Affordable Input Devices for Virtual Reality

Submitted in partial fulfilment of the requirements for the Degree of
BACHELOR OF SCIENCE (HONOURS)
of Rhodes University

Edo Lloyd

November 2000

Abstract
This project examines the need for affordable VR input devices. A basic system of categorisation is devised for VR hardware, and the requirements of the input devices are laid out.
Several different types of input device are examined and considered. The most promising type in each category is then implemented and tested, and the result interpreted.

Acknowledgments
First, and most importantly, I'd like to thank my supervisor, Shaun Bangay. Without his endless hours and infinite patience, this project would never have made it to write-up, and I would have been carted off months ago.
Next, thanks need to go to the Center of Excellence and it's corporate funders, and also to Peter Clayton, who organised the much-needed funds required for this project, without which the development of the prototypes would have been a lot more difficult.
I would also like to thank Richard Grant for making himself available to answer my electronics questions.
Thanks also go to EBV Electrolink and the Electronic Services unit at Rhodes for procuring the required electronics bits in a speedy manner, and dealing with my desparate inability to find datasheets.
Finally, I would like to thank Jeremy Alcock and Charl Botha, who supported and kept me awake for the final 36 hours of write-up, who tirelessly proofread and corrected this thesis, and who are really fantastic people. Thanks, guys, I wouldn't have made it through without you!

Table of Contents

Chapter Numbers 1 - Introduction

Numbers 1.Numbers 1 Problem Statement

Virtual Reality requires specialised expensive equipment. Since the price range of this equipment is beyond the means of the average developer, most people either manufacture their own devices from scratch, or adapt an existing low-cost device (eg. Mattel PowerGlove) to fit their needs. The problem, therefore, is to produce a low-cost set of effective input devices for VR, since many developers are hindered by the cost of commercial devices.

Numbers 1.Numbers 2 Motivation for the Study

Virtual Reality has the potential to be one of the next advances in mass end-user technology. Before this can happen, however, there are certain fundamentals about virtual reality that need to change, most notably the costs involved. At present, the cost of setting up and running a virtual reality system are so prohibitive, only well funded organisations can afford such commercial systems.
In order to demonstrate this, a costing for a basic virtual reality system was assembled, as follows. A basic VR system was considered to consist of a headset, a dataglove, and some sort of position/orientation tracker. The developer was assumed to already have a computer, and access to software. The hardware was chosen from the mid to bottom end of the price range, and consisted of the VR4 headset from Virtual Research ($7900), a CyberGlove from Virtual Technologies [15] ($9800 for the 18 sensor model), and a FasTrak system from Polhemus [16] ($6050), and came to a total of $23,750.00. With the addition of a Tactools XTT1 tactile feedback system from Xtensory ($1500) and an Alphatron 3D sound convolver board from Crystal River Engineering ($495), the total value rises to $25,745.00, quite a hefty price tag for what is, after all, a low-end, entry level VR setup. The prices were taken from the VR Buyers Guide [17].
One way of lowering the costs of a virtual reality system is to build it yourself. Obviously, this is a solution employed by many home users, since most organisations do not have the infrastructure to build such devices on their own. There do exist several companies that buildVR equipment to order, but these are few, and expensive. Home users have come up with a variety of innovative ways to solve the problems involved in designing and building a VR platform, including many ways to adapt existing technology to the needs of VR (eg. the Mattel PowerGlove[1]), as well as completely new classes of equipment to solve the various problems faced by developers.
In order for VR to become a feasible and common platform, the costs involved in the acquisition and setting-up of VR systems must drop. This would require a drop in the quality of the hardware, a reduction in the cost of the required materials, or a redesign of current hardware, which utilised cheaper mechanisms. Obviously, a lowering of the quality of hardware is the least desirable of the available options, while reducing the cost of the materials required is beyond the control of the ordinary researcher; this leaves redesigning the current hardware to take advantage of cheaper materials, while maintaining a degree of quality.
Another strong motivating factor is the need for usable devices by the Virtual Reality Special Interest Group[2] at Rhodes University, a need which has not received much attention in previous years. The scope of this project is limited to input devices only.

1.Numbers 3 Research Objectives

The objectives of this study are firstly to categorise the needs of a virtual reality system in terms of its input hardware; secondly, to investigate possible ways of capturing physical data and transforming it into digital information; and finally, to construct some systems, comprising of hardware and programming interfaces, to test and demonstrate some of the investigated capture and transformation methods.
The emphasis in this study is on making the devices as usable as possible, while utilising readily available and affordable components. The hardware should be a simple as possible, in order to minimize component count (and thus cost), while still doing any data manipulation as required,as efficiently as possible. By making the hardware simple and efficient, the entire system takes one step closer to being real-time.

[1]See http://www.spies.com/~jet/VR/faq-0.3_toc.html
[2]See http://www.cs.ru.ac.za/vrsig/

The systems should provide basic, well-documented capabilities to the programmer, while still allowing for later expansion and upgrade. By providing only the basic required capabilities, the systems can be constructed with maximum simplicity, thus reducing cost; if the systems are easy to upgrade, however, this lack of functionality need not hinder the system developer.

1.NumStersc4ure of the Study

This study is broken down into the following major sections:

-?- Background: *Chapter 2* discusses the data acquisition methods that are crucial to any input system, and some traditional and current methods for VR data acquisition. The division and classification of input devices is also covered.

-?- Core Material: *Chapter 3* covers the construction and design of the ADC hardware, as well as its communication protocols, while *Chapter 4* covers the design, construction, and protocols of the video tracking marker system. *Chapter 5* and *Chapter 6* discuss the different types of input devices as researched by the author, including their construction and implementation.

-?- Results: The results of the performance rating and usability experiments are presented in *Chapter 7*.

-?- Conclusions: *Chapter 8* gives the conclusions drawn, based on the assessment of the results. Possible extensions and future work are also discussed in this chapter.

In addition, *Appendix A* contains source code listings for the drivers written for the DANE VR system[3], which are used with this hardware. Testbed program code is included in *Appendix B*, while *Appendix C* contains the schematics of all circuits constructed by the author, as well as connector details.
The CD-ROM attached to this report contains all research material, including datasheets for allhardware used. The reader is directed to the readme.txt file on the CD-ROM for further details.

[3]DANE is the 4th generation Rhodes University Virtual Reality Environment (RhoVeR). See http://www.cs.ru.ac.za/vrsig/

Chapter 2 - Virtual Reality Input Fundamentals

2.1 Classification of Input Devices

All virtual reality input devices considered in the study can be classified into one of two categories, either position trackers or bend sensors. While it is possible for a given sensor to fall into both categories (see below), all systems designed and prototyped by the author are either one or the other. These categories are not the only possible divisions; for example, the Alpeda Multimedia Course divides VR input devices into 7 separate categories, including 'flying mice and wands' and 'datagloves' [18].

A position (or motion) tracker is one which tracks the position of an object (usually the user), providing the position of the tracked object in three dimensional space. The position may be a simple three dimensional Cartesian coordinate, or it may include components like pitch, roll, and yaw. An example of a motion tracker is the Polhemus system[4].

Bend sensors are, as the name suggests, sensors that measures the angle of a bend. While this may not immediately sound particularly useful in the context of a virtual reality system, this type of sensor is one of the most common input devices used. Bend sensors may be used to position one object relative to another, for example, if the position and orientation of the upper arm is known, and a bend sensor is fitted to the elbow, it is a computationally light task to work out where in 3D space the lower arm is, as well as its orientation with respect to the upper arm; the angle of the lower arm can be read directly from a bend sensor fitted to the elbow, and from this value and the position of the upper arm, the position of the lower arm can be calculated by using basic trigonometry. By extending this principle, it is possible to build up a hierarchy of objects in three dimensions, then, from the position and orientation of one object, calculate the positions and relative orientations of all other objects in the hierarchy. This facility is very useful when working with OpenGL, which uses matrix transforms [19].

It is possible to duplicate the effects of a bend sensor with a motion tracking system, but doing so often increases the load on the host computer. In general, motion trackers tend to be larger,more complex, and more expensive than bend sensors, so using motion trackers to replace bend sensors would not normally be the most efficient way of implementing a system. For example, when a series of very fine joints need to be monitored, especially when they are close together, using a motion tracking system becomes a non-trivial task, since most motion tracking systems either have a granularity which is greater than the distance between the fingers of the average user's hand, or tend to be bulky, and thus not suited to applications where many separate small objects that are very close together need to be tracked. When using bend sensors, this can be (and usually is) relatively simple. A good example of such a situation is tracking the positions and orientations of the user's fingers and hand.

2.2 Analog to Digital Conversion Techniques

The basis of almost all physical data capture by computer is the conversion of the analog data provided by the sensors to digital information, which is usable by the computer. The analog signal is converted into a digital number in proportion to the amplitude of the analog signal. This section will give some idea of the various different techniques available to do this conversion [11].

2.2.1 Parallel Encoder

Please convert an image
OLE Generated Object
to the postscript
manually!

[4]See http://www.polhemus.com/home.html

In the parallel encoder, the analog signal is simultaneously fed into one input of n comparators, the other inputs of which are connected to n equally spaced reference voltages. A priority encoder then generates a digital output based on the highest comparator activated by the input voltage (see **Figure 1**).
Parallel encoding (also known as flash encoding) is the fastestmethod, and is thus expensive. It also tends to get prohibitively bulky beyond 10 bits, due to the fact that it requires a balanced ladder of resistors to be accurate.

2.2.2 Successive Approximation

This method revolves around trying various output codes by converting them to analog levels, then comparing the generated voltage with the input via a comparator. This is usually done by setting all bits to 0, then setting each bit to 1, starting with the most significant bit. If the generated analog voltage does not exceed the input voltage, the current bit is left at 1, otherwise it is reset to 0.
Since this is essentially a binary search algorithm, an n-bit converter will require n clock cycles to achieve the correct code. While this is slower than a parallel encoder, it is still reasonably fast, and is much cheaper than parallel encoders. The problem with successive approximation converters is that they tend to respond badly to rapidly changing signals, and especially to spikes. Although generally quite accurate, these converters can sometimes produce strange nonlinearities and "missing codes".

2.2.3 Single-slope Integration

Single-slope integration is based on the comparison of a generated voltage ramp and an input voltage. A ramp generator is started at the same time as a counter, which counts pulses from a stable clock. When the ramp voltage equals the input, a comparator stops the counter; the count is then proportional to the input voltage, and is output as the digital value.
This method is not used very often, since for any degree of accuracy, the stability and accuracy of the comparator must be guaranteed, more than for other systems, since the comparator plays a fundamental role in the operation of this type of ADC. On the other hand, this method is extremely simple, and is easy to construct and scale, and can produce an efficiency close to other popular methods. The single slope integration method is used mainly for applications that don't require absolute accuracy, or where the successive approximation method would be unsuitable, such as peak detection.

Some Existing Tracking Systems2.3 Motion Tracking

One of the more popular methods for tracking the position of an object is the use of an electromagnetic system, such as the Polhemus ISOTRAK II system [1]. The system consists of three parts: a transmitting antenna, a receiving antenna, and associated electronics. Each of the antenna consists of three mutually orthogonal coils. To determine the position of the receiver, each of the coils is independently excited. As each of the coils is excited, the associated receiver coils detect the magnetic field created from the location of the transmitter. The controlling electronics use the resulting three vectors to determine the location of the receiver. The electromagnetic system is good because it can be used to achieve a high degree of accuracy. A single marker not only gives a three dimensional Cartesian coordinate, but also the roll, pitch, and yaw of the marker. This tracking system also has a very fine granularity.
This system is extremely expensive, however, due to the complexity of the hardware, and the precision to which it must be manufactured. It also has a limited range, due to the decay of the emitted electromagnetic radiation.

2.3.2 Bend Sensing

Bend sensors are usually found in some form in any virtual reality system. There are many methods used for measuring the angle of a bend, but one which has found a lot of popularity is the optical method, which works on the principle of light loss in a medium due to the degree of flex in that medium. A good example of this principle is the DataGlove 5 from 5th Dimension [20].
The DataGlove 5 works by running a length of optical fibre around each finger. At one end of the fibre is an LED, which emits light into the fibre, and at the other end, a light dependant element (such as a resistor or transistor). As the angle of flex of the fibre changes, so the does the amount of light which passes through the fibre, and thus the current in the detecting

element at the other end. By measuring the current in the detection element, an idea of the degree of bend may be obtained.

The DataGlove 5 is effective, in that it provides a reasonable idea of the orientation of the fingers, and is low cost, compared to other devices. The system is simple, and easy to work with.
The main disadvantage of the VR Glove is that it tends to 'wear out' over time. The fibre deteriorates, eventually snapping or fracturing, or the controlling electronics fuse, rendering the device useless.

2.4 Proposed Structure of System

An important test for the system as a whole is whether it can provide an effective system as a whole, rather than just separate components. The devices developed in this project are a part of a larger system, while being separatable devices. This section serves to illustrate the structure of an entire VR system.

Please convert an image
OLE Generated Object
to the postscript
manually!

The system consists of input and output devices (see **Figure 2**). The inputs consist of a set of bend sensors, which provide analog data to the DCU (which is driven by the host computer), and a video tracking system (which consists of the VTMS and the camera). The host computer uses the information provided by the input devices to drive the output devices, which consist of a head mounted display, and a 3 dimensional sound system.
In this study, low cost implementations of the DCU and Video Tracking system have been developed. Two types of bend sensor are also designed and implemented. The rest of the system exists as components of the Dane system.

Chapter 3 - Data Capture Unit

3.1 Problem

The data capture unit (DCU) is possibly the most important part of any piece of virtual reality hardware, since it is the vital link between the user equipment and the host computer. Almost all virtual reality input equipment incorporates some form of DCU, although certain devices (for example, input sensors that connect directly to the parallel port of the host computer) have implicit DCUs.

The most important function of the DCU is to translate the raw data into information understandable by the host computer. This should be done as quickly and accurately as possible, ie. the DCU should not be a bottleneck.

Since all the systems considered by the author produce either an analog voltage or a TTL logic signal, an analog to digital converter is deemed the most effective solution. Historically, analog to digital conversion is the most often used means of converting an some external signal to computer-usable information, and thus is a well researched and readily available answer to the problem.

Having decided on an analog to digital converter unit as the core of the DCU, the next step was to design a circuit to perform the conversion. Since this is a well used procedure, there are many analog to digital converter modules available on a single integrated circuit. The ADC0802 from National Semiconductor was chosen, since it provides all the basic functionality required, while still being readily available and affordable. In addition, the ADC0802 is designed to interface directly to a microprocessor, and so makes the system easy to migrate to a stand-alone microprocessor driven platform (see *Chapter 8*).

The other basic functionality (as described in section 3.2) is provided by 4000 series CMOS logic devices, since they are cheap, reliable, and have very low power requirements. They have also been around for a long time, and are thus considered to be industry standard.

3.2 Required Functionality

The DCU needs to convert the analog readings provided by the sensors into a digital value which can be processed by the computer. It needs to be able to distinguish between several different sensors, and provide unique readings for each. It also needs to provide an interface for a wide range of different sensing devices.

Since the ADC unit needs to operate via existing hardware ports on the computer, it needs to conform to certain other requirements, such as logic levels and timing. It also needs to use as few resources on the host machine as possible, in order to be readily upgradable, as well as needing a very usable set of software drivers, to make software construction as easy as possible.

The ADC unit provides the final interface between the host computer and the sensors, so it needs to provide a set of software drivers or programming libraries. The software needs to provide as much functionality as possible, while remaining as simple as possible. The ADC unit also needs to be able to provide a count of the number of sensing devices attached to it.

The analog to digital converter should have an error of no more than 1 bit, and should produce at least a 6-bit output (64 unique values).

3.3 Building Blocks

Please convert an image
OLE Generated Object
to the postscript
manually!

The analog to digital conversion hardware revolves around the ADC0802 chip (see **Figure 3**), which provides the A/D module. The ADC0802 is an 8-bit successive approximation converter, which utilises a potentiometric ladder [2], which means it is somewhat slower than chips which utilise other conversion methods. It has an error of $\pm\frac{1}{2}$ bit, which is well within the required accuracy. The chip requires a 2.5V reference, which is approximated using a 2.7V zener diode, givingan effective range of approximately 230 unique values, which translates into 230 unique positions for any given sensor, far more accurate than required.

Please convert an image
OLE Generated Object
to the postscript
manually!

The ADC0802 does have one drawback, which is that it only has a single analog input. This is overcome with the addition of an 8 to 1 analog multiplexer, in this case, the CD4051BC[5] (see **Figure 4**). The 4051BC is digitally controlled, and provides a means to address 8 separate devices into a single input, with very low "ON" impedance [3], making the signal attenuation negligible.

Please convert an image
OLE Generated Object
to the postscript
manually!

In order to minimise the I/O line usage on the host computer, the data is transmitted in a synchronous serial form. To convert the parallel output from the ADC0802 into a serial bitstream, a CD4014BC[6] shift register is used (see **Figure 5**). The 4014BC is a parallel input / serial output 8-stage shift register, with Q outputs from the 6th, 7th, and 8th stages. All outputs conform to standard "B" series output drive [4]. The 4014BC has a high noise immunity (typically 45% of the supply voltage, roughly 2.2V in this case), and thus may operate via a fairly long cable to the hostmachine, since any electromagnetic interference will be filtered out by the shift register. The 4014BC has a serial / parallel control line, which is used to control whether data is jammed into the Q stages on the positive clock transition, or whether the data already in the Q stages is left shifted. It also requires a clock signal to trigger data movement.

3.4 How it Works

Due to space constraints, the schematic diagram of the DCU can be found in *Appendix C* as **Figure 12**.
The sensor address is set via the MUXA, MUXB, and MUXC lines by the host computer, selecting one of the available input devices. The input devices draw power directly from the

[5]CD4051BC single 8-channel analog multiplexer / demultiplexer, manufactured by Fairchild Semiconductors®

[6]CD4014BC 8-stage Static Shift Register, manufactured by National Semiconductor®

power rails of the ADC unit. By connecting the V_{EE} line on the 4051BC to ground, the multiplexer provides a swing of 0 to 5 V on an input line. The output of the multiplexer is connected to the $V_{in}(+)$ line of the ADC0802.
f SUB clck = ˜ 1 OVER {1.1 RC}
The ADC0802 is provided with a 2.7V reference via a zener diode and a 1K resistor. This reduces the number of unique outputs slightly, but not to the degree where it would be a problem, and removes the need for a precision 2.5V reference. The ADC0802 has an internal clock, which only needs an RC network, so a 10K resistor and a 150pF capacitor are added, giving an internal clock of roughly 606 KHz, which is obtained from the formula,
The ADC0802 takes a maximum of 114μs to do a conversion [5].
The 8-bit output from the ADC0802 is passed to the 8-bit parallel input of the 4014BC static shift register. By taking the parallel / serial control line (which is connected to the INIT line on the host computer) high, data is jammed into each stage of the register on the positive transition of the clock. The parallel / serial control line is then taken low, allowing the data to be clocked out to the host computer, via the INH line.

The CLK and INIT lines from the host computer are also used to drive the second shift register, which provides a device count to the user. The actual count is provided via a 4-bit DIP switch with pull-down resistors. The data is jammed and clocked in exactly the same fashion as the ADC0802 output, and is transmitted on the INL line to the host computer.

3.5 Control Protocols

The software control protocols are very simple. Essentially, all they need to provide are a 3-bit input device address via MUXA, MUXB, and MUXC, a clock and control line via CLK and INIT respectively, and read in the information from INH and INL.

3.5.1 The get_data protocol

The main protocol is the get_data protocol (see **Figure 6**), which reads in an 8-bit value from a selected input device. The first step is to set the address of the input device, which is done by writing the address to the least significant 3 bits of the DATA byte of the parallel port. Once the input device has been selected, a short delay (of roughly 150 μs) needs to be allowed for the ADC0802 to stabilise.

Please convert an image
OLE Generated Object
to the postscript
manually!

Once the ADC0802 has stabilised, the data may be jammed into the shift register, by taking INIT high, then pulsing CLK. INIT is controlled from the select_in line of the parallel port (bit 3 of the CONTROL byte), while CLK is controlled from the init line (bit 2 of the CONTROL byte).
When the data has been jammed into the shift register, it can then be clocked out by pulsing CLK (bit 2 of the CONTROL byte), andreading the value of INH, which is connected to the paper_empty line (bit 5 in the STATUS byte). It will take 8 clock cycles for the full data byte to be read in. The clock pulse needs to be at least 180 ns wide, or a maximum frequency of 2.8MHz [6].

3.5.2 The device_count protocol

The device_count protocol is a somewhat simpler exchange, and not used as often as the getdata protocol. It would typically be used during the initialisation of a system, when the objects that comprise the virtual environment are being set up. In many ways, the device_count protocol is very similar to the final stages of the getdata protocol.
The first step is to transfer the settings of the 4-bit DIP switch into the static shift register. This is done by taking INIT (bit 3 of the CONTROL byte) high, then pulsing CLK (bit 2 of the CONTROL byte). This causes the shift register to jam the data into the upper 4 Q stages. Once the device count has been jammed into the shift register, INIT should be taken low. The 4-bit device count may then be read in by the host machine via INL (bit 7 of the STATUS byte), by pulsing CLK 4 times.
It is important to note that since the two protocols share control lines (INIT and CLK), calling one is liable to corrupt the data of the other. This means that the entire protocol should be carried out every time it is called.

Chapter 4 - Video Tracking Marker System (VTMS)

4.1 Required Functionality

The job of the VTMS is to provide a series of distinctive markers that may be easily picked up a video camera, to mark out the boundaries of an object to be tracked by the system. The markers also need to be distinctive enough for the image processing software to easily identify and locate them in a video frame.
To simplify the image processing software (which is covered in *Chapter 6*), only one marker may be visible to the camera(s) at any given time. The software also needs to know which of the markers is on at any given time, and needs to be able to turn the current marker off, and the next marker on.
Finally, the VTMS needs to be able to share the parallel port with an analog to digital conversion unit, while still being able to co-exist with it. It should be possible to have either or both devices connected to the host computer at any given time. The VTMS consists of a small 'belt-pack' and a series of markers which are placed on the user or tracked object.

4.2 How it Works

At the heart of the VTMS is the CD4017BC[7], which is a 5-stage divide-by-10 Johnson counter with 10 decoded outputs and a carry over bit [7]. The 4017BC also provides a reset feature. The actual markers themselves are simply red ultra-bright LEDs, which are mounted on long wires, and attached to known positions on the object to be tracked. The clock enable line is tied to ground, ensuring the device is always ready for operation. The schematic diagram for the VTMS can be found in *Appendix C* as **Figure 13**.
Since there are no guarantees of power-on values, RSET (which is connected to the auto_feed line in the parallel port on the host computer, controlled by bit 1 in the CONTROL byte) should be pulled briefly high (for at least 400 ns [8]) at system initialisation. This will ensure thatmarker 0 is on, giving known starting conditions.

Please convert an image
OLE Generated Object
to the postscript
manually!

Once the host computer has captured a video frame from one marker, it pulses the ADV line (strobe on the parallel port, controlled by bit 0 of the CONTROL byte). This will cause the decade counter to increment its count by one, thus changing the output line currently high, and thus the marker that is currently on (see **Figure 7**).
When the host computer has stepped through all the attached markers[8], it should reset the VTMS to the initial state of having marker 0 on. This is done by pulling reset high for a minimum of 400ns, then setting it low again.

[7]CD4017BC, made by National Semiconductor®
[8]The actual number of markers is limited to a maximum of 10, but can be anything less than 10.

Please convert an image
OLE Generated Object
to the postscript
manually!

4.3 Cables and Connectors

The problem of possibly having two devices needing to share one physical port on the host computer is overcome by constructing a special plug (see **Figure 8**). Since the IEEE-1284 standard defines the connector used for parallel ports as the 25-way "D" type[9], with the PC having the male connector, the plug is based on a 25-way "D" type female connector.

The 25-way "D" type female connector provides an interface to the signal (and ground) lines of the parallel port. Since the ADC unit needs the most signal lines, its cable connects directly into the "D" connector. The VTMS unit only requires 2 signal lines and a ground, so its cable is connected via a standard audio $\frac{1}{4}$ inch stereo jack, the socket for which is mounted on the outside of the D-25 shell. This allows the VTMS cable to be detached from the parallel port, without the need to remove the ADC cable.

4.4 Control Protocols

The VTMS software protocols are very simple, and essentially consist of single toggle instructions. This means that the computing power needed to drive the hardware is so minute, as to be negligible.

The first of the two protocols is the advance protocol. This is used when the host computer has obtained a video image of the tracked object with a single marker on, and now needs to turn the next marker on. The ADV line is pulled high for at least 250ns [10], then held low.

The other protocol is the reset protocol, which is used when the host computer has stepped through all available markers and needs to reset to marker 0, or at system initialisation, to initialise the VTMS to its starting state (ie. marker 0 on). To reset the VTMS, the RSET line is pulled high for at least 400ns, then reset to low.

Chapter 5 - Bend Sensors

5.1 Possible Systems

Several possible systems for effective bend sensing were originally considered by the author. Each system was considered, along with previous research work pertaining to that system (if any could be found) from other sources. The most promising two systems were implemented and tested, and rated according to various criteria, as documented in *Chapter 7*.

5.1.1 Semi-Conducting Liquid

This method is based on the assumption that the resistance of a semi-conducting liquid will change as the amount of liquid between two electrodes changes. These sensors were expected to be very cheap and easy to manufacture, far cheaper than any of the other options considered, and thus held a lot of promise as a possible solution to the problem.
Expected problems included electrode corrosion due to electrolysis in the salt water, and excessive current flow causing damage to equipment. Since the sensors cost almost nothing to make, it was assumed that the irritation of having to replace them from time to time was not a problem.

5.1.2 Potentiometer Based

The potentiometer based system works by building a rigid frame that matches the shape of the object to be monitored, with rotary potentiometers fitted at all bends. As the shape of the object changes, so the angles of the potentiometers, and thus their resistance, changes.
It was expected that this system would be slightly uncomfortable to the user, since ergonomic design was not the most important consideration. There was also some possibility that the system might restrict movement to a limited set of degrees of freedom.

5.1.3 Light Based

This system was considered as an alternative to the semi-conducting liquid system, and was largely based on the 5th Dimension 5DT DataGlove[21]. It works by shining a beam of light down an air-filled tube (where the 5DT DataGlove uses optical fibre), and measuring theattenuation of the beam at the other end. From this, the bend of the tube can be deduced. The light based sensor is essentially a simplification of the fibre-optic technique, utilised by many virtual reality hardware manufacturers, such as 5th Dimension. Due mainly to time constraints, this sensor type was not implemented.

5.1.4 Spring Loaded Cable / Potentiometer

Please convert an image
OLE Generated Object
to the postscript
manually!

By far the most complicated sensor considered, the spring loaded cable system is essentially a reproduction of early virtual reality bend sensors. Each individual sensor is essentially a spring-loaded potentiometer with an attached cable (see **Figure 9**). As the cable is extended or retracted, the resistance of the potentiometer changes. The cable is connected along the edge of the bending object, allowing the cable to extend as the object bends.
Since this system seemed more likely to suffer from mechanical failure more than any other, and also required a far more mechanical base than any other considered system, it was not implemented.

5.2 Construction

Two systems were implemented, the semi-conducting liquid based, and the potentiometer based systems.

5.2.1 Semi-Conducting Liquid

The semi-conducting liquid bend sensors were constructed using a solution of salt in water and heatshrink tubing. A copper electrode was sealed into each end of the tube. Care was taken to ensure that no air got trapped in the tubes.
Since the expected resistance of the liquid was very low, external current limiting resistors were added. Also, since the expected change in resistance was very low, an amplifying stage was

Please convert an image
OLE Generated Object
to the postscript
manually!

added. The amplifying stage was constructed from an operational amplifier, and was designed to have a gain of roughly 20. The output from the amplifier was fed into an input on the ADC (see **Figure 10**).
For the tests conducted by the author, the diameter of the tubes and the concentration of salt in the solution were varied.

5.2.2 Potentiometer Based

This system was implemented to track the orientation of a single arm. One potentiometer was fitted to the elbow joint, while two were fitted to the shoulder, thus allowing the shoulder to move in two degrees of freedom. 5V was placed across each potentiometer, with each wiper connected to a sensor input on the ADC unit. The rigid frame was constructed from wooden dowel and wire.
Several different values of potentiometer were used during testing. The speed of the ADC unit was also varied.

Chapter 6 - Position Sensors

6.1 Possible Systems

There were less available ideas for determining the position of a object, and thus only one was implemented. Many commercial systems appear to use a mechanism similar to the Polhemus ISOTRAK system (see *Chapter 2*).

6.1.1 Floor Grid

The floor grid system works by setting up a two dimensional grid of light beams at roughly floor level. As the user moves around, different beams are blocked, thus providing an idea of where the user is standing. The floor grid system was based on an idea proposed by Joseph Gradecki in 1993[12].
This system has the advantage of being very simple to implement, since it requires only a level comparator to provide a digital input to the host computer. The electronics required to do this are simple, common, and cheap, and could very easily be powered by the host computer, thus removing the need for an external power supply. On the other hand, the system is dated, and doesn't provide a very accurate idea of where an object is. Its use is limited to two dimensions, and thus it is not that useful for most virtual reality applications.

Please convert an image
OLE Generated Object
to the postscript
manually!

The floor grid system was not implemented.

6.1.2 Boom Tracking

A boom tracking system consists of a series of rods with sensors to measure the bends of the joints (usually potentiometers) (see **Figure 11**). As the tracked object moves around, the angles of the rod-linkages change, thus allowing the position to be calculated.
Boom tracking is very easy to implement, since it uses essentially the same procedure asthe potentiometer based bend sensing system. The setup considered here is based on the design by Joseph Gradecki [13]. It is, however, unwieldy and limiting to use, since it attaches directly to the user, and can result in a tangled mess. Gradecki suggests adding tilt switches to measure pitch and roll, should these values be required.
By replacing the potentiometers at the joints with small motors, the boom system can be used as a haptic three dimensional feedback device. A good example of this use is the SensAble *Phantom* product line, which allows users to 'touch and manipulate virtual objects'[14].

6.1.3 Camera Tracking

Camera tracking requires the least mechanical equipment of all position trackers considered. It works by taking successive pictures of an object, then comparing them to find how the object has moved. To simplify this procedure, markers are attached to the tracked object.
Camera tracking may be constructed as an inward or outward system. In an inward system, the camera is used to watch the tracked object, while the host computer analyses the marker pattern of the object, thus calculating how its position changes. In an outward system, the camera is fixed to the tracked object, and is used to plot the relative movement of some marker system which is fixed to some external point. In both systems, however, the basic principle remains the same [22].
The system considered, and eventually constructed, by the author consisted of a single inward camera, which was fixed to an external point, and a system of three markers, which were driven by the VTMS described in *Chapter 4*.

6.2 Implementation

Other than the VTMS (see *Chapter 4*), the only hardware required was a camera and its capture card. The low level hardware drivers were already implemented in the Dane system, thus requiring only the tracking modules to be written by the author.

6.2.1 The Dane Model

The Dane system model is based on communicating modules. Each module performs a specific function, and may communicate with any number of other modules, thus allowing the construction of processing pipelines.
Since this system is inherently distributed, it is very easy to spread it across a network of computers, instead of requiring a single host to do all the work. This means that one machine could handle the inputs, while another handled the rendering of the virtual environment, thus obviating the need for high-power 'workhorses'.
The distributed nature of the Dane system also makes software maintenance very easy. Since all processes are simply pipelines of operations, it is very easy to add extra functionality by inserting a new module into a pipeline.

6.2.2 The Video Tracking System

The first part of the video tracking system is the module which deals with the markers. It is implemented in C, with hooks into the Java system that comprises Dane, and provides two functions, allowing the system to advance the marker count or reset the marker count (see *Chapter 4*). In the system implemented by the author, the hardware control module provides a single input port.
The modules which provide an interface to the camera were already implemented in the Dane system. They are used to get a PPM image from the camera.
The most important module in the video tracking system is the module which finds the marker in the video image. It was implemented with a single input port, which was connected to the camera-controlling module, and a single output port, which was connected to the next process in the chain. In the first implementation of the system, the marker was found by doing a linear search through the red components of each pixel until a value that was greater than the threshold was found. When this occurred, the pixels around the area were checked, to ensure the event was not a freak error, but that the marker really was at the suspected co-ordinate. If none of the surrounding pixels were above the threshold values, the binary search was continued. If the endof the image was reached without finding the marker, a dummy co-ordinate of (-1 ; -1) was returned, otherwise the co-ordinates of the marker in the image were returned.

6.3 Experimentation

The experiments carried out on the video tracking system are all done in the processing module (see *6.2.2*). Several services are implemented to test the capabilities of the system.
The first processing module built calculates the distance of the object from the camera. This was done by calculating the distance between the detected co-ordinates of two markers using Pythagorean geometry.
The other system attempted by the author is a very basic reverse kinematics system, which is used to calculate the angle of the shoulder. Markers are placed on the shoulder and elbow, and their positions read in. A line is dropped vertically from the shoulder sensor to the horizontal plane of the elbow sensor, and the length is calculated using Pythagorean geometry. The distance between the sensors is then calculated, from which, using basic trigonometric ratios, the angle of the shoulder could be calculated. This method is also implemented with a third marker on the hip, which provides a point to drop the shoulder line to.
Experiments were also conducted with varying values of colour threshold and background saturation.
The results of the experiments and the conclusions drawn from them are contained in *Chapter 7* and *Chapter 8*.

Chapter 7 - Results

This chapter deals with the results of the various experiments carried out by the author.

7.1 DCU

The DCU is driven by a clock signal, which is provided by the host computer. The frequency of the clock was varied, in an attempt to speed up the data transfer operations. The delays in the algorithms were also altered.
The delay between setting the address of the multiplexer and jamming the data into the shift register had to be at least 200 μs (as defined by the ADC0802 datasheet), otherwise the values produced by the DCU became erratic and unpredictable. The delays which govern the frequency of the clock could be set to 0 without any untoward results.

7.2 Bend Sensors

Two types of bend sensor were implemented by the author: semi-conducting liquid type and potentiometer based. The experiments conducted on each type were different.

7.2.1 Semi-conducting Liquid

The parameters varied for this system were the concentration of the saline solution and the diameter of the tubing. The concentration of the solution was found to have very little effect until it dropped below roughly 3 mg / ml, at which point the ions in the solution became few enough to offer a noticeable resistance to the current. When the solution was weak enough for this to occur, the narrowest tubing was found to give the best results. Three sizes of tube were used, 1mm, 2.5mm, and 4mm.
The electrodes at each end of the tube were copper, and thus highly susceptible to electrolysis. The sensors become unusable due to corrosion after 2 to 4 hours of constant current flow (with varying thickness of tube and concentration of solution).
The resistance of the semi-conducting liquid sensors was to found to vary between approximately5 and 2000 ohms. It was largely determined by the concentration of the saline solution, but was also affected by the ambient temperature, and was found to increase as the electrodes corroded.
Due to the dynamic qualities of the semi-conducting liquid sensors, the results obtained are not reproducible. For the same reasons, they can not be considered to be accurate, since the expected value for a known input is constantly changing.

7.2.2 Potentiometer Based

With this system, the values of the potentiometers were varied, to see what the effect (if any) was on the accuracy of the DCU. The timing of the DCU was kept constant. Values between 500 ohms and 100 kilo-ohms were tried.
The only difference between the different values of potentiometer was the current that flowed through them, since the potentiometers were configured as voltage dividers. These tests had no effect on the actual value read in from the DCU, although it was surmised that making the resistance too high would reduce the current below the operating limit of the integrated circuits of the DCU.
In contrast to the semi-conducting liquid sensors, the potentiometer based system provides readily reproducible results. Allowing for slight variations from test to test for shifting of the sensors on the user, the system produces the same values for the same gestures.

7.3 Video Position Tracking

The first tests used a colour threshold of 150 (with a maximum of 255), and were aimed at seeing how easy it was to pick up the markers against various backgrounds. The algorithm performed well against background colours that were dark, but tended to produce unpredictable results with light colours, especially colours with a highly saturated red value. When the algorithm was adjusted to look for low green and blue values, but high red ones, the markers were easily found, provided there were no other red objects in the frame.
The algorithm for determining distance was effective in that it could tell the difference betweendifferent distances from the camera, but required that the markers used be an exact (known) distance apart. If the markers moved, the system could be fooled into thinking the object had moved. This system also produced different results when the camera was repositioned.
The final system to be tested was the arm position tracker. It had fewer difficulties than the distance system, and was able to provide a good reading for the shoulder angle. In some of the

early tests, which were conducted on a slow computer (Cyrix 6x86 PR166 with 24MB of RAM), when the objects were moving very quickly, the system gave obviously erroneous results, but this problem was solved when the software was run on a faster machine.
In the first implementation, the error rate varied from none against dark backgrounds to almost 90% against backgrounds with a high red saturation. The error rate rose quite sharply as the saturation of the background rose. When the algorithm was adjusted to take into account the green and blue values of the pixel, the error rate did not rise as sharply with rising saturation, but the system still had trouble with red objects in the frame.
In all cases, the accuracy of the detected positions deteriorated as the markers got closer to the camera, which was attributed to the fact that the markers appeared much larger in the frame, and thus 'fooled' the algorithm, which found the edge of a cluster of red pixels.

Chapter 8 - Conclusions

This chapter draws conclusions from the results of the experiments carried out by the author, as described in the previous chapter. It also includes observations made by the author during the period of this project.

8.1 Categorisation of Devices

Virtual reality peripherals are broadly divided into input and output devices. The input devices are further divided into bend sensors and position trackers, with the proviso that these categories are not the only possible categories, but are the most convenient for the purposes of this study.
Bend sensors are defined to be any sensing device which measures the angle of a bend, while position trackers are defined as devices which give a three dimensional co-ordinate which describes the position in space of the tracked object. The most effective virtual reality input system would involve components of both types of device, since each has certain strengths and weaknesses (see *Chapter 2*).
At the time of writing, the commercial virtual reality hardware is either very expensive or not very effective [23]. The devices researched and implemented during the course of this project provide an effective yet cheap solution to this problem.

8.2 Data Capture Unit

The DCU is a fundamental part of any virtual reality input device, since it serves as the link between the sensors and the host computer. Historically, this function is fulfilled by an analog to digital converter; this approach was deemed to be the best available by the author, since many single-chip converters were available, and the methods had been well researched and tested.
The DCU needs to convert the analog signals from a set of input sensors to a digital value, which could be read in by the host computer. An 8 to 1 multiplexer is used to provide 8 uniquely addressable inputs, an 8-bit successive approximation ADC integrated circuit provides the conversion facilities, while a shift register is used to minimise the resources used on the inputport on the host computer. A second shift register and a DIP switch provides a device count.
The DCU proved to be a simple yet effective solution. With a low component count, and very low power consumption, it can easily be clipped to the belt of the user. Since the entire device used two input lines and five output lines from the parallel port, another identical DCU could easily be added. If the device count facility were to be removed, a total of five devices could be interfaced through a standard parallel port. By changing the multiplexer to a 16 to 1 device, 16 unique inputs could be addressed.
The DCU performs well in all tests conducted upon it. The only real problem is the slight delay needed for the ADC0802 to do a conversion. This delay is beyond the control of the author, as it was specific to the device. In all other respects, the DCU is capable of providing information as fast as the host computer can request it.

8.3 Bend Sensors

The best bend sensor tested is the potentiometer based system. The semi-conducting liquid devices proves to be unstable, and ultimately unusable, while the potentiometer based devices are very stable and hard wearing.
The main problem with the semi-conducting liquid sensors is due to the low resistance of said sensors. It is surmised that if a means could be found to sandwich a very thin layer of saline solution between two non-conducting sheets, this problem might be overcome. This would not solve the other problem, however, which is the decay of the electrodes. Since electrolysis would occur to some degree with any conducting material, the electrodes would eventually corrode to the degree where they become useless. Also, since the concentration of the saline solution is being changed by the electrolysis process, its resistance also changes over time, resulting in the need for dynamic recalibration. The saline solution would have to be replaced with some form of semi-conducting liquid which does not give rise to electrolysis for these sensors to be effective.
The author found using the potentiometer based system to be very comfortable, with almost nohinderance to natural motion. The only problem found was that over extended periods of use, the system tended to move around slightly, but this was attributed to the very simple mechanism used to hold the device in place (lycra webbing).

Since the value of the potentiometers used to construct the potentiometer based system proved to be inconsequential to the operation of the device, the author recommends using 10kΩ potentiometers, in order to minimise power consumption in the device. Each 10kΩ potentiometer dissipates approximately 2.5 mW of power ($P = I^2R$), since each passes 0.5 mA of current from a 5V source ($V = IR$). This also means that a power supply which provides 5V at a maximum of 100 mA (which is trivial in cost and effort to construct) would provide ample power for a reasonable full-body sensing system.

8.4 Video Tracking

The video tracking system is successful in that it provides an accurate way of determining the position of an object. It turned out to be somewhat more complex to use than originally anticipated, although it has a lot of potential for expansion.
The experimentation showed that the system works best against a black background. The dark background provides a high contrast to the markers, allowing use of the simplest algorithm to locate markers (the one that only looks at the red part of each pixel).
In all tests, the algorithms were easily confused by other red objects in the video frame. Obviously, the system only works with environments lacking in red objects, but the author considered this to be a worthy sacrifice.
The video tracking system could easily be expanded, since it is based on elements already present in the Dane system, and requires only that extra objects be written as needed. The distance and arm position tracker systems are good examples of this, since they both take the output from the marker finding module and process it to produce some conclusion.
The video tracking system is very dependant on the speed at which the camera can provideframes. When the system was tested on a slow machine, there were serious problems tracking even moderately fast motions. The camera and capture card used in the tests was not the highest quality, and thus probably contributed to this problem.

8.5 Research Conclusions

The study was a success overall, since it met the goals set at the beginning of the research period. A basic system of categorisation was devised and applied to the field of virtual reality; several ways of capturing physical data and transforming it into a digital medium usable by a computer were examined, and the most promising were implemented and tested, demonstrating the possibilities of future research, and proving the value of this project.
The implemented systems provide very usable tools for system programmers. They utilise easily available components, and are all very cheap to build. The hardware systems designed by the author are extremely efficient, and approach real-time capable operation.
This document provides documentation on the various systems implemented, while examples of how to use them appear on the accompanying CD. The systems lend themselves to later expansion and updating.

8.6 Possible Extensions

A possible extension to this project would be to use a microprocessor as the core of the DCU. By utilising a device such as the Atmel 90LS8535 (which has a built-in analog to digital converter), the DCU could become a stand-alone device, which could provide information over a standard shared medium (such as an ethernet LAN) to any computers which required it. This approach would also reduce the number of cables which need to connect the user to the host computer(s); with the addition of a wireless LAN, the cables could be eliminated altogether.

References

1. "*Polhemus - ISOTRAK II*", http://www.polhemus.com/isotrkds.htm, September 1999
2. National Semiconductor, "*ADC0801 / ADC0802 / ADC0803 / ADC0804 / ADC0805 8-bit µP Compatible A/D Converters*", National Semiconductor Datasheets, pp 1, November 1999
3. Fairchild Semiconductor, "*CD4051 Single 8-Channel Analog Multiplexer / Demultiplexer*", Fairchild Semiconductor Datasheets, pp1, January 1999
4. National Semiconductor, "*CD4014BM / CD4014BC 8-Stage Static Shift Register*", National Semiconductor Datasheets, pp1, February 1988
5. National Semiconductor, "*ADC0801 / ADC0802 / ADC0803 / ADC0804 / ADC0805 8-bit µP Compatible A/D Converters*", National Semiconductor Datasheets, pp 3, November 1999
6. National Semiconductor, "*CD4014BM / CD4014BC 8-Stage Static Shift Register*", National Semiconductor Datasheets, pp3, February 1988
7. National Semiconductor, "*CD4017BM / CD4017BC Decade Counter / Divider with 10 Decoded Outputs*", National Semiconductor Datasheets, pp 1, March 1988
8. National Semiconductor, "*CD4017BM / CD4017BC Decade Counter / Divider with 10 Decoded Outputs*", National Semiconductor Datasheets, pp 4, March 1988
9. Mueller, S, "*Upgrading and repairing PCs, LINUX edition*", Que Corporation, United States of America, November 1999, pp 840 - 844
10. National Semiconductor, "*CD4017BM / CD4017BC Decade Counter / Divider with 10 Decoded Outputs*", National Semiconductor Datasheets, pp 3, March 1988

11. Horowitz, P, and Hill, W, "*The art of electronics (second edition)*", Cambridge UniversityPress, Cambridge, 1995; pp 612 - 626
12. Gradecki, Joseph D, "*Interfacing a Motion Device*", PCVR, January/February 1993, pp 6 - 8
13. Gradecki, Joseph D, "*Survey of Available Head Tracking Technology*", September/October 1992, pp 3 - 4
14. "*SensAble Technologies: Product Info*", http://www.sensable.com/products/phantom.htm, 22 September 2000
15. "*Hardware Products / CyberGlove*", http://www.virtex.com/products/hw_products/cyberglove.html, 15 January 2000

16. "*Polhemus - FASTRAK*", http://www.polhemus.com/ftrakds.htm, 18 August 2000
17. "*VR Buying Guide*", http://www.cs.jhu.edu/~feldberg/vr/vrbg.html, July 1999
18. "*MMC - Other VR input devices*", http://plum.ia.polsl.gliwice.pl/~DIP/alpeda/fr_1619.htm, 17 March 1999
19. Bangay, S, "*Computer Graphics*" Computer Science Graphics Course Handout, Department of Computer Science, Rhodes University, 2000
20. "*5DT - 5DT Data Glove 5 (5 sensor)*", http://www.5dt.com/p_glove5.html, 5 July 2000
21. "*5DT - 5DT Data Glove 5 (5 sensor)*", http://www.5dt.com/p_glove5.html, 5 July 2000
22. Gradecki, Joseph D, "*Survey of Available Head Tracking Technology*", September/October 1992, pp 5 - 9

23. "*VR Buying Guide*", http://www.cs.jhu.edu/~feldberg/vr/vrbg.html, July 1999

Appendix A

bendinterfaceimp.c

```
/** @name Implementation file: bendinterfaceimp.c
$Revision: 1.1 $
$Id: bendinterfaceimp.c,v 1.1 2000/08/14 23:19:10 edo Exp $
/
#include <jni.h>
#include "BendInterface.h"
#include <stdio.h>
```

```
#ifdef SYSIO
#include <sys/io.h>
#endif
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <arpa/inet.h>
#include <math.h>
#include <unistd.h>
/* #include <conio.h> */
#define DATA 0x378
#define STATUS DATA+1
#define CONTROL DATA+2
int pots=0;
int value=0;
int j=0;
JNIEXPORT void JNICALL Java_BendInterface_Init(JNIEnv* env, jclass jcl)
{
if (ioperm(0x378, 4, 1)) {
fprintf (stderr, "Unable to get port %x for stick – perhaps superuser access required?\n",
0x378);
return;
}
}
JNIEXPORT jint JNICALL Java_BendInterface_GetDeviceCount(JNIEnv* env, jclass jcl)
{
return pots;
}
void clck() {
int bla = inb(CONTROL);
bla = bla | 0x04;
outb(bla, CONTROL);
bla = bla & 0xFB;
outb(bla, CONTROL); }
int read_line() {
int bog = inb(STATUS);
if (((bog^0x20)&0x20) == 0)
return 1;
else
return 0;
}
JNIEXPORT jint JNICALL Java_BendInterface_GetData(JNIEnv* env, jclass jcl, jint psen)
{
outb(psen, DATA);
value = inb(CONTROL);
value = value & 0xF7;
outb(value, CONTROL);
clck();
value = value + 8;
outb(value, CONTROL);
value = 0;
for (j = 7; j >= 0; j--) {
value = value + read_line()*pow(2, j);
clck();
}
return value;
}
```
bendinterface.java
```
/** @name Java file: bendinterface.java
$Revision: 1.1 $
$Id: bendinterface.java,v 1.1 2000/08/14 23:18:59 edo Exp $
/
class BendInterface {
static public native void Init();
static public native int GetDeviceCount();
static public native int GetData(int pot);
static {
System.loadLibrary("bendinterface");
}
} //BendInterface
```
vidtrack.java
```
/** @name Java file: vidtrack.java
$Revision: 1.0 $
$Id: vidtrack.java,v 1.0 2000/09/23 edo Exp $
/
class VidTrack extends DeviceComponent {

public VidTrack() {
super (this, 3);
VideoCameraDevice cam = new VideoCameraDevice (320, 240);
DeviceSource camsource = new DeviceSource (cam);
SetConnection (0, camsource, 0);
VideoWindowDevice vwd = new VideoWindowDevice ();
DeviceSink vidwin = new DeviceSink (vwd);
SetConnection (1, vidwin, 0);
SetConnection (2, new SpotFindComponent(), 0);
}
public void ProcessVideoImage (int device, VideoImage m) {
switch (device) {
case 0:
PutData(1, m);
PutData(2, m);
break;
case 2:
System.out.println("wierd data from SpotFindComponent!");
break;
}
}
public void ProcessSpotFinderMessage(int device, SpotFinderMessage m) {
switch (device) {
case 0:
System.out.println("wierd data from VideoCameraDevice!");
break;
case 2:
if ((m.x != 0) && (m.y != 0))
System.out.println("Found marker at (" + m.x + "|" + m.y);
break;
}
}
public void ThreadRoutine () {
super.ThreadRoutine ();
}
public static void main(String args[]) {
VidTrack vt = new VidTrack();
while (true)
Component.RunComponents();
}
} // class
```

spotfindcomponent.java

```
/** @name Java file: spotfindcomponent.java
$Revision: 1.0 $ $Id: spotfindcomponent.java,v 1.0 2000/09/23 edo Exp $
/
class SpotFinderMessage implements PortMessage {
// change this if the class definition is changed.
static final long serialVersionUID = 550L;
// coordinates relative to the size of the image. [0 .. 1].
double x;
double y;
}
class SpotFindComponent extends Component {
VideoImage last;
VideoImage curr = null;
SpotFinderMessage sfm = null;
public SpotFindComponent () {
CreatePorts (1, 1);
curr = new VideoImage();
}
private SpotFinderMessage findSpot(VideoImage invid) {
SpotFinderMessage coords = new SpotFinderMessage();
coords.x = 0;
coords.y = 0;
boolean found = false;
int pos = 0;
// find spot
while ((!found) && (pos < invid.width*invid.height*3)) {
if (invid.picture[pos] > 150) {
found = true;
coords.y = ((int)(pos / invid.width)) / invid.width;
coords.x = ((int)(pos % invid.width)) / invid.height;
}
pos += 3;
}
return coords;
}
private void videoCopy(VideoImage src, VideoImage dest) {
dest = new VideoImage(src.width, src.height);
if (src.width*dest.width > 0)
for (int i = 0; i < src.width * src.height * 3; i++)
dest.picture[i] = src.picture[i];
}
public void ThreadRoutine () {
```

```
if ((inports[0] != null) && (!(inports[0].WillBlockDataIn()))) {
videoCopy(curr, last);
curr = (VideoImage)(inports[0].TransferDataIn ());
sfm = findSpot(curr);
System.out.println("Spot found at (" + sfm.x + "," + sfm.y + ")"); }
if ((outports[0] != null) && (!(outports[0].WillBlockDataOut())))
outports[0].TransferDataOut (sfm);
}
} // class
```

Appendix B

Track.cpp
```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#define DATA 0x378
#define STATUS DATA+1
#define CONTROL DATA+2
int value;
void main() {
outportb(DATA, 0);
while (!kbhit()) {
for (int b = 0; b < 4; b++) {
value = inportb(CONTROL);
value = value & 0xFE;
outportb(CONTROL, value);
printf("cycle %d, value = %d ", b, value);
delay(500);
value = value | 0x01;
outportb(CONTROL, value);
delay(500);
}
printf("\nResetting markers\n");
value = inportb(CONTROL);
value = value & 0xFD;
outportb(CONTROL, value);
delay(500);
value = value | 0x02;
outportb(CONTROL, value);
printf("value = %d\n", value);
delay(500);
}
}
```
Glove.cpp
```
/* check status of A/D on parallel port */
/* pin assigns are:
11 busy- input low nibble
12 paper empty- input high nibble
16 init- clock for serial
17 select input- serial control
25 ground- ground
2 data 0- MUX A
3 data 1- MUX B
4 data 2- MUX C */
#include <stdio.h>
#include <conio.h> #include <dos.h>
#include <iostream.h>
#include <math.h>
#define DATA 0x0378
#define STATUS DATA+1
#define CONTROL DATA+2
// program controls
#define POTS 4
#define CLOCK_TIME 1
```

```
// define this for debugging output
// #define DEBUG
int i, j;
int value, bit;
void clck();
int read_line();
void main() {
clrscr();
while (!kbhit()) {
gotoxy(1,1);
for (i = 0; i < POTS; i++) {
outportb(DATA, i);// set MUX address
value = inportb(CONTROL);// jam data in shift register
value = value & 0xF7;
outportb(CONTROL, value);// set init high (inverted logic)
clck();// clock cycle
value += 8;
outportb(CONTROL, value);// set init low (inverted logic)
value = 0;
for (j = 7; j >= 0; j--) {
value += read_line()*pow(2, j);
clck(); // clock cycle to shift bits
}
cout << ”The value of pot ” << i << ” is ” << value << ” \n”;
} // for loop
} // while loop
} // main()

/* the clck function outputs a clock signal on the clock line (pin 16),
with a pulse width of CLOCK_TIME us */
void clck() {
int bla = inportb(CONTROL);
bla = bla | 0x04;// set clock bit high
outportb(CONTROL, bla);
delay(CLOCK_TIME);// wait for specified time
bla = bla & 0xFB;// set clock bit low outportb(CONTROL, bla);
delay(CLOCK_TIME);
} // clck
int read_line() {
int bog = inportb(STATUS);
if (((bog^0x20)&0x20) == 0)
return 1;
else
return 0;
}
```

Appendix C

Please convert an image
OLE Generated Object
to the postscript
manually!

Please convert an image
OLE Generated Object
to the postscript
manually!

Please convert an image
OLE Generated Object
to the postscript
manually!