Chapter 3

Wearable Text and Graphic Input Devices

3.1 Introduction

The purpose of this chapter is to introduce two new computer input devices specifically designed for interaction in a mobile environment. The text input device is called the Chording Glove, a glove-mounted chord keyboard which requires nothing more than a solid surface to tap the fingers against. The graphic input device is called the Biofeedback Pointer. This uses bioamplifiers to detect and recognise muscle movements in the lower arm in order to control a pointer. In this chapter we will describe both devices in detail, including their hardware, software and their usage.

3.2 The Chording Glove

While keyboards are highly efficient when used on a desktop, they have been found to be quite difficult to use with a mobile computer. Chord keyboards can be made much smaller than a standard keyboard, with no loss of performance. The main restricting factor of chord keyboards is that, despite the small number of keys, the board must be nearly hand-sized to allow the user to chord comfortably. Soft keyboards operated with a stylus have been found to be the most usable keyboard for mobile system, since they can be made smaller than chord keyboards. The biggest problem is that the keyboard can take up a large amount of the display. Handwriting recognition, with a soft keyboard as a backup has been found to be much more acceptable and is currently is common use on many handheld systems. However, a boardless text interface is a necessity in order to use a continuous mobile computer. Contact gloves can be used as a boardless text interface by translating gestures into characters. While the system can be used for continuous mobility, the gestures must be made with a high level of accuracy in order to be correctly recognised.

Another boardless interface can be made by reducing a chord keyboard to its most basic parts. The board can be thrown away and the chord keys mounted on the fingers themselves. This is exactly what the Chording Glove does. This combines the benefits of a chord keyboard with the portability of a contact glove. A chord keyboard can be used for general text input, like a standard keyboard. It can be learned quickly and retained in long term memory. A contact glove is uncumbersome, since it is just another part of the users clothing. It is quick to activate and use and to stop and put away.

3.2.1 The Hardware

The Chording Glove has three basic parts. These are the Finger Sensors, the Shift Buttons, and the Function Keys (Figure 3.1).

Finger sensors

The finger sensors are small pressure sensitive triggers sewn to the glove at each fingertip. These sensors are designed to activate when pressed against any solid surface.

The thumb is intended to have two sensors, one at the tip and one on the side. The sensors are in series, so pressing either activates it. Different positions on the thumb are used to chord depending on the orientation of the hand. When the hand is against a flat surface, the side is used. When the hand is chording against a curved surface (like a coffee mug) the tip is used. The prototype does not include the second thumb sensor because the current sensor design is too large to accommodate both sensors. The

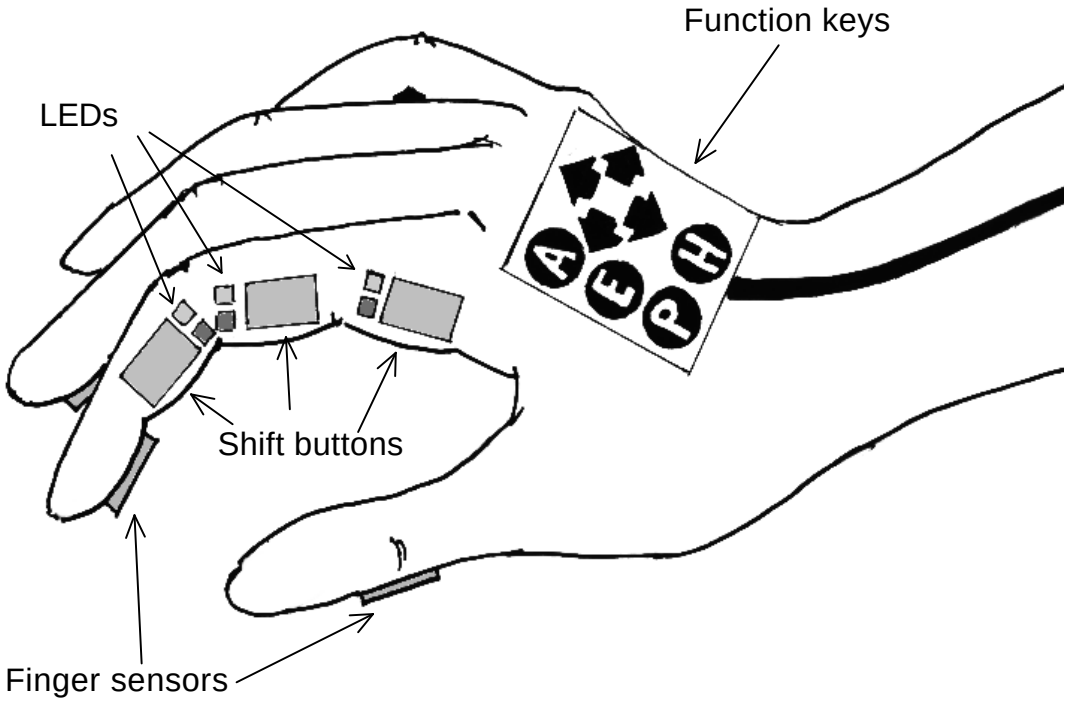


Figure 3.1: Side view of the Chording Glove

sensor on the side of the hand is used because chording against a flat surface is more common.

Several sensor designs were tested before settling on the Hard Copper Plate finger sensor (see Appendix C). The Hard Copper Plate design is a digital button (Figure3.2(a)). Two plastic rectangular plates are connected by a spring at each end. Thin sheets of copper are glued to each piece of plastic backing. Wires are soldered to the copper plates. Additional solder acts as a conductive extension of the copper plates. The entire button is encased in insulating tape to prevent damage and facilitate connecting to the glove.

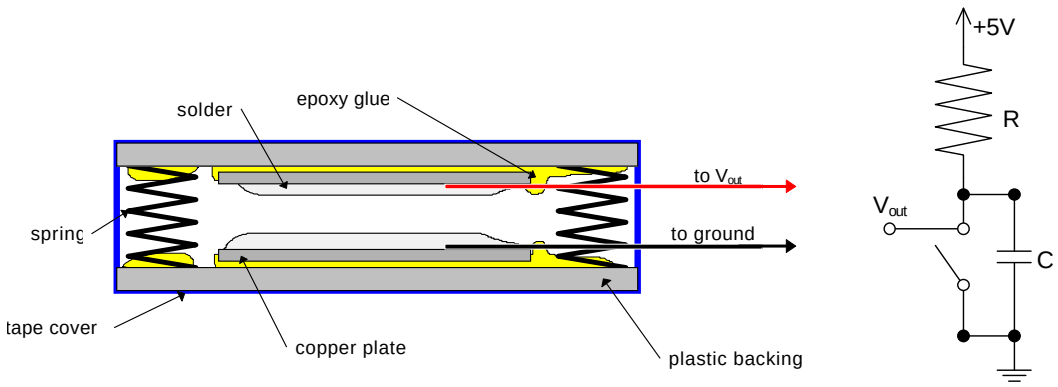
One plate is grounded and the other is connected to 5 volts through a resistor. Compressing the sensor, makes the plates touch, grounding both. Bounce noise is removed by a 48Hz low pass filter (Figure3.2(b)).

Shift buttons

The shift buttons are used to change each shift state: Caps, Number, and Control. When pressed, the shift operates on the next character[1]. When double pressed the shift operates on all characters until pressed again. There is a green LED next to each shift button which lights when the shift is On. A red LED lights when the shift is Locked.

Two designs were tested for the shift buttons (see Appendix C). The first was Taped Aluminium Foil Plates, but these were found to break too easily and the wires would constantly be pulled loose because they could not be soldered in place. The aluminium foil was replaced by copper foil, which was found to

[1]Unless the next chord made is <Space>, <BackSpace>, or <Return>. If one of these is made, the shift stays on until another chord is made.



(a) Cutaway view of a finger sensor

(b) Circuit diagram of a finger sensor

Figure 3.2: The finger sensor

be sufficiently rugged and could be soldered to the wires to keep them in place.

A shift button is made of two small square pieces of copper foil, separated by a thick piece of paper with a hole in the middle. One piece of foil is grounded and the other is connected to 5 volts through a resistor. The foil is flexible enough to connect when pressed, and will return to its original position when released. when the foil pieces connect, they are both grounded, activating the sensor. Bounce noise is removed by a 48Hz low pass filter. This has the same circuit diagram as the finger sensor (Figure3.2(b)).

Function Keys

The function keys are seldom used buttons which are located on the back of the glove. These are intended to be pressed by the other hand. These buttons perform the following operations:

<Pause> - This causes all the finger sensors to be ignored until this key is pressed again. This is to allow the hand to perform other actions without accidental chording.

<Escape> - This has the same effect as on a normal keyboard.

<Help> - A single press of the <Help> key will call an application sensitive help function and a double press displays the chord keymap. The keymap is displayed until <Help> is pressed again. This allows the user to be able to look up chords at any time.

<AutoCaps Toggle> - This turns on and off the AutoCaps feature. The AutoCaps feature automatically capitalises the first letter of a sentence. This is to save effort of using the shift at the start of each sentence. The Caps shift turns on automatically whenever a sentence ending character is chorded: <Period> (.), <Exclamation Point> (!), and <Question Mark> (?). There is an LED next to this button which indicates if it is on or not.

Arrow Keys - The four keys: <Up>, <Down>, <Left>, and <Right> provide basic pointer control, in the same way as on a normal keyboard.

The function keys have the same design as the shift buttons. They are made of foil separated by thick paper with holes for each key. One layer of foil is grounded and the other is connected to 5 volts through a resistor. Bounce noise is removed by a 48Hz low pass filter. The circuit diagram is the same as the finger sensors (Figure3.2(b)).

A more detailed description of the Chording Glove's hardware can be found in Appendix C.

3.2.2 Chord Recognition

There are a variety of ways to programme the chord recognition. Some chord keyboards only have one manner of chord recognition. The recognition starts when the first finger is pressed and stops when the last finger is released. All the fingers pressed in this period are used to generate the chord (Roberts, 1995).

The Chording Glove has a different method of chord recognition. The computer reads the state of the Chording Glove every clock tick (approximately 55ms). If a chord changes or a timeout occurs, a chord is generated in one of three possible ways, *press and hold*, *press and release,* and *change*.

Press and Hold The first case is to make the chord and keep it pressed. When the first finger is pressed, the computer waits 5 ticks. If no fingers are released within those 5 ticks, the computer registers the chord generated from the pressed fingers and outputs it to the keyboard buffer. If the chord remains unchanged for 10 ticks, it is repeated every 2 ticks until it is released.

Press and Release The second case is to make the chord and release it before 5 ticks have elapsed. As above, when the first finger is pressed, the computer waits 5 ticks. If a finger is released before that period the chord generated from all the pressed fingers (before one was removed) is sent to the keyboard buffer. The computer then waits another 5 ticks. If all the fingers are released in this period, the computer stops waiting and is ready to start a new chord. Otherwise, if there are fingers still pressed after 5 ticks, that chord is sent to the keyboard buffer. If this new chord is held, it is repeated as above: first after 10 ticks and then after every 2 ticks.

Change The third case is to change the fingers after a chord is recognised. When the first finger is either added or removed, the computer waits 5 ticks, ignoring any finger changes in that period (as in the Press and Release case). At the end of this time, if any fingers are still pressed, the chord is sent to the keyboard buffer.

The Chording Glove's method for recognising chords has a slight advantage. Instead of needing to release all the fingers after each chord, a new chord can be made by changing the necessary fingers. This can be used to full advantage by the keymap by making frequent digrams and trigrams out of similar chords. This is very similar to having a one character overlap on a standard keyboard. Allowing this "overlap" can potentially speed up the text input rate.

3.2.3 The Chord Keymap

One of the most difficult parts of creating a new keyboard is determining the layout of the keys. This problem is amplified for a chord keyboard, because the user needs to memorise the key positions before they can effectively use the device. The characters associated with all the chords is called the keymap. When generating a new keymap there are five criteria that should be kept in mind.

1. Easy to learn – A good chord keymap can be learned for basic use in less than an hour. It is very important that the chords are easy to memorise, because the user cannot use the keyboard without knowing the all the chords. The time it takes to learn will be proportional to the difficulty in getting people to use the keyboard.

2. Easy to remember – Once learned, a good keymap should stay in long-term memory, even if not used for significant periods of time. There are several ways to help learn the keymap and keep it memorised. Some examples are:

1. Chords can be related to keyboards already learned. For example, a chord can be based on a motion usually used in touch typing, such as using the thumb to press space. This can also be detrimental as it is possible to have the two keyboards interfere with each other, making it difficult to switch between the two keyboards.
2. Chords can be related to sign languages learned. This is similar to being related to keyboards. The advantage is that if the user already knows the sign language, there are less new chords that must be learned.
3. Chords can follow a logical pattern. If chords which are usually typed together (common digrams or trigrams) follow a simple pattern, they are easier to remember. Repeated, rhythmical motion tend to say in memory longer than random, disjointed motion.

3. Minimal work – Because a chord keyboard does not require any hand motion, all motion is concentrated in the fingers. This could cause strain in weak fingers. In order to reduce strain on the user, the finger work should be biased towards the strongest fingers. This reduces muscle strain which could lead to injury.

4. Low error – There is a fine line between where similar chords help memorisation and where they cause confusion. A good rule of thumb to use is that the character which is simpler in shape should have a simpler chord. For example, if the chords for `U` and `W` are related, it would be better to have `W` use more fingers than `U`. Similarly, since <`Comma`> is just a <`Period`> with an appendage, if the chords are similar the <`Comma`> should use more fingers. Another good rule is to have the less frequently typed characters be the less comfortable chords. This reduces strain while reducing errors.

Fast typing – Fast typing is often related to productivity. A user will not want to switch to a new keyboard if their productivity will drop considerably. Chords can be made faster to enter by making trigrams and digrams created by fluid motions. As mentioned above this also aids in memorisation.

Once the keymap is designed, it must go through several iterations of testing and revising before it reaches its final form. Once the final form is settled, it is necessary to test it, both theoretically and empirically.

Developing a Keymap

The first step in creating a new chord keymap is to find which characters are most frequently used. It is possible to find tables of letter probabilities in books on cryptography (Seberry & Pieprzyk, 1988; Denning, 1982; Konheim, 1981). Unfortunately these tend to only give the relative frequency of letters. To make a keymap, all possible typed characters are needed. This includes numbers, punctuation, and control characters. The easiest way to discover this is to collect a large sample of text and count the frequency of characters.

The text which is analysed should contain samples of all possible input which would be typed on a keyboard. This includes normal English, computer languages, operating system commands, data, etc in roughly the proportions that they would be expected to be typed.

The next step in generating a keymap is to divide up the characters in groups. Since there are more typable characters than chord combinations, some way must be found to increase the number of chords. The methods for increasing the number of chords are described in Section 2.2.3. There are two basic concepts behind all these methods. The first concept is to add extra keys. This can be accomplished by using two hands, or by having multiple keys for each finger. The second concept is to add shifts. This can be done by adding extra keys for the thumb, or using sticky shift keys.

Using shift keys has the advantage of dividing the keymap into smaller groups. If divided properly, the smaller groups could be easier to memorise than one large group. The chord groups should be defined using three criteria:

1. All the characters must have a high chance of being typed together. This minimises the need to switch between groups.

2. The characters typed with no shifts on must have the highest frequency. The characters typed with two shifts on must have the lowest frequency.

3. Each group of characters must be obvious and coherent.

The third step in generating the chord keymap is to rate the 31 combinations by comfort. This was done in Seibel (1962). Seibel measured the time it takes to make a chord once a subject is told to make the chord. This is called the Discrimination Reaction Time (DRT). The chance of error of each chord was measured and was found to be roughly proportional to the DRT. When the chords are sorted by the DRT, this gives a list of the most to least comfortable chords.

The next step is to line up the characters on the sorted chord list, with the most frequent characters made by the most comfortable chords. This is now the initial keymap. After this is done, the chords are modified to take into account mnemonics to help remember the chords. This is done is several ways:

1. The chord can resemble the character. The finger combination can have some obvious relation to the shape of the character typed.

2. A sequence of chords can have some meaning and be easy to make. For example a common sequence like `t` $\rightarrow$ `h` $\rightarrow$ `e` can be made a simple sequence of chords, which are easy to make together.

3. One chord can be based on another chord. For example, similar characters, like <Comma> and <Period>, could have similar chords, with only one or two finger changes.

4. A shifted chord can be based on an unshifted chord. For example the chord for <**E**xclamation Point> could be the shifted version of the chord for e.

5. Sequential characters (i.e. numbers) can be made by following a simple pattern.

This is not a comprehensive list, for there are other logical ways to map which can assist in memorisation. Other mnemonics would have to depend on the shift groups and the physical layout of the chord keyboard.

In summary, the process for creating a chord keymap is:

1. Determine character frequency
2. Separate the characters into logical groups
3. Rate chords by comfort
4. Assign characters to chords
5. Revise using mnemonics

Following each of these steps in order will give a good keymap for a specific chord keyboard. There is no one best keymap for a chord keyboard, for many solutions may work equally well. In order to find the quality of the keymap created by following these steps, the keymap must be tested.

Testing the keymap

There are two ways to test the keymap: theoretically and empirically. Of the five criteria mentioned at the start of this section (Page 60), three can be tested empirically: relative finger work, speed and error. We would want a keymap to use the strongest fingers most and the weakest fingers least. One way to determine he relative finger use is to add up all the frequencies of characters made using each finger. Normalising the results will show which fingers are used most and which are used least. The thumb is the strongest finger and can handle larger amounts of work than the other fingers. The thumb would hopefully be used most. The index finger is the next strongest, after the thumb, and should be used almost as much as the thumb. The little finger is the weakest and should be used least. This test gives results which make it quick and easy to compare different keymaps. There are limits to this test, in that it will only determine which fingers are used most, not the ease of the finger combinations. Thus the results of this test must be taken with a grain of salt.

Another theoretical test for the keymap is to find out the speed of the keymap. The theoretical input speed can be approximated by averaging Seibel's DRTs, and weighting each DRT by the frequency of the associated character. This is a heuristic method for estimating the potential chording speed, and although the model is somewhat physiologically flawed, it does give a rough idea of speeds that might be expected. The theoretical error can be calculated the same way using Seibel's errors.

In Seibel (1962), the DRTs were calculated by measuring the time it took for a subject to make a chord once asked to do so. The chords were displayed randomly and were not associated with any characters. When using chords for text entry, the following character is known in advance, giving the subject

a slight time advantage, as knowing the next character in advance allows the user to plan the next chord before the previous one is finished. This implies that it is possible to achieve faster speeds than the theoretical input speed. Further study must be carried out to show the validity of this model.

A common empirical method for testing the keymap is to have several people learn it and measure their progress (Gopher & Raij, 1988; Kirschenbaum et al., 1986). This would give the time it takes to learn the keymap, which is very important in determining how likely people are to use the keyboard. In addition, input speed and rate of increase can be measured. This determines how easy it is to use the keymap.

Developing the Chording Glove Keymap

By using guidelines described above, a new keymap can be created. The following is a step-by-step description of how the chord keymap was made for the Chording Glove.

In order to clarify the finger patterns used in the chords, we will be using a graphical notation to specify the chord. In this notation, the thumb is on the left and lower than the rest of the fingers. A filled dot indicates that the finger is pressed, and an empty dot indicates the finger is not used in the chord. For example, the notation  refers to a chord made by the index and ring fingers. This notation will be used throughout the rest of this thesis.

Determine character frequency The probabilities of all the characters in the ASCII character set were calculated by counting their appearances in a large amount of text in various formats. Details of texts and probability results can be found in Appendix B.

Divide the characters into logical groups There are 100 typable ASCII characters. This requires at least two shift keys. This yields 124 possible chord combinations, which is more than enough. This allows a few characters to be in more than one group.

The 100 ASCII characters divide well into four groups defined as follows:

Lower Case - this is the default group, which is entered with no shifts on.
`a b c d e f g h i j k l m n o p q r s t u v w x y z . , <Space> <BackSpace> <Return>`

Upper Case - these tend to be the next most common, and should be entered with just the first shift key (*caps*) on.
`A B C D E F G H I J K L M N O P Q R S T U V W X Y Z . , <Space> <BackSpace> <Return>`

Numbers - this group consists of numbers and maths-related punctuation. This is the next most common group, so it is entered with just the second shift (*num*) on.
`0 1 2 3 4 5 6 7 8 9 + - = * / ( ) ^ ~ . , <Tab> <Space> <BackSpace> <Return>`

Punctuation - This is all the rest of the characters. Since these are the least frequent, they are made by using both shifts (*caps* + *num*).
`! ? _ " ' ‘ < > @ ^ { } # % ~ | $ \ <Tab> : ; <Space> <BackSpace> <Return>`

Each mode can be locked on if the user needs to enter a several characters from the same group in one stretch. Since `<Space>`, `<BackSpace>` and `<Return>` are likely to be used with any group, it stands to reason that they should be used in all groups.

Table 3.1: The evolution of the Chording Glove keymap. Any character in **bold** has been changed from the previous step.

Chord	1st Guess		Iteration 1			Iteration 2			Iteration 3			Iteration 4			Final Keymap		
	letter	num.	punctuation			shape			sequences			similarity					
○○○●○	**E**	**3**	E	3	**!**	E	3	!	E	3	!	E	3	!	E	3	!
○○●○○	**T**	**2**	T	2		T	2		**H**	2		H	2		H	2	**@**
●○○○○	<Space>		<Space>			<Space>			<Space>			<Space>			<Space>		
○●○○○	**O**	**1**	O	1	**|**	**R**	1	**’**	**T**	1	’	T	1	’	T	1	’
○○○○●	**A**	**4**	A	4	**&**	**I**	4	**?**	I	4	?	I	4	?	I	4	?
●○○●○	**N**	**7**	N	7		**O**	7	**|**	O	7	|	O	7	**|**	O	7	|
○●●○○	**R**		R			**N**		**”**	N		”	N	**(**	”	N	(	”
○○●●○	**I**		I		**?**	**A**		**&**	**G**		**>**	G	**)**	>	G	)	>
●○○○○	**S**	**5**	S	5	**$**	S	5	$	**A**	5	**&**	A	5	&	A	5	&
○●●●○	**H**		**L**	=	**<**	**M**	-	**_**	M	-	_	M	-	_	M	-	_
●○●○○	**L**	**6**	**H**	6		H	6		**R**	6		**S**	6	**$**	S	6	$
●●●○○	**D**	**0**	**G**	0	**>**	**C**	0	**%**	C	0	%	C	0	%	C	0	%
●●●○○	**C**	**9**	C	9	**%**	**G**	9	**>**	**S**	9	**$**	**R**	9	**‘**	R	9	‘
●○○○●	**U**	**8**	U	8		**Y**	8		Y	8		Y	8		Y	8	
○○○●●	<BackSpace>		<BackSpace>			<BackSpace>			<BackSpace>			<BackSpace>			<BackSpace>		
○●○●○	**P**		P	+	#	P	+	#	P	+	#	P	+	#	P	+	#
●●●●●	**M**		M	**-**	**_**	**L**	=	**<**	L	=	<	L	=	<	L	=	<
●○●●○	**G**		**D**	**/**	****	D	/	\	D	/	\	D	/	\	D	/	\
○●●●●	<Period>		.	.	**:**	.	.	:	.	.	:	.	.	:	.	.	:
●●●●●	<Return>		<Return>			<Return>			<Return>			<Return>			<Return>		
●○○○●	**F**		F			**U**	**[**	**{**	U	[	{	U	[	{	U	[	{
●○○●●	**Y**		Y			**F**			F			F			F	<Tab>	<Tab>
●●○●○	**B**		B			B			B			B			B	*****	*****
●○●●●	**W**		W			W			W			**,**	**,**	**;**	,	,	;
●●○○●	<Comma>		,	,	**;**	,	,	;	,	,	;	**W**	**]**	**}**	W	]	}
○●●○●	**K**		K	^	^	K	^	^	K	^	^	K	^	^	K	^	^
●●●○●	**V**		V			V			V			**X**	*****	*****	**J**		
●○●○●	**X**		X	*****	*****	X	*	*	X	*	*	**Z**			Z		
○●○●●	**Z**		Z			Z			Z			**V**			V	<Escape>	<Escape>
●○○●●	**J**		J			J			J			J			**X**	*****	*****
○●●○●	**Q**		Q			Q			Q			Q			Q	**?**	**?**

The mapping of the capital letters is exactly the same as the lower case letters, including <Period> and <Comma>. <Period> and <Comma> have the same mapping in the numerical characters, but are replaced by <Colon> and <Semicolon> in punctuation mode. Each mode has 31 or fewer characters, with the most common characters made without any shifts and the least common made with two. This satisfies the three criteria for grouping characters.

Rate chords by comfort Seibel's (1962) list of DRTs was used to rate the chords. The chords in table 3.1 are ordered using his results, from most comfortable, to least comfortable.

Assigning characters to chords Assigning characters to chords can be done in four simple steps.

The First Guess: Blind Association The first step is allocating chords for <Space>, <BackSpace>, and <Return>. The following chords were chosen for them:

 <Space> This is the same way <Space> is typed on a keyboard, with just the thumb.

 <BackSpace> The <BackSpace> is often on the rightmost side of the keyboard. Furthermore, having <BackSpace> as the last two fingers contrasts well with <Space> space being made with the first finger.

 <Return> Using all fingers simultaneously is easy to remember and has a sense of finality which is good to associate with <Return>.

The easiest way to start is to blindly assign the characters to the chords, aligning the highest probabilities with the lowest DRTs. This is first done for letters (both lower and upper case are the same). The initial keymap can be seen in the two columns under 'First Guess' in Table 3.1.

Numbers follow a special pattern to facilitate memorisation and use. This was generated by using an increasing pattern of the fingers and the thumb as shown in Table 3.2. Not only is this logical, but all the chords are in the easier-to-make half of the DRT list.

Table 3.2: The chord patterns for numbers

<table>
<tr><th></th><th>Index</th><th>Middle</th><th>Ring</th><th>Little</th></tr>
<tr><td>No thumb</td><td>1</td><td>2</td><td>3</td><td>4</td></tr>
<tr><td rowspan="3">Thumb</td><td>5</td><td>6</td><td rowspan="2">7</td><td rowspan="2">8</td></tr>
<tr><td colspan="2">9</td></tr>
<tr><td colspan="3">0</td></tr>
<tr><td></td><td colspan="4"><Return></td></tr>
</table>

After the initial keymap is made, it is repeatedly revised using the methods described above in an iterative process in which each step brings it closer and closer to a workable keymap. Four steps are carried out in between the initial guess and the final keymap. Table 3.1 shows the changes in Chording Glove's keymap for each of these iterations.

The First Iteration: Punctuation Since the characters in the shifted states are not as common as the unshifted characters (letters), assigning them should not be weighted by the probability but by their relation to the unshifted characters. Table 3.3 details the mnemonic relationships used in creating the keymap.

Table 3.3: Mnemonic relations between shifted and unshifted characters

Character	Punctuation	Mnemonic	Comments
A	&	**A**nd	
B	*	**B**y	*as in "multiply 4 **b**y 5"*
X	*		*similar appearance*
C	%	Per**c**ent	
D	/	**D**ivision sign	
/	\		*similar appearance*
E	!	**E**xclamation point	
G	>	**G**reater than	
I	?	**I**nquiry	
Q	?	**Q**uestion	
K	^	**K**arat	
L	=	Equa**l**	
	<	**L**ess than	
M	-	**M**inus	
-	_		*similar appearance*
O	|	**O**r	*used in programming languages like C*
P	+	**P**lus	
	#	**P**ound sign	*also similar in appearance to* +
S	$		*similar appearance*
.	:		*similar appearance*
,	;		*similar appearance*

The Second Iteration: Shape A chord which resembles the character it makes is easier to remember. The characters in the keymap are switched around to allow as many shape relations as possible. The letters `I`, `M`, `N`, and `Y` are based on the letters in American Sign Language (Figure 3.3). Other letters are based on the hand shape resembling the chord, such as `C`, `F` ,`L`, `R`, `U`, <`Quote`>, and <`Apostrophe`> (Figure 3.4).

The Third Iteration: Sequences In the next iteration chords are chosen to facilitate creating common digrams or trigrams. This is done by making minimal finger changes between letters which are likely to be used in sequence. For example, `t` → `h` → `e` is made by the index finger followed by the middle and then the ring fingers. It is the most common trigram and it is one of the easiest chord sequences to make. Table 3.4 shows the chords for some common sequences.

The Fourth Iteration: Similarity It facilitates memorisation if similar characters have similar chords. For example, <`Comma`> is a <`Period`> with an added appendage. The chords should reflect this, by having the chord for <`Comma`> be the same as for <`Period`>, but with the thumb added. Table 3.5 shows the complete list of chords which are based on other chords.

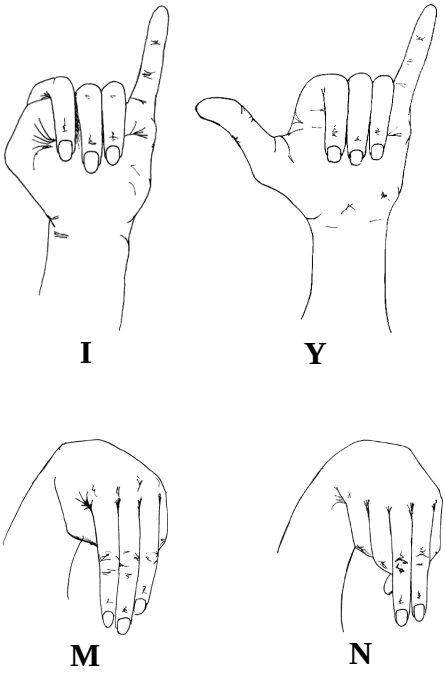


Figure 3.3: The hand positions in American Sign Language for the letters I, M, N, and Y

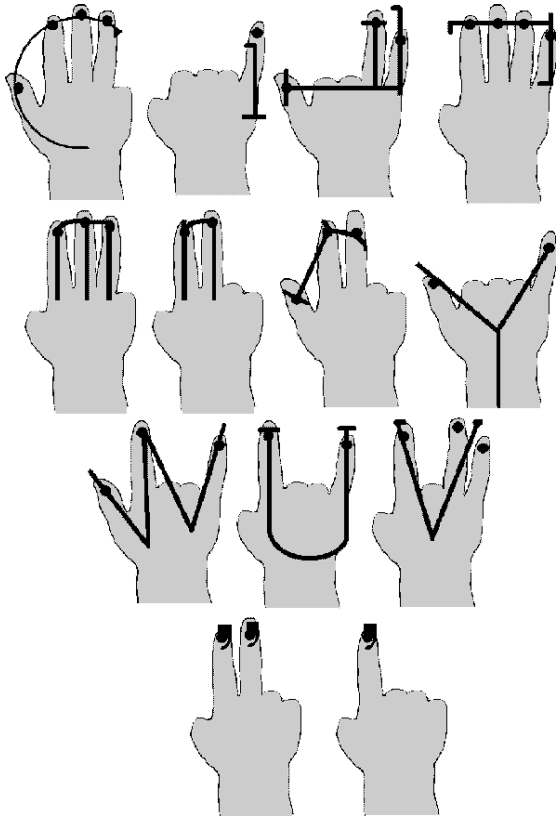


Figure 3.4: The chords which resemble the characters they make. Note: the fingers not shown are not bent forward, they merely do not touch the surface. They are removed from the diagram for clarity.

Table 3.4: Common trigrams and their chords

t → h → e			i → n → g			q → u	
t	○● ○○		i	○○ ○●		q	○●● ○●
h	○○ ●○		n	○●● ○○		u	○●○ ○●
e	○○ ○●	○	g	○○ ●●	○		

Table 3.5: Chords related to other chords

U	○●○ ○●	S	●○ ●○○	.	○○ ●●●	’	○●○ ○○
W	●●○ ○●	Z	●○ ●○●	,	●○ ●●●	"	○●● ○○
V	○●○ ●●					‘	●●● ○○

[	○●○ ○●	{	○●○ ○●	(	○●● ○○
]	●●○ ○●	}	●●○ ○●	)	○○ ●●○

The Final Keymap The final step in generating the keymap is to switch any last few characters which are still in an inconvenient position and then place all the left-over characters. First X and J were switched because, in practice it was found that the chord ●●○ ●● was easier to make than ●●● ○●, and thus was associated with the more common letter. Next, @ and ~ are assigned to the easiest punctuation

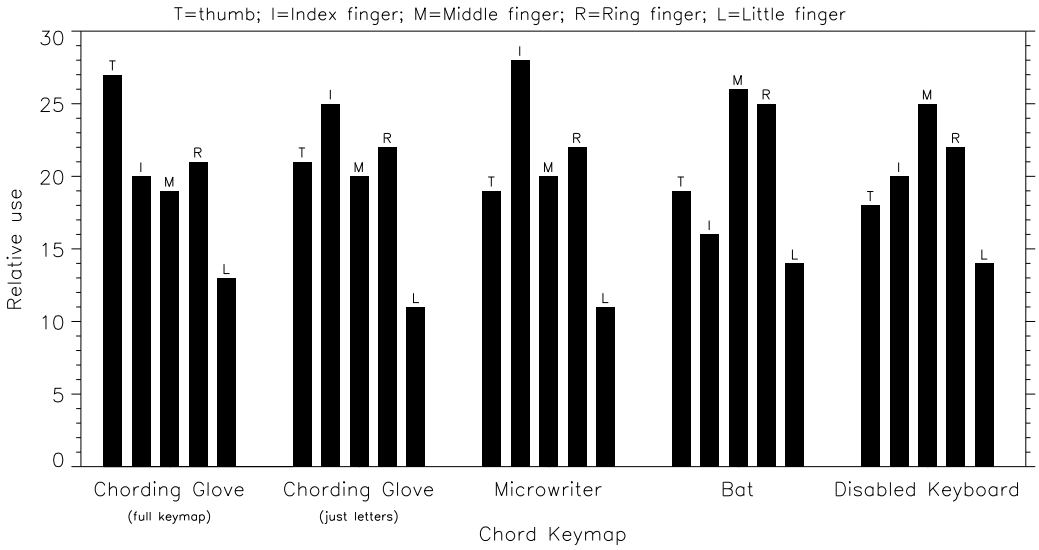
chords still available. Then secondary chords were found for * and ? because there were no shifted chords already assigned to B and Q and it seemed logically appropriate. Lastly the control characters <Tab> and <Escape> were arbitrarily placed in the easiest shifted chords still available.

Theoretical Testing As mentioned above, the keymap can be tested by determining the relative finger usage. The normalised sums of frequencies for each finger are shown in Table 3.6 and displayed as a bar graph in Figure 3.5. Each keymap is normalised to that the sum of the relative use of each finger is 100%. This is to enable a comparison of the relative work for each finger. Because of the differences between the methods for entering special characters on the various chord keyboards, only the chords for the letters were used. Using the full keymap will yield slightly different results.

Table 3.6: Normalised finger work distribution

Keymap used	Thumb	Index Finger	Middle Finger	Ring Finger	Little Finger
Chording Glove (full keymap)	27%	20%	19%	21%	13%
Chording Glove (just letters)	21%	25%	20%	22%	11%
Microwriter (Roberts, 1995)	19%	28%	20%	22%	11%
Infogrip's Bat (Microsoft Corporation, 1995)	19%	16%	26%	25%	14%
Disabled Keyboard (Kirschenbaum et al., 1986)	18%	20%	25%	22%	14%

Figure 3.5: The relative work performed by each finger for different keymaps



The results are favourable: the index finger performs a majority of the work. This is good because the index finger is relatively strong. The little finger is weak and, as designed, does the least work. The other fingers roughly share the rest of the work equally. However, since <Space> makes up approximately 15% of all characters typed, and each of the keymaps use just the thumb for <Space>, the work values

for thumb would be noticeably increased if <Space> were included. This is obvious by comparing the results for the Chording Glove's full keymap to the one for just the letters.

The other methods mentioned above for testing the keymap are finding the theoretical input speed and error rate. These values for various chord keymaps are listed in Table 3.7. Like the situation with work distribution, comparing keymaps is difficult because they do not entirely share the same input mechanisms. Therefore only the chords for letters are used. Again, the full keymap yields slightly different results.

The last three entries are used for a comparison only. *Best possible* show the results from the fastest DRTs associated with the most frequent letters, and the lowest errors associated with the most frequent letters. Note that these are not the same keymap, just the best possible values for these attributes. *Best random* is generated from the average value of the fastest 26 DRTs and lowest errors, which would tend to be the values of a randomly generated keymap. *Worst possible* is results from the slowest DRTs and highest errors associated with the most frequent letters.

Table 3.7: Theoretical Input Speed and Error Rate

Keymap used	Speed		Errors
	in ms	in WPM	
Chording Glove (full keymap)	305	39.3	7.1%
Chording Glove (just letters)	306	39.1	7.0%
Microwriter (Roberts, 1995)	308	38.9	8.1%
Infogrip's Bat (Microsoft Corporation, 1995)	313	38.3	7.3%
Disabled Keyboard (Kirschenbaum et al., 1986)	305	39.4	6.7%
Best possible	302	39.7	5.3%
Best random	313	38.3	7.8%
Worst possible	336	35.7	15.2%

The results show that the Chording Glove achieves its design specifications of having a fast input speed and low error rate. It also compares favourably with all but the Disabled Keyboard, which rates only slightly better. Note the relatively small range of speeds for the various keymaps. The difference between 'Best possible' and 'Worst possible' is only 4wpm. Like with the alternative layouts on standard keyboards, there is not much difference in input speed. The big differences are in finger work and error rates, and can be seen in the graphs in Figure 3.5, and in the 'Errors' column in Table 3.7.

Alternative Keymaps

The Chording Glove's keymap is designed for general English text input. This is not the only situation in which the Chording Glove can be used. Entering text in a foreign language is the most obvious reason for changing the keymap, but there are others. Changing the keymap on a standard keyboard would require either changing or ignoring the labels on the keys, neither of which is particularly desirable. If a user does create a customised layout on their own computer, interference between the layouts would cause their per-

formance on anyone else's computer to be worse than if they never learned their new layout. Since the chord recognition and display of the chord list are both done in software, changing the Chording Glove's keymap can be as easy as loading a new file. Consequently, choosing a new keymap for use with another language, specialised application, or even for reasons of personal preference, becomes a trivial task, effectively equivalent to adding an unlimited number of customisable shift states. The portable nature of the Chording Glove avoids the problems of using a non-customised device. Since the Chording Glove can be taken with the user, it can be just plugged into the new machine and used normally.

While creating a new keymap is not trivial, it is hoped that the method provided here will greatly simplify the process, allowing specialised keymaps to be easily created for any desired purpose. One example could be to make a keymap intended to write Fortran programs. Since the language is not case-sensitive, one could switch the caps and punctuation modes. This way, entering common punctuation like `$` and `'` would be easier since they would require only one shift, while switching to capital letters, a less common task, would be harder, requiring two shifts. Any set of 124 characters (plus another 124 control characters) can be created in order to maximise the efficiency of the user's text entry.

3.2.4 Uses for the Chording Glove

The combination of high mobility and the chording-style input makes the Chording Glove particularly suited to certain tasks. One advantage of the Chording Glove is the fact that, as a chord keyboard, no visual supervision is necessary. This means that the Chording Glove can be used to enter text in situations where one must pay attention to outside events, such as taking notes, driving, or walking in difficult terrain. The display can take the form of either a heads-up display, or even an audio display, to provide feedback to the user without interrupting their other activities.

The one-handed nature of the Chording Glove frees the other hand to perform separate or parallel tasks. For example, the other hand could be used for real-world tasks, such as operating light machinery, carrying objects, etc. For parallel tasks, the other hand could operate a graphic input device to allow simultaneous text entry and graphical interaction. Performing separate tasks in parallel can have improved performance over performing the tasks separately. Experiments have shown this to work for two graphic inputs devices in parallel (Buxton & Myers, 1986; Kabbash et al., 1994), but it may work for graphic and text tasks as well.

Using only one hand has the disadvantage of a slower text input rate as compared to a full-size desktop keyboard. This means the Chording Glove should be used in situations where a full-size desktop keyboard is inconvenient, such as in a mobile environment. Alternatively, it could be used in situations where speed is not an issue, such as in casual text entry tasks like interacting with the operating system, searching or reading text, and possibly basic text composition. This issue is discussed in more detail in Section 6.2.3.

The Chording Glove has the potential to avoid some of the problems linked to RSI. Touch typing on standard keyboard requires the hands to be held in an unnatural position for long periods of time. There is no 'standard' hand orientation for the Chording Glove. The user can position their hand in any manner they find comfortable, as long as they have a surface to chord against. This position can be easily changed

if it becomes uncomfortable after a while.

Suggestions for specific applications for the Chording Glove can be found in Section 3.4.

3.3 The Biofeedback Pointer

In addition to a method for entering text in a mobile environment, there must be a method for pointing and selection. Again, this must be highly economical in its use of space, and be accurate enough for day-to-day interactions - such as those associated with word processing, browsing and data input. Desktop interfaces such as the mouse use up too much space to be of any use. Pointers designed for portable computers such as trackballs, trackpads and trackpoints are board-mounted and while they take up very little space, they still require holding or manipulating something. Glove-mounted virtual reality-style graphic input devices take up no space and do not require anything to be held, but they tend to be too complicated and expensive for everyday use as a simple two-dimensional pointer.

The speed and power of modern computers makes it possible to measure and analyse bioelectric data in real time. The extra hardware required to do this is little more than a few bioamplifiers and a medium to fast analogue to digital converter. The user tapes electrodes and amplifiers to their forearm, which are connected via a junction on the belt to the computer. This is light weight and uncumbersome. By using this device the user can control a pointer simply by moving their hand. Selection can be accomplished by a set of buttons on the side of the index finger which can be pressed by the thumb. These are provided by the Chording Glove. When in text mode the buttons act as shifts. In pointer mode, the buttons act as selection buttons.

3.3.1 Measuring Bioelectric Signals

EEG, GSR, and EMG were all considered as possible bases for the Biofeedback Pointer. The advantages and disadvantages of each were explored in detail and are described below.

An EEG is a collection of low frequency and low amplitude waveforms generated by electrical activity in the brain. Low amplitudes require better amplifiers and noise filtering. The low frequencies can limit the rate of information transmission, reducing the speed of an input device. The potential discomfort caused by the liquid gel required to make the ohmic connection though the hair makes measuring EEGs even more difficult. GSRs, on the other hand, are large enough to be fairly easy to read under most conditions. However, GSR is extremely low frequency, which places limits on its usability as an input device. The GSR is sensitive to imperfections (cuts, scars, etc) and perspiration in the skin, degrading the potential performance of the device. These same imperfections cause no problems with sensing EEGs and EMGs, and in some case imperfections actually *improve* the signals (Geddes, 1972).

An input device based on EMGs needs to be able to detect skin voltages on the order of microvolts (Carroll, 1984). The frequencies in question are, for the most part, less than 500Hz, fast enough to not limit the input speed, while slow enough to read with a 1kHz sample rate. EMGs can be measured with easy-to-apply surface electrodes which are clean and pose no health or safety risk to the user. Consequently, EEGs and GSR were ruled out as potential input devices in favour of EMGs, which were easier to read, control, and measure.

3.3.2 Which Muscles to Use?

Before continuing, readers may find it useful to review some basic anatomical terminology. The following terms will be frequently used in the following sections. The *sagittal*, or *median* plane of the body is the plane of symmetry, i.e. between left and right halves of the body. *Medial* is the direction toward the median plane. The little finger is medial to the thumb (with the palm facing forwards). *Lateral* is the opposite direction, outward from the median plane. The *anterior* or *ventral* side of the forearm is the front, when the palm is facing forward. The *posterior* or *dorsal* side is the back of the arm. *Distal* is the direction away from the root of a limb. The hand is distal to the elbow. *Proximal* is toward the root of a limb, the opposite of distal. A full description of all the physiological terms used in this thesis is available in Appendix A.

Selecting a Muscle Group

There are many muscle groups which can be used to control a computer pointer, but some groups are easier and more appropriate to use than others. There are four main factors to keep in mind when selecting a muscle group. First, the location of the electrodes should be in an area to which they are easy for the user to attach. One would prefer not to need a mirror or another person's assistance for attaching the electrodes. Another factor is that they should be in a socially convenient place. The head and the arm remain as the most workable zones, as they are usually left bare, or with a minimum of covering. A third factor is the size of the muscles and their EMG signals. The neck and face muscles tend to be smaller in size. This requires smaller electrodes to prevent the electrodes from physically interfering with motion. The arm muscles are longer, flatter, and have larger EMGs. This makes them both easier to measure and easier to attach electrodes to.

The fourth and last factor is the type of motion the muscle controls. The arm has joints in the shoulder, elbow, wrist and fingers. All have EMGs large enough to measure. However, the types of motions controlled are quite different. Collecting EMGs for some of the finger motions require electrodes in rather inconvenient locations, making the fingers a less than ideal control group. The shoulder has three degrees of freedom: adduction/abduction, rotation in the sagittal plane, and rotation about the axis of the arm. The elbow has one degree of freedom, making it useless as a two dimensional pointer (Figure 3.6). The wrist has the three degrees of freedom (Figure 3.7): up and down (extension/flexion), left and right (adduction/abduction), and clockwise and anti-clockwise rotation (supination/pronation). Only two degrees of freedom are necessary for a two-dimensional pointer, but having more degrees than necessary is an added bonus which will be addressed in more detail in Section 6.2.3.

Another advantage the wrist has over the other arm muscles is that the relaxed state is in the middle of each degree of freedom. That is to say, the wrist can move from a relaxed state to the left, right, up, or down, or rotate clockwise or anti-clockwise. Few other joints can do this. For example, the elbow, in a relaxed state is straight, when used, it can only move in one direction. This makes it more difficult to create a metaphor to control the pointer. The easier the metaphor, the easier it is to visualise the motion, and, consequently, to learn it. Following this logic, the best results are most likely to come from the linear wrist motions (extension/flexion and adduction/abduction). It is easy to visualise the relationship

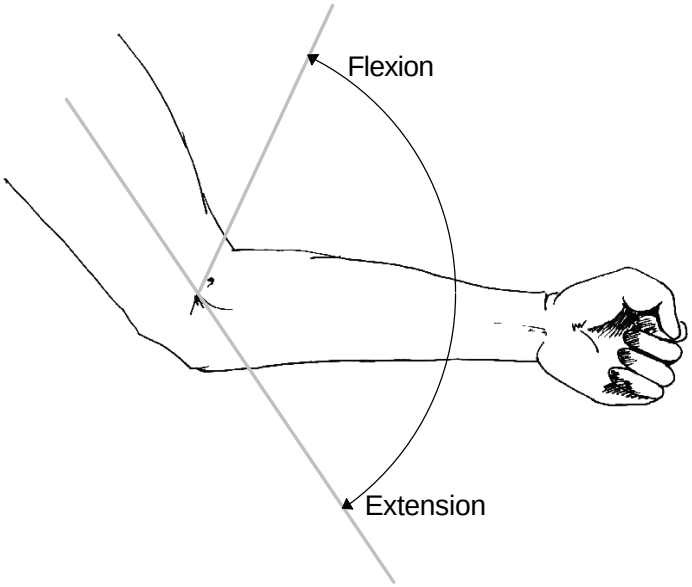


Figure 3.6: The one degree of freedom of the elbow

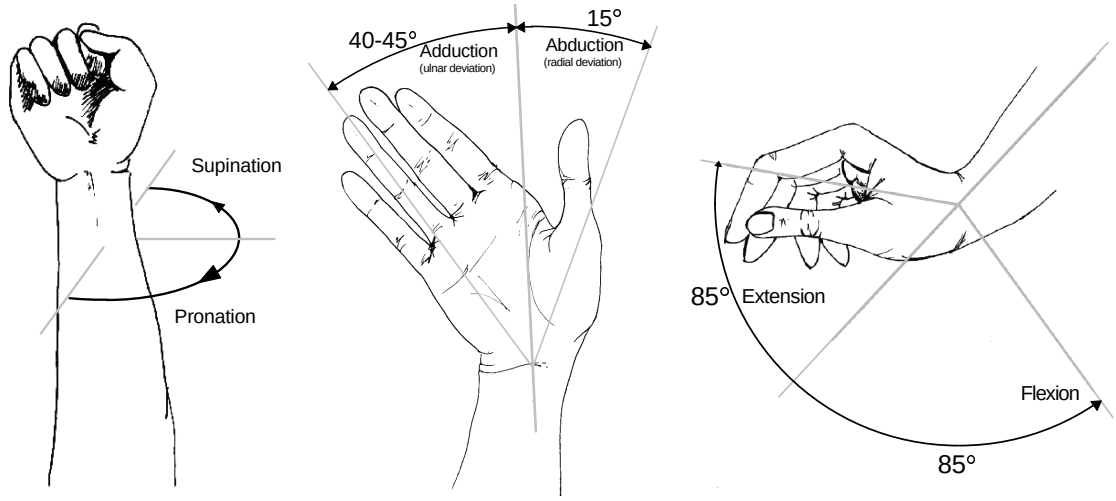


Figure 3.7: The three degrees of freedom of the wrist

between extending the wrist and moving a pointer upwards. Rotations have a less obvious relationship, which would impede learning.

An added bonus of the using the wrist is that the forearm has the least microorganism population density of any part of the body (Geddes, 1972). This means that it is safer to keep electrodes on for longer periods of time, which is important when considering the extended periods for which the device is likely to be used as a computer input device.

A final argument for using the wrist is its relatively high bandwidth in Fitts' Law tasks. The index finger, used by itself, has an IP of 3.0 bits/s. The wrist and forearm have a roughly equal IP of 4.1 bits/s. Slightly higher in bandwidth is the combination of thumb and index finger, with an IP of 4.5 bits/s (Balakrishnan & MacKenzie, 1997).

For these reasons the wrist was chosen as the muscle group to control the pointer for all the experiments (further information about using other muscle groups to control the pointer can be found in Section 6.2.2)

Selecting the Appropriate Wrist Muscles

There are six separate wrist motions: *supination* - rotating the palm to face forward, *pronation* - rotating the palm to face backwards, *flexion* - moving the wrist downward, *extension* - moving the wrist upward, *adduction* or *ulnar deviation* - sideways motion in the direction of the little finger, and *abduction* or *radial deviation* - sideways motion in the direction of the thumb. These motions are controlled by 19 muscles (Figure 3.8). The motions and their controlling muscles are detailed in Table 3.8.

Of these 19 muscles, 14 control the linear motions. Of those, only 6 are superficial and not obscured, and consequently easily measurable. Of the remaining six, the following were chosen: flexor carpi ulnaris (FCU), flexor carpi radialis (FCR), extensor carpi ulnaris (ECU), and extensor digitorum (ED). The extensor digitorum was chosen instead of the extensor carpi radialis longus and brevis pair because it is more superficial and has a stronger EMG. Table 3.9 summarises the motions which these four muscles control.

This choice of muscles gives an orthogonal group which can be used to determine the motion. There are four sensors. One sensor on the FCR is activated upon flexion or radial deviation. Another sensor on the ECU is activated upon extension or ulnar deviation. So far, this gives a one dimensional pointer which can go up or down. A third sensor on the FCU is senses either flexion or ulnar deviation. If the FCR and FCU sensors both detect a signal, the wrist is flexing, i.e. pointing down. If the ECU and FCU are both active, the wrist is moving in the ulnar direction, i.e. to the right (for the right hand). The FCR activated by itself yields radial deviation, i.e. leftward motion. ECU by itself means the wrist is extending, i.e. moving up. These three sensors by themselves are theoretically enough to uniquely determine the wrist's orientation. A fourth sensor was used to improve the accuracy. This sensor was placed on the ED, which shows extension, or upward movement of the wrist.

3.3.3 Hardware

Attaching the Electrodes

The placement of the electrodes is very important, since the Biofeedback Pointer works better with stronger signals. The electrodes need to be placed on the largest part of the muscle in order to get the clearest signal. In this section we describe the recommended positions for the electrodes and how to find them. The level of detail in this section tends to give the impression that the task of applying electrodes is somewhat daunting. In practice the task is simpler than it sounds, and can be performed rather quickly after a little practice. Precise details are given in this section to maximise consistency and repeatability for the tests in Chapter 5 and to avoid any confusion by the reader.

It helps to stretch the skin taut before attaching on the electrode. This is to prevent the skin from stretching later and pulling the electrode loose. This often occurs when the electrode is placed when the elbow is bent. When the elbow is straightened, the skin stretches and the electrode may come off. All the

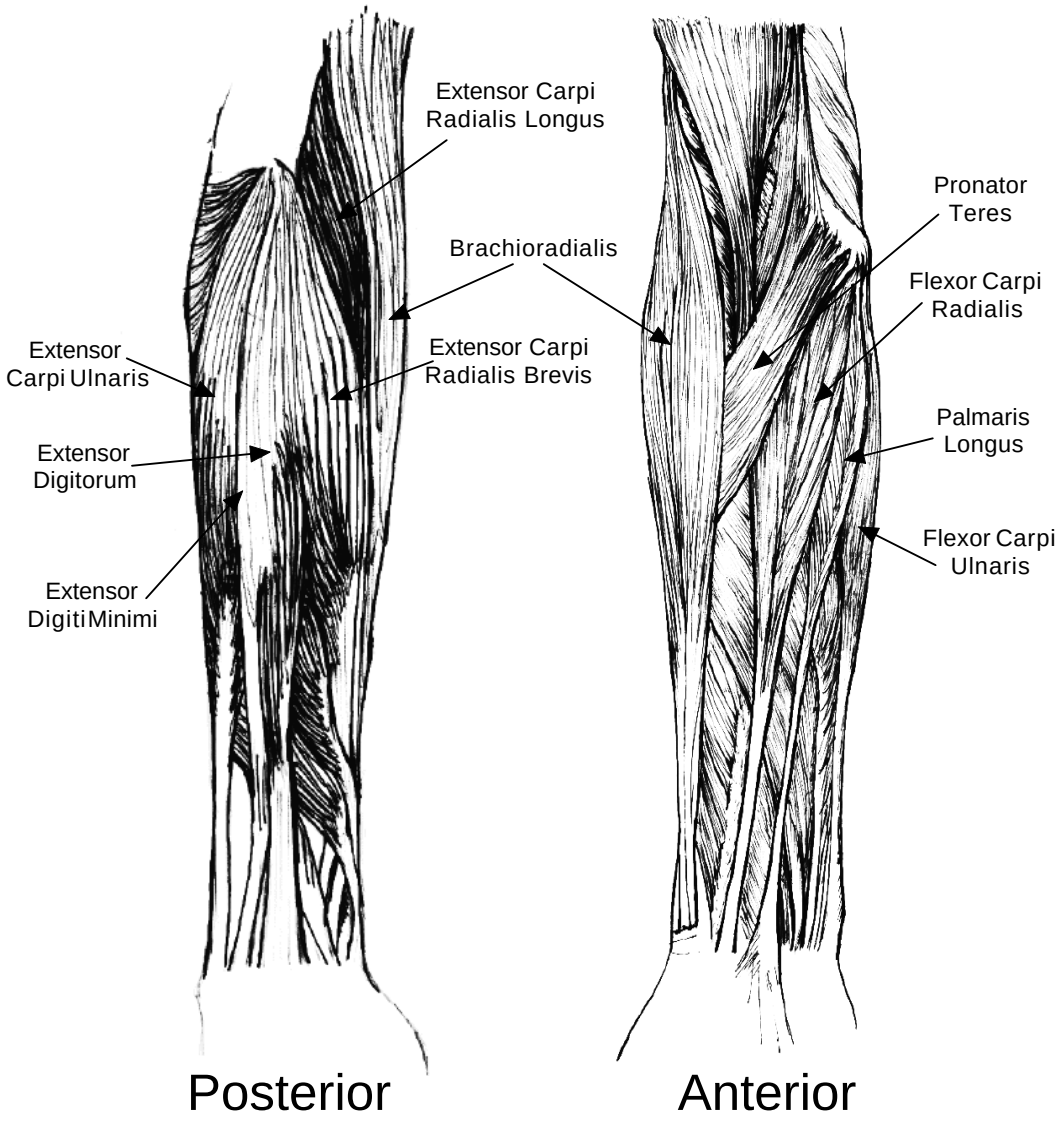


Figure 3.8: The superficial muscles in the lower right arm which control wrist motion. These are based on diagrams found in Moore (1985)

Table 3.8: The muscles controlling movement of the wrist. Since muscles are listed by the motions they perform, some muscles are listed more than once. The primary actions controlled by the muscle are in **boldface**. Motions are for the wrist unless otherwise specified. This is summarised from the muscle descriptions in Chapter 4 of Palastanga et al. (1990).

Motion	Muscle	Movements controlled	Level	Notes
Pronation	Pronator Teres	**pronation**, weak elbow flexion	superficial	
	Pronator Quadratus	**pronation**	deep	
	Brachioradialis	**elbow flexion**, pronation, supination	superficial	Used to return from extreme pronation or supination
Supination	Supinator	**supination**	deep	
	Biceps Brachii	**elbow flexion**, shoulder flexion, supination	superficial	With supinator for strong twisting/pulling motions, e.g. using a corkscrew.
	Brachioradialis	**elbow flexion**, supination, pronation	superficial	Used to return from extreme pronation or supination
Flexion	Flexor Carpi Ulnaris	**flexion**, **adduction**, finger extension	superficial	with PL and FCR
	Flexor Carpi Radialis	**flexion**, **abduction**, pronation, elbow flexion	superficial	with PL and FCU
	Palmaris Longus	slight metacarpophalangeal flexion, flexion	partly-obscured	weak muscle, absent in 10% of people
	Flexor Digitorum Superficialis	**metacarpophalangeal flexion**, **proximal interphalangeal flexion**, flexion	partly-obscured	
	Flexor Digitorum Profundus	**distal interphalangeal flexion**, proximal interphalangeal flexion, metacarpophalangeal flexion, flexion	deep	
	Flexor Pollicis Longus	**thumb interphalangeal flexion**, thumb metacarpophalangeal flexion, flexion	partly-obscured	
Extension	Extensor Carpi Radialis Longus	**extension**, **abduction**, elbow flexion	partly-obscured	with ECRB and ECU
	Extensor Carpi Radialis Brevis	**extension**, **abduction**	superficial	with ECRL and ECU
	Extensor Carpi Ulnaris	**extension**, **adduction**	superficial	with ECRL and ECRB
	Extensor Digitorum	**metacarpophalangeal extension**, interphalangeal extension, extension	superficial	
	Extensor Indicis	**assists ED**, extension	deep	
	Extensor Digiti Minimi	**little finger metacarpophalangeal extension**, little finger abduction, extension	partly-obscured	
	Extensor Pollicis Longus	**thumb extension**, abduction, extension	deep	
	Extensor Pollicis Brevis	**thumb metacarpophalangeal, carpometacarpal extension**, abduction, extension	deep	
Adduction (ulnar deviation)	Flexor Carpi Ulnaris	**Adduction**, **flexion**, finger extension	superficial	with ECU
	Extensor Carpi Ulnaris	**Adduction**, **extension**	superficial	with FCU
Abduction (radial deviation)	Flexor Carpi Radialis	**Abduction**, **flexion**, pronation, elbow flexion	superficial	with ECRL and ECRB
	Extensor Carpi Radialis Longus	**Abduction**, **extension**, elbow flexion	partly-obscured	with ECRB and FCR
	Extensor Carpi Radialis Brevis	**abduction**, **extension**	superficial	with ECRL and FCR
	Extensor Pollicis Brevis	**thumb metacarpophalangeal, carpometacarpal extension**, extension, abduction	deep	
	Extensor Pollicis Longus	**thumb extension**, abduction, extension	deep	

Table 3.9: The muscles used by the Biofeedback Pointer and the motions they control

	ED	ECU	FCU	FCR
Radial deviation				■
Flexion			■	■
Ulnar deviation		■	■	
Extension	■	■		

electrode pairs should be placed on along the arms axis, with the end with the lead facing proximally and the far end facing distally.

The electrodes used are 3M Red Dot^{TM} pediatric monitoring electrodes. Any electrodes will work, but these were chosen because they are small, cheap, and easy to order and apply. The electrodes are slightly too big for the amplifiers. This necessitates cutting the last half-centimetre off one side of the electrode. This can be easily done with scissors or a sharp razor. Since only some of the sticky tape is cut off, it has no effect on the quality of the electrode. The reference electrode is on its own and should not be cut.

It is often helpful to write the number of the amplifier on one of its electrodes. This greatly facilitates matching up the electrodes to the amplifiers if the amplifiers are removed and later re-attached. It is also a useful tool if attempting to diagnose any problems with the hardware. Since each of the amplifiers is numbered on the underside, the number cannot be seen when the amplifier is worn. Numbering the electrode makes the identity of each amplifier clear at a glance.

Before placing the electrodes, the hardware should be plugged into the computer and turned on. The software should be running, and EMG measurements turned on (see Appendix H). This will display the EMG from each electrode on the screen. This is useful when placing the electrodes, as the display will immediately show how strong the signal is.

Acromium Process (Reference Electrode) The easiest electrode to place is the reference electrode. This can be placed on any bony part of the body. The reason for this is that the skeleton acts as a common ground. Since there is no fat or muscle underneath the reference electrode the voltage will be constant. The acromium process in the right shoulder is the most convenient location for this electrode. The acromium process can be easily found as it is the larger and more medial of the two bony projections on the shoulder. The other bony projection, the coracoid process can also be used for the reference electrode. The acromium process was arbitrarily chosen for the sake of consistency.

Flexor Carpi Radialis The first electrode pair is placed over the Flexor Carpi Radialis. Hold out the right arm, with the anterior side facing up. At the wrist the FCR is the lateral-most tendon, just medial to the position where the pulse can be read. Follow this muscle proximally and medially along the arm to about an inch or two distal to the elbow. The muscle is easier to detect if the wrist is flexed. The electrode should be placed here, on the widest part of the muscle. This position should be easy to find by flexing

and relaxing the wrist.

Extensor Digitorum The second electrode pair is placed over the Extensor Digitorum. Reach around to the dorsal side of the arm while keeping the anterior side up. Place your fingers about halfway down the arm and then extend and relax the wrist. You should be able to feel the muscle moving. When you can feel the muscle, place the electrodes there.

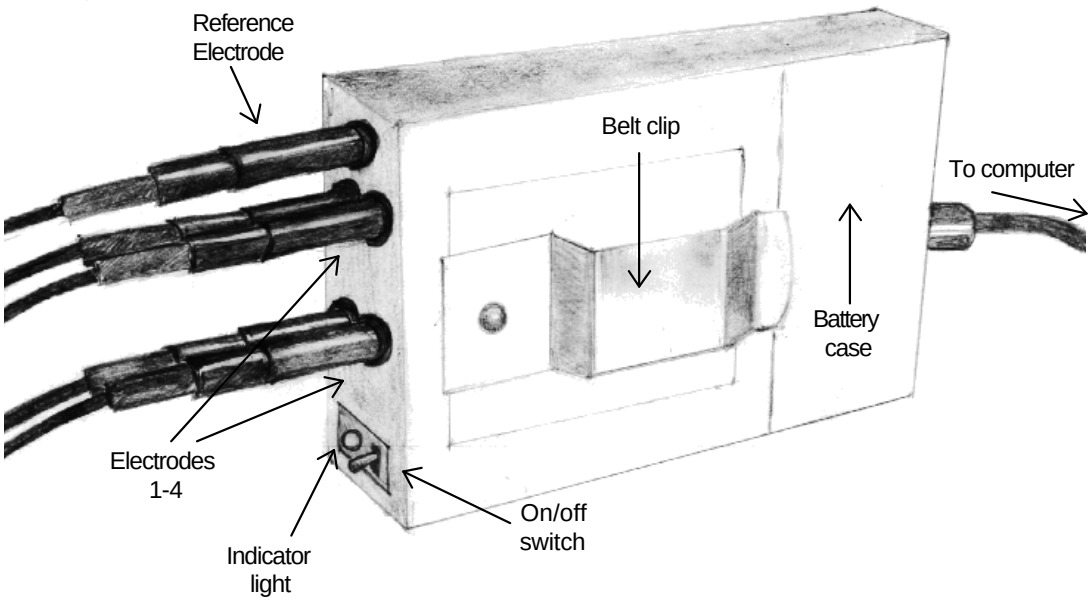
Flexor Carpi Ulnaris The third electrode pair is placed over the Flexor Carpi Ulnaris. Keep the arm anterior side up. Feel on the left side of the arm about an inch or two distal to the elbow. Adduct and relax the arm. Place the electrodes at the point where you feel the strongest muscle movements.

Extensor Carpi Ulnaris The fourth and last electrode pair is placed over the Extensor Carpi Ulnaris. Turn the arm dorsal side up. Feel approximately midway down the forearm, slightly left of centre. Abduct and relax the wrist. You should feel the muscle contracting under the skin. Place the electrode at the widest part of the muscle.

Data Collection

The majority of the Biofeedback Pointer's hardware is contained within a small belt-mounted box (Figure 3.9). This contains the power supply and circuitry to isolate the bioamplifiers from the computer. The remainder of the hardware consists of the bioamplifiers and an analogue to digital converter (ADC). A schematic of the data collection hardware is shown in Figure 3.10. Each electrode pair is plugged directly into a Remote Physiological Pre-Amplifier. The direct connection minimises potential background noise which can be generated by long leads. The physiological amplifier is made up of five parts, all contained in a solid epoxy resin housing. First, the current from each electrode is limited to 500μA. This is a safety precaution necessary in order to prevent a power surge from reaching the user (British Standards, 1993). The EMG signals are fed into the first amplifier with a gain of 100. Low frequencies of 16Hz or less

Figure 3.9: The main hardware of the Biofeedback Pointer



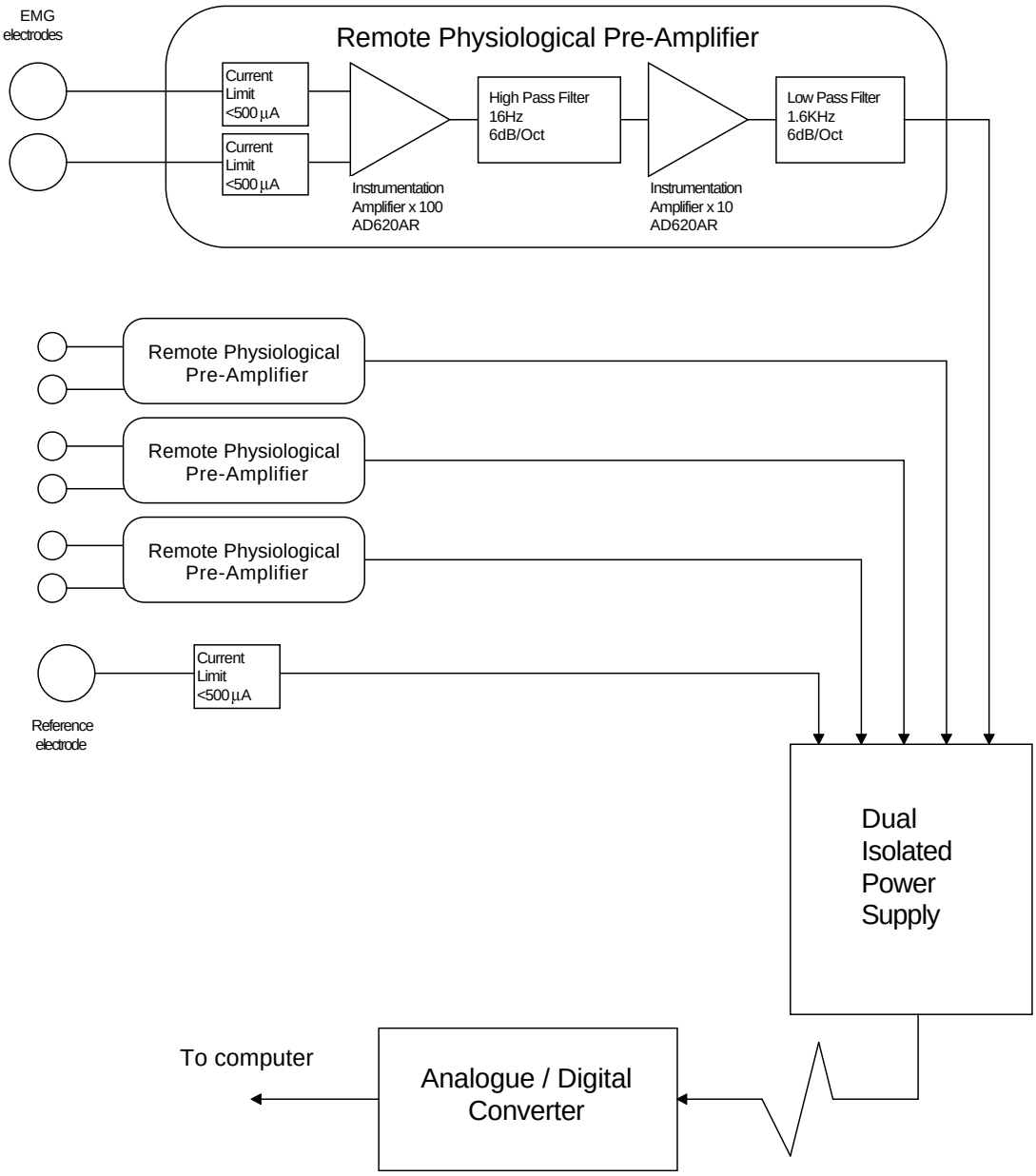


Figure 3.10: Diagram of the bio-amplifier setup

are then filtered out. Next the signal is sent through the second amplifier with a gain of 10. Lastly, high frequencies over 1.6kHz are filtered out. The specifications for the physiological amplifier are detailed in Appendix G.

All four amplified EMGs and the reference electrode are sent into the box containing the dual isolated power supply (Figure 3.9). The reference electrode is also current limited to 500μA. A single 9V battery powers the amplifiers, plus an LED which blinks when the power is on. The four amplified signals emerge from the box in a long cable which ends in a male 25 pin D-shell connector.

The analogue to digital converter is a PICO ADC-11, which is a eleven channel ADC which is attached to the parallel port. The ADC is powered by the PC and its operation is controlled in software. It measures voltages in the range from 0 to 2.5V and can run at speeds up to 18kHz. The technical specification for the ADC-11 can be found in Appendix G.

Signal Processing

Signal processing can be done by several different methods. Three of those were described in Section 2.3.10. Of all the methods, the neural network system (Hiraiwa et al., 1993) seemed to have the most advantages. This method was adapted to work with the above hardware, using an increased update rate of 16Hz.

The process of data acquisition and analysis is pictured in Figure 3.11 and described as follows. Each of the digitised signals is read every millisecond (1). Every 64ms a Fast Fourier Transform (FFT) (2) is performed on the past 512 data readings. The real and imaginary parts are combined to give a 256 point array (3) which is the square of the absolute value of the frequency spectrum. The spectrum is then linearly rebinned to an 8 point array (4). Each of the four reduced spectra (5) are then fed into the neural network (6), which converts the data into pointer motion (7).

3.3.4 Pattern Recognition

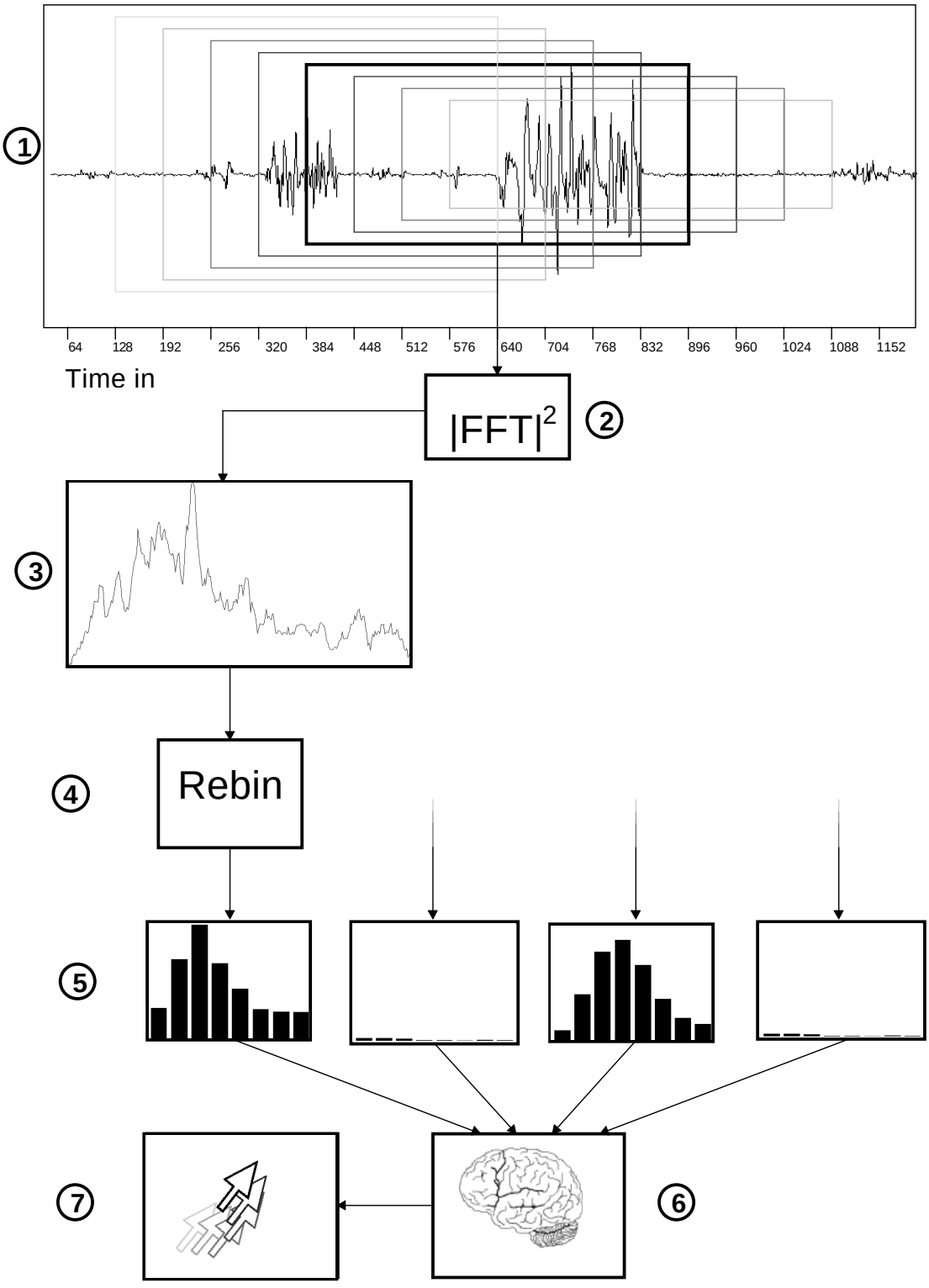
Accurate recognition of the complicated EMG waveforms is essential to the performance of the Biofeedback Pointer. In Section 2.3.10 we described three methods for EMG analysis. Of these three, only the neural network provides the continuous values and a fast enough update rate to be used as a pointing device. For this reason a modified version of Hiraiwa et al.'s (1993) neural network was used to analyse the EMG waveforms.

The Neural Network

Hiraiwa et al.'s (1993) neural network analysed two EMG inputs to provide 10 finger joint angles as outputs. The Biofeedback Pointer has four inputs and x and y coordinates as outputs. By exploiting biofeedback, the Biofeedback Pointer's neural network did not need to perform all the work, allowing for a somewhat larger margin of error than Hiraiwa et al.'s (1993). Consequently, a much simpler neural network could be used for the Biofeedback Pointer. A simple perceptron was found to give the most accurate results.

A *simple perceptron* is a one layer feed-forward neural network (Hertz et al., 1991). This type of neural network takes an array of N input values and produces an array of M output values. One way to

Figure 3.11: The signal processing



implement this network is to multiply the input array by an $N \times M$ matrix of weights. Sometimes the input array is prepended with an additional unit element to accommodate for a uniform offset. This is to allow the output to be non-zero when all the inputs are zero.

The Biofeedback Pointer uses four 8 point arrays as inputs. Combining these arrays with an additional point to act as an offset gives a 33 point input array ($\mathbf{a}$). Since the output is a two-dimensional vector, it is simpler to conceptualise if we use separate arrays for the weights, $\mathbf{w}_x$ and $\mathbf{w}_y$. The dot product of the input array by each of the weight arrays gives the output values of (x, y):

$$x = \mathbf{w}_x \cdot \mathbf{a} \tag{3.1}$$

$$y = \mathbf{w}_y \cdot \mathbf{a} \tag{3.2}$$

Figure 3.12 shows a diagram of this neural network.

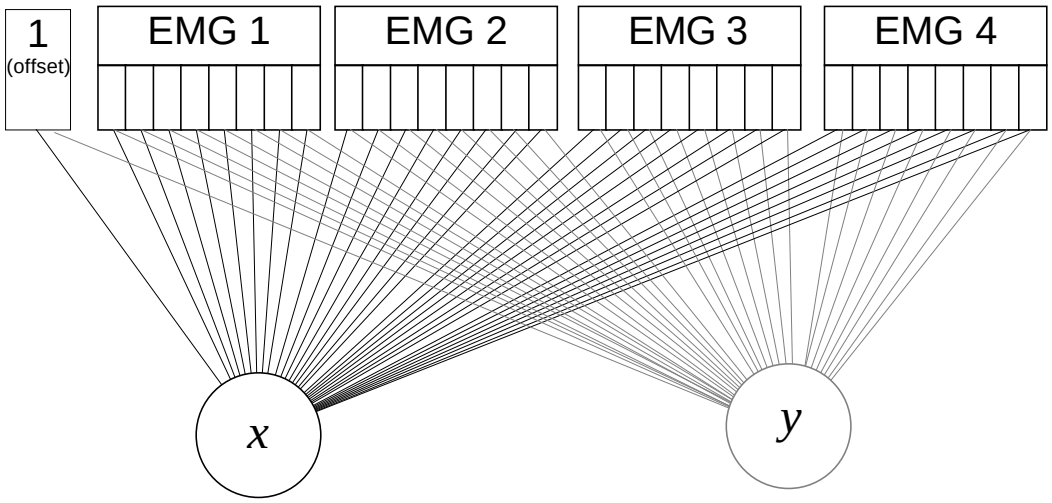


Figure 3.12: The Biofeedback Pointer's simple perceptron neural network

Training

When the pointer is first used, the neural network must be trained. This is done by having the computer move the pointer around the screen while the user follows the motion with their hand. The pointer starts at the centre of the screen and pauses. Each motion begins from the centre, moves out, pauses, then moves back to the centre, where it pauses before beginning the next motion. The motions are: diagonally down and right, diagonally down and left, diagonally up and left, diagonally up and right, down, left, up, and finally right (Figure 3.13). After all the motions, the pointer pauses at the centre of the screen while the weights are calculated.

At each time step the pointer position (x_i, y_i) and the reduced EMG spectra $(\mathbf{a}_i)$ are saved. When the training is complete, we have $N = 378$ sample points saved in the vectors $\mathbf{x}$ and $\mathbf{y}$, and the matrix a, generated by N arrays of EMG spectra, each with 33 values. The weights are then calculated by the following equations:

$$\mathbf{w}_x = \mathsf{R}^{-1}\mathbf{q}_x \tag{3.3}$$

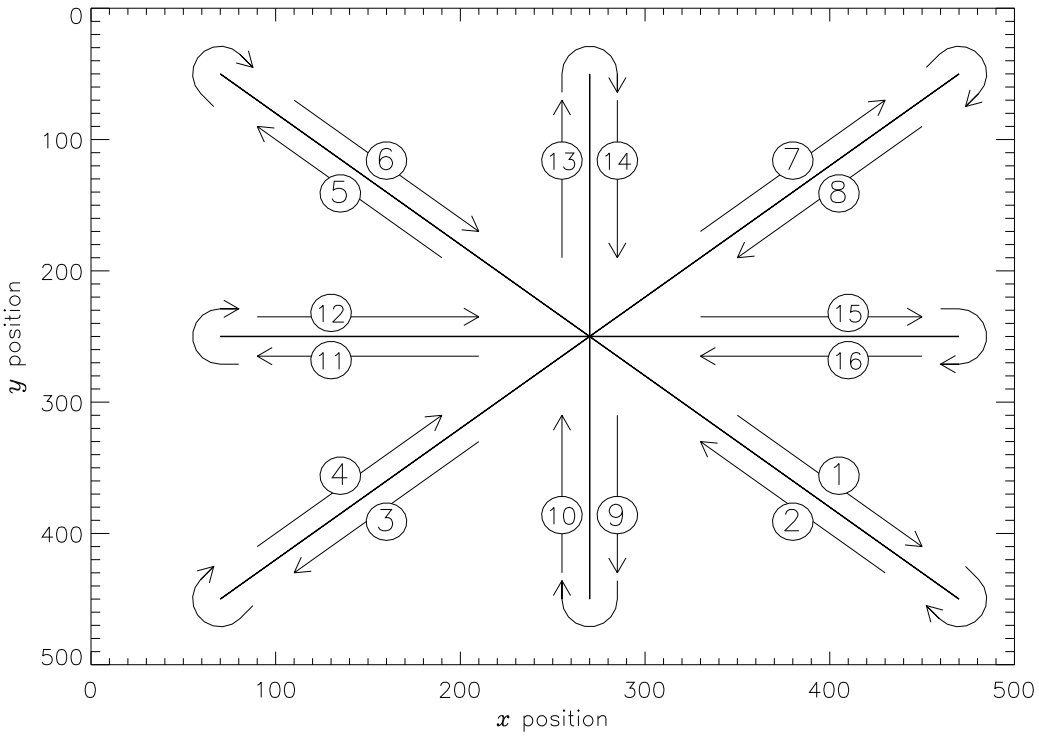


Figure 3.13: The pointer training motion

$$\mathbf{w}_y = \mathsf{R}^{-1}\mathbf{q}_y \tag{3.4}$$

where the variables R, $\mathbf{q}_x$, and $\mathbf{q}_y$ are defined as

$$\mathsf{R} = \mathsf{a}\mathsf{a}^{\mathsf{T}} \tag{3.5}$$

$$\mathbf{q}_x = \mathbf{x}\mathsf{a}^{\mathsf{T}} \tag{3.6}$$

$$\mathbf{q}_y = \mathbf{y}\mathsf{a}^{\mathsf{T}} \tag{3.7}$$

The derivation of these equations can be found in Appendix F.

It is possible that when the pointer is trained, the user's motions will lag behind the motion on the screen. To compensate for this problem with reaction time delay, the weights are calculated eight times, with time offsets from zero to seven steps. This can compensate for reaction time delays up to $^{7}/_{16}$ seconds. The weights which yield the least error are saved and used in the neural network. The error in this network is defined as the average value of the distance from the real points $(\mathbf{x}, \mathbf{y})$ to the calculated points $(\mathbf{x}', \mathbf{y}')$. In other words:

$$\text{error} = \sum_{i=1}^{N} \frac{\sqrt{(\mathbf{w}_x^{\mathbf{t}}\mathbf{a}_i - x_i)^2 + (\mathbf{w}_y^{\mathbf{t}}\mathbf{a}_i - y_i)^2}}{N} \tag{3.8}$$

Using

Once the neural network weights are calculated, it immediately starts translating the EMG input into a two-dimensional position vector ($\mathbf{r}$). The earliest versions of the Biofeedback Pointer used this vector as

the position of the pointer on the screen. This was very noisy and hard to control. The biggest deficiency was that the centre of the screen was easy to point to, and less central points were difficult to target. To solve this problem it was decided that the wrist motion would control the velocity of the pointer. The velocity ($\mathbf{v}$) was calculated from the position vector ($\mathbf{r}$) by the piecewise function:

$$\mathbf{v} = \begin{cases} \frac{\mathbf{r}}{s} & r < cs \\ \frac{\mathbf{r}}{s}(3 - \frac{2c}{r}) & r \geq cs \end{cases} \tag{3.9}$$

This continuous function scales the larger motions by a factor of three. In addition, this is controlled by two variables, c and s. s is the overall scaling, which converts the (x, y) position into a reasonable value for velocity. Since r ranged up to 300, scaling it down by a factor of $s = 16$ provided the reasonable range of values. c is the cutoff magnitude. Below this value, the velocity is unchanged, but above it is scaled by a factor of 3. A value of $c = 6$ was settled on by a process of trial-and-error. Below this value the effect of scaling was unnoticeable, and larger values of c required excessively large motions to trigger the scaling.

Using these values allowed the pointer to perform detailed small movements and faster long movements. This type of scaling is common with continuous graphic input devices and is called an *adaptive control-to-display (C/D) ratio* (Foley et al., 1990).

Other Neural Networks

The simple perceptron was the most effective neural network for the Biofeedback Pointer. In addition to the simple perceptron mentioned above, a Back-propagation neural network (BPN) was also tried with a variety of nodes and levels. Both networks took in 32 inputs (4 spectra of 8 points each) and produced 2 outputs (x and y). The actual process of calculating the weights and outputs varied greatly.

A test was designed to compare the networks. In this, a large number of sets of training data was collected. Part of this data was used to train each of the neural networks. The rest of the data was used to determine the average error which was measured as the distance from the actual to the predicted point. All the possible parameters of each network were varied to determine the best values.

The Back-propagation Network The Multi-level BPN was based on a combination of the algorithms found in Freeman & Skapura (1991) and Hertz et al. (1991). In training the BPN, all the weights were set to random values between 0 and 1. For each step in the trial the outputs are calculated and weights adjusted. This is repeated until the weights converge to constant values. The forward and back propagation process is as follows:

1. Propagate the inputs through the network. The sum of all the weighted inputs to the node (h) is scaled by the function:

$$g(h) = \frac{1}{1 + e^{-h}} \tag{3.10}$$

   which is the output of the node.

2. Once the final outputs are found, the error (δ) is calculated by the function:

$$\delta_i = g'(h_i)(t_i - O_i) \tag{3.11}$$
$$= O_i(1 - O_i)(t_i - O_i) \tag{3.12}$$

where t_i is the target value and O_i is the output value.

3. These errors are propagated backwards by:

$$\delta_{i,k-1} = g'(h_{i,k-1}) \sum_j w_{j,i,k}\delta_{j,k} \tag{3.13}$$

This is done for each node in each layer.

4. The weights are then updated by:

$$\Delta w_{j,i,k}(t) = \eta\delta_{i,k}O_{j,k-1} + \alpha\Delta w_{j,i,k}(t-1) \tag{3.14}$$

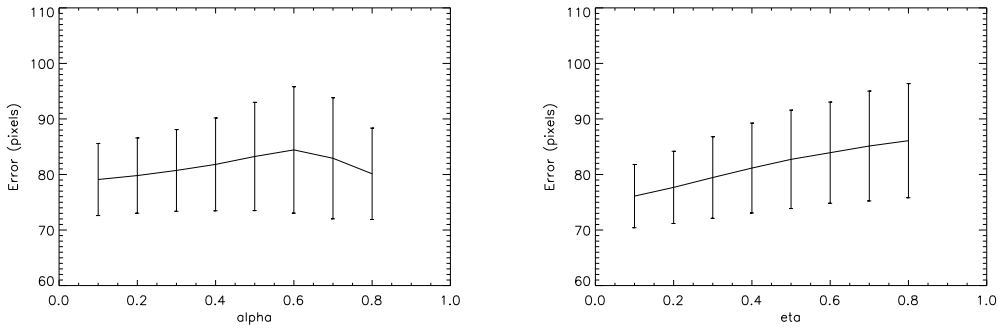
where α is the momentum term and η is the learning rate.

5. This is then repeated until the weights converge.

The BPN had three levels. One input layer of 32 nodes, one output layer of 2 nodes and one hidden layer of n nodes. This gave three parameters: α (momentum), η (learning rate), and n (nodes). In the test, α and η were varied from 0.1 to 0.8. The number of nodes (n) ranged from 2 to 40.

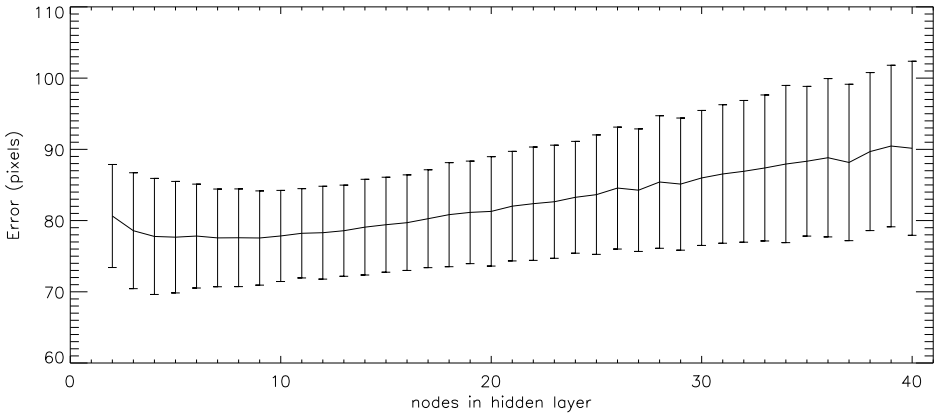
Figure 3.14 shows the effects that varying these parameters has on the errors. Smaller values η give less error. Very large or very small values of α give better results. The best number of nodes is 9. The smallest values of α and η with 9 nodes yields an error of 76 ± 23. This error is the distance in pixels from the computed point to desired one. This was calculated using 13 sets of training data. Other BPN setups were tried, but none gave as low errors. The lowest errors on the other BPNs ranged from 90 to 150.

The Simple Perceptron Network The Simple Perceptron neural network collected all the training data before calculating the weights. The best values for the weights were calculated by a multidimensional least squares fit, as described in Appendix F. This did not require any initial guess because an exact solution was found by solving the matrix equations, instead of using an iterative process to estimate the weights. There were no parameters which could be changed for optimisation, making it easier to determine whether this method was effective or not. The average error after one set of training data was 82 ± 17, which is slightly higher than the best BPN, but the difference is not significant on the 5% level. On the other hand, this method achieved this result 13 times faster and with an order of magnitude fewer calculations per step, making it clear that this was the better method.



(a) Variation in α

(b) Variation in η



(c) Variation the number of nodes

Figure 3.14: Error levels of the Backpropagation Neural Network. Errors are given in pixel units *before* scaling (Equation 3.9)

3.3.5 Biofeedback

Once the neural network is trained to recognise the arm motions, its learning process is over. From then on the user's performance with the pointer is improved by biofeedback. *Biofeedback* is a process in which one or more biological signals are measured and converted into a form which can be displayed to the person (visual, audio, etc). Through a process of trial and error, the person soon figures out how to control the biological function.

This process occurs in the Biofeedback Pointer. As users manipulate the pointer, they slowly learn exactly which arm motions correspond to the pointer motions. The muscles used are not exclusive to the trained motions. For example, the extensor digitorum is not only used in extending the wrist, but is also used in extending the fingers. As a consequence, moving one or more fingers can give similar signals to moving the whole wrist. It has been tested empirically that by experimenting with different motions, the user can find that they achieve the same motion by a slight movement of the middle finger that they would otherwise have used their entire hand. This provides a process in which the user progressively learn the easiest method for interacting with the computer through the Biofeedback Pointer.

3.3.6 Uses for the Biofeedback Pointer

In operation, the Biofeedback Pointer is essentially equivalent to a joystick: the forearm is the equivalent to the base of the joystick and the hand is equivalent to the joysticks shaft. As a consequence we would expect that the Biofeedback Pointer would be best suited for similar applications. As stated in Section 2.3.3, the joystick works best for navigation and low precision pointing. It would stand to reason that the Biofeedback Pointer also be most efficient in those situations.

Navigation refers to both two and three dimensional graphic navigation. This includes such applications as video games, exploration of 3D data, operating a virtual trackball, etc. This also includes navigation in a windows-style environment, such as menu selection and scrolling. Low precision pointing consists mostly of selecting objects (point and click), or moving objects (click and drag). The Biofeedback Pointer would be ineffectual at higher precision tasks such as drawing or fine manipulation tasks.

One disadvantage of the Biofeedback Pointer is that it will continuously move, even when not being used. The only way to keep the pointer still is to completely relax the arm, preventing the user from doing anything else with that arm. This is not a problem for most tasks, since the window manager usually does not care where the pointer is when it is not performing any selection. A situation where this is a problem is an environment with pointer-driven keyboard focus. In this environment, all text input is sent to the window which contains the pointer. It would be very difficult to both control the position of the pointer and enter text at the same time. There is a way around this, and that is to be able to turn off the pointer when it is not needed. This issue is discussed in more detail in Section 6.2.3.

3.4 Applications

While the Biofeedback Pointer and Chording Glove can be used individually or with other input devices, the full advantage comes from using the two devices together as an input for a wearable computer. In this system, the Chording Glove is worn on the dominant hand, while the Biofeedback Pointer's bio-amplifiers are attached to the forearm. These are connected to a wearable computer with a heads-up display. In this system, there is no externally visible difference between the active and passive states of computer use. The only difference would be that in the active state the user would see that the display is on. Switching between the active and passive states can be accomplished simply by pressing a button or two. Consequently, this system rates rather well when judged by the four portability factors introduced in Section 2.4.

Using bioelectric-based graphic input devices for people with severe motor disfunction has been accomplished with EMG, EEG, and EOG. Chord keyboards have been shown to be more usable than a standard keyboard for people with motor disabilities like cerebral palsy and, muscular dystrophy and dyslexia (Kirschenbaum et al., 1986). The reason for this is that the minimal motions involved in chording are much easier to perform for people with these kinds of disabilities. The system introduced here can be used, in one form or another, for people of varying levels of disability. Those with minor motor disabilities could use the Biofeedback Pointer and Chording Glove together to interact with a computer in the same manner as an able-bodied person. For those with more severe disabilities who cannot use the Chording

Glove, the Biofeedback Pointer could be used both as a normal pointer, and to select characters from a soft keyboard.

The portable nature of the Chording Glove and Biofeedback Pointer allows them to be used in several kinds of hostile environments. In Space there is very little room for extraneous equipment. This system takes up very little room and can even be integrated into a space suit. Zero-gravity has its own special problems. A standard keyboard cannot be used because of Newton's Law of action and reaction: by typing a keyboard one could slowly launch oneself across the room (Matias et al., 1996). If the Chording Glove is operated by using the body as the chording surface, this problem is avoided completely. Our system also has benefits for military usage. The devices are silent and are very fast to put away, an important factor in potentially life-threatening situations. In underwater situations, where voice input is near-impossible, our system provides a method for text and graphic interaction with minimal motions.

A more everyday application would be text or graphic interaction for fieldwork. An excellent application would be archaeology. A map can be scrolled or zoomed to determine the present location. The location of artefacts can be recorded, along with notes about them. The computer can then be used to look up further information about the artefacts. With this system, all this can be performed in the field, without even needing to look away from the task at hand.

3.5 Summary

The Chording Glove is a text input device designed for efficient use in a mobile environment. Chords are input by pressure sensitive switches on each finger tip. The shift modes are controlled by small sticky-shift buttons located on the side of the index finger in reach of the thumb. Useful, but infrequently used function keys are located on the back of the hand and can be pressed by the opposite hand.

The chord keymap is designed to take full advantage of the glove's intended use by making the most common characters require the least work to create. The keymap is also designed to maximise speed, minimise errors, and be easy to learn. If necessary, the keymap can be easily changed, to allow more efficient text entry in other languages or for specialised applications.

The lack of visual supervision and the requirement of only one hand for text entry are major factors in allowing the Chording Glove to work in a mobile environment. The lower input speed, as compared to a standard keyboard, suggest the Chording Glove should be used only for causal text entry. Fortunately one would not expect to perform more intensive text entry in a mobile situation. In addition to being a design requirement for a mobile environment, the variety of workable hand orientations for text entry, has the potential to reduce some of the factors that have been linked to RSI on a standard keyboard.

The Biofeedback Pointer is designed to control a computer pointer in a mobile environment. Nothing need be held or manipulated, wrist motion alone is used to control a 2D pointer. The wrist was chosen primarily for the ease of applying the electrodes and measuring the EMGs. Added incentives were high IP of wrist motion and the fact that electrodes could be safely kept on the forearm for relatively long periods of time.

The EMGs from four of the major muscles in the forearm are measured by bio-amplifiers, converted to a digital signal, and transformed into power spectra before being analysed by a neural network. These

four spectra are then translated into an x and y velocity for the pointer. The neural network is trained by requiring the user to follow the cursor as it moves trough a series of motions. When training, the EMG spectra and the cursor position are fed into the neural network which finds the best set of weights to derive the cursor position from the EMG spectra.

The Biofeedback Pointer's design makes it best suited for tasks requiring navigation or low precision pointing. Examples of these tasks are basic windows-style GUI interaction and exploring graphical data, both of which are likely to be performed by a mobile computer. The continuous motion of the Biofeedback Pointer can cause problems in certain environments or applications. Consequently there must be a method for turning the pointer off when motion is unwanted.

Both the Chording Glove and the Biofeedback Pointer used theoretical tests during their construction to determine if certain aspects of the devices would work. These experiments were very limited in the scope of what they could test. In order to fully understand the devices, they must be empirically tested in operation. Experiments were designed and carried out for the Chording Glove and the Biofeedback Pointer. These are the subjects of the next two chapters.