The Cybernetic Chordic Keyer: A simple low cost high performance multikeyer for wearable computers or the like

Steve Mann, N1NLF (US), VA3NLF (Canada)

Abstract— **Humanistic Intelligence is intelligence that arises in a natural cybernetic way, through having a constancy of user–interface, by way of an "always–ready" computer system. This article describes a handheld keyboard that can be easily custom built for each user, and can be used while doing other things such as jogging, running up and down stairs, or the like. It can also be used to secretly type messages while standing and conversing with other people in a natural manner.**

Keywords—**Keyboards, Keyers, Input devices, Human factors, Mobile communication, Cybernetic sciences, Humanistic Intelligence, Consumer electronics**

I. Introduction

WHAT is described, is a simple conformable keyboard, for use with computer systems that work in extremely close synergy with the human user. This close synergy is achieved through a *user-interface* to signal processing hardware that is both in *close physical proximity* to the user, and is *constant*.

The constancy of user-interface (interactional constancy) is what separates this signal processing architecture from other related devices such as pocket calculators and Personal Digital Assistants (PDAs).

Not only is the apparatus operationally constant, in the sense that although it may have power saving (sleep) modes, it is never completely shut down (dead as is typically a calculator worn in a shirt pocket but turned off most of the time). More important is the fact that it is also interactionally constant. By interactionally constant, what is meant is that the inputs and outputs of the device are always potentially active. Interactionally constant implies operationally constant, but operationally constant does not necessarily imply interactionally constant. Thus, for example, a pocket calculator, worn in a shirt pocket, and left on all the time is still not interactionally constant, because it cannot be used in this state (e.g. one still has to pull it out of the pocket to see the display or enter numbers). A wrist watch is a borderline case; although it operates constantly in order to continue to keep proper time, and it is conveniently worn on the body, one must make a conscious effort to orient it within one's field of vision in order to interact with it.

S. Mann is with the department of Electrical and Computer Engineering, University of Toronto, 10 King's College Road, S.F. 2001, Canada, M5S 3G4, E-mail: mann@eecg.toronto.edu http://wearcomp.org.

Special thanks to Xybernaut, Kodak, Digital Equipment Corporation, Compaq, and various others

II. WearComp as an interactionally constant apparatus

The WearComp apparatus of the 1970s and early 1980s was an example of an interactionally constant wearable multimedia computer system for collaboration in computer mediated reality spaces.

Physical proximity and constancy were simultaneously realized by the 'WearComp' project[1] of the 1970s and early 1980s (Fig 1) which was a first attempt at building an intelligent "photographer's assistant" around the body, and comprised a computer system attached to the body, a display means constantly visible to one or both eyes, and means of signal input including a series of pushbutton switches and a pointing device (Fig 2) that the wearer could hold in one hand to function as a keyboard and mouse do, but still be able to operate the device while walking around. In this way, the apparatus re-situated the functionality of a desktop multimedia computer with mouse, keyboard, and video screen, as a physical extension of the user's body. While the size and weight reductions of WearComp over the last 20 years, have been quite dramatic, the basic qualitative elements and functionality have remained essentially the same, apart from the obvious increase in computational power.

An important, and so far unpublished, aspect of the WearComp has been the keyer, which serves to enter commands into the apparatus. The keyer has traditionally been attached to some other apparatus, such as a flashlamp, or lightcomb, so that the effect is a hands free data entry device (hands free in the sense that one would need to hold onto the flashlamp, or the like, anyway, so no additional hand is needed to hold the keyer).

The operation of a keyer takes place over seven stages.

III. The seven stages of a keypress

The original keyer had five keys, one for each of the 4 fingers, and a fifth one for the thumb, so that characters were formed by pressing the keys in different combinations. The computer can read when each key is pressed, and when each key is released, as well as how fast the key is depressed. The velocity sensing capability arises from using both the naturally closed (NC) and naturally open (NO) contacts, and measuring the time between when the common contact (C) leaves the NC contact and meets the NO contact.

[1]For a detailed historical account of the WearComp project, and other related projects, see [?][?].

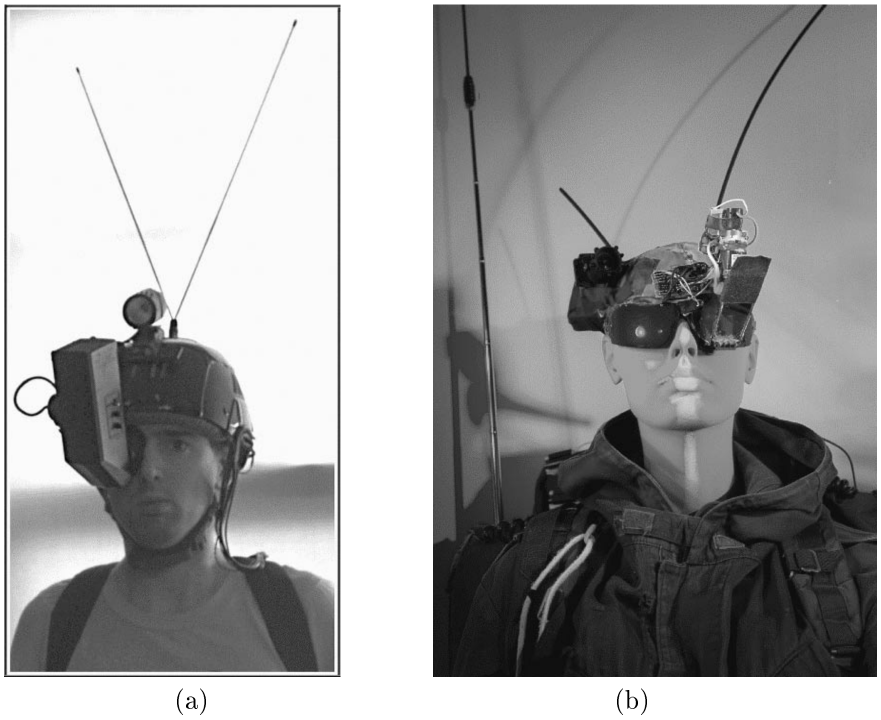

Fig. 1. Early embodiments of the author's original "photographer's assistant" application of Personal Imaging. (a) Author wearing WearComp2, an early 1980s backpack-based signal processing and personal imaging system with right eye display. Two antennas operating at different frequencies facilitated wireless communications over a full-duplex radio link. (b) WearComp4, a late 1980s clothing-based signal processing and personal imaging system with left eye display and beam splitter. Separate antennas facilitated simultaneous voice, video, and data communication.

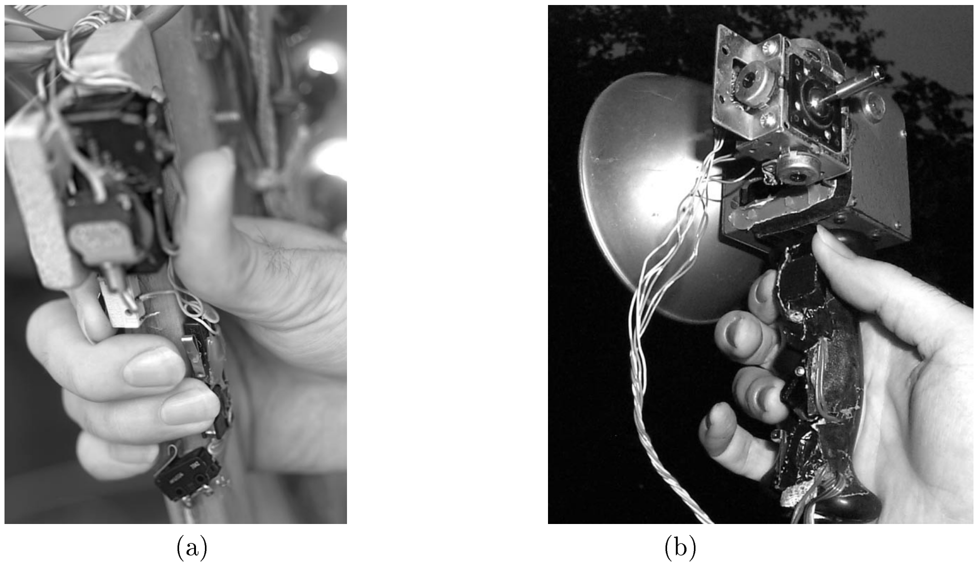

Fig. 2. Some of author's early keyer inventions from the 1970s ("keyboards" and "mice") for WearComp: (a) input device comprising pushbutton switches mounted to a wooden lamp pushbroom hand-grip; (b) input device comprising five microswitches mounted to the handle of an electronic flash. A joystick (controlling two potentiometers), designed as a pointing device for use in conjunction with the WearComp project, is also present.

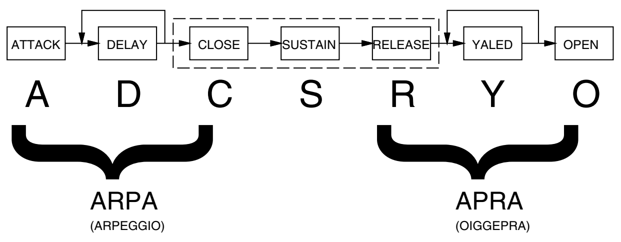

Fig. 3. **The seven stages of the keypress: A** Attack is the exact instant when the first switch is measurably pressed. (e.g. when its common moves away from its first throw if it is a double throw switch, or when the first switch is closed if it is a single throw switch). **D** Delay is the time between Attack and when the last switch of a given chord has finished being pressed. Thus Delay corresponds to an arpeggiation interval (ARPA, from Old High German harpha, meaning harp, upon which strings were plucked in sequence but continued to sound together). This Delay may be deliberate and expressive, or accidental. **C** Close is the exact instant at which the last key of a desired chord is fully pressed. This Closure of the chord exists only in the mind (in the first brain) of the user, because the second brain (e.g. the computational apparatus, worn by, attached to, or implanted in the user) has no way of knowing whether there is a plan to, or plans to, continue the chord with more switch closures, unless all of the switches have been pressed. **S** Sustain is the continued holding of the chord. Much as a piano has a sustain pedal, a chord on the keyboard can be sustained. **Y** Yaled is the opposite of delay (yaled is delay spelled backwards). Yaled is the time over which the user releases the components of a chord. Just as a piano is responsive to when keys are released, as well as when they are pressed, the keyboard can also be so responsive. The Yaled process is referred to as an APRA (OIGGEPRA), e.g. ARPA (or arpeggio) spelled backwards. **O** Open is the time at which the last key is fully (measurably) released. At this point the chord is completely open and no switches are measurably pressed.

There are seven stages associated with pressing a combination of keys. (See Fig 3.)

It should be noted that the Close, Sustain, Release progression exists only in the firstbrain of the user, so any knowledge of the progression from within these three stages must be inferred, for example, by the time delays.

Arbitrary time constants can be used to make the keyboard very expressive, e.g. characters can be formed by pressing keys for different lengths of time. Indeed, a single key alone could be used to tap out Morse code, so that only one key would really be needed, if we were willing to use arbitrary timing information. Two keys would give the iambic paddle effect, similar to that described in a Jan. 12, 1972 publication, by William F. Brown, U.S. Pat. No. 3757045, which was further developed in U.S. Pat. No. 5773769, so that there would be no need for a heavy base (it could thus be further adapted to be used while worn).

Another example of time dependent keyboards include those with a Sustain feature, such as those with the well–known "Typematic" (auto repeat) function of most modern keyboards. A key held down for a long time behaves differently than one pressed for a short time. The key held down for a short time produces a single character, whereas the key held down for a long time produces a plurality of the same character.

Some problems arise with time dependent keying, however. For example, a novice user typing very slowly may accidentally activates a timing feature. Although many input devices (e.g. ordinary keyboards and mice, as well as ordinary iambic paddles) have user adjustable timing constants, the need to adjust or adapt to these constants is an undesirable feature of these keyboards.

Moreover, there are problems and issues of velocity sensing, which is itself a timing matter. Some problems associated with velocity sensing include the necessity of selecting a switch with less deadband zone ("snap") than desired, for the desirable amount of tactile feedback. There were some other undesirable attributes of the velocity sensing systems, so in this paper, the non–velocity sensing version will be described for simplicity.

Without using any timing constants whatsoever and without using any velocity sensing, nor Sustain, nor measurement of the timing in the Delay and Yaled stages), (Fig 3) a very large number of possible keypresses can still be attained.

Consider first, for simplicity, a two key keyboard. There are four possible states: 00 when no keys are pressed, 01 when the least significant key (LSK) is pressed, 10 when the most significant key (MSK) is pressed, and 11, when both are pressed. It is desired to be able to have a rest position when no characters are sent, so 00 is preferably reserved for this rest state, otherwise the keyboard would be streaming out characters at all times, even when not in use.

In a dynamic situation, keys will be pressed and released. Both keys will be pressed at exactly the same time, only on a set of measure zero. In other words, if we plot the time the LSK is pressed on the abscissa, and the time that the MSK is pressed on the ordinate, of a graph (See Fig 4), each keypress will be a point, and we will obtain a scatterplot of keypresses in the plane. Simultaneity exists along the line $t_0 = t_1$, and the line has zero measure within the plane. Therefore, any symbol that requires simultaneous pressing of both keys (or simultaneous release of both) will be inherently unreliable, unless we build in some timing tolerance. Timing tolerances require timing information, such as a timing constant or adaptation, so for now, for simplicity, let us assume that such timing tolerances are absent. Therefore, let us only concern ourselves with whether or not the key presses overlap, and if they do, let us only concern ourselves with which key was pressed first, and which was released first.

This limitation greatly simplifies programming (e.g. for programming on a simple 6502 microprocessor or the like), and greatly simplifies learning, as the pace from novice to expert does not involve continually changing timing parameters and various other subjectively determined timing constants. Accordingly, without any timing constants or timing adaptation, we can, with only two switches, obtain six possible unique symbols, not including the Open chord (nothing pressed). (See Fig 5.)

The operation of the cybernetic keyer is better understood by way of a simple example, illustrated in Fig 6.

The timespace graph of Fig 4 is really just a four dimensional time space collapsed onto two dimensions of the

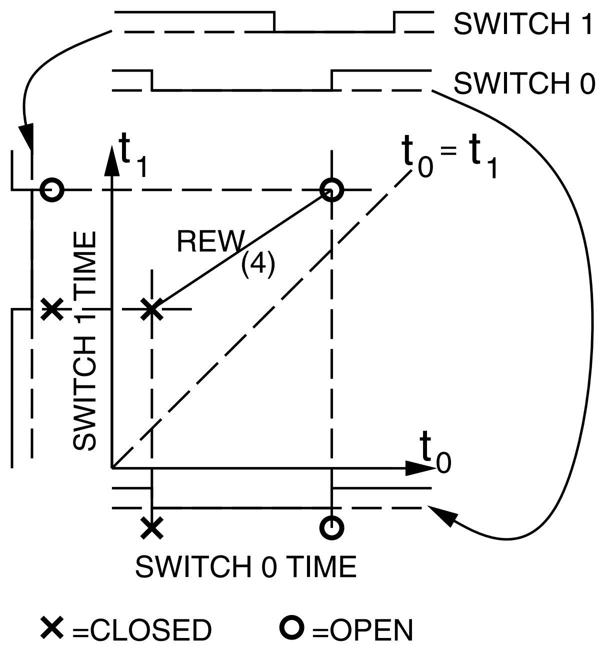

Fig. 4. **Cybernetic keyer timing:** Two keys would be pressed or released at exactly the same time, only on a set, denoted by the line $t_0 = t_1$, which has measure zero in the (t_0, t_1) plane, where t_0 is the time of pressing or releasing of SWITCH 0, and t_1 is the time of pressing or releasing of SWITCH 1. To overcome this uncertainty, the particular meaning of the chord is assigned based ordinally, rather than on using a timing threshold. Here, for example, SWITCH 0 is pressed first and released after pressing SWITCH 1 but before releasing SWITCH 1. This situation is for one of the possible symbols that can be produced from this combination of two switches. This particular symbol will be numbered (4) and will be assigned the meaning of REW (Rewind).

page. Accordingly, we can view any combination of key presses that involves pressing both switches within a finite time, as a pair of ordered points on the graph. There are six possibilities. Examples of each are depicted in Fig 7.

IV. The pentakeyer

With three switches instead of two, there are many more combinations possible. Even if the three switches are not velocity sensing (e.g. if they are only single throw switches), there are still 51 combinations, which can be enumerated as follows:

- Choose any one of the three switches (one symbol each)
- Choose any pair of switches (e.g. omit any one of the three switches from the chord). For each of these three choices, there are four possible symbols (corresponding to the symbols 3 through 6 of Fig 5).
- Using all three switches, at the ARPA (arpeggio, Fig 3) stage:
  - there are three choices for First switch;
  - once the first switch is chosen, there remains the question as to which of the remaining two will be pressed Next;
  - then there is only one switch left, to press Last.

Thus at the ARPA stage, there are $3*2*1 = 6$ different ways of pressing all three switches. At the APRA (oiggepra, Fig 3) stage, there are an equal number of ways of releasing

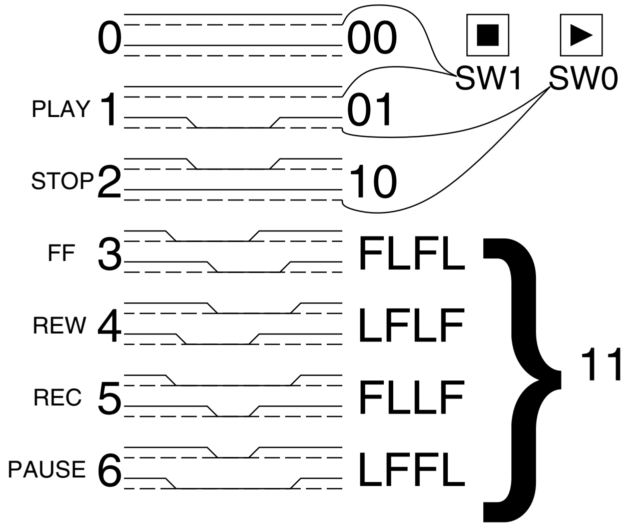

Fig. 5. **The cybernetic keyer:** Timing information is depicted as dual traces: SWITCH 0 is depicted by the bottom trace and SWITCH 1 by the top trace. The zeroith symbol 00 depicts the open chord (no switches pressed). The first symbol 01 depicts the situation in which only SWITCH 0 is pressed. The second symbol 10 depicts the situation in which only SWITCH 1 is pressed. The third through sixth symbols 11 arise from situations in which both switches are pressed and then released, with overlap. The third symbol FLFL depicts the situation in which SWITCH 1 is pressed First, switch 0 is pressed Last, SWITCH 1 is released First, and switch 0 is released Last. Similarly LFLF denotes Last First Last First (fourth symbol). FLLF denotes the situation in which SWITCH 1 is held down while SWITCH 0 is pressed and released (fifth symbol). LFFL denotes the situation in which SWITCH 0 is held down while SWITCH 1 is pressed and released (sixth symbol). The zeroith through sixth symbols are denoted by reference numerals 0 through 6, respectively. Each of the active ones (e.g. other than the Open chord, 0) are given a meaning in operating a recording machine, with the functions PLAY, STOP, FastForward (FF), REWind, RECord, and PAUSE.

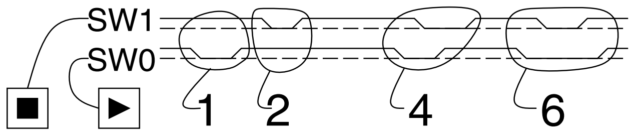

Fig. 6. **Cybernetic Keyer timing example:** In this example, the top trace denotes SWITCH 1, and the bottom trace SWITCH 0. Initially, SWITCH 0 is pressed and then SWITCH 1 is pressed. However, because there is no overlap between these switch pressings, they are interpreted as separate symbols (e.g. this is not a chord). The separate symbols are 1 (PLAY) and 2 (STOP) respectively. This results in the playing of a short segment of video which is then stopped. Then, a little while later, SWITCH 0 is pressed and then SWITCH 1 is pressed. However, because there is now overlap, this action is considered to be a chord. Specifically it is an LFLF (Last First Last First) chord, which is interpreted as symbol number 4 (REWIND). A REWIND operation on a stopped system is interpreted as high speed rewind. A short time later, SWITCH 0 is held down while SWITCH 1 is pressed briefly. This action is interpreted as symbol number 6 (PAUSE). Since PAUSE would normally be used only during PLAY or RECORD, the meaning during REWIND is overloaded with a new meaning, namely slow down from high speed rewind to normal speed rewind. Thus we have full control of a recording system with only two switches, and without using any time constants as might arise from other interfaces such as the iambic Morse code keyers used by ham radio operators.

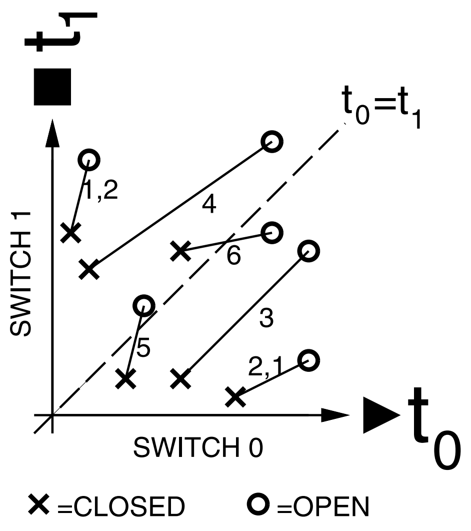

Fig. 7. **Cybernetic keyer timings:** The symbol "X" denotes pressing of the two keys, and exists in the first pair of time dimensions, t_0 and t_1. The releasing of the two keys exists in the first second pair of dimensions, which, for simplicity (since it is difficult to draw the four dimensional space on the printed page), are also denoted t_0 and t_1, but with the symbol "O" for Open. Examples of symbols 3 through 6 are realized. Two other examples, for when the switch closures do not overlap, are also depicted. These are depicted as 1,2 (symbol 1 followed by symbol 2) and 2,1 (symbol 2 followed by symbol 1).

these three switches that have all been pressed. Thus there are six ways of pressing, and six ways of releasing, which gives $6*6=36$ symbols that involve all three switches. Therefore, the total number of symbols on the three switch keyer is $3+12+36=51$. That's a sufficient number to generate the 26 letters of the alphabet, the numbers 0 through 9, the space character, and four additional symbols.

Uppercase and control characters are generated by using the four additional symbols for SHIFT, CONTROL, etc., of the letter or symbol that follows. Thus the multiplication sign is SHIFT followed by the number 8, and the at sign is SHIFT followed by the number 2, and so on.

It is preferable to have all the characters be single chords, so that the user gets one character for every chord. Having a separate SHIFT chord would require the user to remember state (e.g. remember whether the SHIFT key was active), and would also slow down data entry.

Accordingly, if a fourth switch is added, we can:

- Choose any one of the four switches (one symbol each);
- Choose any pair of switches. For each of these $\frac{4!}{2(4-2)!}=6$ choices, there are four possible symbols (corresponding to the symbols 3 through 6 of Fig 5);
- Choose any three switches (e.g. omit any one of the four switches from the chord). For each of these 4 choices, form the chord in any of the $3*2*1=6$ possible ways, and unform the chord in any of six possible ways, giving $6^2=36$ ways to create and uncreate the chord of the three chosen switches, as described in the three switch example above;
- Using all four switches, at the ARPA (arpeggio) stage:
  – there are four choices for First switch;
  – once the first switch is chosen, there remains the question as to which of the remaining three switches will be pressed Second;
  – once the second switch is chosen, there remains the question as to which of the remaining two switches will be pressed Third;
  – then there is only one switch left, to press Last.

Thus at the ARPA stage, there are $4*3*2*1=4!=24$ different ways of pressing all four switches. At the APRA (oiggepra) stage, there are an equal number of ways of releasing these four switches that have all been pressed. Thus there are twenty four ways of pressing, and twenty four ways of releasing, which gives $24*24=576$ symbols that involve all four switches.
Therefore, the total number of symbols on the four switch keyer is

$$\frac{4!}{1!(4-1)!}(1!)^2+\frac{4!}{2!(4-2)!}(2!)^2+\frac{4!}{3!(4-3)!}(3!)^2+\frac{4!}{4!(4-4)!}(4!)^2$$

$$=4*1^2+6*2^2+4*6^2+1*24^2=748. \qquad (1)$$

That's a sufficient number to generate the 256 ASCII symbols, along with 492 additional symbols which may be each assigned to entire words, or to commonly used phrases, such as a sig (signing off) message, a callsign, or commonly needed sequences of symbols. Thus a callsign like "N1NLF" is a single chord. A commonly used sequence of commands like ALT 192, ALT 255, ALT 192, is also a single chord. Common words like "the", "and", etc., are also single chords.

The four switches can be, one each associated with the thumb, and three largest fingers, leaving out the smallest finger. Claude Shannon's information theory, however, suggests that if we have a good strong clear channel, and a weaker channel, that we can get additional error free communication by using both the strong and weak channels than we can by using only the strong channel. Therefore, we can and should use the weak (smallest) finger, for at least a small portion of the bandwidth, even though the other four will carry the majority of the load. Thus, referring to Fig 2, especially Fig 2(b), we can see that there are four strong double throw switches for the thumb and three largest fingers, and a fifth smaller switch having a very long lever for the smallest finger. The long lever makes it easy to press this switch with the weak finger but at the expense of speed and response time. In fact, each of the five switches has been selected specifically knowing the strength and other attributes of what will press it. This design gives rise to the pentakeyer.

The result in (1) can be generalized. The number of possible chords for a keyer with N switches, having only Single Throw (ST) switches, and not using any looping back at either the Delay or Yaled (Fig 3) stages of chord development, is:

$$\sum_{n=1}^{n=N}\frac{N!}{n!(N-n)!}(n!)^2 \qquad (2)$$

Equation 2 simplifies to:

$$\sum_{n=1}^{n=N} \frac{N!n!}{(N-n)!} \quad (3)$$

Thus the pentakeyer gives us 5+40+360+2880+14400 = 17685 possible chords, without the use of any loopback, velocity sensing, or timing constants.

V. Redundancy

The pentakeyer provides enough chords to use one to represent each of the most commonly used words in the English language. There are, for example, enough chords to represent more than half the words recognized by the UNIX "spell" command with a typical /usr/share/lib/dict/words having 25143 words.

However, if all we want to represent is ASCII characters, the pentakeyer gives us $17685/256 > 69$, e.g. more than 69 different ways to represent each letter. This suggests that, for example, we can have 69 different ways of typing the letter "a", and more than 69 different ways of typing the letter "b", and so on. In this way, we can choose whichever of these ways follows most conveniently in a given chord progression.

In using most musical instruments, there are multiple ways of generating each chord. For example, in playing the guitar, there are at least two commonly used "G" chords, both of which sound quite similar. The choice of which "G" to use depends on which one is easiest to reach, based on what chord came before it, and what chord will come after it, etc.. Thus the freedom in having two different realizations of essentially the same chord makes playing the instrument easier.

Similarly, because there are so many different ways of typing the letter "a", the user is free to select the particular realization of the letter "a" that's easiest to type when considering whatever came before it and whatever will come after it. Having multiple realizations of the same chord is called chordic redundancy. Rather than distributing the chordic redundancy evenly across all letters, more redundancy is applied where it is needed more, so that there are more different ways of typing the letter "a" than there are of typing the letter "q" or "u". Part of this reasoning is based on the fact that there are a wide range of letters that can come before or after the letter "a", whereas, for example, there is a smaller range of, and tighter distribution on, the letters that can follow "q", with the letter "u" being in the center of that relatively narrow distribution.

Redundancy need not be imposed on the novice, e.g. the first–time user can learn one way of forming each symbol, and then gradually learn a second way of forming some of the more commonly used symbols. Eventually, an experienced user will learn several ways of forming some of the more commonly used symbols.

Additionally, some chords are applied (in some cases even redundantly) to certain entire words, phrases, expressions, and the like. An example with timing diagrams for a chordic redundancy based keyer is illustrated in Fig 8

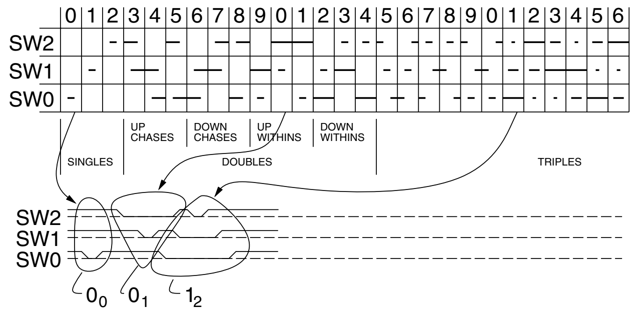

Fig. 8. Example of Keyer with a functional chordic redundancy generator or keyer having functional chordic redundancy. This keyer is used to type in or enter the numbers from 0 to 9 using three single throw switches. Each symbol (each number from 0 to 9) may be typed in various ways. Thus if we wish to type "001", we can do this as follows: first press and release switch SW0, to obtain symbol 0_0 (the zeroith embodiment of symbol 0); then to speed up the process (rather than press the same switch again) we press switch SW1 while holding SW2, to obtain symbol 0_1 which is another realization of the symbol 0; we then choose a realization of the symbol 1, namely 1_2, that does not begin with switch SW2. Thus before the chord for symbol 0_1 is completely released (e.g. at the Yaled stage), we begin entering the chord for symbol 1_2, starting with the available switch SW0.

This approach, of having multiple chords to choose from, in order to produce a given symbol, is the opposite of an approach taken with telephone touchpad style keyboards in which each number could mean different letters. In U.S. Pat. No. 6,011,554, issued January 4, 2000, assigned to Tegic Communications, Inc. (Seattle, WA), King; Martin T. (Vashon, WA); Grover; Dale L. (Lansing, MI); Kushler; Clifford A. (Vashon, WA); Grunbock; Cheryl A. (Vashon, WA) describe a disambiguating system in which an inference is made as to what the person might likely be trying to type. A drawback of this Tegic system is that the user must remain aware of what the machine thinks he or she is typing. There is an extra cognitive load imposed on the user, including the need to be constantly vigilant that errors are not being made. Using the Tegic system is a bit like using command line completion in Emacs. While it allegedly purports to speed up the process, it can, in practice, slow down the process by imposing an additional burden on the user. In some sense, the Tegic system is a form of anti–redundancy, giving the user less flexibility. For example, forming new words (not in the dictionary) is quite difficult with the Tegic system, and when it does make mistakes, they are harder to detect because the mistakes get mapped onto the space of valid words.

Indeed, chordic redundancy (choice) is much more powerful than anti–redundancy (anti–choice) in how characters can be formed.

VI. Ordinally conditional modifiers

A modifier key is a key that alters the function of another key. On a standard keyboard, the SHIFT key modifies other letters by causing them to appear capitalized. The SHIFT key modifies other keys between two states, namely a lowercase state and an uppercase state.

Another modifier key of the standard keyboard is the

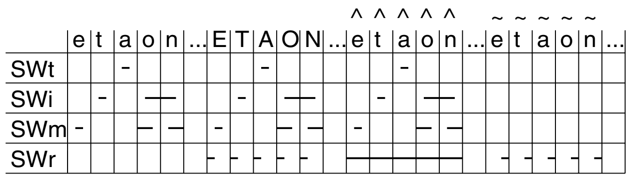

Fig. 9. Example of keyer with ordinally conditional modifier. Letters are arranged in order of letter frequency starting with the letter "e" which is the most commonly used letter of the alphabet. Each of the 26 letters, the ten numbers, and some symbols are encoded with the 51 possible chords that can be formed from 3 switches, a middle finger switch, SWm, an index finger switch, SWi, and a thumb switch, SWt. (The more common letters such as e, t, a, etc., are also encoded redundantly so that there is more than one way to enter, for example, the letter "e".) A ring finger switch, SWr, is the ordinally conditional modifier. If SWr is not pressed, an ordinary lowercase character is assumed. If a chord leads with SWr, the character is assumed to be an uppercase character. If the chord is entered while holding SWr the character is assumed to be a control character. If a chord trails with SWr, it is assumed to represent a meta character. The ordinally conditional modifier is also applied to numbers to generate some of the symbols. For example, an exclamation mark is entered by leading with SWr, into the chord for the number 1.

control key. A letter key pressed while the control key is held down is modified so that it becomes a control character. Thus the letter "a" gets changed to "Â" if it is pressed while the control key is held down.

With the cybernetic keyer, an approach is to have a modifier that is ordinally conditional, so that its effect is responsive to where it is pressed in the chord. See Fig 9 for an example of how a four key ordinally conditional modifier is implemented.

VII. Rollover

One reason chording keyboards can be slow is that they often don't provide rollover. A regular QWERTY... keyboard allows for rollover. A typical 1984 IBM MODEL M keyboard, for example, will be responsive to any key while the letter "q" is held down. When the letters "q" and "w" are held down, it is responsive to most keys (e.g. all those except keys in the q and w columns). When the letters "q", "w", and "e" are held down, it is responsive to other keys except from those three columns. When "q", "w", "e", and "r" are held down, it is still responsive to keys in the right hand half of the keyboard (e.g. keys that would ordinarily be pressed with the right hand). Only when five keys are held down, does it stop responsing to new keypresses. Thus the MODEL M has quite a bit of rollover. This means that one can type new letters before finishing the typing of previous letters. This ability to have overlap between typing different letters allows a person to type faster because a new letter can be pressed before letting go of the previous letter.

Commercially available chording keyboards such as the Handykey Twiddler and the BAT don't allow for rollover. Thus typing on the Twiddler or BAT is a slower process.

A goal of the cybernetic keyer is to be able to type much more quickly. Therefore, an important feature, is the tradeoff between loopbacks at the Delay and Yaled stages (Fig 3), and rollover. If we decide, by design, that there will be no loopback at the Delay or the Yaled stages, we can assume that a chord has been committed to at the Release stage. Thus once we reach the Release stage, we can begin to accept another chord, so long as the other chord does not require the use of any switches that are still depressed at the Release stage. However, because of the sixty nine–fold chordic redundancy, it is arranged that, for most of the commonly following letters, there exists at least one new chord that can be built on keys not currently held down, at the Release stage.

A. Example of rollover on a cybernetic keyer

With reference to the two switch keyer, suppose we press SWITCH 1, then press SWITCH 0, and then release SWITCH 1. We are now at the Release stage, and can enter a new command with SWITCH 1, since we know that there will be no Yaled loopback (e.g. since we know that the chord will not involve pressing SWITCH 1 again). Thus pressing SWITCH 1 again can be safely used as a new symbol, prior to releasing SWITCH 0. In this way, symbols can rollover (overlap).

VIII. Further increasing the chordic redundancy factor: A more expressive keyer

The number of possible chords can be increased from 17685 to 7 + 84 + 1260 + 20160 + 302400 + +3628800 + 25401600 = 29354311 by simply adding three switches at the thumb position. This provides more than twenty nine million symbols, which is enough that each word in the English language could be represented in approximately a thousand different ways. This degree of chordic redundancy could provide for some very fast typing, if this many chords could be remembered. However, rather than increasing the number of switches, it is preferable to increase, instead, the expressivity of each one.

The pentakeyer is a crude instrument that lacks an ability to provide for tremendous "expression" and sensitivity to the user. It fails to provide a rich form of Humanistic Intelligence in the sense that the feedback is cumbersome, and not continuous. A guitar, violin, or real piano (e.g. an old fashioned mechanical piano, not a computerized synthetic piano data input keyboard), provides a much richer user experience because it provides instant feedback, and the user can experience the unique response of the instrument.

Even using both rails of each switch (e.g. double throw and even velocity sensing) still fails to provide a truly expressive input device. Accordingly, a better keyer was built from continuous transducers in the form of phonographic cartridges salvaged from old record players. These devices provide continuous flow of pressure information along two axes (each phono cartridge is responsive to pressure along two independent axes at right angles to one another, originally for playing stereo sound recordings from the grooves of a record). The dual sensor is depicted in Fig 10, and may comprise either a continuous sensor, or two on–off switches

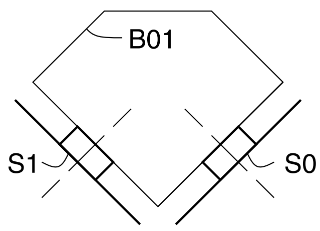

Fig. 10. Dual sensor keyer. Sensors S0 and S1 may be switches or transducers or other forms of sensory apparatus. Sensors S0 and S1 are operable individually, or together, by way of rocker Block B01. Pressing straight down on Block B01 will induce a response from both sensors. A response can also be induced in only one of sensors S0 or S1 by pressing along its axis.

operable on separate axes.

In this embodiment, the keyer described here is similar to the ternary data entry keyboard described by Langley, October 4, 1988, in U.S. Pat. No. 4,775,255, in the sense that there are also to axes of each key, so that a given key can move towards or away from the operator, and has a central "off" position, and a spring detent to make the key return to the central position in the absence of pressure from the finger. An important difference, though, is the fact that the keyboard of U.S. Pat. No. 4,775,255 has no ability to sense how fast, how hard, or how much each switch is pressed, other than just the ternary value of 0, +1, or -1. Also, in the keyer of U.S. Pat. No. 4,775,255, the axes are not independent (e.g. one can't press +1 and -1 at the same time).

The sensor pair of Fig 10, on the other hand, provides two independent continuous dimensions. Thus a keyer made from five such sensor pairs provides a much more expressive input. Even if the transducers are quantized, to binary outputs, there are four possibilities: $0, -1, +1$, and ± 1. With five continuous transducers, one for each finger or thumb position, the user interface involves squeezing out characters, rather than clicking out characters. The input space is a very richly structured ten dimensional timespace, producing ten traces of time–varying signals. This ten dimensional timespace is reduced to discrete symbols according to a mapping based on comparison of the waveforms with reference waveforms stored in a computer system. Since comparison with the reference waveforms is approximate, it is done by clustering, based on various kinds of similarity metrics.

IX. Including one time constant

If we relax the ordinality constraint, and permit just one time constant, pertaining to simultaneity, we can obtain eleven symbols from just two switches. Such a scheme can be used for entering numbers, including the decimal point.

```
    00 (Open chord not used)
0   01
1   10
2   FLFL
3   FLLF
4   FLW
5   LFFL
```

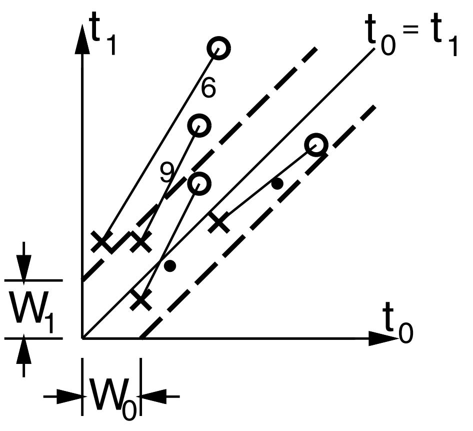

Fig. 11. Example showing Keyer Chords within timing tolerances W_0 and W_1. Time differences within the tolerance band are considered to be zero, so that events falling within the tolerance band defined by W_0 and W_1 are considered to be effectively simultaneous.

```
6   LFLF
7   LFW
8   WFL
9   WLF
.   WW
```

Using this simple coding scheme, the number zero is entered by pressing the LSK. The number one is entered by pressing the MSK. The number four, for example, is entered by pressing the MSK first, then pressing the LSK, and then releasing both at the same time, Within a certain time tolerance for which time is considered the same. (The letter "W" denotes Within tolerance, as illustrated in Fig 11.) The decimal point is entered by pressing both switches at approximately the same time, and releasing both at approximately the same time. The "approximately the same time" is defined according to a band around the line $t_0 = t_1$ in Fig 4. Such a timing band is depicted in Fig 12.

This number system can be implemented either by two pushbutton switches, or by a single vector keyswitch of two components, as illustrated in Fig 10. In the latter case, the entire set of number symbols can be entered with just one switch, e.g. by just one finger. Note that each number involves just a single keypress, unlike what would be the case if one entered numbers using a small wearable Morse code keyer, or the like. Thus the cybernetic chordic keyer provides a much more efficient entry of symbols.

X. Making a conformal keyboard

Wearable keyers are known in the art of ham radio. For example, in U.S. Pat. No. 4,194,085, March 18, 1980, Scelzi describes a "Finger keyer for code transmission". The telegraphic keyer fits over a finger, preferably the index finger, of an operator, for tapping against the operator's thumb or any convenient object. It is used for transmission of code with portable equipment. The keyer is wearably operable when walking, or during other activities.

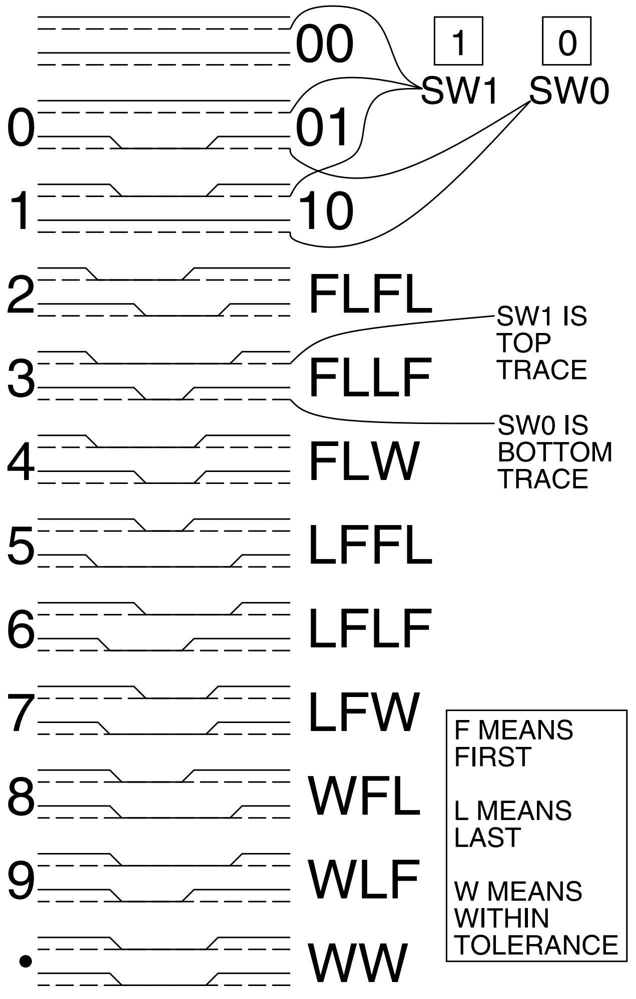

Fig. 12. Chordic Keyer with timing tolerances. In addition to the unused Open chord, there are eleven other chords that can be used for the numbers 0 through 9, and the decimal point.

Keyers, such as previously known keyers, as well as the proposed keyers, such as the pentakeyer, and the continuous ten dimensional keying system, are much easier to use if they are custom made for each user. The most important aspect is getting the hand grip to fit well.

A subject of ongoing work, therefore, is in designing ways of moulding the keyers to fit the individual hand of the wearer. Presently, this is done by dipping the hand in icewater, and draping it with heated plastic material that is formed to the shape of the hand. Once the handpiece is formed, sensors are selected and installed so the keyer will match the specific attributes of the user's hand geometry.

XI. Conclusion

Learning to use the pentakeyer is not easy, just as learning how to play a musical instrument is not easy. The pentakeyer evolved out of a different philosophy, more than twenty years ago. This alternative philosophy knew nothing of so–called "user–friendly" user interface design, and therefore evolved along a completely different path.

Just as playing the violin is much harder to master than playing the TV remote control, it can also be much more rewarding and expressive. Thus if we were only to consider ease of use, we might be tempted to teach children how to operate a television because it is easier to learn than how to play a violin, or how to read and write. But if we did this, we would have an illiterate society in which all we could do were things that were easy to learn. It is the author's belief that a far richer experience can be attained with a lifelong computer interface that is worn on the body, and used constantly for ten to twenty years. On this kind of time scale, an apparatus that functions as a true extension of the mind and body, can result. Just as it takes a long time to learn how to see, or to read and write, or to operate one's own body (e.g. it takes a number of years for the brain to figure out how to operate the body so that it can walk), it is expected that the most satisfying and powerful user interfaces will be learned over many years.

XII. Acknowledgements

Simon Haykin, Woodrow Barfield, Richard Mann, Ruth Mann, Bill Mann, and Steve Roberts (N4RVE), helped much in the way of useful feedback, constructive criticism, etc., as this work has evolved.

Dr. Carter volunteered freely of his time to help in the design of the interface to WearComp2 (a 6502-based wearable computer system of the early 1980s), and Kent Nickerson similarly helped with some of the miniature personal radar units and photographic devices involved with this project throughout the mid 1980s.

Programming of one of the more recent embodiments of the Keyer was done in collaboration with Adam Wozniak, and elements of the recent embodiment borrow code from Wozniak's PICKEY. Students, including Eric Moncrieff, James Fung, and Taneem Ahmed, are continuing with this work.

Additional Thanks to Xybernaut Corp., Digital Equipment Corp., Compaq, etc... for lending or donating additional equipment that made these experiments possible.

XIII. References

Material will be deleted from the manuscript, to make room for the references. Exactly what material will be deleted, as well as references cited, will be responsive to reviewers' comments.