# PEYE: Toward a Visual Motion Based Perceptual Interface for Mobile Devices

Gang Hua[1], Ting-Yi Yang[2], and Srinath Vasireddy[2]

[1] Microsoft Live Labs Research, One Microsoft Way, Redmond, WA 98052, USA
[2] Mirosoft Live Labs Engineering, One Microsoft Way, Redmond WA 98052, USA
{ganghua,tingyiy,srinathv}@mirosoft.com

**Abstract.** We present the architecture and algorithm design of a visual motion based perceptual interface for mobile devices with cameras. In addition to motion vector, we use the term "visual motion" to be any dynamic changes on consecutive image frames. In the lower architectural hierarchy, visual motion events are defined by identifying distinctive motion patterns. In the higher hierarchy, these visual events are used for interacting with user applications. We present an approach to context aware motion vector estimation to better tradeoff between speed and accuracy. It switches among a set of motion estimation algorithms of different speeds and precisions based on system context such as computation load and battery level. For example, when the CPU is heavily loaded or the battery level is low, we switch to a fast but less accurate algorithm, and vice versa. Moreover, to obtain more accurate motion vectors, we propose to adapt the search center of fast block matching methods based on previous motion vectors. Both quantitative evaluation of algorithms and subjective usability study are conducted. It is demonstrated that the proposed approach is very robust yet efficient.

## 1 Introduction

Video cameras have become a standard setting for all kinds of mobile devices, such as mobile phones, Pocket-PCs and personal digital assistants (PDAs). However, so far their functionalities are mostly limited to taking pictures or small video clips, mainly for entertainment or memorandum purpose. As an alternative to small keyboard, D-pad or touch screens, we can indeed leverage the rich visual information from the cameras for more convenient and non-invasive human machine interaction. This becomes of special interest due to the recent advancement of computer vision based human computer interaction (HCI) techniques [1,2,3] and the ever increasing computational power the mobile devices have.

To summarize some previous works, TinyMotion [4,5] may be one of the first general visual motion sensing systems for camera phones, which estimates the motion vectors by full search block matching (FSBM) [6] on grid samples. Other earlier work include the bar-code readers [7,8], where the phone cameras are used for reading highly regular shaped bar-code sheet to fire commands for the mobile phones. Hannuksela et. al [9] proposed a sparse motion estimation algorithm for vision based HCI on mobile devices, but they did not really come out of any real

M. Lew et al. (Eds.): HCI 2007, LNCS 4796, pp. 39–48, 2007.

systems to use that. Some other research works even have investigated using mobile phone cameras for interacting with large displays [10].

Notwithstanding the demonstrated success of the systems discussed above, they have not fully leveraged the richness of the visual information we can leverage. For example, even the most recent TinyMotion [4,5] system has only utilized a fraction of the distinctive motion patterns to trigger the commands. What is worse, systems like bar-code readers [10,7] are obviously of limited use because they must be played with some distinctive visual patterns in a planar surface.

In this paper, we present the architecture design of a visual motion based perceptual interface for mobile devices. Here we use the term "visual motion" to incorporate any dynamic changes of image features. The overall architecture has two layers: at the lower-level of the hierarchy, we define a set of visual motion events, e.g., "left", "right", "up", "down", "rotate clockwise", "rotate anti-clockwise", "blurred", and "darkened", based on different characteristic motion patterns. These visual events are fired through an integrated application programming interface (API), which we call *portable eye*, or in short PEYE. At the higher hierarchy, the applications can then be manipulated by these visual events. In view of this, we provide more freedom for the application developers to utilize these events for their own purpose.

The visual motion we exploited include motion vector, blur-ness and lightness changes, among which motion vectors still account for the majority of the visual events. Unlike TinyMotion [4,5], where FSBM is adopted for motion estimation, we explore a set of fast block matching methods such as three step search (TSS) [11], four step search (FSS) [12], diamond search (DS) [13], hexagon search (HS) [14], and the adaptive multiple-mode search (AMMS) [15]. To further improve the accuracy of all the above methods, we propose an adaptive search center scheme by pre-matching three points along the previous motion vectors. We then design an online scheme to switch among the different motion estimation algorithms, which makes a compromise between matching quality and computational expenses. For example, when the CPU is heavily loaded or the battery is low, we switch to a faster but less accurate method such as AMMS, DS, or HS, and vice versa to a more accurate but slower algorithm such as TSS and FSS. We call such a scheme *context aware motion estimation*.

Our contributions reside in three folds: firstly, we presented a novel hierarchical architecture design of visual motion based perceptual interface for mobile devices. Secondly, we designed a context aware motion estimation scheme, which better tradeoffs speed and accuracy for motion estimation. Thirdly, to the best of our knowledge, we are the first to utilize visual cues such as blur-ness and dark-ness changes to define visual events for HCI. Our system runs $25-30$ frames per second, depending on the frame rate of the mobile devices.

## 2 Platform Architecture

The PEYE API is developed using DirectShow under Windows CE. The overall platform architecture is presented in Fig. 1. It can be divided into three

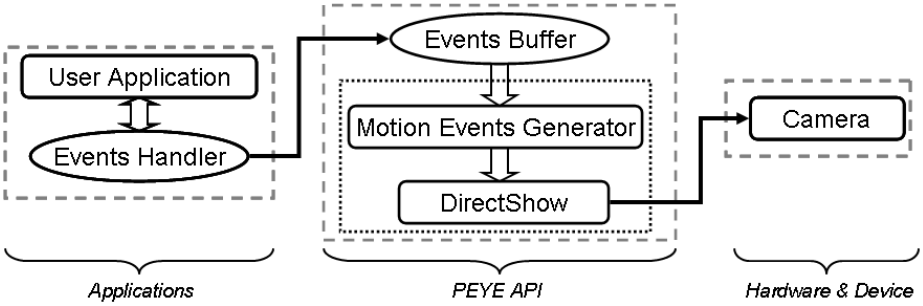


**Fig. 1.** The overall architecture of the PEYE system

hierarchies: the applications, the PEYE API, and the hardware devices (e.g., the camera). The core is the PEYE API, which analyzes the captured videos, and sends the *visual events* to the user applications. When the user applications receive these events, it responds just as it responds to any other *Windows messages*. We must note that under other mobile operating system, the video interface may need to be changed but the overall architecture is still valid.

The architecture design of our system distinguishes itself from previous ones (e.g., the TinyMotion [4,5]) in the sense that the motion analysis module is clearly separated from the applications. All communications between the user applications and the PEYE API are carried out through the message system of the OS. In other words, the user applications do not need to code any additional interfaces to export the results from the visual motion analysis module. Our hierarchical design also enables the user applications to have more flexibilities to respond to the different visual events.

## 3 Visual Events

Based on the different visual cues utilized, the visual events we exploited in PEYE fall into three categories: motion gestural, lightness, and blur-ness events.

### 3.1 Motion Gestural Events

The motion vectors provide the richest source of visual events, such as "left", "right", "up", "down", "pan", "tilt", etc. To detect these motion patterns, we need to estimate the motion vectors firstly.

**Context Aware Motion Estimation with Adaptive Search Center.** Due to the limited computational resource for PEYE, we can not afford dense optical flow estimation. Thus we partition the image into 4 equal regions, and perform fast block matching for the 4 center blocks of size $16 \times 16$, as illustrated in Fig. 2. We use the sum of squared difference (SSD) of pixel intensities as the matching

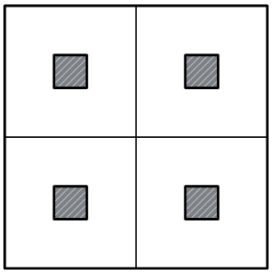

**Fig. 2.** Each 16 × 16 matching blocks (shaded small rectangles) is centered at one of the four equally sized partitions of the images

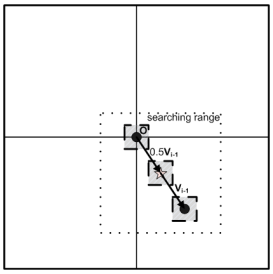


**Fig. 3.** The adaptive search center: $\boldsymbol{v}_{i-1}$ represents the previous motion vector. Three blocks are checked and the center of the best match is the new search center.

criterion. In our implementation, the square of pixel differences are implemented using a look-up table, which speeds up the matching at least 8 times.

Instead of utilizing a single matching method, we explored a set of fast block matching methods including TSS [11], FSS [12], DS [13], HS [14], and the AMMS [15]. These different fast matching algorithms are all faster than FSBM [6], although the gain in speed is at the expense of the accuracy to different extent. For example, in general HS is the fastest but least accurate one, whilst DS may be among the slowest several but it is more accurate. We refer the readers to the corresponding references for details of the different algorithms.

We improve all these fast block matching methods by introducing an adaptive search center scheme. Denote $\boldsymbol{v}_{i-1}$ as the motion vector estimated from the previous frame, also denote $\mathbf{O}$ as the zero search center, which can be the center of any of the shaded blocks in Fig. 2. We first match the template block at three points $\mathbf{O}$, $\mathbf{O} + \frac{1}{2}\boldsymbol{v}_{i-1}$, and $\mathbf{O} + \boldsymbol{v}_{i-1}$. The one with the smallest SSD will be chosen as the search center to start any of the aforementioned fast block matching methods, as illustrated in Fig. 3. We add the suffix "AC" to denote adaptive search center, e.g., FSSAC stands for FSS with adaptive search center.

Our study reveals that these different methods may perform differently over different motion patterns and/or different scenes. This motivated us to design an online selection scheme to switch among this set of motion estimation algorithms. The followings are general selection conditions with descending priorities:

1. If the battery power of the mobile device is below a certain level, then we always switch to HSAC (i.e., HS with adaptive search center) since it is the fastest one.
2. If the CPU is heavily loaded, we will select only between HSAC and AMMSAC, otherwise select among all algorithms, using Condition 3.
3. Based on the historical matching accuracies of each of the candidate methods, choose the most accurate one. Note the historical matching accuracy of each method is obtained by evaluating its accuracies every $K$ frames and is accumulated over a shifting time window.

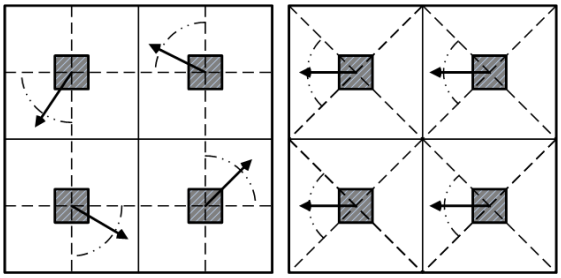

**Fig. 4.** The motion patterns associated with "rotate clockwise" and "right" events. The arcs indicate the direction ranges the motion vectors should fall into. The events are named according to camera motion, which is the reverse of camera motion. (Left figure: "rotate clockwise"; Right figure: "right")).

The above scheme enables the system to switch to a more accurate but slower matching method when more computation resource is available, and vice versa. We call it as *context aware motion estimation*. The motion vectors for each block are averaged over 5 consecutive frames to reduce the effects of jittering noise. Another engineering issue we should highlight here is that all the methods discussed above work with grey-scale images. In some of the mobile devices, the video stream is in YUV format, in that case the Y component is used directly. When the video stream is of the format $RGB$, we use bit shift to transform it to a grey scale image [5], i.e., $Y = (R >> 2) + (G >> 1) + (G >> 3) + (B >> 3)$, which approximates to the RGB to Y conversion formula.

**Gestural Events.** The motion gestural events can then be defined based on the four motion vectors estimated for each frame. For example, if all four motion vectors are going *left*, then the camera motion and thus the gestural event is "right", since we are indeed moving the camera to the *right*. Fig. 4 presents two motion patterns, which defines the motion gestural events "rotate clockwise" and "right". Other motion gestural events are defined in a similar fashion.

### 3.2 Lightness Events

The overall lightness of the image is also a very useful visual cue to define visual events, i.e., the "darkness" change events. Assuming that the environment lighting is not too dark, the image can only be dark if the camera view is occluded. Since the users can very conveniently cover the camera view by hand, it is natural and non-invasive for them to use it to interact with the mobile applications.

We estimate the darkness of the images by estimating the average pixel intensity $I_{ave}$ over the four matching blocks defined in Fig. 2. If $I_{ave}$ is below a threshold $T_I$, then it fires the event "darkened". The threshold $T_I$ is set by collecting a set of images using the mobile cameras under different lighting conditions during which we also intentionally use our hand to block the view of the camera from time to time. We fit a Gaussian distribution with mean $\mu_I$

and variance $\sigma_I^2$ on the average pixel intensities over all these image frames we collected. The threshold is set to be $\mu_I - 3\sigma_I$.

### 3.3 Blur-ness Events

There are mainly two types of blurs: the *de-focus* blur and the *motion* blur. De-focus blur happens when the mobile camera is out of focus, and motion blur happens when either the object or the camera is moving fast. In spite of the different causes, the blurred images present common characteristics – the lack of sharp object boundaries. This results in a very simple yet effective method to judge if an image is blurred. Denote $I_x(i,j)$ and $I_y(i,j)$ as the image gradient in $x$ and $y$ directions at pixel location $(i,j)$, respectively. We define

$$I_{Blur} = \sum_{k=1}^{4} \sum_{(i,j)\in B_k} |I_x(i,j)| + |I_y(i,j)| \tag{1}$$

where $B_k$ indicates the $25 \times 25$ block centered at the $k^{th}$ block defined in Fig. 2. If $I_{Blur}$ is smaller than a threshold $T_{Blur}$, the image is regarded as being blurred.

To distinguish between the de-focus blur and the motion blur, we exclude the case where both of them present simultaneously. Notice that for motion blur, the background scene is subject to dramatic change due to the large motion, while for de-focus blur, the frame differences should be subtle since the motion is small. Denote $I_t(i,j)$ as the pixel intensity at time frame $t$. We then define

$$I_m = \sum_{k=1}^{4} \sum_{(i,j)\in B_k} (I_t(i,j) - I_{t-1}(i,j))^2. \tag{2}$$

If $I_m$ is larger than a threshold $T_m$ and $I_{Blur} > T_{Blur}$, then the "motion blur" event will be alarmed, while if $I_m \leq T_m$ and $I_{Blur} > T_{Blur}$, the "de-focus blur" event will be issued. We reuse the square operation look-up table we built for estimating motion vectors to evaluate Eq. 2. Both $T_{Blur}$ and $T_m$ are determined empirically and are fixed in PEYE.

## 4 Applications

We have integrated PEYE with a bunch of interesting applications such as key-free mobile web browsing, pen-free sketchy, game playing, and automatic phone pick-up, as visualized in Fig. 5. We present more details as follows.

**Key-free Mobile Web Browsing.** Not until the recent development of the DeepFish system [16] has Web-browsing on mobile devices been an enjoyable experiences. We have integrated PEYE with the DeepFish browser to use motion gestural events to scroll up/down and zoom in/out the web pages. It provides a much more natural-to-use and non-invasive interaction scheme for the users to interact with the web browser. The first two figures in Fig. 5 show two views of using PEYE for web-browsing, the second is a zoom-in version of the first.


**Fig. 5.** Sample PEYE applications: a) key-free web browsing (left two figures), b) pen-free sketchy (the third), and c) games (last figure shows the game *1942*)

**Pen-free Sketchy.** PEYE provides a natural means to place sketches, where the users just need to move the mobile devices to place the sketches. We let the users to push the D-pad once to start or end one sketch. The third figure in Fig. 5 presents a screen shot on how we can draw a complex sketch. To erase the sketches, the users just need to use his hand to wipe off the camera view to trigger a "darkness" event. Possible extensions include recognizing sketches for inputting characters, especially for east Asian languages [5,4].

**Mobile Games.** When playing games on mobile devices, it is not that convenient to respond quickly using either small key-board or D-pad. We demonstrate that using PEYE, the users can respond very quickly since all he/she need to do is to move the mobile devices in a certain way. To show one example game we have deployed, the last figure in Fig. 5 presents a screen shot of an air-fighter game called *1942*. Certainly, we are also exploring means to let two or more users to play games together, each with his/her own mobile device.

**Automatic Phone Pick-up.** When the users put the mobile devices on the table, the cameras are usually facing down. If there is an incoming phone call and the users grab the mobile devices, PEYE will automatically detect a darkness transition event. Upon receipt of this event, the application can automatically answer the phone call instead of waiting for the users to push the pick-up button.

## 5 Evaluation

In this section, we evaluate the performance of the PEYE algorithms, and conduct usability studies for the mobile web browsing experiences.

### 5.1 Evaluation of Algorithms

In this section we evaluate the performances of the different block matching methods in terms of both speed and accuracy, with or without adaptive search

**Table 1.** The average number of matching positions for the different methods

| | Average number of matching positions (#/frame) | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Methods\Video# | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | Overall |
| FSBM | 225 | | | | | | | | | 225 |
| TSS | 25 | | | | | | | | | 25 |
| FSS | 20.6 | 21.6 | 24.6 | 24.7 | 24.6 | 19.6 | 24.5 | 20.6 | 21.7 | 22.9 |
| DS | 18.9 | 20.4 | 26.6 | 25.5 | 25.5 | 16.8 | 24.7 | 18.6 | 20.5 | 22.6 |
| HS | 14.2 | 15.3 | 15.2 | 17.6 | 18.3 | 13.1 | 17.6 | 14.4 | 15.2 | 16.0 |
| AMMS | 11.4 | 14.4 | 25.2 | 25.3 | 25.5 | 8.9 | 22.9 | 11.8 | 14.7 | 18.9 |
| TSSAC | 26.4 | 25.8 | 26.9 | 26.9 | 26.9 | 26.3 | 27.0 | 26.6 | 26.7 | 26.8 |
| FSSAC | 20.2 | 20.3 | 25.6 | 24.2 | 24.8 | 18.8 | 24.6 | 18.9 | 20.0 | 22.3 |
| DSAC | 17.2 | 17.2 | 26.3 | 23.5 | 24.9 | 14.4 | 24.0 | 14.6 | 16.6 | 20.5 |
| HSAC | 13.7 | 14.4 | 17.0 | 16.4 | 17.0 | 12.8 | 16.5 | 13.3 | 14.2 | 15.3 |
| AMMSAC | 9.3 | 10.0 | 22.6 | 18.6 | 20.8 | 6.8 | 18.8 | 7.2 | 9.4 | 14.6 |

**Table 2.** The average SSD of the matching results for the different methods

| | Average SSD /frame | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Methods\Video# | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | Overall |
| FSBM | 3.24 | 9.11 | 14.35 | 12.11 | 19.95 | 9.10 | 9.51 | 21.71 | 28.98 | 14.23 |
| TSS | 2.28 | 9.86 | 15.13 | 12.93 | 21.33 | 10.27 | 10.34 | 23.94 | 30.95 | 15.33 |
| FSS | 3.33 | 10.57 | 15.50 | 13.15 | 21.99 | 9.57 | 10.70 | 23.02 | 33.40 | 15.69 |
| DS | 3.32 | 9.95 | 14.77 | 12.58 | 20.84 | 9.28 | 10.44 | 23.18 | 33.08 | 15.27 |
| HS | 3.38 | 15.34 | 14.74 | 14.02 | 22.78 | 10.08 | 11.25 | 24.82 | 36.24 | 16.50 |
| AMMS | 3.39 | 14.45 | 14.86 | 12.51 | 20.74 | 9.47 | 10.57 | 24.99 | 35.07 | 15.79 |
| TSSAC | 3.23 | 9.05 | 11.59 | 8.53 | 14.38 | 9.37 | 8.44 | 21.71 | 28.13 | 13.13 |
| FSSAC | 3.24 | 9.15 | 12.15 | 8.64 | 15.60 | 9.34 | 8.77 | 21.98 | 29.21 | 13.12 |
| DSAC | 3.24 | 9.12 | 11.56 | 8.43 | 14.29 | 9.27 | 8.78 | 22.06 | 29.02 | 12.86 |
| HSAC | 3.32 | 10.50 | 13.34 | 11.75 | 19.43 | 9.86 | 10.33 | 23.66 | 33.78 | 15.10 |
| AMMSAC | 3.27 | 9.32 | 11.88 | 8.65 | 14.58 | 9.36 | 9.05 | 22.20 | 29.68 | 13.11 |

centers. The speed is measured by the number of matching locations. The less the number is, the faster the algorithm is. The accuracy is quantified by the SSD. The smaller it is, the better the match is. We use the results of FSBM as a baseline. Note all the matching are confined in a block of size $15 \times 15$.

Since we can not afford to do all the evaluations on the mobile device. We recorded 9 video sequences, which were taken against different background and/or with different motions patterns using a Samsung Pocket-PC. Then we evaluated all the algorithms on a PC. We summarize the evaluation results in Table 1 and Table 2. As we can easily observe, except for TSS where only accuracy is improved, the adaptive center scheme does improve all the other algorithms in terms of both speed and accuracy. For example, AMMSAC on average only evaluates 14.6 positions (v.s. 18.9 for AMMS). DSAC achieves the best average matching error as low as 12.86 (v.s. 15.27 for DS). Another observation is that

w.r.t. either speed or accuracy, none of the methods is constantly the best. This is the main motivation for us to design the third selection scheme in Sec. 3.1.

### 5.2 Usability Study

We perform usability study on the application of web-browsing using PEYE. With just some brief how-to-use descriptions, we let the users to browse a web using PEYE for interaction. The users then rank their experience in 5 grades, with 5 being "very satisfactory" and 1 being "very unsatisfactory". We also ask the users to compare their PEYE browsing experiences with that of using D-Pad. It is also ranked in 5 grades, with 5 being "much better" and 1 being "much worse". The users are also categorize to be "novice", "intermediate" or "advanced" based on how long they have been using mobile devices. 15 users participated in our usability study (6 novice, 8 intermediate, and 1 advanced).

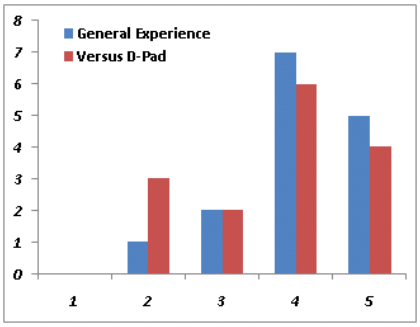


**Fig. 6.** The blue bar presents the distribution of the experiences of the users using PEYE, and the red bar displays the distribution of the users' experiences compared with that of using D-Pad

The histograms of the users' scores of the two studies are summarized in Fig. 6. As we can observe, 80% users rank their experiences using PEYE to be either *satisfactory* or *very satisfactory* (46.7%+33.3%). For comparison with the users' experience using D-Pad, 66.7% of the users think that it is *much better* or *better* to use PEYE than using D-Pad. On the other hand, three users (i.e., 20%) think that it is more convenient to use D-Pad. We found that two of the three users are in the intermediate level, and the other is in the advanced level. Moreover, the only user who ranked his/her experience with PEYE as unsatisfactory is an intermediate level user. This implies that they may have been biased toward using D-Pad since they have got used to it. We expect the users to prefer PEYE more, once they became more familiar with it.

## 6 Conclusion and Future Work

In this paper, we have presented PEYE: a novel computer vision based perceptual interface for mobile devices. Although the current implementation is under Windows CE, the algorithms as well as the architectural design can certainly be applied to other embedded OS. Our algorithm evaluation and usability study demonstrate the efficacy of the proposed approach. Future work includes identifying more visual events and developing novel applications based on PEYE.

## References


1. Zhang, Z., Wu, Y., Shan, Y., Shafer, S.: Visual panel: Virtual mouse, keyboard and 3D controller with an ordinary piece of paper. In: Proc. ACM Perceptive User Interface Workshop, ACM Press, New York (2001)
2. Wilson, A., Oliver, N.: Multimodal sensing for explicit and implicit interaction. In: Proc. of 11th International Conference on Human Computer Interaction, Las Vegas, NV (July 2005)
3. Wilson, A.: Robust vision-based detection of pinching for one and two-handed gesture input. In: Proc. of Intenational Symposium on User Interface Software and Technology (UIST), Montreux, Switzerland (October 2006)
4. Wang, J., Canny, J.: Tinymotion: Camera phone based interaction methods. In: Proc. alt.chi of ACM CHI, Montreal, Canada (April 2006)
5. Wang, J., Zhai, S., Canny, J.: Camera phone based motion sensing: Interaction techniques, applications and performance study. In: Proc. of Intenational Symposium on User Interface Software and Technology (UIST), Montreux, Switzerland (October 2006)
6. Furht, B., Greenberg, J., Westwater, R.: Motion Estimation Algorithms for Video Compression. In: The International Series in Engineering and Computer Science, Springer, Heidelberg (2004)
7. Rohs, M., Zweifel, P.: A conceptual framework for camera phone-based interaction techniques. In: Gellersen, H.-W., Want, R., Schmidt, A. (eds.) PERVASIVE 2005. LNCS, vol. 3468, Springer, Heidelberg (2005)
8. Rekimoto, J., Ayatsuka, Y.: Cybercode: Designing augmented reality environments with visual tags. In: Proc. of DARE, pp. 1–10 (2001)
9. Jari Hannuksela, P.S., Heikkila, J.: A vision-based approach for controlling user interfaces of mobile devices. In: CVPR 2005. Proc. of IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, Washington, DC, USA, p. 71. IEEE, Los Alamitos (2005)
10. Ballagas, R., Rohs, M., Sheridan, J.G.: Sweep and point and shoot: phonecam-based interactions for large public displays. In: CHI 2005 extended abstracts on Human factors in computing systems. Conference on Human Factors in Computing Systems archive, Portland, OR, USA, pp. 1200–1203 (2005)
11. Koga, T., Iinuma, K., Hiranoa, A., Iijima, Y., Ishiguro, T.: Motion compensated interframe coding for video conferencing. In: Proc. of IEEE NTC 1981, pp. G.5.3.1–G.5.3.4 (1981)
12. Po, L.M., Ma, W.C.: A novel four-step search algorithm for fast block motion estimation. IEEE Trans. on Circuits and Systems for Video Technology 6(3), 313–317 (1996)
13. Zhu, S., Ma, K.K.: A new diamond search algorithm for fast block-matching motion estimation. IEEE Trans. on Image Processing 9(2), 287–290 (2000)
14. Zhu, C., Lin, X., Chau, L.P.: Hexagon-based search pattern for fast block motion estimation. IEEE Trans. on Circuits and Systems for Video Technology 12(2), 355–394 (2002)
15. Liu, Y., Oraintara, S.: Adaptive multiple-mode search algorithm for fast block-matching motion estimation. In: IEEE Trans. on Circuits and Systems for Video Technology (September 2003)
16. `http://labs.live.com/deepfish/`