

A Conceptual Framework for Camera Phone-based Interaction Techniques

Michael Rohs and Philipp Zweifel

Institute for Pervasive Computing
Department of Computer Science
Swiss Federal Institute of Technology (ETH) Zurich, Switzerland
`rohs@inf.ethz.ch`

Abstract. This paper proposes and evaluates interaction techniques for camera-equipped mobile phones. The proposed techniques are based on a visual code system that provides a number of orientation parameters, such as target pointing, rotation, tilting, distance, and relative movement. Our conceptual framework defines a set of fundamental physical gestures that form a basic vocabulary for describing interaction when using mobile phones capable of reading visual codes. These interaction primitives can be combined to create more complex and expressive interactions. A stateless interaction model allows for specifying interaction sequences, which guide the user with iconic and auditory cues. In using the parameters of the visual code system as a means of input, our framework enhances the currently limited input capabilities of mobile phones. Moreover, it enables users to interact with real-world objects in their current environment. We present an XML-based specification language for this model, a corresponding authoring tool, and a generic interpreter application for Symbian phones.

1 Introduction

Today's camera-equipped mobile phones and PDAs combine a large number of features and provide computing and communication capabilities comparable to those of earlier desktop PCs. Yet much of this functionality is hard to use. If we consider the input capabilities of camera phones, we soon discover that mobile device technology has outgrown the ability of the interface to support it. For example, small keypads are ill-suited for input tasks of moderate complexity. These input problems are only slightly mitigated with the integration of joysticks and touch screens.

What is missing is an interface that takes mobile users' constantly changing context into account and allows for spontaneous interaction with physical objects that users encounter while on the move. If we can provide a simple way to access information services associated with these objects, then we can create a more useful mobile Internet. Also, if we can provide a consistent interaction model for objects in the user's environment, then we can greatly increase the usability and functionality of mobile devices.

An essential feature of the visual code system is mapping points in the image plane to corresponding points in the code plane, and vice versa (see Figure 1, right). With this feature, the pixel coordinates of the camera focus (the point the user aims at, indicated by a crosshair during view finder mode) can be mapped to corresponding code coordinates. Each code defines its own local coordinate system that has its origin in the upper left edge of the code and is independent of the orientation of the code in the image. Areas that are defined with respect to the code coordinate system are thus invariant to projective distortion.

The following section gives a brief overview of related work. Section 3 outlines a number of application scenarios. Section 4 discusses interaction primitives, their combinations, and how they are indicated to the user. In Section 5, we define our user interaction model, which describes how to create visual code image map applications. Also, we describe an XML-based specification language, an authoring tool for visual code image maps, and a corresponding parser and interpreter on the phone. In Section 6, we report about a usability study in which we evaluated our interaction techniques. We wrap up with a number of conclusions and ideas for future work.

2 Related Work

Several projects have investigated linking physical objects with virtual resources, using technologies such as RFID tags or infrared beacons [1, 7]. However, these projects were mainly concerned with physical linking per se or with the infrastructure required for identifier resolution. They were limited in the richness of user interactions and typically allowed just a single physical gesture (for example presence in the reading range of an RFID reader) and thus just a single action per object. We, in contrast, allow the semantics to be a function of both the *object* and the *gestural sequence*.

A number of projects focused on improving interaction with the device itself rather than with the user's environment. No objects in the environment were integrated in the interaction. In 1994, Fitzmaurice et al. [8, 9] prototyped a spatially aware palmtop device with a six degrees-of-freedom tracker to create a porthole window into a large 3D workspace. The actual processing was done on a graphics workstation. The palmtop sensed its position and orientation in space and combined input control and output display in a single unit – thus integrating “seeing” and “acting”. One could explore the 3D workspace with the palmtop using an *eye-in-the-hand* navigation metaphor. The goal was to step out of the “claustrophobic designs and constraints” imposed by the form factor of handheld devices. While this was a vision in 1994, similar interfaces can be built today with handheld devices and interaction techniques as presented in this paper.

Our work can be seen as an instance of an *embodied user interface* [3], in which the user directly interacts with the device itself – for example by tilting it – and both the manipulation and the virtual representation are integrated within the same object. Tilting of a handheld device has been explored as an

input parameter for menu selection, scrolling, navigation, text entry, and 3D object manipulation [10–15]. Readability problems of tilted displays, which we also experienced in this work, are described in [11] and [12].

In [2] Fishkin analyzed the idea of a physical grammar, and in [3] he addressed the issue of multi-gesture command sequences. Bartlett [12] used gestures – like tilting, snapping, shaking, tapping, and fanning – as a vocabulary of commands for a handheld electronic photo album. Our work tries to build compound interactions from a vocabulary of interaction primitives.

Bartlett [12] commented on some of the limits of embodied user interfaces: “perceived motion on the screen is the sum of the motion of the embodied device and the changes made to the display by the device. As you interact with the device by moving it, the orientation of the display to the user changes, then in response to that motion the display contents move on the display.” This effect is especially severe for fast movements; however such movements are not required in our design. Rather, our work is more concerned with subtle yet easily and manually controllable changes.

Our interaction model allows us to define state spaces of phone postures in 3D. These issues are similar to those experienced with augmented reality environments and with 3D input devices [16, 15]. In our case however, our 3D environment is the physical world perceived through the camera lens. The Video-Mouse [15] is an optical mouse for 3D input based on video sensing and although it shares some similarities with our system, it is different in that it provides only very limited height sensing.

3 Application Scenarios

As outlined in the introduction, our system enhances the general input capabilities of mobile devices and provides a way to access mobile information services related to physical objects within a user’s vicinity. Our system allows fine-grained control and access to various information items and services that can be physically hyperlinked [7] to objects in the environment. Typical objects that a mobile user might encounter include product packaging, vending and ticketing machines, posters and signs, as well as large electronic displays [4]. A few application scenarios are outlined below.

- **Tram stop.** The left and middle sections of Figure 2 show a tram stop information panel tagged with a visual code which allows users to access tram arrival times and to obtain further information by rotating the phone. To obtain information about the route of interest, users focus on the corresponding route number.
- **Vending machine.** The right part of Figure 2 shows a vending machine tagged with visual codes. To purchase products and confirm the purchase, users aim at the desired object. Of course in this scenario, a payment method needs to be in place.
- **Campus map.** Visual code image maps can help to find the location of an event on a campus map. A visitor to the campus could focus on an area



Fig. 2. A tram stop (left and middle) and a vending machine (right) equipped with visual code image maps.

labeled “current events” to get information about conferences, talks, and other events. The location of the event could then be highlighted on the mobile phone screen.

- **Augmented board games.** Computer-augmented board games are another good candidate for using visual code image maps since such games could benefit from a wide range of interaction possibilities that do not tie the user to a desktop computer.

4 Interaction Techniques

This section introduces the interaction techniques that are used in visual code image map applications. These techniques rely on sensing visual codes from different perspectives. We describe how interactions are combined from basic building blocks and how interaction cues guide users in the interaction process.

4.1 Interaction Primitives and Interaction Cues

Combined interactions are constructed from basic building blocks, called *interaction primitives*. *Static interaction primitives* require the user to aim their camera phone at a visual code from a certain orientation and to stay in that orientation. We defined two kinds of *dynamic interaction primitives*, which either involve “sweeping” the camera across a visual code or simply moving the phone relative to the background.

To facilitate information access and guide users in their interaction flows, each interaction primitive is associated with one or more *interaction cues* in the form of an icon. They appear on the mobile device’s display and provide users with an indication of the possible interaction postures. Visual cues can optionally be combined with auditory icons.

For instance, the leftmost *rotation* interaction cue in table 1 indicates to users to rotate the mobile phone either clockwise or counterclockwise in order

to access more information. The rightmost cue for the *distance* primitive means that more information can be obtained by moving the phone closer to the code – relative to the current posture.

An interaction cue should be both intuitive when encountered for the first time and easy to remember. Interaction cues should also be unambiguous so that it is easy to distinguish between different interaction primitives. In our design of interaction icons, we use color extensively since it facilitates distinguishing between different interaction primitives, and color displays are available on all camera phones. We restrict icon size, since the interaction cues must occupy only a small part of the phone display. They have to be rather simple and plain in order not to unnecessarily distract the user or clutter the interface.

4.2 Input and Output Capacity

Static interaction primitives map certain orientations of the mobile phone, also called *postures*, to individual information aspects. The posture of the device is determined with respect to a visual code in the camera image. With the term *input capacity* we denote the number of discrete information aspects that can be sensibly encoded in each of the interaction primitives. The input capacity is a measure of how many discrete interaction postures can be easily and efficiently located and selected by users. An important performance aspect is the time it takes a user to locate an individual information item among a set of available information items. This time depends on the kind of interaction primitive, the number of available postures, as well as the quality of feedback that is provided to the user. For static interaction primitives, discrete postures are possible, like focusing a particular information area, as well as more fine-grained forms of input, like the continuous adjustment of a value by moving closer to or away from a code. For each interaction primitive, we will give an estimation of its input capacity, which has been obtained experimentally and during user testing. In this work, discrete postures activate associated information aspects. It would also be conceivable to use voice commands for this purpose. In addition, voice commands can be taken as a way to get further input once a certain posture has been reached. This combination of postures and audio input would realize a multi-modal user interface.

The output capabilities of mobile devices are limited due to the small screen size. Thus, the amount of textual and graphical information that can be shown on a single screen is limited. Fortunately, the interaction postures are very well suited for structuring the presentation of data by distributing it across several related interaction postures. With the proposed approach, text and graphics can be overlaid over the camera image as known from augmented reality applications. Graphical elements can be registered with objects in the image, i.e. shown at the same position and resized and adapted to the viewing perspective. This makes the connection between the physical object and the information shown on the display more obvious to the user and avoids its isolated presentation without any other context.

Output can also be used to realize a feedback loop to the input side, which has an impact on the input capacity. To create the feedback loop, characteristic icons can represent interaction primitives and indicate interaction possibilities to the user. Mobile devices typically have an audio output channel which can be also used for establishing a feedback loop. Characteristic audio cues (“earcons” [17]) can be permanently associated to different information or interaction types. Auditory cues have the advantage that they do not take up any space on the device display. Designing audio feedback needs to be done with care, because it has privacy repercussions or might be distracting to some users. Another interesting option to support the feedback loop between output and input is available with the phone’s vibration feature (“tactons” [18]).

4.3 Static Interaction Primitives

Static interaction primitives are based on the parameters of the visual code system, as well as the focused area, key presses, and the time stayed in a given interaction posture. For the user, this means finding a posture in view finder mode guided by the interaction cues, staying in that posture, and optionally taking a high-resolution picture to “freeze” the current posture. The “information freezing” feature stops the view finder mode and shows the information related to the last captured phone posture. The available static interaction primitives, their estimated input capacity, and the associated icons are shown in table 1.

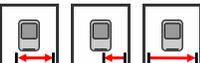
Static interaction primitive	Input capacity	Interaction cues
pointing	number of information areas	information area is highlighted
rotation	7	
tilting	5 (+4 if using NW,NE,SW,SE)	
distance	8	
stay	unlimited (time domain)	 (icon has a highlighted display)
keystroke	12 (keypad) + 5 (joystick)	 (icon has a highlighted keypad)

Table 1. Input capacity and interaction cues of static interaction primitives.

Pointing. The *pointing* interaction primitive requires targeting an area with the crosshair shown on the device display in view finder mode. The area is defined in terms of code coordinates. The input capacity is only limited by the number of areas that can be reached with the crosshair while the associated visual code is in the camera view. Section 5.5 presents techniques for extending the scope of reach. The borders of an area are highlighted when the associated visual code is recognized and the focus point is inside that area.

Rotation. The *rotation* interaction primitive associates rotation regions with discrete information items. For usability purposes, the rotation of the phone should be limited to $\pm 90^\circ$ from the upright position. To improve legibility, text should be aligned vertically if the rotation is greater than 45° . Users can complete up to 7 discrete postures, which correspond to regions that cover about 30° each, centered at 0° , $\pm 30^\circ$, $\pm 60^\circ$, and $\pm 90^\circ$. *Rotation* is also usable as a continuous input control for a parameter value, such as the volume of a hi-fi system.

Tilting. During user testing, *tilting* turned out to be the most challenging interaction primitive for users since it requires turning the head in order to follow the device screen. We therefore do not use precise tilting values, but only an indication of the direction (“north”, “west”, “south”, “east”, and central position). This results in an input capacity of five postures. It is straightforward to extend this by “north-west”, “north-east”, “south-west”, and “south-east”, resulting in an overall input capacity of 9 postures.

Distance. The *distance* is measured during view finder mode and has an input capacity of 8 easily distinguishable distances. Distance is another a good candidate for continuous input.

Stay. The *stay* interaction primitive requires the user to stay in a certain posture. It automatically changes the provided information after a certain amount of time. The time interval can be freely specified, but should depend on the amount of information shown on the device screen. For a few lines of information it would typically be a couple of seconds. This primitive can be combined with the *keystroke* primitive described next, in order to realize a “timeout kill” mechanism as used for multi-tap text entry [14]. The input capacity is unlimited in principle, requiring the user to wait.

Keystroke. Finally, the *keystroke* interaction primitive consists of pressing a button on the device’s keypad or using the device’s thumb-operated joystick. Our target device has a 12 button numeric keypad and a non-isometric joystick with five states (left, right, up, down, press). The input capacity of this interaction primitive is obviously limited by the number of available keys.

The numbers given for the discernible input capacity of each interaction primitive decrease, if the basic primitives are combined with each other, as shown in the next sections.

4.4 Dynamic Interaction Primitives

There are two kinds of dynamic interaction primitives. With the first, the phone is moved (“swept”) across the code in a certain direction while the camera is in view finder mode. The direction of movement is sensed by the mobile device

and used as the input parameter. Interaction symbols for this kind of dynamic interaction primitive are not shown on the device display, but printed next to the code. For each possible direction of movement, a label is given, informing the user about the operation that will be triggered when the code is “swept” in the indicated direction. These interaction primitives are suitable for “blind” operation, in which a single operation is selected and immediately triggered after the movement. Sweep primitives can be regarded as the equivalent of a crossing-based interface for visual codes [19]. The input capacity amounts to 4 for both horizontal and vertical movement as well as for diagonal movement. A combination of both movement types seems to be too complex. With current phone hardware, the movement must not be too fast, in order for the codes to be reliably detected at multiple positions in the image. The input capacity and interaction cues are depicted in table 2.

Dynamic interaction primitive („sweep“)	Input capacity	Interaction cues (printed next to the code)
horizontal or vertical movement	4	
diagonal movement	4	

Table 2. Input capacity and interaction cues of *sweep* interaction primitives.

The second kind of dynamic interaction primitives is based on the optical movement detection algorithm that does not require a visual code in the camera image. It provides relative linear movement and relative rotation. It is not suited for discrete input, but for continuous adjustment of parameter values or for direct manipulation tasks. The corresponding interaction cues can be shown on the device display, printed next to a code, or shown on an electronic display to indicate that its objects can be directly manipulated by movement detection. Table 3 contains the capacities and interaction cues of these interaction primitives.

A *clutching mechanism* is required to prevent incidental motions of the phone from triggering unwanted dynamic interaction primitives. In our system, the relative movement tracking is active while the phone’s joystick button is held down. Releasing the button exits the relative movement detection state. This is also known as a *quasimode* as defined by Raskin in [20]. In Buxton’s model [21], pressing the joystick button down corresponds to a state transition between state 1 (“tracking”) and state 2 (“dragging”), releasing the button again transitions back to state 1.

Dynamic interaction primitive (relative movement)	Input capacity	Interaction cues
relative linear movement	4 (continuous)	
relative rotation	2 (continuous)	

Table 3. Input capacity and interaction cues of relative movement interaction primitives.

4.5 Combinations of Interaction Primitives

The basic interaction cues are designed in such a way that they can be combined to form more complex interaction cues. Table 4 shows the possible combinations of two static interaction cues. When the mobile display shows a combination interaction cue, this means that the user has a choice to select between more than one interaction primitive to reach further information items. The usability of such combinations is discussed in Section 6. Combinations of more than two interaction cues should be avoided in order not to confuse the user. Even with combinations of only two static interaction cues, a large number interaction possibilities results.

Some of the static interaction primitives can be combined with the dynamic *sweep* interaction primitives. Each of the eight directions of movement can be combined with the following static interactions: rotation, tilting, and distance. The idea is to move the camera across the code in the chosen direction while keeping a certain rotation, tilting, or distance. In the case of rotation, for example, it should be easy to hold the phone rotated 90° counterclockwise from the upright position.

Combinations of static primitives and dynamic primitives that sense relative movement seem to be more practical. Even if they cannot be executed simultaneously, performing a dynamic after a static interaction primitive is useful. A user first selects a certain parameter using a static interaction primitive – like tilting – and then uses relative linear movement to adjust the value. The relative movement detection is activated while the user is holding the joystick button down. This kind of combination resembles a “point & drag” transaction in classical GUI interfaces [21].

5 Visual Code Image Maps

In this section, we describe how combinations of interaction primitives can be applied in entire *visual code image map* applications. Visual code image maps consist of a number of areas, which are located close to a visual code and associated with multiple information aspects or specific operations. Areas can cover

Combination	Interaction cue	Combination	Interaction cue
pointing & rotation	 + highlighted area	rotation & stay	
pointing & tilting	 + highlighted area	rotation & keystroke	
pointing & distance	 + highlighted area	tilting & distance	
pointing & stay	 + highlighted area	tilting & stay	
pointing & keystroke	 + highlighted area	tilting & keystroke	
rotation & tilting		distance & stay	
rotation & distance		distance & keystroke	

Table 4. Combinations of two static interaction primitives with example interaction cues.

a certain region in the vicinity of a code, occupy the same space as the code, or even be defined as infinitely large. Area locations and shapes are defined in terms of the coordinate systems of the visual codes located near them. Area-related information is accessed by varying the input parameters provided by the visual code system. The input parameters are abstracted to a set of postures that are easily discoverable and applicable by users. The postures are specified as combinations of interaction primitives in an image map definition.

Figure 3 shows an example interaction flow for a simple image map. To the left of the screenshots, the enlarged interaction cues are drawn. An elliptical region next to a visual code is associated with six information items. At a farther distance (depicted in the upper three screenshots), three different information items are presented. The user just has to *stay* at that distance. The *stay* user interaction symbol indicates that more information will be displayed by waiting. Moving closer to the code plane, the interaction cue changes and another information aspect is displayed (depicted in the lower three screenshots). In the near distance posture, more information can be accessed by rotating the phone to the left (counterclockwise) and to the right (clockwise). The underlying user interaction model is discussed in more detail below.

When designing overlays over the camera image, design guidelines as described in [22] should be taken into account. The visual context given by the camera image should be maintained as far as possible. A graphical representation should be chosen such that the underlying context is revealed. This avoids

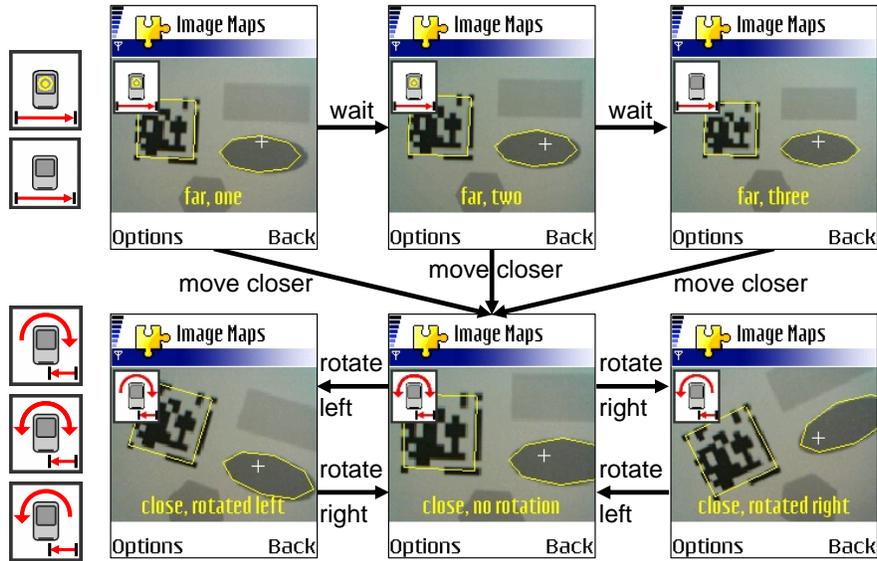


Fig. 3. An example interaction flow in a visual code image map.

the issue that the user has to split visual attention between the camera image (the “context”) and the generated graphical overlay. It enables *dual attention*, which is characteristic of see-through tools. In addition, unnecessary information, i.e., information that is not part of the currently pointed area should be hidden. This is especially important for small displays.

5.1 User Interaction Model

The user interaction model determines how a user can browse information or trigger actions in a visual code image map. We use a stateless model that only considers the currently sensed parameters. Each interaction posture is associated with a rule. A rule consists of a condition and a result that is activated when the condition is met. A condition is made up of a set of constraints. A constraint restricts the valid range of a single input parameter. The rules are continuously checked and their results activated if their conditions are met. The visual code image map designer has to ensure that conditions are mutually exclusive. If they are not, the order of execution is undefined. For non-idempotent functions, it is important that they are not activated multiple times. Checking such constraints is part of the semantics of each action result and not specified in the interaction model. The stateless model is easy to understand for users, since they always see the same result if the same input posture is chosen. For image map designers, image map applications are easy to specify in this model. In a completely stateless model, some of the proposed combinations of static

and dynamic interaction primitives cannot be realized. It is therefore slightly extended as described below.

State-based models are inherently difficult to understand for users since the system can behave differently on the same input parameters if it has different states. We therefore limited the notion of input state in the system to the relative movement interaction primitives. In order to activate relative movement detection, the user has to hold the joystick button down. The user's last posture receives relative movement updates while the joystick button is held down. Releasing the button exits the relative movement detection state. This *quasi-mode* scheme ensures that the user is not inadvertently locked in a state. The second notion of state is introduced with the *stay* static interaction primitive. It becomes true when the time stayed in a certain posture is within a predefined time range. The state is thus defined by the set of other constraints of a condition, without the *stay* interaction primitive. A rule containing a *stay* interaction primitive fires when all other constraints have been true for the specified amount of time. The timeout is reset when the rule becomes invalid again.

5.2 Visual Code Image Map Specification Language

Based on the interaction model described above, we developed a visual code image map specification language. The specification language is XML-based. Depending on the visual code value, different measures are taken to retrieve the specification of an unknown image map. The XML description is loaded from the local file system, obtained via Bluetooth or the mobile phone network. It is parsed and the extracted information is used to present information and provide functionality according to the image map. We assume that up-to-date information is inserted into the XML file on the server, e.g. via PHP scripting.

5.3 Information Results

Information results can consist of auditory cues, textual overlay over the camera image, bitmap overlays, and overlays of graphical shapes. Textual overlays can appear at a constant position in the mobile's display, which is the default in the current implementation. The text position can also be tied to specific code coordinates and thus appear as an overlay of an element in the image map. Bitmap overlays can either appear at a constant display position or located at specific code coordinates. As with textual output, free rotation of images is an expensive operation for current mobile devices and can thus not be performed in real time on current devices. The Symbian operating system, for example, only provides functions for rotating text in steps of 90°, which is sufficient for legibility in the case of rotation. Graphical overlays, such as rectangles, ellipses, and polygons are automatically adapted to perspective distortion by using the code coordinate system mapping. It is possible to specify multiple textual and graphical outputs in a single information result.

5.4 Action Results

Triggering an action result consists of starting the requested application on the device and dispatching the provided arguments in the format the application requires. The semantics of the arguments depend on the given application. In the simplest case, the argument string provided in the XML description is simply passed on to the application. In a more complex case, it requires parsing the argument string and calling multiple methods on the phone application. The action result needs to define whether it has to be executed on each image update while the corresponding condition is valid, once as the rule first becomes active, or only when the joystick is additionally held down. Example action results are starting the WAP browser with a specific URL as an argument, storing vCard and vCalendar entries, placing a phone call to the number given in the argument string, invoking the SMS editor, or sending a predefined text message without invoking the SMS editor. Other action results include opening a Bluetooth connection to report relative movement updates and visual code sightings.

5.5 Focus Point Adaptation

A problem with visual code image maps comes from the fact that at least one code needs to be present in the camera view in order to compute the mapping from the image to the code coordinate system. This restricts the radius of action for moving the focus point. This situation is shown in the left section of Figure 4. The focus point is typically indicated to the user with a cross hair that is located in the middle of the display. The reachability problem can be solved in a number of ways. First, multiple codes can be dispersed throughout an image map. This raises aesthetical concerns and restricts the designer of an image map, because more space is occupied by visual markers. This might be mitigated in the future with higher-resolution cameras, allowing much smaller codes to be used. Additionally, with zoom cameras, a wide angle setting can be used to cover a larger part of the image map. Second, there is no reason why the focus point has to be located in the middle of the screen. One option would be to include the most suitable position of the cursor as a parameter in the image map specification. If a visual code is located to the left of a vertical arrangement of areas, for example, the focus point might be set horizontally to the right for easier targeting.

A third option is to dynamically adapt the position of the focus point depending on the position of the code center on the screen. This is shown in the middle and right of Figure 4. The focus point is computed as the mirror point of the code center point through the image center point. In usability tests, this smooth adaptation style seemed to be more predictable than another adaptation style, in which discrete focus point positions had been used. The smooth adaptation of the cursor position requires more dexterity than a fixed cursor position, but is manageable after a short time. If no code is present in the image for a certain time, the focus point is repositioned to the display center. If multiple codes are available, the nearest one is chosen for adaptation. If multiple codes are visible

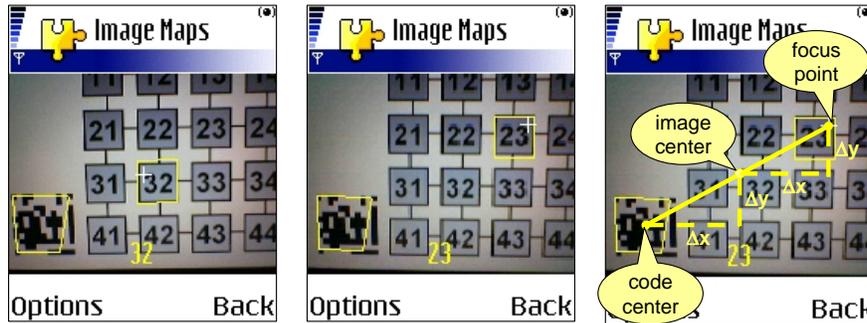


Fig. 4. Central focus point (left) and adapted focus point (middle and right).

in the camera image, the adaptation can be disabled, because the reachability problem is no longer given. With dynamic adaptation, the reachable radius is increased by up to 100% compared to a focus point which is centered on the display.

5.6 Visual Code Image Map Editor and Interpreter

We have developed a visual code image map editor in Java. The editor produces image map specifications from jpg, gif, and png images (typically a picture of the visual code in the real world) so that users can draw areas and specify constraints, interaction cues, and results. The resulting output is an XML file that can be stored on a server and which can be downloaded to the mobile phone.

On the device side, we have developed a generic visual code image map interpreter in C++ for Symbian devices. For each detected code, the interpreter tries to locate the corresponding image map and continuously checks for satisfied conditions in the available rules. As long as the conditions of a rule are satisfied, the corresponding information or external application is shown.

6 Usability Evaluation

6.1 Goals and Design

To understand the strengths and weaknesses of the individual interaction techniques, as well as the approach as a whole, we designed a qualitative usability study. It consisted of a questionnaire, two task execution parts, and a final interview. The questionnaire covered basic biographic data and asked about users' level of experience with mobile phones, text messaging, and playing computer games. The two task execution parts served different purposes. The aim of the first one was to evaluate individual interaction primitives and their combinations independently from the semantics of a specific application. The second one used

the campus map scenario outlined in Section 3 to help users understand the implications of using the interaction concepts in a broader context. The dynamic interaction primitives have not been evaluated in this study.

A number of technical factors influence the users' satisfaction with the interaction techniques, such as the size and quality of the display and the response time and reliability of the visual code system. However, we can expect that most of these technical factors are likely to improve. The usability evaluation thus tried to focus on issues that are inherent to the design of the proposed interaction techniques.

The first part of the study consisted of 15 individual tasks. Users employed the various interaction techniques to try and find a secret number and a secret letter in a particular image map. The first few tasks tested the dexterity required for the basic interaction primitives as well as how easily users were able to remember and distinguish between various interaction cues. The remaining tasks tested combined interactions. The second part of the study allowed users to get a feel for a possible real-word application. Lastly, in post-test interviews, users were asked to express their opinion about the overall system, rate the individual interaction primitives, and to give feedback about the presented scenario.

When observing tasks, we used the think-aloud technique. Tasks were performed under quiet, laboratory-like conditions. Our evaluation procedure was adapted from the guidelines proposed in [23].

The execution of the study, including the initial questionnaire and the final interview, lasted approximately one hour per user. Eight users took part in our study, with ages ranging between 17 and 35. All of our users had some experience with personal computers, and all regularly used mobile phones for making phone calls and writing text messages. Some of them were heavy phone users, who often played mobile phone games and accessed information via WAP.

6.2 Findings and Discussion

Our results indicate the most challenging interaction primitive for users to do was *tilting*. The *pointing*, *distance*, and *stay* primitives were rated the best, followed by *keystroke* and *rotation*. For combinations which used *pointing* with other static primitives, we found two groups of people who preferred different user interactions. The first group, consisting of five participants, preferred user interactions that demand less manual dexterity. They favored the *pointing & stay* interaction, followed by the *pointing & keystroke* interaction. The second group, consisting of three participants, preferred the *pointing & distance* and *pointing & rotation* combinations. One possible explanation for this difference is dexterity. Since the first group seemed to have more problems with manual dexterity, they preferred passive user interactions, like *stay*, and well-known interactions, like *keystroke*. The second group, which had less problems with manual dexterity, liked combinations which used *distance* and *rotation* primitives the best because this gave them immediate control over the visual code application. With the *stay* primitive, the system forces the user to pause. This can be problematic, since the user cannot control the duration of each state. The *pointing & tilting*

combination was by far the most difficult interaction. The reason seems to be that this combined interaction which asks users to simultaneously focus on an area while keeping a visual code in the camera image and tilting the phone to the required position is very demanding for first time users.

During view finder mode, camera images are continuously sampled and the display is updated accordingly. We provided an “information freezing” feature that stops the view finder mode and shows the information related to the last captured phone posture. Some users used this feature as soon as they reached the correct phone posture. The feature is extremely important in that it provides users with some sense of permanence and stability.

We observed some learning effects during the usability test. In general, participants managed the *rotation* interaction primitive in task 13¹ more easily and more rapidly than in task 3² although task 13 was more difficult. Evidently, participants had improved their skills in handling the interaction techniques during the performed tasks and, moreover, the tasks did not seem to exhaust them.

All user interaction cues seemed to be easy to learn and remember. However, two participants first confused the *tilting* and *rotation* interaction cues. But after this first mistake they had no further problems. The interaction cues have been redesigned in the meantime and now use different colors for indicating “rotation” (red) and “tilting” (blue), which should improve distinguishability.

In the second part of the study, users had to look up a building on a campus map that was printed on a poster and attached to the wall. Observation showed that the application was not self-explanatory. Most users needed some instructions on how to use it. The following observations have been made during the second part:

- If the information areas are not clear, users tend to focus on the visual codes since they assume that they contain information items. The observation of this behavior offers two conclusions: first, a visual code image map designer should pay attention to design obvious information areas. Second, in a complex image map application, focusing directly on the visual codes should trigger a help menu or a description of the application.
- Most users tended to read printed information that was captured by the camera and shown on the phone display by looking directly at the printed information. They seemed to avoid the display since the size and quality is not yet good enough. However, users had no problems to read generated textual information and graphical overlays over the camera image directly on the display.
- Two users spontaneously remarked that they find it easier to access information aspects with the information map application on the wall than with the newspaper-like tasks on the table.
- Reading distances differed between users, but all users managed to find a suitable distance between a printed code and the camera after a short time.

¹ *rotation* plus *pointing* and only one visual code in range

² *rotation* plus *pointing* and two visual codes in range

7 Conclusions and Future Work

We have used a visual code system to augment camera phones with physical gestures in order to turn them into versatile interfaces to real-world objects. The proposed conceptual framework allows constructing rich interaction sequences by combining a few basic interaction primitives in a flexible way. Even though the input capacity of each individual interaction primitive is limited, their combination results in a large number of input postures. The chosen stateless interaction model, or rather its realization as an XML-based specification language, adequately describes visual code information maps, including input postures, phone movements, information results, and action results. An authoring tool and a generic interpreter application for Symbian phones enable the creation and usage of visual code image map applications.

The user evaluation showed that most of the postures are easily and quickly discoverable with the help of graphical and auditory cues. It also showed that users generally like the proposed interaction paradigm. A few undesirable combinations of interaction primitives have been revealed that should be avoided by visual code image map designers.

A possible next step in this research would be to investigate how data found in an image map can be actively manipulated instead of just accessed. Another idea would be to examine if users develop a kind of “posture memory” – in the sense of “muscle memory” – when they repeatedly access the same information items. Furthermore, investigating the proposed interaction techniques in computer-augmented board games would be very interesting.

Acknowledgments

We thank the anonymous reviewers for their valuable comments and suggestions for related work. We also thank Jennifer G. Sheridan (Lancaster University), Rafael Ballagas (RWTH Aachen), Friedemann Mattern, Marc Langheinrich, and Harald Vogt (Institute for Pervasive Computing at ETH Zurich) for constructive comments and discussions. Last, but not least, thanks to our test users.

References

1. Want, R., Fishkin, K.P., Gujar, A., Harrison, B.L.: Bridging physical and virtual worlds with electronic tags. In: Proceedings of the SIGCHI conference on Human factors in computing systems, ACM Press (1999) 370–377
2. Fishkin, K.P.: A taxonomy for and analysis of tangible interfaces. *Personal Ubiquitous Comput.* **8** (2004) 347–358
3. Fishkin, K.P., Moran, T.P., Harrison, B.L.: Embodied user interfaces: Towards invisible user interfaces. In: Proceedings of the Seventh Working Conference on Engineering for Human-Computer Interaction, Kluwer, B.V. (1999) 1–18
4. Ballagas, R., Rohs, M., Sheridan, J.G., Borchers, J.: Sweep and Point & Shoot: Phonecam-based interactions for large public displays. In: CHI '05: Extended abstracts of the 2005 conference on Human factors and computing systems, ACM Press (2005)

5. Rohs, M.: Real-world interaction with camera-phones. In: 2nd International Symposium on Ubiquitous Computing Systems (UCS 2004), Tokyo, Japan (2004) 39–48
6. Rohs, M., Gfeller, B.: Using camera-equipped mobile phones for interacting with real-world objects. In Ferscha, A., Hoertner, H., Kotsis, G., eds.: *Advances in Pervasive Computing*, Austrian Computer Society (OCG) (2004) 265–271
7. Kindberg, T.: Implementing physical hyperlinks using ubiquitous identifier resolution. In: *Proceedings of the eleventh international conference on World Wide Web*, ACM Press (2002) 191–199
8. Fitzmaurice, G.W., Zhai, S., Chignell, M.H.: Virtual reality for palmtop computers. *ACM Trans. Inf. Syst.* **11** (1993) 197–218
9. Fitzmaurice, G., Buxton, W.: The Chameleon: Spatially aware palmtop computers. In: *CHI '94: Conference companion on Human factors in computing systems*, ACM Press (1994) 451–452
10. Rekimoto, J.: Tilting operations for small screen interfaces. In: *Proceedings of the 9th annual ACM symposium on User interface software and technology*, ACM Press (1996) 167–168
11. Harrison, B.L., Fishkin, K.P., Gujar, A., Mochon, C., Want, R.: Squeeze me, hold me, tilt me! an exploration of manipulative user interfaces. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. (1998) 17–24
12. Bartlett, J.F.: Rock 'n' scroll is here to stay. *IEEE Comput. Graph. Appl.* **20** (2000) 40–45
13. Hinckley, K., Pierce, J., Sinclair, M., Horvitz, E.: Sensing techniques for mobile interaction. In: *Proceedings of the 13th annual ACM symposium on User interface software and technology*, ACM Press (2000) 91–100
14. Wigdor, D., Balakrishnan, R.: TiltText: Using tilt for text input to mobile phones. In: *Proceedings of the 16th annual ACM symposium on User interface software and technology*, ACM Press (2003) 81–90
15. Hinckley, K., Sinclair, M., Hanson, E., Szeliski, R., Conway, M.: The VideoMouse: A camera-based multi-degree-of-freedom input device. In: *Proceedings of the 12th annual ACM symposium on User interface software and technology*, ACM Press (1999) 103–112
16. Zhai, S.: User performance in relation to 3D input device design. *SIGGRAPH Comput. Graph.* **32** (1998) 50–54
17. Brewster, S.A.: Using nonspeech sounds to provide navigation cues. *ACM Transactions on Computer-Human Interaction* **5** (1998) 224–259
18. Brewster, S., Brown, L.M.: Tactons: Structured tactile messages for non-visual information display. In: *Proceedings of the fifth conference on Australasian User Interface*, Australian Computer Society, Inc. (2004) 15–23
19. Accot, J., Zhai, S.: More than dotting the i's — foundations for crossing-based interfaces. In: *Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press (2002) 73–80
20. Raskin, J.: *The humane interface: New directions for designing interactive systems*. ACM Press/Addison-Wesley Publishing Co. (2000)
21. Buxton, W.: A three-state model of graphical input. In: *Proceedings of the IFIP TC13 Third International Conference on Human-Computer Interaction*, North-Holland (1990) 449–456
22. Tapia, M.A., Kurtenbach, G.: Some design refinements and principles on the appearance and behavior of marking menus. In: *Proceedings of the 8th annual ACM symposium on User interface and software technology*, ACM Press (1995) 189–195
23. Gomoll, K., Nicol, A.: Discussion of guidelines for user observation. From: *User Observation: Guidelines for Apple Developers* (1990)