Development and Testing of an Adaptive Computer Keyboard Interface

Kevin Barclay, Dr Samuel Mann,
Peter Brook & Dr Andy Doonan
School of Information Technology & Electrotechnology
Otago Polytechnic
Dunedin, New Zealand
smann@tekotago.ac.nz

ABSTRACT

This paper describes the background and technical development of an Adaptive Computer Keyboard Interface (ACKI). This is a microprocessor device that allows the generation of text information from simple control devices (e.g. the formulation and manipulation of text by movement of joystick). The system is shown to be successful. This paper also describes project plans to further develop the ACKI to the level where it may be considered a commercial product. This will include a second prototype that has more programmable variables and research into usability and human computer interaction.

1. INTRODUCTION

This paper describes the development and testing of a prototype device that has considerable potential for research and commercial development. This device is the Adaptive Computer Keyboard Interface. In short, this device sits between the keyboard and computer, takes information from almost any control device (joystick, keypad, blow-straw etc) and converts it to text strings (Figure 1). While this may have many uses, the intended initial use is in assistive technologies for people with physical disabilities who cannot operate a standard keyboard. This includes those with severe disabilities and localised damage such as OOS.

Current assistive technology systems are extremely expensive or custom made. Neither are user programmable meaning that for each new user or input device a new program has to be written. The ACKI overcomes these problems. Advantages of the system include:

- Platform independence: will work on any computer system
- Transportable: no software is required on the client computer, the ultimate 'plug and play'
- Easily programmable: a simple setup wizard assigns text strings to positions of the input devices. The microprocessor embedded in the ACKI is non-volatile meaning it maintains its settings until it is re-assigned. This wizard operates through the same keyboard cable so no extra set up is

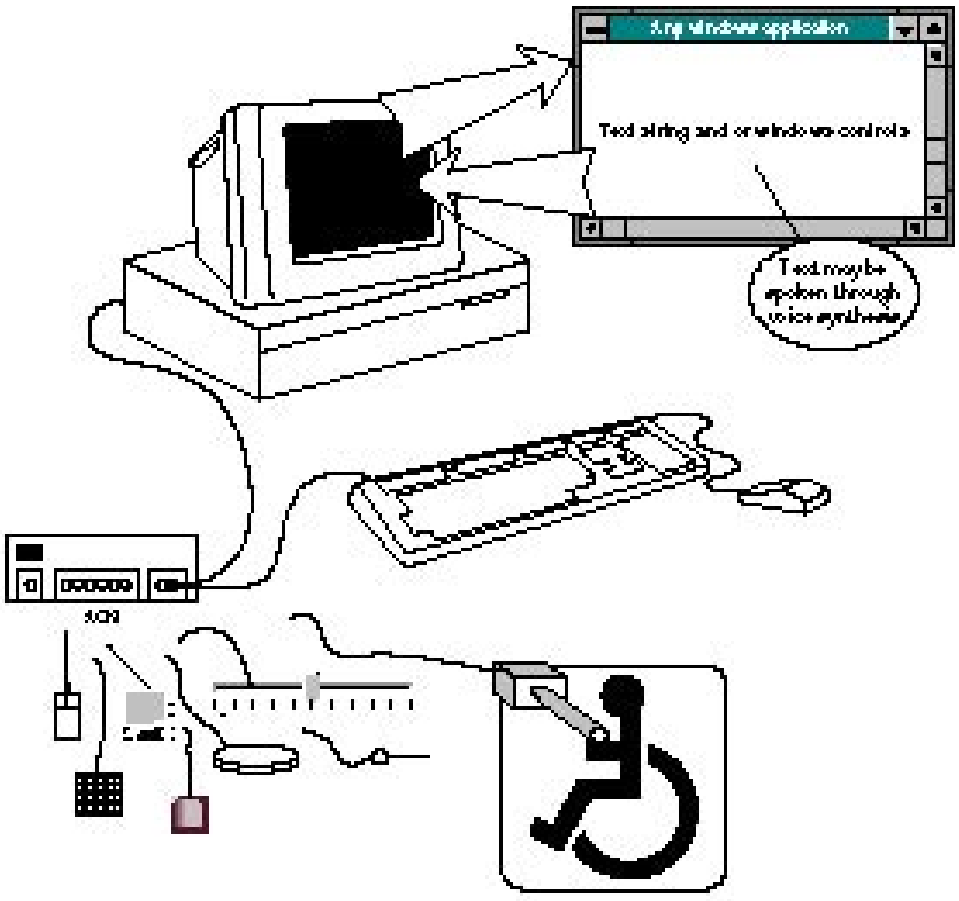

Figure 1:

ACKI sits between the keyboard and computer and accepts input from almost any other device from joysticks and dials to air-pressure pads. It can be very simply programmed to send text strings to the computer.

required

- Input device independence: Any analogue control device can be plugged into the ACKI and positions assigned to text strings.

2. DEVELOPMENT

3.1 Technical Development

The ACKI project has several aims, however at its core is an issue about using a microcontroller to emulate the functionality of a keyboard.

It is interesting that the keyboard-pc link has been one of the most stable constants in modern computing hardware. As computer users, we press keys on the keyboard and single characters magically appear on the screen. However, when one steps back from it and views the process objectively, it seems remarkably clumsy.

Aside from being clumsy, another interesting thing about keyboards is that they are essentially platform

independent – it doesn't matter whether they are connected to a system which is running DOS, Windows 3.X, Windows 9X, OS2 or LINUX etc. The translation of thumps on the keys to characters on the screen is thoroughly standardised and encapsulated in the machine hardware (BIOS).

By programming a microcontroller to behave like a keyboard and giving it access to the keyboard port on a computer essentially abstracts the keyboard interface back to a programming interface where we can create enhanced functionality, but importantly also have this as a (Operating System) platform independent device.

The Adaptive Computer Keyboard Interface (ACKI) project is a research and development project whose major aims are to:

1: Implement a microcontroller based keyboard interface.
2: Add some enhanced functionality to the unit so that external device states can be used to choose, and trigger, the sending of 'character data' from the ACKI unit to the computer.
3: Allow for flexibility, variation and choice of the 'external devices'.
4: Make the device programmable by the user so that 'character data' sent by the unit to the computer can be easily modified.
5: Make the user-defined 'character data' non-volatile.
6: Allow a traditional keyboard to work in conjunction with the ACKI.

Having decided upon the major system requirements, the ACKI project quickly decomposed into four main development areas:

1: Device analog interface.
2: Serial driver interface into the keyboard I/O lines.
3: Digital storage solution for non-volatile character data in the ACKI unit.
4: Programming interface to change character data

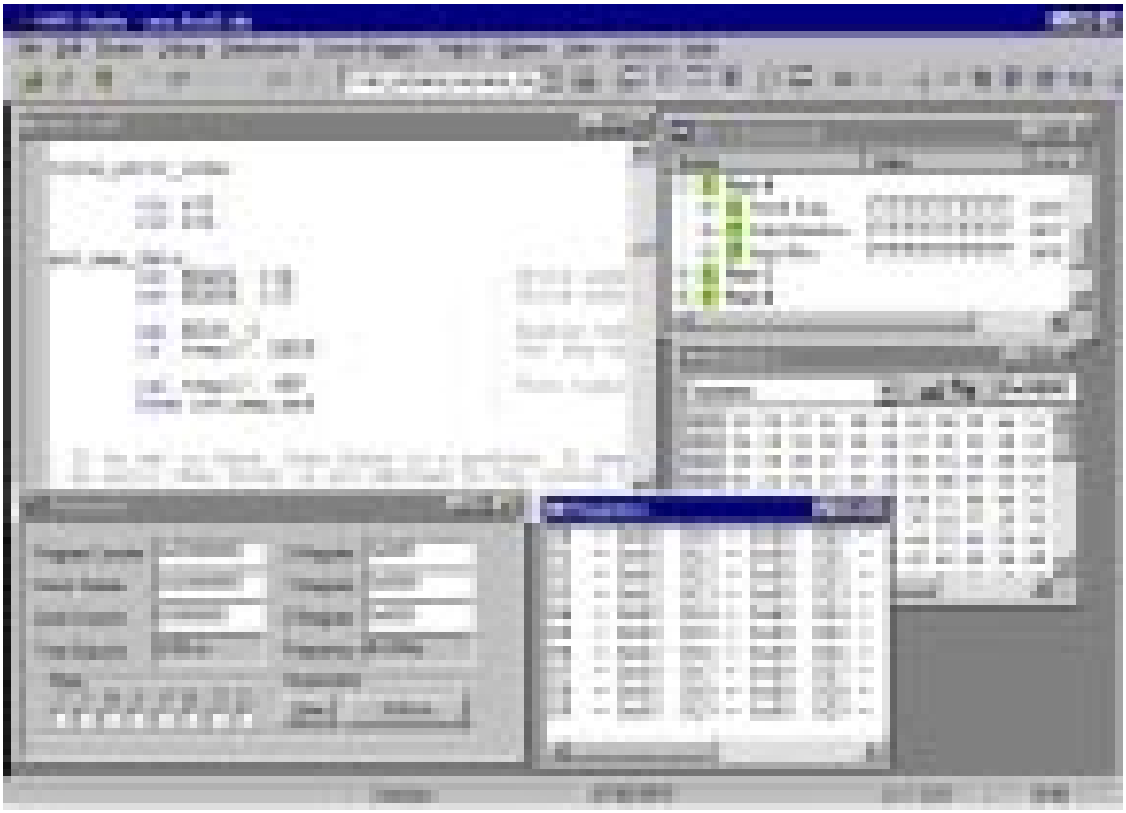

Figure 2:

The system was developed on ATMEL microprocessors using AVR studio.

stored in the ACKI.

3. DEVICE ANALOG INTERFACE

The intention of the ACKI development is to be able to cater for almost any device, however, some assumptions are required:

- The ACKI unit would have two external devices connected to it at any time, and these would be used together to choose which of the stored strings of character data would be sent to the PC, and also trigger that sending process.
- The ACKI unit had to cater for storage of approximately 64 different strings of character data, each string a maximum of about 200 characters. It followed then that each of the external devices would need to be able to represent eight different states, so that between the two devices 64 unique inputs could be generated to select the required character data to be sent to the PC.

It was decided to proceed using two different types of external devices for the prototype. One type would be a conventional joystick where the eight points of the compass would be used, and the other would be an array of eight push buttons. However, any combination of the input devices would need to work together with only minor (or preferably no) intervention by the user.

Pure analog inputs from the joysticks are converted with an analog to digital converter (ADC) to quantify measurements and evaluate the physical position of the joystick. This also allowed frugality with a limited number of I/O pins on the microcontroller. Even though the push buttons could have been read directly on microcontroller port lines, a hardware interface was created that used ADCs for both the joystick and push button inputs. This utilized a Texas Instruments chip (TLC542) which had several advantages, but primarily it allowed a six wire interface to the hardware, that had the capacity to sample from the worst case scenario which was both external devices being push buttons i.e. a total of 16 inputs to deal with.

Each ADC chip had the capacity to sample from 11 different input channels, and a separate chip was used for each of the four possible device interfaces, even though only two device interfaces would be used at any given time. The three ADC control lines were placed on a common bus, and each ADC chip was uniquely controlled by its own Chip Select line (CS). This was all done under control of code running in the microcontroller.

Several versions of the device analog interface were built as tests were made, and the design and program progressed.

Figure 3 shows the second version, which was a full implementation of the device interface. It was set up for four ADCs, and was tested fully with both joystick and press button inputs. Complete sample time for a Joystick (2 channels) is 423 microseconds and for a press button device (8 channels) is 1200 microseconds. So the worst-case scenario is 2 press button devices, total sample time being 2435 microseconds. This equates to over 400 complete samples occurring every second.

4. SERIAL DRIVER FOR KEYBOARD

The core functionality of the ACKI project is to be able to connect into the IO lines between a keyboard and the computer, and drive information into them so that it looks like a keyboard is sending data.

The physical interface between the keyboard and computer is a relatively simple interface and well defined. This is not the case for the data that actually

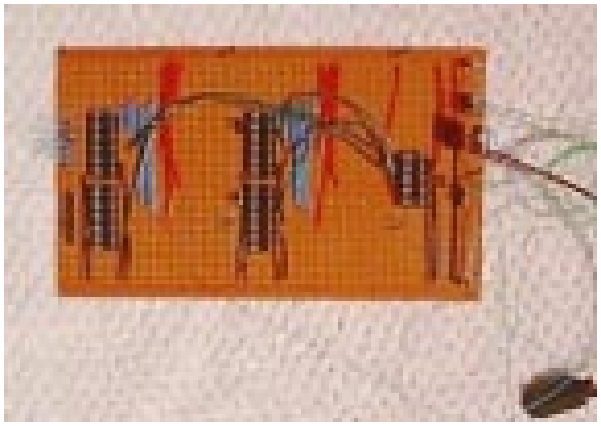

Figure 3:

The Analog-Digital Interface

propagates along the keyboard IO lines. Different sources gave a (slightly) different story about the operation and format of keyboard data, a favourite quote from a Keyboard FAQ where an author said that:

> " getting the AT keyboard to work in my first project was a gut-wrenching, hair-pulling experience."

4.1 The Keyboard Protocols

The first task is simply trying to intercept and read the serial data moving between a keyboard and PC. There are two serial lines of interest: clock and data.

The protocol between the keyboard and PC is an awkward one, because it is bi-directional. Each device can send information to the other, but the keyboard always runs the clock. So if the PC wants to send messages to the keyboard, it must first signal to the keyboard to start running the clock, and then it manipulates the data line so that it will be read on the falling edges of the clock cycle. If the keyboard needs to send data to the host, it simply checks the status of the data and clock lines, and if all is normal will start running the clock and manipulating the data line.

The data is in an 11 bit format, which consists of 1 Start, 8 Data, 1 (Odd) Parity, 1 Stop bit. This was a first major difficulty: as the keyboard data and clock lines were monitored, the data could be reliably identified. Eventually it was discovered that although the keyboard makes 11 clock cycles to clock in 11 bits of data, - as an acknowledgement the PC will pull the clock line low briefly, to show the successful receipt of a character and prevent the keyboard from sending any further data. Thus, there are actually 12 clock cycles, as opposed to 11. The AVR in-built UART was used to talk to a terminal program on a PC via an RS232 serial link, so whenever a key was pushed on the keyboard, the microcontroller would intercept and clock in the data internally and then send it out via the serial link to allow an ASCII representation of the data being displayed on the terminal program.

The codes being sent between the keyboard and PC (referred to as Scan Codes) really represent an (almost) arbitrary mapping of the keys on a keyboard. They have no correspondence to ASCII or any other psuedo-standard. Apparently, when IBM created the original keyboard specification, they wanted the scan-codes that represent individual key-presses to be as 'flexible' as possible, to cope with different character sets used in different countries, -so effectively any key press can be interpreted as any character.

Whenever you push and release an ordinary key, there are actually three scan codes generated. This is because the key-release sequence includes a generic key release code ($F0) and a repetition of the original scan code.

For example, on a standard AT keyboard, the scan code for a 'z' is hex $1A.

	Keyboard Action:	ScanCode
1.	Keypress <z>	Scan Code sent: $1A
2.	KeyRelease <z>	BreakCode sent $F0
3.	Scan Code resent	$1A

The code that a key press generates is the same regardless of whether it is a <shifted> character or not. The difference between a shifted and unshifted character is only that, when you press the shift key, a scan code is sent to the PC which tells it to interpret the following scan codes in its <shifted> state:

	Keyboard Action:	ScanCode
1.	Keypress (left) <shift>	Scan Code sent: $12
2.	Keypress <z>	Scan Code sent $1A
3.	KeyRelease <z>	Break Code sent $F0
4.		Scan Code resent $1A
5.	Keyrelease (left) <shift>	Break Code sent $F0
6.	Scan Code resent	$12

The final complication to this scenario is that some keys generate an extended set of scan codes. This can be seen for keys like <Alt>, <PrintScreen> and the <Windows Key>.

A number of codes exist that the PC can send to the keyboard. These are for changing various keyboard parameters, resetting the keyboard and performing other maintenance functions. Interestingly, the

LEDS on a keyboard (Num Lock, Caps Lock etc) are controlled by these commands, and are not directly switched from the keys on the keyboard.

After many hours of testing, a number of things became apparent. The most important of these was that having been able to read the keyboard data (albeit in a crude un-decoded state) there was some confidence that keyboard data could be simulated by taking control of the clock and data lines. This also meant isolating the actual keyboard during that time so as not to become confused by any signals generated on the clock and data lines.

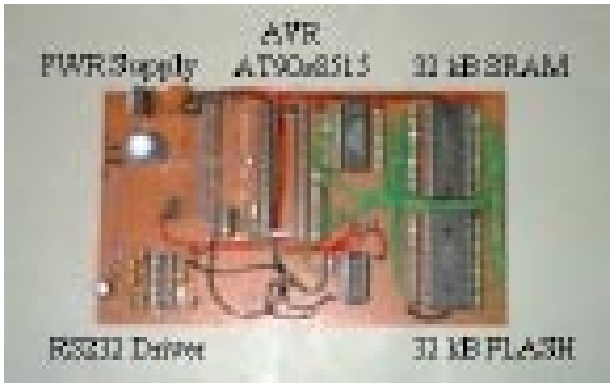

Figure 4: Finished Storage/Microprocessor Board. (Some ICs are not inserted)

4.2 Hardware Interface

The hardware interface for the serial driver had to accomplish two major functions: Be able to drive the clock and data lines under control of the AVR, and be able to isolate the keyboard.

The data and clock lines are both normally held high (+5v) by pull-ups at both the keyboard and PC. This enabled two transistors to pull the lines to ground and generate the clock and data pulses. The isolation of the keyboard was more difficult as the lines (data and clock) essentially had to be treated as bi-directional lines. A pure analog device was used to do the switching - a bi-lateral transmission gate.

The final version of the keyboard interface contains transmission gates for keyboard isolation, transistors for driving the clock and data lines and an extra package of inverting buffers to provide some electrical isolation and also flip the control signal to the transistors (because of an inverting function which occurred as a result of the mode they were used in). The finished product looks remarkably uncomplicated, but is effective, simple and implements a reasonably elegant solution to the problem.

4.3 Software

As well as the hardware interface, a huge effort was spent implementing code to actually control and drive into the unit. Essentially, the basic keyboard protocol had to be emulated and welded together with the hardware. A thorough understanding of the protocol, and plenty of preliminary testing was the key to coming up with a good solution here.

Because the ACKI unit will store strings of text in ASCII representation, a conversion routine was needed to change ASCII codes into numerical codes. This table was implemented at no expense to code space, by placing the table into the ESEG (EEPROM segment) of the AVR memory architecture. The code handles shifted and unshifted alphanumeric characters, as well as a relatively complete set of punctuation. Support for more characters is available by a simple modification to the lookup table.

5. STORAGE SOLUTION

Because of the requirement for the ACKI to be an independent and transportable device, it was necessary to be able to store 16kb of text data in non-volatile memory so that it could be transmitted at any time. ATMEL FLASH memory was chosen for this task (Figure4).

The ATMEL microprocessor used is able to control 64kB of external memory (16 bit address) and so was able to map both SRAM and FLASH devices directly into the processors address space. The SRAM is used as a high-speed buffer during download of changed text information by the user.

By implementing the SRAM buffer, any download errors that occur can be detected and programming of the FLASH aborted. The buffer is also used because

Figure 5:

Typical ASCII strings that the user sets

writing to the FLASH device is moderately time consuming - it is loaded in 64 byte sectors and the programming operation then requires about 10mS. So, without implementing some flow control on the serial programming link, this hardware buffer offers a good solution.

6. SOFTWARE

Although the ACKI is intended as a transportable device, requiring no software on the client computer, some software is needed to configure the device for an individual user. An application was written in Delphi that allows the association of inputs to ASCII text strings (Figure 5). The ASCII characters (up to 255 for each of the 64 positions) can be written and sent to the ACKI in a few minutes. This is also sent to a configuration file for easy editing.

7. INTEGRATION

With all the component prototypes functioning, an integrated design was developed and constructed (Figures 6 and 7).

Figure 6 shows the analog interfaces on the top board microcontroller/storage systems on the lower board. The keyboard/PC is connected at the top right, keypad and joysticks on the top left. The RS232 link is connected midway on the right hand side.

8. TESTING

8.1 Performance

The ACKI device performs as expected. Instead of keys, a series of external controls from ACKI can be used to feed the computer with data. Currently

Figure 6:

The completed circuit boards

Figure 7:

The full prototype

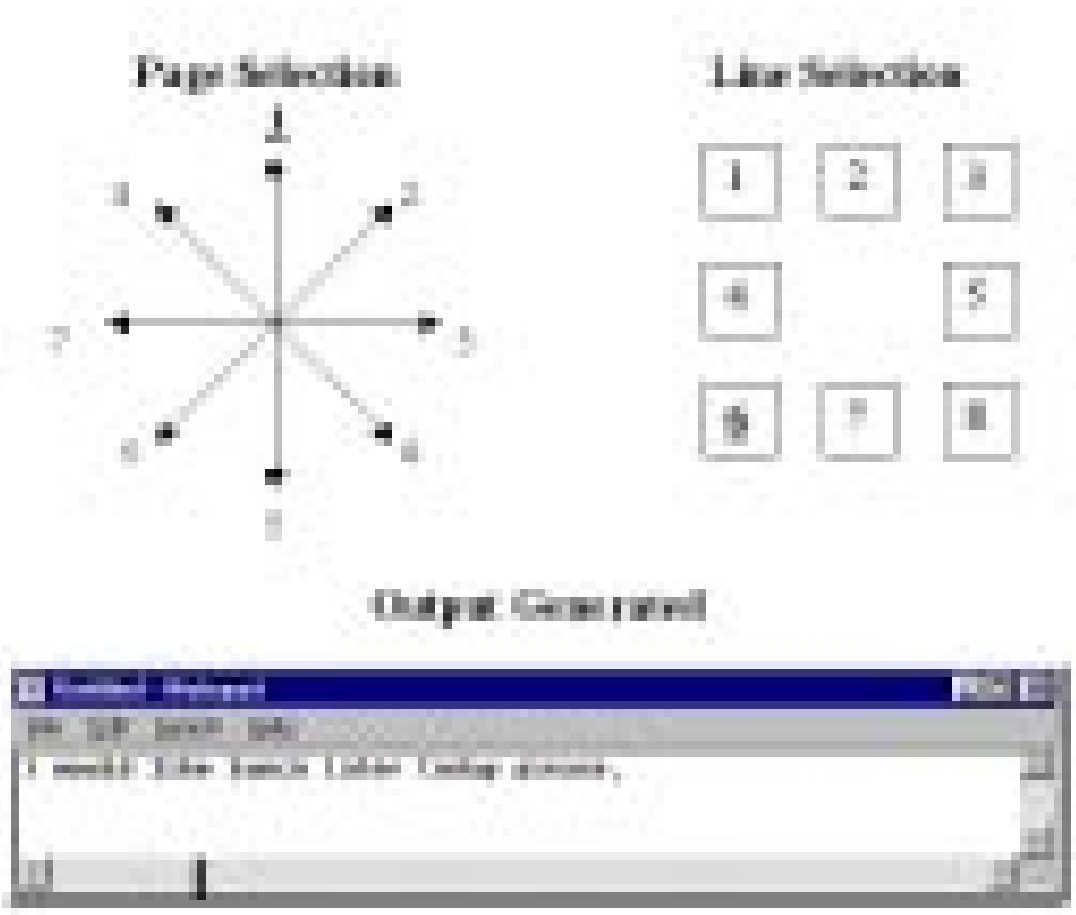

Figure 8:

One line is selected from eight pages of eight lines of text

these controls consist of combinations of joysticks and pushbutton pads, either of which selects from eight options, giving a total of 64 possible strings from which to select.

The ACKI unit was designed to store approximately 64 different strings of character data where each string contains a maximum of 200 characters. The system was also designed to be a transportable and independent device, where data could be transmitted at any time, so it was necessary to be able to store 16kb of text data in non-volatile memory. This means that the predetermined strings can be ready programmed and stored permanently within the ACKI device. Users may then select from eight pages of eight text strings using a combination of two input devices (Figure 8).

9. FURTHER RESEARCH

9.1 Human Computer Interaction

The prototype can now be used for user testing. This will include user and task modelling techniques, along with keystroke modelling and GOMS testing.

Preece *et al.* (1994) described four components, which make up the usability requirements of a system. Learnability, throughput, flexibility and attitude make up these guidelines.

The GOMS method is also a useful construct. The Goals describe the user's goals (the tasks he or she wants to achieve). For example, the user wishes to

notify a caregiver of a need by selecting a specific line of text by using the ACKI device. Operators are the basic physical actions that a user must perform in order to use the system. In the case of the ACKI, the user could move the joystick, press a button on the keypad, or press the joystick trigger. Methods involve a goal being split into smaller tasks, and there may be more than one method of doing so. Using different input devices with which to select ACKI text represented alternative possible methods. For example, to select a text page and then a line of pre-programmed text from the existing ACKI system, a combination of input devices would be used. This describes which method (described above) it is that a user chooses to use. The user can decide to use any two devices from the range of joysticks and keypads. Potential users may be unaware of the alternatives, but these options would be addressed within user documentation and training.

9.2 Second Prototype

A goal of ongoing research is to further develop the system with a second functional prototype suitable for continuing research and commercial application. The first prototype has a limited range of keystrokes available: namely the alphanumeric characters. To allow a full range of input, keystrokes such as backspace and return should be added. To allow capitals, a shift register is needed, perhaps operating like the keyboard caps key.

A second prototype should also parameterise some of the control structures of the device. One limitation of the first prototype is the inability to vary the sequence and timing of positions that results in a 'keystroke'. This could be brought out as a variable, to be programmed from the PC.

9.3 Commercialisation

This project plans to further develop the ACKI to the level where it may be considered a commercial product. Further work will involve an assessment of commercial viability and a business plan. This will also involve consideration of intellectual property and commercial partners. There is also potential for development in other application areas. A simple, transportable device that can act as an interface between sensors and PCs may have many applications beyond keyboard emulation.

ACKNOWLEDGEMENTS

The continued development of this project is being funded by the Otago Polytechnic's Research and Development Committee. Diana Kassabova, Bruce Fergus and Matthew Dooher are working on this project. John Brennan and Peter Church assisted in HCI testing.

REFERENCES

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., Carey, T. (1994). *Human-computer interaction.* Sydney: Addison-Wesley.