This is the reason why a crystal of such a seemingly peculiar oscillation rate of 18.432Mhz was selected during the design.

Thus, when the desired baud rate is $BAUD = 115200$ the UBRR register should be given the value of:

$UBRR = \frac{18432000}{16(115200)} - 1 = 9$

Note that there is no remainder. In mathematical terms, the baud rate is selected such that the numerator in the equation is a divisor of the clock generator.

Enabling the transmit and receive circuitry is done by setting the TXEN(Transmitter Enable) and RXEN(Receiver Enable) bits of the UCSRB(USART Control and Status Register B). Similar to the ADC, the USART can operate by using interrupt signalling a received word or a transmitted word. However, as stated

in the paragraph on programming the ADC it is desirable to stop non-ADC clocks while converting for noise reduction. Thus, it is undesirable to allow the USART circuitry to wake the CPU before the ADC has finished a conversion. Therefore, the USART will not be running in interrupt mode. Instead, after the ADC has finished sequentially sampling all its ADC channels, the USART will serialize and transmit the converted values in a tight loop, continuously polling the TXC(Transmit Complete) bit for each word. After the digital values of all six ADC channels have been transmitted, all non-ADC clocks are again shut down for another sequential reading of all ADC channels.

USART reception will only occur in the beginning of the program, while it is waiting for the reception of commands in an ASCII format. Thus, the USART reception circuitry is not enabled when the microcontroller is sampling and transmitting.

Summary Summary and conceptualization the operation of the microcontroller program.

The IDLE state does the following

1. Enables and sets up the the USART circuitry(i.e baud rate and and RXEN/TXEN)
2. Continously polls the USART reception registers for the ASCII commands "?n" and "s".
   (a) Upon receiving the "?n" command, the microcontroller program transmits "?n 6\r\n\0x00" which tells the other party how many analog values will be reported.
   (b) Upon receiving the "s" command, the microcontroller program transitions to the ADC INIT state.

The ADC INITIALIZE state does the following

1. Enables global interrupts(Must do this to allow any form of interrupts to be processed)
2. Sets the sleep mode registers to allow shutting down non-ADC clocks while converting.

3. Initializes the ADC(i.e enabling the adc, setting the prescaler, reference voltage and enabling ADC interrupts)

4. Transitions to the SAMPLE state

The SAMPLE state does the following

1. Iteratively samples the ADC channels, shutting down non-ADC clocks before each conversion. Each completed conversion triggers an interrupt which stores the result in an indexed array.

2. When all ADC channels are sampled, the state transitions to the TRANSMIT state.

The Transmit state does the following

1. Transmits all ADC channels over the USART in the format: "s <ADC0> <ADC1> <ADC2> <ADC3> <ADC4> <ADC5>\r\n\0x00". Each <ADCx> value is substituted by its respective digital ADC channel value and is transmitted in a four-character width ASCII format. It could be argued that this is an inefficient encoding, however it is selected for interoperability with the CyberGlove such that the same device driver can be used for both gloves.

2. Transitions to the SAMPLE state

Note that there is no sink in the state diagram, thus the device is reset to the IDLE state by either power cycling or pulling the RESET line low.

D.7 VRPN Device Driver

One of the major reasons VRPN was selected in the design was to use the same device layer for all the devices in the application. This reduces the number of libraries which must be used(and learned). The tracker used in the application is a FOB(Flock-of-Birds) tracker and a VRPN device driver for it is already available. The data glove implementation as well as the CyberGlove do not have a VRPN device driver. Thus, a VRPN device driver to support these gloves must be implemented. This is the reason why the messaging protocol of

the data glove was made interoperable with the CyberGlove. Only one VRPN device driver needs to be implemented, regardless of which glove is used.

VRPN is an object-oriented library written in the C/C++ programming language(s). Several base-classes are available for the purpose of being extended(in OO terms) to support new hardware. For example, the Flock-of-Birds VRPN device driver is an extension of the vrpn_Tracker base class. This allows the vrpn client or the application to use a tracker as an abstract interface describing a class(both in OO terms and general terms) of devices. Thus, replacing the actual tracker with a different VRPN-supported tracker would require few if any changes to the application.

It is possible to implement new VRPN device drivers by defining an entirely new class of devices. There is no "data glove" device class in the VRPN library. However, there is a VRPN device class for analog devices, called vrpn_Analog. This class of devices operate by sending an array of floating point numbers, called channels, to the client. Although the values transmitted by the data gloves are integers, this device class is similar enough to base the VRPN data glove implementation on.

There is little point in translating the implementation code into english. However, a short discussion showing the simplicity of extending VRPN is appropriate.

The implementation of the data glove VRPN device is named vrpn_CyberGlove. This name reflects the serial messaging protocol supported by the device driver. The vrpn_CyberGlove device class extends vrpn_Analog. This allows the client to use the original VRPN client library without needing to compile in the vrpn_CyberGlove source code. However, if the client will also be running the server, the vrpn_CyberGlove source code must be included in the library.

The vrpn_CyberGlove device class overrides the vrpn_Analog constructor. The overriding constructor establishes connection with the serial communication port by using convenience functions provided by the VRPN library(vrpn_Serial.h). The device driver then transmits the "?n" message to query the number of analog channels reported. Subsequently, the reply message from the device is used to set the number of vrpn analog channels which are available. Finally, the "s" message is sent to the device in order to start the continuous transmission of converted analog values.

After the vrpn_CyberGlove object is constructed, the generic VRPN server will

continuously execute the mainloop which is defined in the vrpn_CyberGlove device class. This mainloop reads the bytes from the serial port, parsing the ASCII representation of the ADC values. Then, control is transferred to the VRPN library to report any changes to the channels.

The discussion above shows how simple it is to extend the VRPN library to support new devices.

The client application creates a vrpn_Analog_Remote object with the appropriate arguments to create a connection with the VRPN server. The client application must execute the VRPN client mainloop in its own mainloop, and is notified of any changes through callback functions.

E Microcontroller and VRPN Data Glove Code

vrpn_CyberGlove.C and vrpn_CyberGlove.h together with the microcontroller code can be found in the attached compressed file.

E.1 VRPN Program Code

E.1.1 VRPN Header File

```
#ifndef VRPN_CYBERGLOVE_H
#define VRPN_CYBERGLOVE_H

#include "vrpn_Connection.h"
#include "vrpn_Analog.h"

/*
0.   thumb rotation/TMJ (angle of thumb rotating across palm)
1.   thumb MPJ (joint where the thumb meets the palm)
2.   thumb IJ (outer thumb joint)
3.   thumb abduction (angle between thumb and index finger)
4.   index MPJ (joint where the index meets the palm)
5.   index PIJ (joint second from finger tip)
6.*  index DIJ (joint closest to finger tip)
7.** index abduction (sideways motion of index finger)
```

```
8.   middle MPJ
9.   middle PIJ
10.* middle DIJ
11.  middle-index abd'n (angle between middle and index fingers)
12.  ring MPJ
13.  ring PIJ
14.* ring DIJ
15.  ring-middle abduction (angle between ring and middle fingers)
16.  pinkie MPJ
17.  pinkie PIJ
18.* pinkie DIJ
19.  pinkie-ring abduction (angle between pinkie and ring finger)
20.  palm arch (causes pinkie to rotate across palm)
21.  wrist pitch (flexion/extension)
22.  wrist yaw (abduction/adduction)
*/

class VRPN_API vrpn_CyberGlove : public vrpn_Analog {

  public:
        vrpn_CyberGlove(const char * name, vrpn_Connection * c);
    vrpn_CyberGlove(const char * name, vrpn_Connection * c,     const char * dev
        ~vrpn_CyberGlove(void);

    /// Sets the size of the array;  returns the size actually set.
    /// (May be clamped to vrpn_CHANNEL_MAX)
    /// This should be used before mainloop is ever called.
    vrpn_int32 setNumChannels (vrpn_int32 sizeRequested);

        virtual void mainloop();
protected:
        const char * _device;
        const char * _baud;
        int dev_fd;

        int             nSensors;
```

```
        static const int buf_size = 256;
    unsigned char        buffer[256];
    unsigned char str[20];
        int              value[22];

        struct timeval timestamp;

                /// send report iff changed
        virtual void report_changes
                    (vrpn_uint32 class_of_service = vrpn_CONNECTION_LOW_LATENCY);
        /// send report whether or not changed
        virtual void report
                    (vrpn_uint32 class_of_service = vrpn_CONNECTION_LOW_LATENCY);

    void     UpdateData();
        void     Wait(double t);


};

#endif
```

E.1.2 VRPN C Code File

```
#include "vrpn_Analog.h"
#include "vrpn_CyberGlove.h"
#include "vrpn_Connection.h"
#include "vrpn_Serial.h"

#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <string.h>


vrpn_CyberGlove::vrpn_CyberGlove(const char * name,
```

```
                                        vrpn_Connection * c,
                                        const char * p_device, const char * p_ba
        vrpn_Analog (name, c)
{
    this->dev_fd = vrpn_open_commport(p_device, atol(p_baud));
    if (this->dev_fd < 0) {
            fprintf(stderr, "vrpn_CyberGlove: Could not open %s\n", p_device);
            return;
    }
    vrpn_flush_input_buffer(this->dev_fd);
    vrpn_write_characters(this->dev_fd, (const unsigned char*)"?n", strlen("?n")

    struct timeval wait;
    wait.tv_sec = 1;
        vrpn_read_available_characters(this->dev_fd, this->buffer, 8, &wait);
        nSensors = atoi((const char*)(&buffer[3]));
    printf("CyberGlove: Channels: %d %s\n", nSensors, buffer);

        this->setNumChannels(nSensors);

        // Start ASCII stream mode
        vrpn_write_characters(this->dev_fd, (const unsigned char*)"s", strlen("s
}

vrpn_CyberGlove::~vrpn_CyberGlove(void) {
        vrpn_close_commport(this->dev_fd);
}


void vrpn_CyberGlove::report_changes (vrpn_uint32 class_of_service)
{
  vrpn_Analog::timestamp = timestamp;
  vrpn_Analog::report_changes(class_of_service);
}

void vrpn_CyberGlove::report (vrpn_uint32 class_of_service)
```

```
{
        vrpn_Analog::timestamp = timestamp;
        vrpn_Analog::report(class_of_service);
}


vrpn_int32 vrpn_CyberGlove::setNumChannels (vrpn_int32 sizeRequested) {
  if (sizeRequested < 0) sizeRequested = 0;
  if (sizeRequested > vrpn_CHANNEL_MAX) sizeRequested = vrpn_CHANNEL_MAX;
  num_channel = sizeRequested;
  return num_channel;
}


void vrpn_CyberGlove::mainloop (void) {
        //fprintf(stderr, "vrpn_Cyberglove::mainloop");
        server_mainloop();
        vrpn_gettimeofday(&timestamp, NULL);
        vrpn_read_available_characters(this->dev_fd, this->buffer, 255, NULL);
        this->buffer[255] = NULL; // Prevent reading past end
        unsigned char * ptr = this->buffer;
        unsigned char character = 0;
        int position = -1;
        int n = 0;
        while(ptr < buffer+255) {
            n = 0;
        if(character == 's') {
            position = 0;
            character = 0;
            report_changes();
        } else if(position != -1 && position < num_channel) {
            //n = sscanf((const char*)ptr, "%d", &this->value[position]);
            value[position] =  strtol((const char *)ptr, (char**)&ptr, 10);
            if(value[position] != 0)
                channel[position] = value[position];
            //channel[position] = value[position] = atoi((const char *)this->str
```

```
            position++;
        } else {
            character = *ptr;
            position = -1;
            ptr++;
        }
        //sscanf((const char *)ptr, "%s", this->str);
        }
        /*
        if(buffer[0] != 's') {
            return;
         }
        char * token = strtok((char*)buffer, " ");
        for (int i=0; i<nSensors; i++) {
            token = strtok(NULL, " ");
            if(token != NULL)
                channel[i] = value[i] = atoi(token);
        }
        */

}
```

E.2 Microcontroller Program Code

```
//#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>
#include <stdio.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#define F_OSC 18432000
#define BAUD 115200 //38400
//#define MYUBRR 9 //#define MYUBRR FOSC/16/BAUD-1
```

```
#define MYUBRR (F_OSC/16/BAUD - 1)


#define ADC_maxchannel 6
volatile unsigned ADC_recent_value[ADC_maxchannel]={0};
volatile unsigned char ADC_channel=0;

#define ADC_avg_max 10
volatile unsigned ADC_avg_value[ADC_maxchannel][ADC_avg_max];
volatile unsigned char ADC_avg_pos=0;

#define GLOVE_STATE_IDLE  0
#define GLOVE_STATE_QUERY 1
#define GLOVE_STATE_QUERY_N 2
#define GLOVE_STATE_STREAM  3
unsigned char glove_state = GLOVE_STATE_IDLE;

void USART_Init()
{

UBRR0H = (unsigned char)(MYUBRR>>8);
UBRR0L = (unsigned char)MYUBRR;

UCSR0B = (1<<RXEN0)|(1<<TXEN0);              // Rx Tx enable

UCSR0C = (3<<UCSZ00);                        // Set frame format: 8data, 1 stop bit

}



void USART_Transmit( unsigned char data )
{

while ( !( UCSR0A & (1<<UDRE0)) );
```

```
UDR0 = data;


}



unsigned char USART_Receive( void )
{

while ( !(UCSR0A & (1<<RXC0)) );


return UDR0;
}


void USART_Transmitbuf(char * data, int n) {
        for(int i = 0; i < n; i++) {
                USART_Transmit(data[i]);
        }
}

void ADC_Init() {


      //ADMUX |= (1 << REFS0) | (1 << REFS1); // 11 = 1,1V
   //ADMUX |= (1 << ADLAR); // left adjustment
   //ADCSRA |= (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); // prescaler = 128 ->
   //ADCSRA |= (1 << ADATE); // adc auto trigger enable
   //ADCSRA |= (1 << ADIE); // enable interrupt
   //ADCSRA |= (1 << ADEN); // Enables ADC
   //ADCSRA |= (1 << ADSC); // Start Conversion is enabled
//    ADCSRB = (1 << ADTS2); // default is zero -> free running mode

   ADCSRA=
```

```
        (1<<ADEN)|//converter enable
        (0<<ADSC)|//dont start conversion yet
        (0<<ADATE)|//no ext triggering
        (0<<ADIF)|//clear flag, stop pending conv
        (1<<ADIE)|// interrupt enable
        (1<<ADPS2)|
        (1<<ADPS1)|
        (1<<ADPS0);// ADC Prescaler Select Bits: Division Factor 128
    ADMUX=
        (0<<REFS1)|
        (1<<REFS0)|//AVCC
        (0<<ADLAR)|//right-aligned
        (0<<MUX3)|
        (0<<MUX2)|
        (0<<MUX1)|
        (0<<MUX0);//channel 0, single-ended

}


ISR(ADC_vect) {
    const unsigned char adcl=ADCL;
    const unsigned char adch=ADCH;
    ADC_avg_value[ADC_channel][ADC_avg_pos] =
                            ADC_recent_value[ADC_channel]=((unsigned)adch<<8) | adcl


}

void ADC_capture()
{
    unsigned char channel;
    // Capture:
    //ADC_channel = 0;
    //ADMUX = (0 << REFS1) | (1 << REFS0) | ADC_channel;
    //ADCSRA |= (1<<ADSC);
```

```
    //sleep_cpu();
    for(channel=0; channel<ADC_maxchannel; channel++)
    {
        ADC_channel=channel;
        ADMUX =
                    (0<<REFS1)|
                    (1<<REFS0)|//AVCC
                    channel;
        //ADCSRA|=(1<<ADSC)|(1<<ADIE);// Sleeping will start conversion automagica
            SMCR |= (1<<SE);
        sleep_mode();
            SMCR &= ~(1<<SE);

    }

}

int calc_chan_avg(unsigned char channel) {
        unsigned int val = 0;
        for(unsigned char i = 0; i < ADC_avg_max; i++) {
                val += ADC_avg_value[channel][i];
        }
        return val/ADC_avg_max;
}

int main (void)
{



    USART_Init();
    unsigned char rcv;
    while(glove_state != GLOVE_STATE_STREAM) {
        rcv = USART_Receive();
        switch(rcv) {
                case 'n':
```

```
                USART_Transmitbuf("?n 6\r\n", 6);
                USART_Transmit(0);
                break;
        case 's':
                glove_state = GLOVE_STATE_STREAM;
                break;
        default:
                break;
    }
}


ADC_Init();
sei();
SMCR |= (0<<SM0) | (1<<SE);
unsigned char channel;
char buffer[100];
unsigned char index = 0;
for(;;) {

        for(unsigned char pos = 0; pos < ADC_avg_max; pos++) {
                ADC_avg_pos = pos;
                ADC_capture();


                index = 0;
                buffer[index++] = 's';
                for(channel = 0; channel < ADC_maxchannel; channel++) {
                index += sprintf(buffer+index, " %4d",calc_chan_avg(chan
                }
                buffer[index++] = '\r';
                buffer[index++] = '\n';
                buffer[index++] = 0;
        USART_Transmitbuf(buffer, index);
                while ( !(UCSR0A & (1<<TXC0)) );
```

```
                    }
    }
}
```

F Data Glove History

Data gloves are not a new idea. The first data glove, the Sayre Glove was made at the Electronic Visualization Laboratory (EVL) in 1977. It was based on optical flexible tubes. The amount of light received at the other end indicated the amount of bending.

The different data gloves available differ in their sophistication based on the number of sensors, resolution and sampling rate. The number of sensors largely determine if you can isolate joint movement or just provides a total bend for the whole finger. Resolution is important when fine movement is involved. Both the resolution and sampling rate is important when the data glove is used as a basis for haptic feedback.

The Nintendo Power Glove is one early commercially available data glove made for gaming. It was produced by Mattel toy company and released in 1989 for use with the Nintendo Entertainment System[12]. The glove could detect four different degrees of bending for the total curvature of a finger. It had a rudimentary tracker based on ultrasonic microphones for spatial position and orientation. Later data gloves would be an improvement on this basic design.

The 5DT glove has similiar to the PowerGlove, a single sensor per finger. The resolution is claimed to be at around 10 bits, or 1024 discrete values, with a minimum sampling rate of 75Hz. It comes with an USB or Bluetooth interface and does not include a tracker. It costs around USD1000. A higher end version of the 5DT, the 5DT 14 Ultra has two sensors per finger and abduction sensors between the fingers with a total of 14 sensors.

In 2002, the P5 "glove" came out. It was a low cost alternative to the higher end gloves and had only one sensor per finger. An optical tracker comes with the glove. The glove is discontinued.

Virtual Technologies inc. produced the CyberGlove which was commercially available very early considering its sophistication. It was first available in the year 1990. Today many consider the CyberGlove as being the de facto standard

in high end data gloves and they are used in a number of fields like telerobotics and visualization.

A new development in data gloves is the ShapeHand. It is based on the ShapeTape technology which is based on fiber optics. The ShapeTape "knows" the position of each segment along its length when it is bent and twisted. Thus, by overlaying one of these on a finger, the bending of each joint is captured.

Data gloves are not cheap. They cost thousands of dollars and come with proprietary software development kits. This thesis will attempt to design and create a prototype data glove for an affordable alternative data glove with sophistication comparable to the CyberGlove, in the sense that it should have high resolution, one sensor per joint and high sampling rate.

G Parts List

Partlist

Exported from hanske.sch at 6/11/10 5:37 PM

EAGLE Version 5.9.0 Copyright (c) 1988-2010 CadSoft

Part	Value	Device	Package	Library	Sheet
C1		C-EU025-050X050	C025-050X050	rcl	1
C2		C-EU025-050X050	C025-050X050	rcl	1
C3		C-EU025-050X050	C025-050X050	rcl	1
C4		C-EU025-050X050	C025-050X050	rcl	1
C5		C-EU025-050X050	C025-050X050	rcl	1
IC1	ATMEGA48/88/168-PU	ATMEGA48/88/168-PU	DIL28-3	atmega8	1

JP1	SENSOR	PINHD–2X6	2X06
pinhead	1		
JP2	AREF	PINHD–1X2	1X02
pinhead	1		
JP3	FTDI	PINHD–1X4	1X04
pinhead	1		
JP4	ISP	PINHD–2X3	2X03
pinhead	1		
LED1		LEDSQR2X5	LED2X5
led	1		
Q1		CRYSTALHC18U–V	HC18U–V
crystal	1		
R1		R–EU_0204/5	0204/5
resistor	1		
R2		R–EU_0204/5	0204/5
resistor	1		
R3		R–EU_0204/5	0204/5
resistor	1		
R4		R–EU_0204/5	0204/5
resistor	1		
R5		R–EU_0204/5	0204/5
resistor	1		
R6		R–EU_0204/5	0204/5
resistor	1		
R13	10k	R–EU_0204/5	0204/5
resistor	1		
R14		R–EU_0204/5	0204/5
resistor	1		

Externals:
FTDI 3v3 Level Shifter