Real-World Interaction with Camera Phones

Michael Rohs

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

Abstract. With the integration of cameras, mobile phones have evolved into networked personal image capture devices. Camera phones can perform image processing tasks on the device itself and use the result as an additional means of user input and a source of context data. In this paper we present a system that turns such phones into mobile sensors for 2-dimensional visual codes. The proposed system induces a code coordinate system and visually detects phone movements. It also provides the rotation angle and the amount of tilting of the camera as additional input parameters. These features enable applications such as item selection and interaction with large-scale displays. With the code coordinate system, each point in the viewed image – and therefore arbitrarily shaped areas – can be linked to specific operations. A single image point can even be associated with multiple information aspects by taking different rotation and tilting angles into account.

1 Introduction

With the integration of CCD cameras, mobile phones have become networked personal image capture devices. As image resolution improves and computing power increases, they can do more interesting things than just taking pictures and sending them as multi media messages over the mobile phone network. For example, programmable camera phones can perform image processing tasks on the device itself and use the result as an additional means of input by the user and a source of context data.

In this paper, we present a visual code system that turns camera phones into mobile sensors for 2-dimensional visual codes. For that, we elaborate on and extend our initial ideas presented in [1]. We assume scenarios where camera phones are used to sense a scene that contains one or more visual codes. By recognizing a code tag, the device can determine the code value, the targeted object or image element (even if the object or image element itself is not equipped with a code tag), as well as additional parameters, such as the viewing angle of the camera. The system is integrated with a visual phone movement detection scheme, which provides three degrees of freedom and turns the mobile phone into an optical mouse. Code recognition and motion detection are completely performed on the phone itself. The phone's wireless communication channel is used to retrieve online content related to the selected image area or to trigger actions (either in the background infrastructure or on a nearby larger display), based on the sensed code and its parameters.

These features enable local interaction with physical objects, printed documents, as well as virtual objects displayed on electronic screens in the user's vicinity. Mobile phones are in reach of their users most of the time and are thus available in many everyday situations. They are therefore ideal bridging devices between items in the real world and associated entities in the virtual world. Visual codes provide visible "entry points" into the virtual world, starting from the local surroundings. This offers a natural way of local interaction and strengthens the role of mobile phones in a large number of usage scenarios. The visual code system also provides the basis for superimposing textual or graphical information over the camera image in close real-time in the sense of augmented reality. This entails a manifold of application possibilities in situations where information is to be closely linked to physical objects. An example is the maintenance of devices or apparatuses in the field: Individual parts of an apparatus are associated with visual codes. With the help of the visual code system, graphical information which is aligned with the items in the image, is superimposed over the camera image.

The novelty of the proposed system is its code coordinate system, the visual detection of phone movement, and the determination of the rotation angle and amount of tilting. These features enable interesting applications, beyond simple item selection, such as interaction with nearby active displays. The recognition algorithm precisely determines the coordinates of a targeted point in the code coordinate system, which is independent of the orientation of the camera relative to the code tag (distance, rotation, tilting) and also independent of the camera parameters (focal distance, etc.). This enables the association of each point in the viewed image – and therefore arbitrarily shaped areas – with specific operations. A single visual code can be associated with multiple such areas and a single image point can be associated with multiple information aspects using different rotation and tilting angles.

2 Related Work

Sony's *CyberCode* [2] is related to our approach, but does not operate on mobile phone class devices and does not use phone movement and other additional parameters for interaction in the way we propose. CyberCodes store 24 bits of data. In addition to the ID, the 3-D position of the tagged objects is determined. The proposed applications for CyberCodes are augmented reality systems, various direct manipulation techniques involving physical objects, and indoor guidance systems.

TRIP [3] is an indoor location tracking system based on printable circular markers, also called "TRIPtags". It employs CCD cameras plugged into standard PCs for code recognition, 3-D location, and orientation detection. TRIPtags have an address space of just 19683 (= 3^9) possible codes, which makes them impracticable to encode universally unique IDs, like Bluetooth MAC addresses. In contrast to our system, TRIP is designed for use with stationary cameras which are distributed in a networked environment. It relies on a CORBA infrastructure and a centralized recognition engine named “TRIPparser”. In our system, code recognition is completely done on the mobile phones, which enhances scalability, and code sightings are distributed wirelessly.

The *FieldMouse* [4] is a combination of a barcode reader and a pen-shaped mouse. The mouse detects relative (Δx, Δy) movement. If the location of a barcode on a flat surface is known to the system, absolute locations can be computed by first scanning the barcode and then moving the FieldMouse. This enables various kinds of paper-based GUIs.

A number of commercial efforts exist to recognize product codes with mobile phones. An example is *AirClic* (www.airclic.com), which provides tiny barcode readers that can be attached to mobile phones. The disadvantage of this approach is the necessity of an additional device, which increases the physical size and weight of the mobile phone and consumes additional energy. Barcode readers also do not provide the orientation and selection features of camera based approaches.

SpotCodes (www.highenergymagic.com) are a commercialized derivative of the TRIP tags mentioned above for use with camera phone devices. They recognize the rotation of the code tag in the image, but do not provide an orientation-independent code coordinate system and do not detect relative camera movement independent of codes in the camera image. A number of interaction possibilities are described on the Web page and in [5], such as rotation controls and sliders.

An increasing number of companies offer mobile phones with the ability to read *QR Codes* [6]. They implement the core functionality of decoding QR Codes. They do not, however, have the code coordinate system, rotation, tilting, and visual movement detection features that are integrated in our system.

The same applies to Semacode (semacode.org), which uses standard *Data Matrix* [7] codes to implement physical hyperlinks and load Web pages in the phone’s browser. Example applications are live urban information, such as the position of GPS-equipped buses, information on nearby shops and services, and semacodes on business cards and conference badges.

3 Visual Code, Recognition Algorithm, and Phone Movement Detection

The mobile phone devices we consider have severely limited computing resources and often lack a floating point unit. Hence, the use of floating point operations has to be minimized. The typical phone camera generates low to medium quality color images in VGA (640×480 pixels) or lower resolution. The relatively poor image quality determines the minimal size of code features that can be reliably recognized. The code features therefore have to be more coarsely grained than those of most available visual codes. In our evaluation it became clear that color should not be used as a code feature, because of the large differences in color values, depending on varying lighting conditions. Moreover, color codes are more expensive to print and harder to reproduce than simple black-and-white codes.

Because of the mobility inherent to camera phones, scanned codes might appear at any orientation in the camera image. They can be arbitrarily rotated and tilted, which complicates image recognition. We decided to constructively use these characteristics by measuring the amount of tilting and rotation of a code tag in the image and use them as additional input parameters. Another feature we deemed essential is the ability to map arbitrary points in the image plane to corresponding points in the code plane, i.e. to compute the code coordinates of arbitrary image pixels, and vice versa. In particular, this enables the conversion of the pixel coordinates of the camera focus – which is the point the user aims at – into corresponding code coordinates and the selection of image elements located at these code coordinates. This coordinate mapping can also be used for removing the perspective distortion of image parts.

These characteristics mark out the design space for the visual codes and form the basis for the further discussion. The code layout is pictured in Fig. 1. It consists of the following elements: a larger and a smaller guide bar for determining the location and orientation of the code, three cornerstones for detecting the distortion, and the data area with the actual code bits. The combination of larger and smaller guide bars is beneficial for detecting even strongly tilted codes. In the bottom of Fig. 1 the code coordinate system is shown. Each code defines its own local coordinate system with its origin at the upper left edge of the code and one unit corresponding to a single code bit element. Depending on the code size, the mapping between points in the image plane and points in the code plane is more precise than a single coordinate unit. The x-axis extends in horizontal direction to the left and to the right beyond the code itself. Correspondingly, the y-axis extends in vertical direction beyond the top and bottom edges of the code. For each code found in a particular input image, the code recognition algorithm establishes a bijective mapping between arbitrary points in the code plane and corresponding points in the image plane.

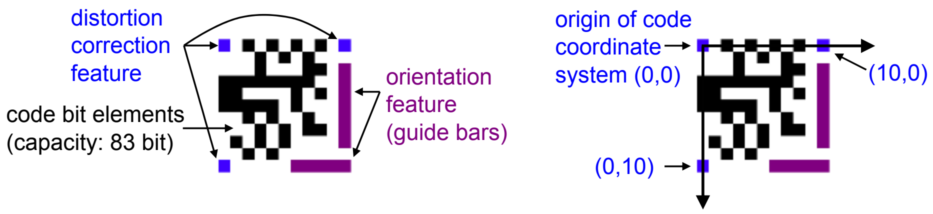


Fig. 1. The layout of the visual code (left) and the code coordinate system (right)

3.1 Recognition Algorithm

The recognition algorithm performs the following main steps on the camera image and produces a code information object for each detected code.

- **Input:** Camera image
- **Output:** Set of code information objects, comprising
  - the code value,
  - the image pixel coordinates of the corner stones and guide bars,
  - the rotation angle of the code in the image,
  - the amount of horizontal and vertical tilting,
  - the distance of the camera to the code,
  - a projective warper object for the code, which implements a planar homography (see below) used to transform image coordinates to code coordinates and vice versa,
  - the width and height of the originating image,
  - a flag indicating the result of error checking.

Gray Scaling and Adaptive Thresholding. To produce a gray scaled version of the colored input image, we use the formula $gray = (red + green)/2$, instead of the ITU-standardized formula for luminance $Y = 0.2126 \times red + 0.7152 \times green + 0.0722 \times blue$. The blue color component is omitted, since it has the lowest quality in terms of sharpness and contrast. Our simple formula is computationally efficient and produces an adequate starting point for thresholding. Efficiency in this step is of utmost importance for the performance of the whole recognition algorithm, because every single image pixel has to be gray scaled.

An adaptive thresholding method is taken to produce a black-and-white version of the gray scaled image, because the brightness of the camera image is not constant and the visual code may be unevenly illuminated. We slightly modified the adaptive thresholding algorithm described in [8], where the basic idea is to use a weighted moving average of the gray values while running through the image in a snake-like fashion (alternating left to right and right to left scanline traversal). Our adaptation takes the previous scanline of each examined scanline into account in order to avoid artifacts in every other line, resulting from the zigzag traversal of the scanlines. The average $g_s(n)$ is updated according to

$$g_s(n) = g_s(n-1) \cdot (1 - \frac{1}{s}) + p_n$$

with p_n denoting the gray value of the current pixel and $s = \frac{1}{8}w$ the width of the moving average (w is the image width). g_s is initialized with $g_s(0) = \frac{1}{2}cs$, where c is the maximum possible gray value. The color of the thresholded pixel $T(n)$ is then chosen as ($t = 15$):

$$T(n) = \begin{cases} 1 \text{ if } p_n < \frac{g_s(n)}{s} \cdot \frac{100-t}{100} \\ 0 \text{ otherwise} \end{cases}$$

Gray scaling and adaptive thresholding turned out to be the most time consuming phase of the recognition process. Therefore, we replaced any floating point operations in this part by shifted integer operations, which resulted in a significant performance improvement.

Region Identification and Labeling. This step consists of finding regions of neighboring black pixels, counting them, and assigning a number to each. The algorithm we use is a well known two-phase method: In the first phase, the image is traversed row by row, assigning preliminary labels to the regions found. During this process, it may happen that two regions with different labels turn out to be in fact the same region. In this case, the equivalence of the two temporary labels is stored in a table. The second phase resolves the equivalencies by merging the corresponding regions and assigns a final label to each region.

In the implementation, gray scaling, adaptive thresholding, and the first phase of region labeling are bundled for performance reasons and are done in a single scan through the image, i.e., pixels that are thresholded as foreground pixels are immediately assigned a label and any label equivalences are recorded.

Calculation of Region Shapes and Orientations. In order to identify candidates for orientation bars among the regions found, the notion of second-order moments is used [9]. From these moments, the major and minor axis of each region is determined. The ratio of the lengths of these axes is a good measure for the "eccentricity" of a region: perfect circles and squares have a ratio equal to one whereas line segments have a ratio close to zero. This is very useful to identify regions with a bar-like shape.

The second-order moments of a region consisting of the set of pixels R and having the center of gravity $(\bar{x}, \bar{y})$ are defined as follows:

$$\mu_{xx} = \frac{1}{|R|} \sum_{(x,y)\in R} (x - \bar{x})^2,$$

$$\mu_{yy} = \frac{1}{|R|} \sum_{(x,y)\in R} (y - \bar{y})^2,$$

$$\mu_{xy} = \frac{1}{|R|} \sum_{(x,y)\in R} (x - \bar{x})(y - \bar{y}),$$

$$\text{where} \quad \bar{x} = \frac{1}{|R|} \sum_{(x,y)\in R} x, \qquad \bar{y} = \frac{1}{|R|} \sum_{(x,y)\in R} y$$

From these moments, an ellipsis $E = \{(x, y)|dx^2 + 2exy + fy^2 \leq 1\}$ that has the same major and minor axis as the region can be defined by setting:

$$\begin{pmatrix} d & e \\ e & f \end{pmatrix} = \frac{1}{4\mu_{xx}\mu_{yy} - \mu_{xy}^2} \begin{pmatrix} \mu_{yy} & -\mu_{xy} \\ -\mu_{xy} & \mu_{xx} \end{pmatrix}$$

Furthermore, the orientation vector of the major axis is calculated as

$$\begin{pmatrix} -\sin\alpha \\ \cos\alpha \end{pmatrix}, \quad \text{where} \quad \alpha = \frac{1}{2} \arctan \frac{2e}{d - f}.$$

Locating and Evaluating the Codes. Locating codes in the image is done by looking for guide bar candidates and by finding corresponding cornerstones. Guide bar candidates are found by simply selecting those regions whose axis ratio lies in a certain range around the expected ideal axis ratio. The range has to be large enough to allow for tilted codes. For each of these candidates, the size and orientation of the region is used to estimate the expected positions of the second guide bar and the three cornerstones. It is then checked whether these features are actually present at the estimated positions. Cornerstone candidates found are only accepted if their axis ratio is above a certain limit.

Computing the Projective Mapping from Code Coordinates to Image Coordinates. Since the code elements are coplanar, there exists a unique homography (projective transformation matrix) between the code plane and the image plane. The projective mapping can be calculated once four corresponding points are known [10]. In our algorithm, the correspondences are the centers of the three cornerstones plus the center of the second guide bar:

Code element	Image coordinates	Code coordinates
upper left cornerstone	(x_0, y_0)	$(0, 0)$
upper right cornerstone	(x_1, y_1)	$(10, 0)$
second guide bar	(x_2, y_2)	$(8, 10)$
lower left cornerstone	(x_3, y_3)	$(0, 10)$

Code coordinates (u, v) are mapped to image coordinates (x, y) with

$$x = \frac{au + bv + 10c}{gu + hv + 10}, \quad y = \frac{du + ev + 10f}{gu + hv + 10}.$$

The parameters a to h are calculated from the four reference points (x_i, y_i), $i \in \{0, \ldots, 3\}$, as follows:

$$\Delta x_1 = x_1 - x_2 \qquad \Delta y_1 = y_1 - y_2 \qquad \Delta x_2 = x_3 - x_2 \qquad \Delta y_2 = y_3 - y_2$$

$$\Sigma x = 0.8x_0 - 0.8x_1 + x_2 - x_3 \qquad \Sigma y = 0.8y_0 - 0.8y_1 + y_2 - y_3$$

$$g = \frac{\Sigma x \Delta y_2 - \Sigma y \Delta x_2}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} \qquad a = x_1 - x_0 + gx_1 \qquad d = y_1 - y_0 + gy_1$$

$$h = \frac{\Sigma y \Delta x_1 - \Sigma x \Delta y_1}{\Delta x_1 \Delta y_2 - \Delta y_1 \Delta x_2} \qquad b = x_3 - x_0 + hx_3 \qquad e = y_3 - y_0 + hy_3$$

$$c = x_0 \qquad f = y_0$$

Computing the Projective Mapping from Image Coordinates to Code Coordinates. The inverse mapping to the one described above is important for applications which select items visible in the image. Given the coordinates of a pixel, the corresponding code coordinates can thus be obtained. Image coordinates (x, y) are mapped to code coordinates (u, v) as follows:

$$u = 10 \cdot \frac{Ax + By + C}{Gx + Hy + I}, \quad v = 10 \cdot \frac{Dx + Ey + F}{Gx + Hy + I}, \quad \textit{with}$$

$$\begin{array}{lll} A = e - fh & D = fg - d & G = dh - eg \\ B = ch - b & E = a - cg & H = bg - ah \\ C = bf - ce & F = cd - af & I = ae - bd \end{array}$$

Rotation Angle. The rotation angle gives the rotation of the visual code in the image in degrees counterclockwise (0-359°). A code that has the same orientation as the image has rotation angle 0° (like the ones in Fig. 1). The rotation is determined by mapping the points (0,0) and (100,0) from the code coordinate system to the image coordinate system, resulting in the image points (a_x, a_y), and (b_x, b_y). The rotation angle is then determined as the arc tangent of the difference quotient of a and b.

Horizontal and Vertical Tilting. The term *tilting* denotes the amount of inclination of the image plane relative to the code plane. *Horizontal tilting* is the amount of inclination of the image plane relative to the horizontal axis of the code. Analogously, *vertical tilting* denotes the amount of inclination of the image plane relative to the vertical axis of the code. Tilting values of 1 mean no tilting, a value less than 1 means tilting towards the left/top, and a value greater than 1 means tilting towards the right/bottom.

The tilting parameters are computed as follows: Four image points with constant distance h (image height) from the image center point in the axis directions of the code coordinate system are computed. They are mapped to corresponding code coordinates and their distances to the center point are computed. The ratios of these distances determine the tilting parameters t_x and t_y. They are independent of the size of the code in the image. From these ratios the tilting angles t_x^α and t_y^α can be determined, if a constant r is known that depends on the camera parameters. It can be obtained experimentally. For the Nokia 6600, e.g., this parameter has the value $r = 1.64177$.

$$\begin{aligned} i &= \text{image coordinates of the image center point} \\ c &= \text{CodeCoordinates}(i) \end{aligned}$$

$$\begin{aligned} x &= \text{ImageCoordinates}(c + (1,0)) - i \\ y &= \text{ImageCoordinates}(c + (0,1)) - i \\ u &= x/|x| \\ v &= y/|y| \end{aligned}$$

$$\begin{aligned} l &= |\text{CodeCoordinates}(i - hu) - c| \\ r &= |\text{CodeCoordinates}(i + hu) - c| \\ t &= |\text{CodeCoordinates}(i - hv) - c| \\ b &= |\text{CodeCoordinates}(i + hv) - c| \end{aligned}$$

$$t_x = l/r$$
$$t_y = t/b$$

$$t_x^\alpha = \arctan\left(r\frac{t_x - 1}{t_x + 1}\right)$$
$$t_y^\alpha = \arctan\left(r\frac{t_y - 1}{t_y + 1}\right)$$

Code Distance. If the real code size s_{real} (the real distance between the centers of the upper left and the upper right cornerstones) and the camera's focal distance f are known, the metric distance from the camera to the untilted visual code can be computed from s_{image} (the pixel distance between the centers of the upper cornerstones in the camera image) using the pinhole model as (w_{image} is the pixel width of the image)

$$D_{camera,code} = \frac{s_{real} \times f}{s_{image}/w_{image}}.$$

Since s_{real} and f are typically not known and we want to use the code distance for interaction purposes rather than measuring its exact value, we define the distance in terms of the size of the visual code in the image. We set $d_{camera,code} := 100$ for the farthest distance at which a code is recognized in view finder mode. For the Nokia 6600 this is the case when $s_{image} = 25$ pixels, which amounts to 15.625% of the image width. Hence the distance is computed as

$$d_{camera,code} = \frac{15.625}{s_{image}/w_{image}}.$$

Should s_{real} and f be known, the metric distance can still be computed from $d_{camera,code}$. For the Nokia 6600, the range of distances at which codes are recognized in view finder mode are: $11 - 46$ cm for $s_{real} = 69$ mm, $3.5 - 14$ cm for $s_{real} = 21$ mm, $2.3 - 9$ cm for $s_{real} = 13.6$ mm.

Reading the Encoded Bits. Once the positions of the guide bars and corner stones have been identified and a suitable projective mapping has been computed, reading the encoded bits is simply a matter of testing the appropriate pixels (x, y) of the black-and-white image, using code coordinates (u, v) with $u, v \in \{0, ..., 10\}$ and $(x, y) = \text{ImageCoordinates}((u, v))$.

Error Detection. In order to detect pixel errors and false orientation features, the code bits are protected by an (83,76,3) linear code that generates an 83-bit code word from a 76-bit value and has a Hamming distance of 3.

3.2 Phone Movement Detection

The code recognition algorithm is combined with a phone movement detection algorithm that solely relies on image data obtained from the camera. It does not require any additional hardware components, such as accelerometers. It is integrated with the visual code recognition algorithm in such a way that the latter only examines images for visual codes when the detected phone movement is below a certain threshold. If the phone is quickly moved, it is very unlikely that the user aims at a specific code. Trying to locate codes in the image in such a case would not be sensible.

The algorithm provides the relative x, y, and *rotational* motion of the phone, representing three degrees of freedom. With the movement detection, the camera phone thus becomes an optical mouse. The algorithm works as follows: Successive images from the camera are dispatched to the view finder to render them on the device display. Every n-th frame (depending on the performance of the phone) is used for phone movement detection. The image is divided in 16×16 pixel blocks. For each block, 16 pixels are sampled (out of the 256 available pixels in each block) and their average gray value is computed. Then, the blocks of the current image are compared to the blocks of the previously sampled frame. The block arrays are displaced against each other in x and y direction using values for Δx and Δy from $\{-3, \ldots, 3\}$. The difference values are computed and normalized with the number of compared blocks (which depends on the amount of displacement) and the minimal difference is chosen as the most likely amount of $(\Delta x, \Delta y)$ movement relative to the image before. Relative rotation θ is computed in a similar fashion, but rotating the block images against each other. The current block image is rotated by $\Delta\alpha$ values between $-24°$ and $24°$, with a step width of $6°$. The rotational coordinate mappings are precomputed and stored in tables for performance reasons. Again, the differences of the resulting block images are compared to the previous block image and the minimal difference is chosen as the most likely amount of relative rotation.

This simple algorithm works quite reliably and detects the relative motion even if the sampled backgrounds only have a limited number of features, like a wall or a floor. Because only a few pixels are sampled, the algorithm performs quickly and leaves enough time for doing the actual code recognition. On a Nokia 6600, it produces about five (x, y, θ) triples per second.

The code recognition and motion detection algorithms were implemented in C++ for Symbian OS (v6.1, v7.0s, and v8.0a). Replacing floating point operations by shifted integer operations reduced the time consumption of the thresholding phase from 2000 ms to less than 400 ms on a Nokia 7650 for a 640×480 pixel camera image. The total execution time of the recognition algorithm on the same device amounts to about 700 ms if less than 5 codes are present, and up to 1500 ms if 30 codes are present – which is rather uncommon in typical applications. The picture-taking process for 640×480 pixel images takes about 850 ms, resulting in an overall average delay of about 2000 ms. Low resolution 160×120 pixel images that are generated during the view finding process are recognized much faster, i.e. in close real-time as the device moves relative to the detected code(s).

4 Item Selection Using Relative Focus Position, Rotation Angle, and Tilting Determination

In this section we show how the additional input parameters that the code recognition algorithm provides can be combined to realize novel interaction patterns.

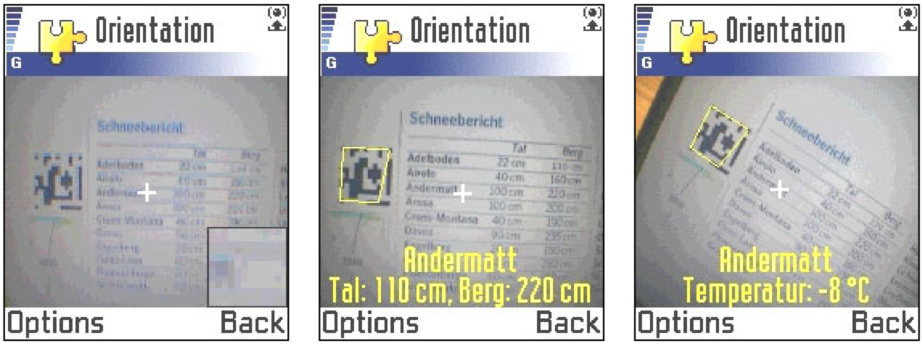
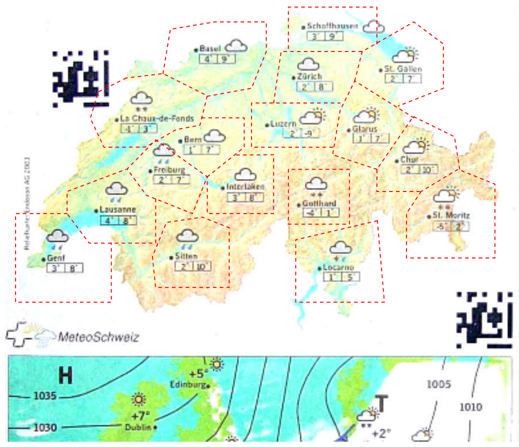


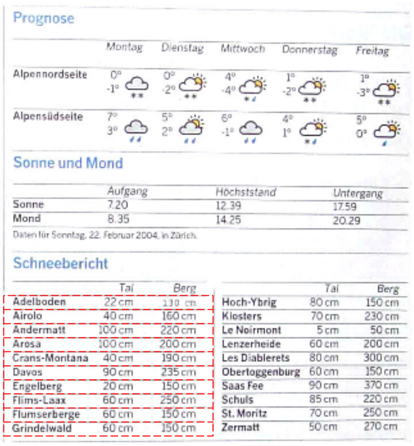
Fig. 2. Selection from a table: the code coordinates determine the table row, the camera rotation specifies the concrete information aspect to display

When aiming the phone camera at the target item, the image of this target item appears on the display. It is continuously updated as the phone is moved. The center of the display contains a crosshair to facilitate precise selection, as can be seen in the screenshots in Fig. 2. To further facilitate item selection, the display contains a magnified portion of the area around the display center. The level of magnification can be adjusted with the joystick. The mapping from image coordinate system to code coordinate system enables the precise selection of items in the image, requiring just a single code element for multiple items. Image items may be menu entries, arbitrarily shaped regions in a picture, or the cells of a table.

Further input parameters comprise the rotation of the code tag in the image and the amount of tilting of the image plane relative to the code plane. The tilting parameter identifies the viewing position (from left, from right, from top, from bottom). Both parameters can be used to associate multiple information aspects with a single point in the code coordinate system.

For an effective interaction, the user has to be provided with indications about the possible interactions. This can be achieved by superimposing visual cues on the display image that indicate at what rotation angles and viewing positions what kind of information is to be expected. We currently investigate different kinds of symbols that guide the user in his or her interactions with visual codes. An indication of user interaction normally consists of a symbol denoting the kinds of physical interaction – like movement, rotation, or tilting – and a set of symbols for the associated actions that are triggered as a consequence of the interaction. The latter comprise symbols for typical functions of a mobile phone, such as placing a phone call or starting the WAP browser. Another possibility is to print interaction cues next to the code. This was realized with a visual code dialer application. The printed code contains a phone number and is surrounded by words indicating the function that is triggered when the phone is tilted in that direction: Just below the code it says “Call”, to the left it says “SMS”, and to the right the word “Store” is printed. Scanning from a central position immediately places a call, scanning from the left opens the phone’s SMS editor with the number already entered into the appropriate field, and scanning from the right looks up the contact information on a server and stores it on the phone.





Prognose

	Montag	Dienstag	Mittwoch	Donnerstag	Freitag
Alpennordseite	0° -1°	0° -2°	4° -4°	1° -2°	1° -3°
Alpensüdseite	7° 3°	5° 2°	6° -1°	4° 1°	5° 0°

Sonne und Mond

	Aufgang	Höchststand	Untergang
Sonne	7.20	12.39	17.59
Mond	8.35	14.25	20.29

Daten für Sonntag, 22. Februar 2004, in Zürich.

Schneebericht

	Tal	Berg
Adelboden	22 cm	130 cm
Airolo	40 cm	160 cm
Andermatt	100 cm	220 cm
Arosa	100 cm	200 cm
Crans-Montana	40 cm	190 cm
Davos	90 cm	235 cm
Engelberg	20 cm	150 cm
Flims-Laax	60 cm	250 cm
Flumserberge	60 cm	150 cm
Grindelwald	60 cm	150 cm

	Tal	Berg
Hoch-Ybrig	80 cm	150 cm
Klosters	70 cm	230 cm
Le Noirmont	5 cm	50 cm
Lenzerheide	60 cm	200 cm
Les Diablerets	80 cm	300 cm
Obertoggenburg	60 cm	150 cm
Saas Fee	90 cm	370 cm
Schuls	85 cm	220 cm
St. Moritz	70 cm	250 cm
Zermatt	50 cm	270 cm

Fig. 3. Example of a weather forecast newspaper page containing visual codes. The 17 regions on the map and all entries in the table are individually mapped to different URLs and thus hyperlinked to specific online content

In newspapers, online background information to articles, advertisements, or information which quickly gets obsolete, like weather forecasts or stock quotes, can be linked via visual codes. Fig. 3 shows a cut-out of a newspaper page containing a geographic map with the current weather data and a table containing the snow conditions for various regions. The dotted lines drawn on the newspaper page indicate sensitive areas that are individually linked to online content. Such a mapping can be easily created with suitable content creation software. As a prototype, we developed a mapping tool for drawing the areas in the image and specifying the associated URL for each region. The tool computes the coordinates of these areas in the coordinate systems of the codes present in the image, and stores this data as an XML file. Multiple URLs can be specified for each region by taking into account rotation, distance, and tilting. As shown in Fig. 3, a single code suffices to select any one of the multiple areas and table entries, respectively. By rotating the mobile device, different aspects of the online information can be chosen: In the example, vertical orientation shows the snow depth for the selected area, while a slight rotation shows the current temperature. Other conceivable operations include showing the currently open skiing trails, calling the local tourist information office, and booking rail and lift tickets. The current weather data is retrieved from a server and the display of the phone is updated in real time as the crosshair is moved across the table entries and geographic regions and as the phone is rotated clockwise and counterclockwise. A video that demonstrates this type of interaction is available at `visualcodes.sourceforge.net`.

The ability to link multiple items to a single code based on their code coordinates and to associate multiple information aspects to a single point depending on rotation and tilting has a number of usability advantages. In the example above, it would of course be possible to present a table of the current snow conditions of all regions on the map to the user once the code has been recognized. But it is difficult to effectively show a table containing all the attributes on the small amount of available display space. It also requires the user to scroll through the – possibly lengthy – table and locate the data of interest in a second step. The presented approach avoids both of these problems. It gives direct and immediate feedback to the user and presents exactly the scanned item and selected information aspect.

5 Visual Codes and Large-Scale Displays

A compelling class of applications for visual codes, the code coordinate system feature, and the relative movement detection involves the interaction with large-scale wall displays, as they are increasingly found in public or semi-public places. Today's wall displays are mostly limited to the passive reception of information. At public places, keyboards and other input devices are often not installed, because of potential problems with vandalism. If passers-by carry their own interaction devices in the form of a camera phone, this problem is eliminated. Example locations for interactive public wall displays are train stations, airports, bus stops, shopping malls, and museums.

To investigate the interaction possibilities, we set up a projected display that contains a 3-D model of a number of visual codes arranged on an invisible sphere as shown in Fig. 4. The model is implemented using Java 3D [11]. The screen model is controlled by the motion of the camera phone. The phone and the screen are connected via Bluetooth. Motion updates are sent as (x, y, θ) triples to the active display. Phone movement in horizontal direction results in a rotation of the sphere around the y-axis, vertical motion results in a rotation around the x-axis, and rotating the phone results in sphere rotation around the z-axis. Because of the relatively low update rate, the movement of the sphere is interpolated between motion updates from the phone, in order to obtain a smooth visually pleasing movement. The updates are frequent enough to be able to effectively control the display contents.

In addition to rotating the sphere containing the codes, aiming at a visual code shown on the wall display brings up an associated menu. Individual menu

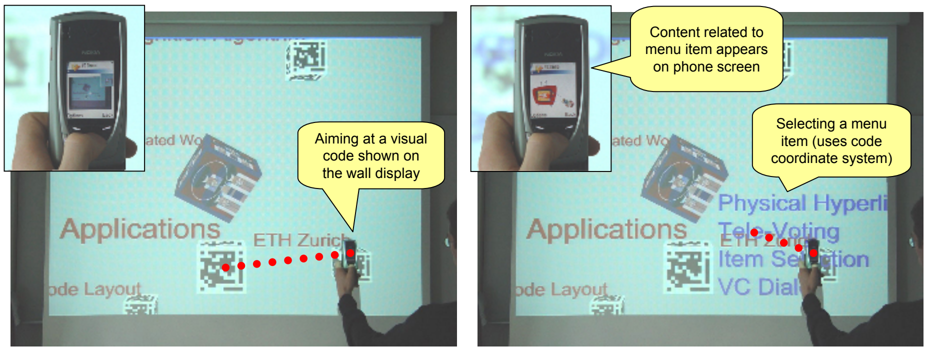

Aiming at a visual code shown on the wall display
Content related to menu item appears on phone screen
Selecting a menu item (uses code coordinate system)
Applications
ETH Zurich
Physical Hyperli
Tele-Voting
Item Selection
VC Dial

Fig. 4. Phone movement detection and item selection for interaction with a projected wall display

items can now be selected, whereupon the related content is transferred to the phone and shown on the device screen. In the demo application, the visual codes are permanently visible. They could also be superimposed over the large-scale display image just before scanning them. This could be synchronized by the Bluetooth connection between the phone and the display.

The motion detection was informally tested by a number of subjects and worked very well. We provided the test subjects with a number of tasks, such as rotating a certain menu to the foreground and selecting a specific menu item. After a short period of practice, the subjects quickly became familiar with this type of interaction. This application was shown as a demonstration at [12].

We are currently investigating the idea of a "Photo Wall", which uses a large-scale display to organize photos taken with the camera phone, because this is difficult to do on the device itself. The large-scale display is also used as an access point to a photo printing service or one's online photo collection. The interaction mechanisms described above are investigated to perform interaction tasks such as navigating through, selecting, deleting, rotating, and manipulating the photos.

6 Conclusion

In this paper we have presented extended features of a visual code system for camera equipped mobile phones. It performs well on resource constrained phone devices with low to medium resolution cameras. Besides detecting visual codes in the user's vicinity and thus linking physical objects to online content, it has a number of supplementary features. It provides the code coordinates, the code rotation angle, and the tilting of the image plane relative to the code plane as additional input parameters. These parameters can be determined without prior calibration. Phone movement detection is integrated with the visual code system. It provides (x, y, θ) motion parameters and turns the mobile phone into an optical mouse.

We have shown how these input parameters can be used and combined to provide novel interaction patterns with objects in the user's local environment. The user can pick up multiple information items which are located at known code coordinate positions relative to a single code tag, by aiming the camera focus at the appropriate location. By slightly rotating or tilting the phone, the user has the opportunity to select between different information aspects. We have also shown how phone movement detection and visual code recognition can be combined to interact with individual items on a large-scale wall display.

In the future, camera phones might play a prominent role as ubiquitous personal image recognition devices and for local interaction with physical objects that their users encounter in everyday settings. New services can be associated with printed documents, wall displays, TV programs, or general consumer products when they are made interactive by techniques as described in this paper.

References

1. 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, Vienna, Austria, Austrian Computer Society (OCG) (2004) 265–271
2. Rekimoto, J., Ayatsuka, Y.: CyberCode: Designing augmented reality environments with visual tags. In: DARE '00: Proceedings of DARE 2000 on Designing augmented reality environments, ACM Press (2000) 1–10
3. de Ipiña, D.L., Mendonça, P.R.S., Hopper, A.: TRIP: A low-cost vision-based location system for ubiquitous computing. Personal Ubiquitous Comput. **6** (2002) 206–219
4. Siio, I., Masui, T., Fukuchi, K.: Real-world interaction using the FieldMouse. In: UIST '99: Proceedings of the 12th annual ACM symposium on User interface software and technology, ACM Press (1999) 113–119
5. Madhavapeddy, A., Scott, D., Sharp, R., Upton, E.: Using camera-phones to enhance human-computer interaction. In: Sixth International Conference on Ubiquitous Computing (Adjunct Proceedings: Demos). (2004)
6. International Organization for Standardization: Information Technology – Automatic Identification and Data Capture Techniques – Bar Code Symbology – QR Code. ISO/IEC 18004 (2000)
7. International Organization for Standardization: Information Technology – International Symbology Specification – Data Matrix. ISO/IEC 16022 (2000)
8. Wellner, P.D.: Adaptive thresholding for the DigitalDesk. Technical Report EPC-93-110, Rank Xerox Research Centre, Cambridge, UK (1993)
9. Veltkamp, R.C., Hagedoorn, M.: State of the art in shape matching – principles of visual information retrieval. In Lew, M.S., ed.: Series in Advances in Pattern Recognition, Springer (2001)
10. Heckbert, P.S.: Fundamentals of texture mapping and image warping. Master's Thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley (1989)

11. Sun Microsystems: Java 3D API Specification, version 1.3. Available at: java.sun.com/products/java-media/3D (2002)
12. Rohs, M., Gfeller, B.: Visual code recognition for camera-equipped mobile phones. Pervasive 2004 Demonstrations D06. Demonstration description available at: www.pervasive2004.org/program_demonstrations.php (2004)

Appendix

We have explored the integrated phone movement detection features in a number of ways. As the camera detects phone movement relative to the background, the content of the phone display is continuously updated. No visual code needs to be present in the view of the camera. With this technique we have built a camera controlled wireframe model of a house, a pong game whose slider can be controlled by tilting the wrist left and right, and an application showing a large subway map that is scrolled in response to phone movement. These applications are shown in Fig. 5 and are available for download at `visualcodes.sourceforge.net`.

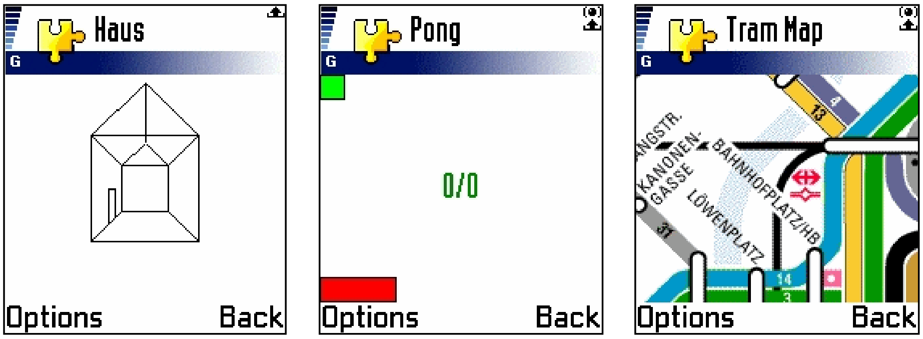


Fig. 5. Wireframe model (left), pong game (middle), and subway map (right), all controlled by the visual detection of phone movement