Keyboard Mapping and Object-Aware Pointing

Tolga Özuygur, Piet Zwart Institute

tolga at odcaf.com

ABSTRACT

Human brain has a remarkable capacity to map and match input and output surfaces with different scales. Almost every pointer technology we use, except touch screen technology, uses this ability of our brain. For example, when we are using a touchpad, which is an industry standart pointing device on notebook class devices, our brain succesfully maps the 15" screen with a 3" touchpad, allowing us to control the pointer on the screen with precision. With the Keyboard Mapping software that we have developed, we are going to experiment on using a large sized, low resolution, touchpad style interaction and will point out its advantages and disadvantages. The main objective of developing such interface is eliminating the need of physically switching from keyboard to a pointing device in order to increase interaction speed between the computer and the user. Also we will experiment with the interaction style we call Object-Aware Pointing which has been prepared to increase the effectiveness of the Keyboard Mapping interface. Object-Aware Pointing does not strictly need to be used with Keyboard Mapping and further usage of it shall be investigated.

Keywords

Interfacing, input devices, surface input mapping, touch-sensing devices.

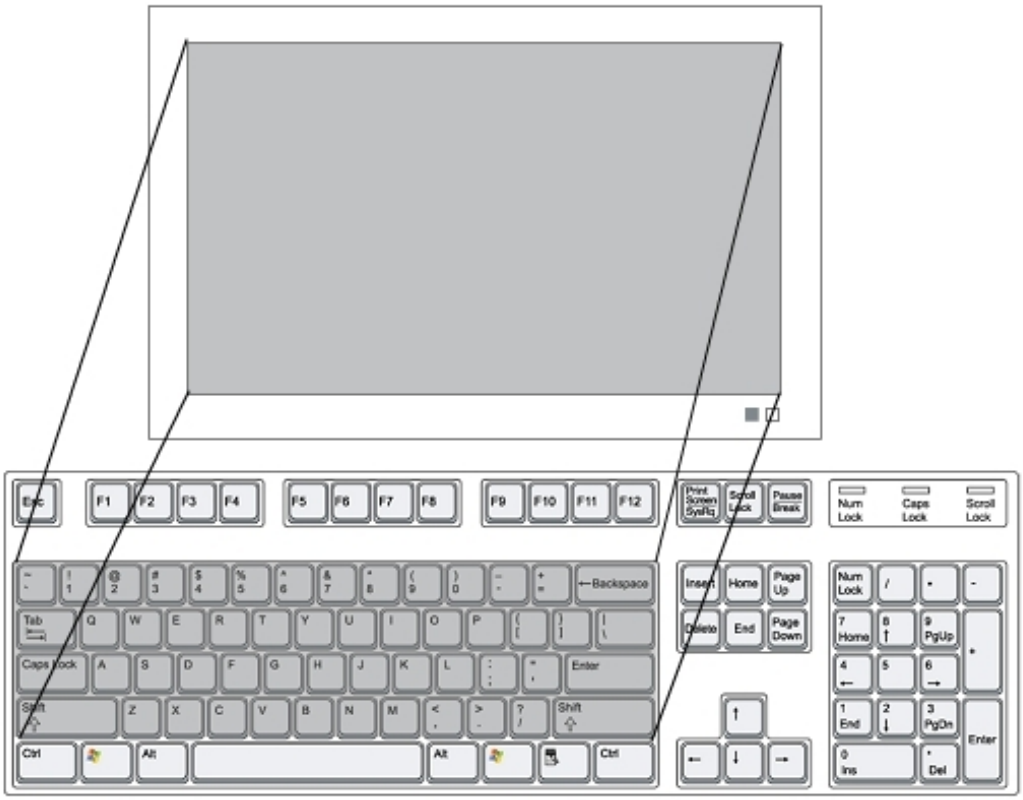

Fig 1. Mapping the Keyboard to the screen coordinates respectively.

INTRODUCTION

Introduction of the keyboard and mouse combination to the world has been a great development in computer technologies. The precision of mouse combined with the keyboard enables user to enter efficient information to a computer system and it is quite easy to learn how to use. Yet with the latest technologies, we have been introduced with many different interaction techniques with an attempt to surpass the weaknesses of the mouse-keyboard combination. One of the weaknesses of this combination, which is the one we will be trying to overcome, is the need of switching between keyboard and the mouse frequently. This action surely is time consuming when analyzed in long-term computer usage. It can also be tiring for the user. The current touch screen style interfaces fix this problem but in the meantime it evades many benefits of using an actual keyboard and mouse. In the Keyboard Mapping technique, we will be focusing on a keyboard that can both be used as a keyboard and as a pointing device. By switching between these two modes, a user can both type and point without the need of moving their hands off the keyboard. The main logic behind this interface design is roughly mapping the keys on the keyboard to the according coordinates on the screen (*Fig 1*). A partition of the keyboard, vertically starting by the number keys and ending before the space bar key is assigned to move the pointer to certain coordinates on the

screen. The user can move their fingers on the keyboard, just like using a touchpad in order to move the cursor. Whenever they want to type, they can simply press the Control key to switch back to the typing mode. In further experiments any key combination can be assigned for this job, the Control key is selected for the first experiments.

As this interaction method offers the user a very low resolution control over the pointer, a technique which we came up with, called Object-Aware Pointing was required. This technique is embedded in the Keyboard Mapping software and is keeping a list of the selectable objects on the screen in realtime. When a user is using the Keyboard Mapping mode, the program selectively tries to understand which button or section user intended to point to and guides the pointer to the potential target. This allows users to interact with an interface with much more precision using this low resolution interface.

In order to observe the effectiveness of these techniques, we have prepared a test interface which offers users variable amount of buttons while increasing the resolution of the interface gradually as they pass the levels. We expect the users to complete the test both with classic keyboard-mouse combination and the Keyboard Mapping software and ask them to evaluate their experience.

PREVIOUS WORK

As it is stated by Ken Hinckley and Mike Sinclair in their article "Touch-Sensing Input Devices" [1] there has been prior attempts to create a taxonomy of touch sensitive pointing devices. The first taxonomy has been developed by Buxton [2] and extended by Card, Mackinlay and Robertson [3]. Yet their taxonomy has been proved inefficient in Hinckley and Sinclairs work, since they are offering a different category which touch sensitive pointing devices could be used. Their work is about adding touch sensitive capacities to existing technologies such as classic mouse and trackball devices. Their work is significiant in the development of our Keyboard Mapping technique, since they offer adding a new level of interaction on the existing technologies, which makes our approach similar. Yet the main approach remains different, while they do not intervene with the switching problem between the keyboard and the pointer, their work offers a more efficient interaction with the existing pointing tools. Also their attempts on improving these devices were on hardware level, while our approach is on software level. Another similar point in both of the works are that they are designed to improve interaction with specific software environments. Hinckley and Sinclair are focused on increasing the user-friendliness of text authoring tools like Microsoft Word.

In Keyboard Mapping, we are focused on increasing the user-friendliness on web based platforms that include interfaces specifically designed to be interacted by this technique. Also, we presume in the future this approach may include hardware based solutions in order to design a keyboard that can offer a much better solutions when combined with Hinckley and Sinclairs touch sensitive surface solutions.

		CONTACT		NON-CONTACT
		Touch-sensing	**Pressure / Force**	**Proximity**
DISCRETE	**Single channel**	Touchpad touch tablet touchscreens (except IR) touch-based switches PadMouse [1]	push button membrane switch Palm Pilot screen (pressure required) supermarket floor mats car seat: weight sensors for airbag	motion detectors electro-magnetic field sensor [11] Light-level sensor Sidewinder force-feedback joystick (IR beam sensor) IR touchscreens
	Multi-channel	TouchMouse TouchCube [12] touch-sensitive joystick [21] Pinch Gloves [17]	Psychic Space [13] (A grid of floor tiles that can sense which tiles a user is standing on.)	
CONTINUOUS	**Single channel**	contact area (e.g. some touchpads & touchscreens)	pressure-sensitive touch tablet [4] vector input touchscreen [9] torque sensor isometric joystick	laser rangefinder stud finder
	Multi-channel		Multi-touch tablet w/ pressure [15] pressure sensors on handhelds [7] Haptic lens (deform-ation at multiple points) [23]	HoloWall [18] Field-sensing devices [24][26]

Table 1. Suggested in "Touch-Sensing Input Devices" [1]
Our Keyboard Mapping can be listed in Discrete - Single Channel - Pressure/Force

Donald S. Santilli [4] approaches this problem similar to our solution with his design "Computer Keyboard Pointing Device" U.S. Pat. 5,675,361. His solution is embedding touch sensitive surfaces on certain keys on the keyboard, in other words an attempt to move the touchpad on the keyboard to remove the need to switch from pointer to keyboard with a large physical movement. His solution requires a major hardware modification which can offer precise pointer movement, while our Keyboard Mapping system is applied as a software solution that can not offer a precise pointer movement. In order to maximize the resolution and precision, we have used as many key on the keyboard as possible instead of limiting it with 4 keys, unlike demonstrated in the design of Santilli.

KEYBOARD MAPPING / TECHNICAL DETAILS

While developing the Keyboard Mapping software, even though we have taken touchpads as the most similar interaction way, the coordinate processing of Keyboard Mapping is in the Absolute Position Movement category. This means that the coordinates on the screen are directly matched with the keyboard surface respectively. An example to devices using this basic algorithm is a Wacom style drawing tablet. In contrary, touchpads use the Relative Position Movement algorithm. Which means that the pointing you do will take the cursor position as the starting point. In Absolute Position Movement the starting point is always the same, can be considered as the left top edge of the keyboard and the screen (0, 0).

If we need to define the boundaries of the detection area on the keyboard, we will be taking an English keyboard as the example. But this software is working on every keyboard since it is using key codes to process instead of character codes.

The top-left key is the "~" key (key code: 192).
The top-right key is the Backspace key (key code: 8).
The bottom-left key is the "Z" key (key code: 90).
The bottom-right key is the right Shift key (key code: 16).

The main problem in defining these boundaries was about the shift keys. Since there are two shift keys both on the bottom left and right of our rectangle, we had the need to disable one. The one on the left is disabled since on many keyboard layouts it usually covers less area and the demand for resolution on the right side of the rectangle was higher due to the size of the return key, which is lowering the resolution in the area inevitably.

The software is written in Adobe Action Script 3. Since the software was intented to be used with web platforms, we have decided to use the object oriented authoring platform Adobe Flash in order to be able to apply the Object Aware Pointing technique. This way the software can be turned into a library and can be used by third party developers.

The basic working principle of the software can be examined with the following flowchart (Fig 2.).

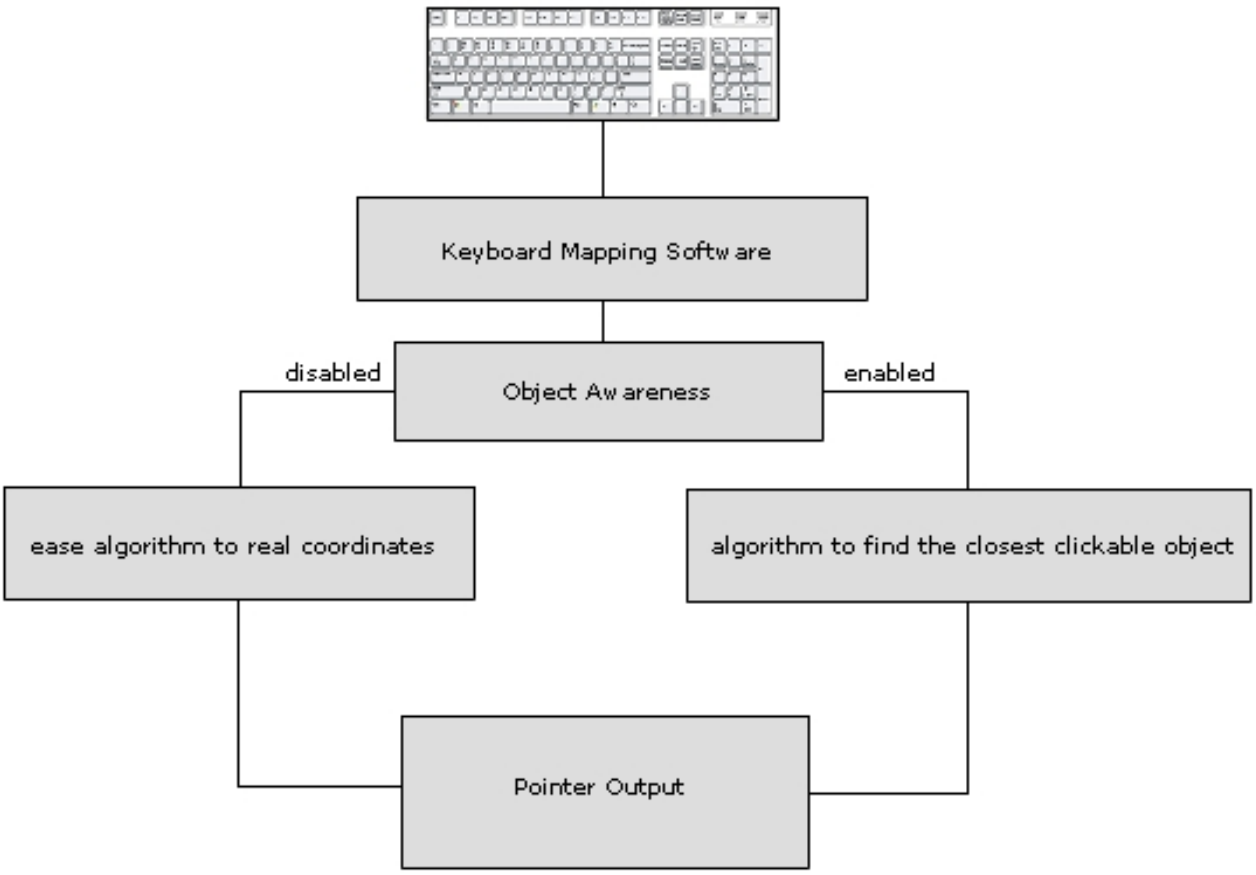

Fig 2. Basic Flowchart of the Software

The software continuously listens to a key input. When the user presses a key the software takes the key code [5] and uses the pre-generated look-up table to turn it into useful coordinates. Once the coordinates are received, the software will check if the Object Awareness is enabled or not. If it is disabled, which may be required on some circumstances, the software will move a virtual pointer in to the fetched coordinates with an ease motion. If it is enabled the Object Aware Pointing algorithm will try to find which object did the user intended to click and manipulates the coordinates of the pointer accordingly.

When the desired coordinate is reached, the user can simply re-press the last key pressed to perform a click action.

OBJECT AWARE POINTING / TECHNICAL DETAILS

Object Aware Pointing is a term and concept that has been invented during the development of Keyboard Mapping software in order to increase its accuracy. But this concept should not be limited with the Keyboard Mapping software and can be used on other applications.

The version we have used here is a very simple application of the concept and can be called Distance Based Object Awareness. Basically, the software is aware of every single clickable area on the screen and compares the user input coordinates to perform a distance calculation to point to the closest object, since that object is most likely to be the object that the user has intented to click. By using this algorithm, the low resolution and low precision interface of Keyboard Mapping can be used in more complex and higher resolution interfaces. Yet it still needs to be used in a specially designed interface.

In future applications, the Object Aware Pointing can be used in capacitive touch screen devices. Most of the smart phones and tablets on the market use this technology and due to the finger-based interaction with the interface, it is often that the user makes the wrong selections through the interface. The interface is needed to be specially designed with large sized objects in order to minimize this types of error. With an Object Awareness algorithm it might be possible to add more and smaller sized objects to the screen. The algorithm can be improved to include a simple artificial intelligence that records the previous user selections in order to optimize the awareness depending on the user to provide a better interaction. It can learn from the user, and decide which object user intented to click depending on the previous interactions of the user with the system.

In our experiments using the Keyboard Mapping technique both with Object Awareness enabled and disabled, it became obvious that when it is enabled it is much easier to navigate through the interface.

EXPERIMENTATION

We have prepared a simple experimentation interface with the Keyboard Mapping software. The interface is specifically designed to test the pointing precision of the software (Fig 3.). This experiment has a very small subject pool of 6 people and has been performed to see the user-friendliness of the software, very roughly. The test includes several levels of interaction. The main objective is clicking the red button which is positioned between numerous blue buttons. As the user passes a level, a new level including a higher resolution button map is being presented. As the level increases the need of the Object Aware Pointing becomes more obvious. It has been observed that as users play around with the interface, they get used to it quickly and be able to click to the right button on their first try.

After the experimentation, we have asked them several questions and expected them to answer in a grade between 1 and 5.

Users have graded the "What do you think about the precision of the pointer?" question with an average of 3.8. This result was unexpectedly high and shows the success of the Object Awareness used in a low resolution interface to increase its precision.

They average grade of the "How comfortable did you feel about using this interface?" question was 4. And the "How easy did you find the learning progress?" resulted with a 4.8. We assume these high scores are the result of using an already existing interface that they are already used to, which is the keyboard. Also, it is the result of our brains capability to map the screen with an interaction surface easily and in short time.

Yet these questions and experiments can not be enough to fully evaluate the system. More precise and planned experiments must be conducted in a platform that can offer an appropriate budget and time.

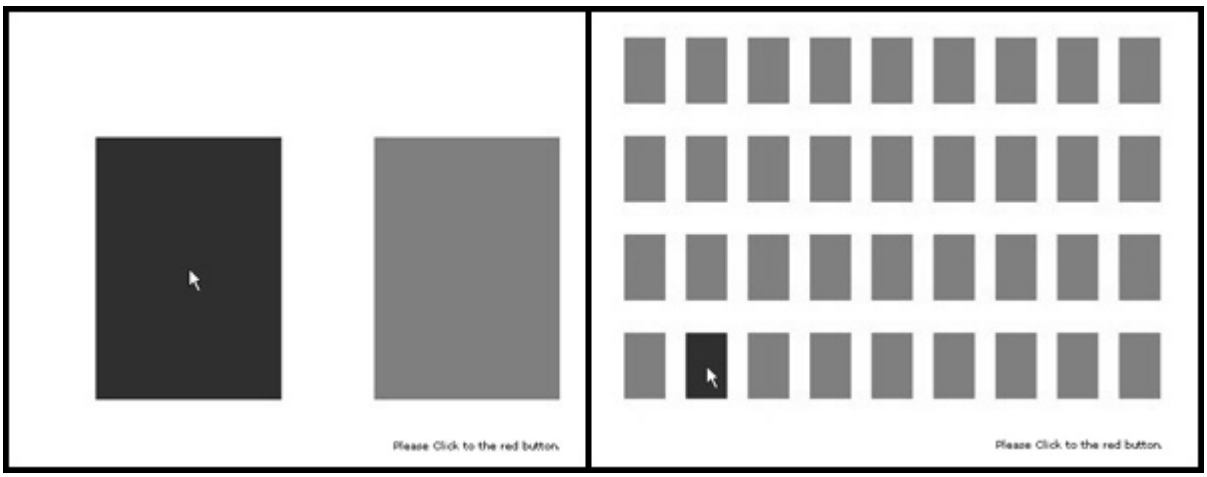

Fig 3. Experiment interface [6]. Low button resolution (left), High button resolution (right).

CONCLUSION AND FUTURE WORK

As a conclusion, this interface software seems to be a good idea to use in specially designed interfaces. But it is clearly inefficient to be used with the current GUI design. The responses from the users are quite promising and shows the need for future experimentations. When coupled with a nicely designed interface this interaction technique can let users perform faster than the keyboard-mouse combination. The software can be released as an Adobe Flash library and can be used by third party developers to provide a new way of interaction to their users.

Object Aware Pointing must be investigated and augmented for further usages. It can be a productive contribution for touch based interfaces, smartphones, tablet computers and kiosks. By adding more complex algorithms, the system can provide a higher precision for the users using a low resolution input technology. It can improve the interfaces by letting developers create more complex and higher resolution interfaces, coupled with capacitive touch screens.

We plan to continue the development of Object Aware Pointing and add simple artificial intelligence features on it in order to test its potential on different pointing devices.

REFERENCES

1. Ken Hinckley, Mike Sinclair "Touch-Sensing Input Devices" CHI'99, 1999
2. Buxton, W. "Touch, Gesture and Marking" *in readings in Human-Computer interaction: Toward the year 2000,* 1995 Morgan Kaufmann Publishers p. 469-482
3. Card, S., Mackinlay, J., Robertson, G., "The Design Space of Input Devices" CHI'90 Conf on Human Factors in Computing Systems, 117-124
4. Donald S. Santilli, U.S. Patent Number: 5,675,361 Oct. 7. 1997
5. Keyboard Keys and Key Code Values, http://help.adobe.com/en_US/AS2LCR/Flash_10.0/help.html?content=00000520.html
6. Keyboard Mapping Experiment: http://swf.odcaf.com/keyboard_mapping/