Intuitive Application-specific User Interfaces for Mobile Devices

Stefan Winkler
winkler@nus.edu.sg

Karthik Rangaswamy
elerk@nus.edu.sg

Jefry Tedjokusumo
jefry@nus.edu.sg

ZhiYing Zhou
elezzy@nus.edu.sg

Interactive Multimedia Lab
Department of Electrical and Computer Engineering
National University of Singapore
Singapore 117576

ABSTRACT

The user interfaces on current-day mobile devices are limited owing to their small form factor. In this paper, we propose intuitive user interfaces for two applications on mobile phones with built-in cameras, namely, a car-racing game and a map navigation application. These user interfaces are based on the visual detection of the device's self-motion. We discuss three different methodologies to implement same and conduct user studies to analyze people's acceptance of the interfaces in comparison to controlling the applications with keys on the phone. Results show that our proposed interfaces are well appreciated for their intuitiveness, even though there is a learning curve.

1. INTRODUCTION

The small form factor of mobile devices call for new and more natural approaches in user interface (UI) design. Using the small, non-ergonomic keys for control in applications other than calling is tedious and not always intuitive.

In order to overcome these problems, we have developed novel, motion-based controls as a user interface for mobile devices. The controls are based on the detection of the 3D self-motion of the device using the integrated camera. Possible movements include translation and rotation with a total of 6 degrees of freedom. These movements can then be used to control applications on the device. 3D motion tracking, which forms the backbone of this research, involves motion detection, computing the pose of the phone in 3D space from the motion registered and tracking an object of reference if necessary [3]. The process has to run in real-time under the constraints of the limited memory and computing resources available on a hand-held device. Fortunately, the processing power of mobile devices is continuously being improved upon, with the latest trend being addition of exclusive hardware for processing of graphical information.

We adopt three methodologies to implement our motion tracking system, namely, marker-based, face tracking, and TinyMotion.

Markers are commonly used as references for Augmented Reality-based applications on the PC and the Mobile platforms. Although one might consider markers as an additional burden, we think that markers can also be carried along with the phone wherever the user goes, for instance in the wallet or in a mini-compartment inside the phone's carrying case. The interfaces we built depend only on a single marker for processing, thus minimizing the burden.

Face tracking requires a face as a reference. There are various methods available to implement face tracking, but very few use the concept to control applications. FotoNation [2] claimed to be the first company to implement face detection for mobile devices. Their product is mainly used in digital cameras and mobile phones for adjusting the white balance of the image, adjusting the camera's exposure, and determining the image's orientation. Their algorithm is capable of detecting multiple faces in an image. Yet it is limited to detecting faces, not tracking them. It cannot estimate face orientation or distance accurately. Hansen et al. [4] implemented a face tracking algorithm on Nokia 6680 and 7610 with a 220 MHz processor that runs at 75-90 ms per frame. Our implementation runs faster at about 20-35 ms per frame on a PocketPC. With better hardware, more sophisticated and processing-intensive applications can be developed on top of the face tracking algorithm.

TinyMotion developed by Wang et al. [9] uses both frame differencing and correlation of image blocks for motion estimation, similar to the methods used by video encoders like MPEG2 and MPEG4. It does not require any points of reference.

We developed two applications as test-benches to evaluate our motion based interfaces. Games have a major share in

the software content of today's mobile phones. Here, the first application we developed was a car-racing game. In addition to games, many of the current applications are being ported to the mobile platform as well, like Google Maps for instance. Our second application to test the proposed interfaces is a similar one, where the phone becomes a looking glass for a virtual map.

The car-racing game's primary controls include steering and acceleration, while the map navigation application requires controls to scroll the map and zoom through different resolutions of the map. We pre-define certain intuitive phone movements that translate into these basic controls on the user interface to drive the applications.

The paper is organized as follows: Section 2 explains in detail the three methodologies mentioned above to track the self-motion of the mobile device. Section 3 describes the applications developed and the user interfaces designed to control them. Section 4 discusses the user study conducted to analyze people's reactions to such interfaces. The paper concludes with Section 5.

2. METHODOLOGIES

We tested our applications on a Hewlett-Packard rw6828 PocketPC equipped with a 416 MHz Intel PXA272 processor and 64MB RAM, running Windows Mobile 5.0 OS. The PDA has a built-in 2-megapixel camera that is used for motion-tracking at a video resolution of 176x144 pixels. We have experimented with three different methods to perform visual motion tracking with video feedback from the phone's camera. For this purpose, we developed a DirectShow FrameGrabber filter to capture incoming frames from the camera. The following sub-sections explain the implementation details of each methodology. An overall summary of all three implementation methods is described in Table 1.

2.1 Marker Tracking

In this method, a marker is used as a reference to estimate the pose of the phone in 3D space. The marker tracking is implemented using ARToolkitPlus [8], a modified version of ARToolkit [5] that can be compiled for the mobile platform. ARToolkitPlus includes many new features over the original ARToolkit, namely new pixel formats for images returned by the mobile phone's camera (like RGB565), fixed point arithmetic to replace some of the computationally intensive floating-point operations, automatic thresholding to compensate for varying lighting, and ID-encoded markers instead of template-based recognition that require no prior training and at the same time do not affect the speed of operation. Figure 1 shows an ID-encoded ARToolkitPlus marker.

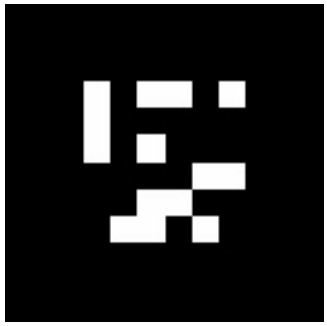
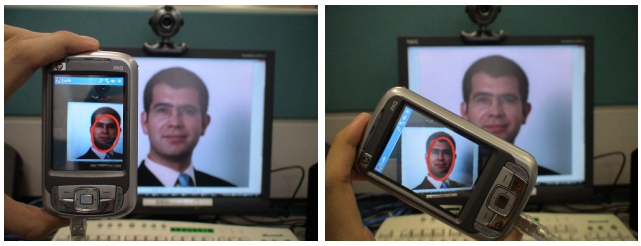
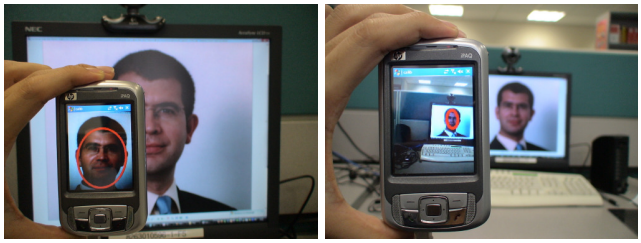


Figure 1: ARToolkitPlus ID-encoded marker.



(a) normal position (b) rotated position



(c) zoom-in position (d) zoom-out position

Figure 2: Face tracking in action. A photo is used for experimental purposes.

ARToolkitPlus accepts video frames and returns the transformation matrix describing the pose in 3D (6 degrees of freedom) if a marker is detected in the frame. The average execution time of the algorithm is approximately 30 ms per frame. The obtained translation and orientation components are then mapped to different controls on the user interface to drive the application. The mathematical details of this process are explained in [5].

There is an intrinsic limitation to translating along the depth direction with an object of reference: As the phone is moved closer to the object, the usable movement range is reduced as the area of the object of reference in the camera view increases.

2.2 Face Tracking

Human faces are a common feature in our daily life, and the user's face is always available for tracking. We use CamShift [1], which was developed in 1998 for a Pentium II 300 MHz processor. Today's mobile phone processors have similar speeds, so any algorithm developed on a PC from that era can easily be ported to the mobile platform without major modifications or optimizations.

CamShift needs an initial face position before it can continually do the tracking. To automate this process, we used an OpenCV implementation of a face detection algorithm based on Haar object detection [6, 7].

CamShift uses probabilistic color histogram to do the tracking; hence it tracks any distinguished uniformly colored object. But it can also be easily distracted by other objects

Table 1: Summary of Implementation Methods

Motion-tracking Method	Implementation	Object of Reference	Number of DOF Detected	Processing time per video frame
Marker-based	ARToolkitPlus	marker	6	30 ms
Face Tracking	CamShift, OpenCV	face	4	20-35 ms
TinyMotion	OpenCV	none	2	$\leq$5 ms

that have similar color (e.g. if there is more than one face in the image). Details on how the algorithm reacts to distraction, occlusion, lighting conditions etc. can be found in [1].

CamShift face tracking supports 4 degrees of freedom, namely 3 translations and rotation about the depth direction, as shown in Figure 2. The time taken for face tracking itself varies between 20-35 ms. The speed of the overall application also depends on the graphical complexity of the application.

2.3 TinyMotion

TinyMotion [9] is an alternative if no reference objects are available in the scene. The algorithm consists of the following steps:

- Color Space Conversion: A bit shifting method is used to convert every image returned by the FrameGrabber filter from 24-bit RGB format to 8-bit gray scale.
- Grid Sampling: A multi-resolution sampling technique is applied on the gray scale images to reduce the computational complexity and memory bandwidth for the subsequent calculations. The grid-sampled frames are denoted as motion-blocks.
- Motion Estimation: The motion estimation technique involves a full-search block matching algorithm on neighboring frames that were grid-sampled in the previous step.
- Post Processing: The relative movements detected in each motion estimation step are accumulated from the beginning of execution, to provide an absolute measurement from a starting position.

We utilized the OpenCV library to implement the TinyMotion algorithm. All of the above steps are realized efficiently by integer-only operations. A detailed mathematical treatment to the above algorithm is given in [9].

Each grid-sampled frame is split into 2 or 4 sectors, and the phone's rotation can be interpreted from the sectors' relative motions. A sample frame which was split into four sectors with their individual motion vectors is shown in Figure 3. These motion vectors were calculated for the grid-sampled version of the video frame in the display.

The algorithm's execution time depends on the size of the search window in the block-matching algorithm and the size of the grid-sampled frame. The current implementation uses a 7x7 window on a 9x11 grid-sampled frame and consumes less than 5 ms to process each video frame.

Figure 3: Four sectors of a video frame with their individual motion vectors.

TinyMotion is still under study as motion detection is restricted to only two degrees of freedom, namely translation along the xy plane. Rotation about the depth direction can be interpreted, but the extent of rotation is difficult to measure accurately, because the in-built camera is not the center of rotation (it is located at the corner of the phone). This is the case with most commercially available phones today. For the same reason, TinyMotion cannot robustly detect translation of the phone along the depth direction either.

Feature tracking and optical flow methods are good alternatives, although they require good resolution for robust motion vector analysis. Unfortunately, even if the phone could support higher resolutions, the processing power of the phone poses serious constraints on the algorithmic complexity of the problem. Hence we are currently working on methods that can retain accuracy of motion-tracking and detect more degrees of freedom while not drastically reducing the speed of operation.

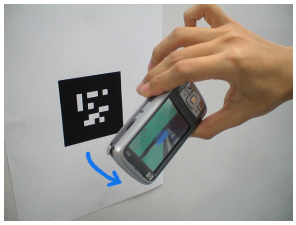
3. APPLICATION AND UI DESIGN

The applications were written in C++ making use of the Direct3D Mobile libraries for graphics. The following subsections describe the user interfaces we designed.

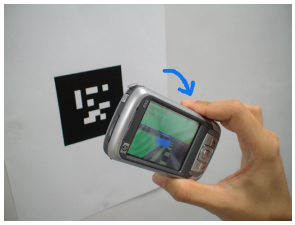
3.1 Virtual Steering Wheel and Accelerator

The car-racing game is a third person view simulation of a car on a winding road with a digital speed indicator. The main controls of the game are the steering wheel and the accelerator. In our interface the user can handle the phone directly as both a steering wheel and an accelerator in order to maneuver the car. To steer the car, the user rotates the phone to the left or right about the depth direction. For acceleration and deceleration, respectively, the user moves the phone towards or away from the reference object along

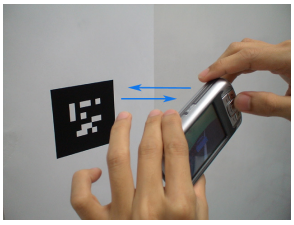
the depth direction, as shown in Figure 4.



(a) Turning left



(b) Turning right



(c) Accelerating and decelerating

Figure 4: The car is steered left or right by tilting the phone about the depth direction. The car is accelerated or decelerated by moving the phone towards or away from the reference object (a marker in this example).

(a) True-horizon mode

(b) Relative-horizon mode

Figure 5: In true-horizon mode, the in-game horizon is parallel to the ground (a). In relative-horizon mode, the in-game horizon is parallel to the phone's base (b).

We use the distance between the phone and the point of reference along the depth direction as an indicator of the car's speed, and the rotation about the depth direction to determine the direction and extent of turning.

In addition to the basic controls, we introduced two different modes of display for the game, namely a true-horizon mode and a relative-horizon mode, which are illustrated in Figure 5. In true-horizon mode, the in-game horizon remains parallel to the ground irrespective of the tilt of the phone. This assumes the orientation of the object of reference with respect to the ground is fixed and known. In relative-horizon mode, the in-game horizon is parallel to the base of the phone's display. Finally, we implemented an engine sound whose pitch is proportional to the car's speed as an additional speed indicator.

3.2 Looking Glass over a Virtual Map

The map navigation application displays maps as textures on a vertex grid. The maps are downloaded from Google Servers online whenever the user scrolls to unexplored regions or zooms to a different resolution of the map. The display includes a video feedback window on the top-left corner of the application and a zoom level indicator. The video feedback window displays the object of reference in the instantaneous view. This window was provided such that the user can keep track of the object of reference within the field-of-view of the phone's camera. Figure 6 shows the map navigation application running on our PocketPC.

The vectors obtained from the motion-tracking algorithm are used to translate a virtual camera over the map textures by a pre-defined offset for every rendered frame, thus giving the notion of continuous scrolling. Zooming is implemented by translating the virtual camera towards or away from the vertex grid to view different resolutions of the map.

Taking the constraints of the mobile device keys into consideration, we designed a strategy for our application to enable scrolling the maps at any arbitrary direction along the plane of the map so as to give the user more freedom in interacting with the application.

We designed a user interface wherein both the scrolling and zooming operations could be achieved with a single point of reference. For scrolling, the user has to move the phone over the reference point such that the map scrolls in a direction defined by a vector drawn from the optical axis to the reference point on the plane containing the reference point. In the case of marker or face tracking, the reference point would be the center of the marker or the facial region being tracked, respectively. The zooming operation is achieved by moving the phone towards or away from the reference point.

Furthermore, we added a feature wherein the user can control the scrolling speed of the map based on the relative distance of the reference point with respect to the optical axis. At the optical axis the scroll speed is zero, but it increases in proportion to the distance between the reference point and the axis on the reference plane.

In addition to this, the phone can also be rotated by any arbitrary angle to view a wider region of the map as shown in Figure 6, similar to how real maps are viewed. The map can be navigated consistently even if the phone is held at arbitrary angles.

In summary, the phone becomes a looking glass which people can use to view and navigate over a virtual map.

As explained in Section 2.1, the usable scroll range is reduced at higher zoom levels when the phone is very close to the object of reference. Hence the scroll speed is made constant at these zoom levels, and only the direction of the reference point relative to the optical axis is used for scrolling. Additionally, a small thresholded region around the optical axis is defined where the scroll-speed drops to zero. The user can move the reference point to this region in order to pause scrolling.

(a) (b)

(c)

Figure 6: The phone can be rotated to view wider regions of the map.

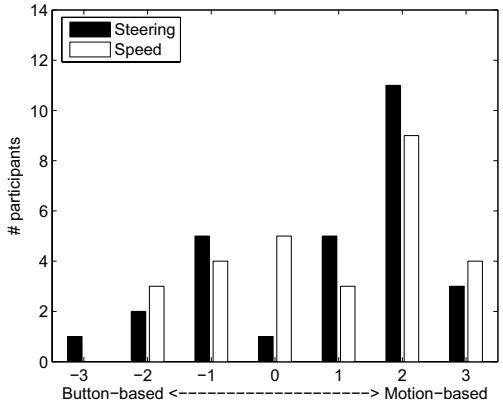
4. USER STUDIES

We conducted user surveys in order to compare the proposed motion-based interface for the two applications with the traditional key-based interface. The factors to compare were accuracy of controls, comfort in usage, sensitivity and responsiveness of the system. For this purpose, we developed key-based interfaces for the two applications. For the car-racing game, the four directional buttons are used to maneuver the car. The left and right arrow buttons steer the car in either direction, and the up and down arrow buttons accelerate and decelerate the car, respectively. The key-based interface for map navigation involves the four directional keys to scroll the maps along the four cardinal directions and two other keys allocated for zooming in and out of map resolutions incrementally. The order in which the interfaces were tested were key-based first followed by the motion-based interface.

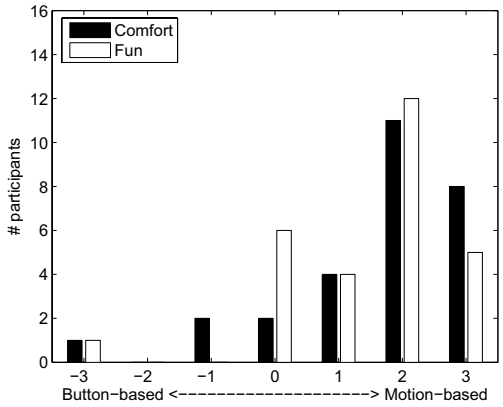
4.1 Car-Racing Game

15 male and 13 female participants between 15 and 16 years of age took part in the user study for the car-racing game. All the participants had prior experience with mobile phone games, and over 90% of them had played a car-racing game on a mobile phone before. Each question in the survey required the user to give a rating between -3 and 3, comparing each aspect of the game listed above for both interfaces; the ends of the scale indicate a clear preference for one interface or the other, and zero represents the neutral point.

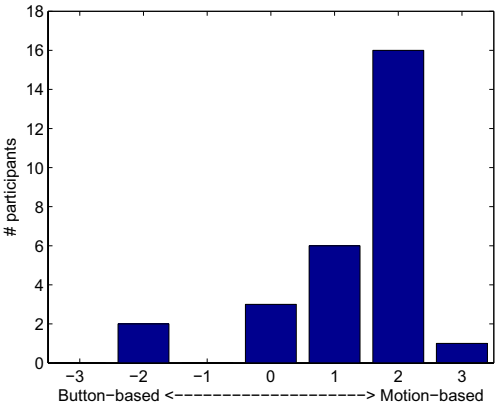
The user ratings for accuracy in steering and speed control with both interfaces are shown in Figure 7(a). The average ratings for the individual distributions are 0.85 for steering and 0.82 for speed control, with variances of 2.77 and 2.58, respectively. We expected a higher rating for steering, but



(a) Accuracy in steering and acceleration



(b) Comfort and fun during gameplay



(c) Overall rating

Figure 7: Comparison of user ratings for car-racing game with button-based and motion-based interfaces.

the location of the camera at the corner on the back of the phone plays a major role. When the phone is tilted to steer the car, the reference object has a high probability of disappearing from the camera's field-of-view. Having a camera at the center of the phone's back would certainly be bet-

Table 2: Average User Ratings for Virtual Map Navigation

Avg. Ratings (0 to 10)	Key-based Interface		Motion-based Interface	
	Scrolling	Zooming	Scrolling	Zooming
Intuitiveness of the Interface	7.3	7.3	7.3	7.1
Comfort in Usage	6.8	6.7	7.3	6.0
Responsiveness of the System	6.2	6.3	6.6	5.9

ter for our interface. A simple solution would be to have a video feedback window, similar to that provided with the map navigation application. Despite this minor drawback which required getting used to for the participants, most of them agreed that our new design was a better alternative to steering with buttons on the phone.

The game played by the participants was only a simple simulation of a car on a road without much environmental detail, as we only wanted to test the controls and not develop a full-fledged game. For acceleration and deceleration, the users opined that although the interaction method was intuitive, additional graphical features like textures and objects by the side of the road would give them a better notion of current speed. On the other hand, if the game is to be made more graphics-intensive, the frame-rate is sure to decrease further, unless better graphics hardware becomes available for mobile devices.

In an informal feedback session, almost all the participants appreciated the innovative approach to game play and wanted such a technology to drive games and other applications on the mobile platform. They also wanted the car-game to be developed into a complete product and released commercially. This is further confirmed by their ratings for the comfort in usage, the fun generated during game-play and overall enjoyment, shown in Figure 7(b) and Figure 7(c).

4.2 Virtual Map Navigation

12 male and 4 female participants between 16 and 28 years of age took part in the user survey for virtual map navigation. Almost all of them had experience with a similar application on the PC platform, while only one of them had used it on a mobile PDA. Owing to this lack of experience, we let the participants experiment with the key-based interface for map navigation on the mobile phone for some time in order to give them a feel of the application on a mobile platform. Once they were comfortable, they were asked to navigate maps with the motion-based interface. The participants were then asked to rate scrolling and zooming of the map with each interface independently on a scale from 0 to 10 for intuitiveness, comfort and responsiveness. The average ratings of all participants are shown in Table 2.

Though the average ratings in Table 2 did not allow us to draw any firm conclusions, the individual ratings were widespread. Hence, we felt that rather than choosing an interface based on the above ratings, it would be wise to identify what led to the users' decisions in making their ratings.

It can be inferred from the ratings for comfort in usage and responsiveness that the motion-based interface was much more comfortable than the key-based interface for scrolling.

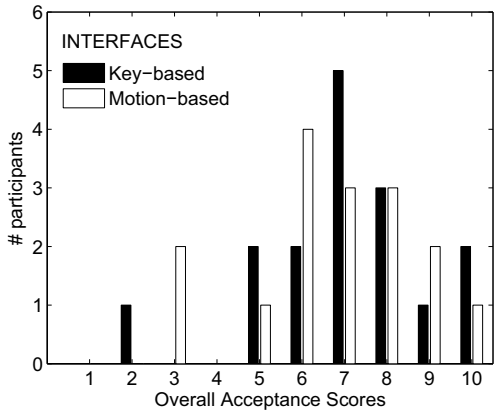


Figure 8: Distribution of user's overall acceptance scores for virtual map navigation with key-based and motion-based interfaces.

Yet, the motion-based interfaces suffered in ratings for the zooming operation. The main reason appears to be experience with such interfaces, especially in applications where user interaction is constrained by online downloading of data and network speed. A similar influence can also be seen in the equal ratings for intuitiveness of the two interfaces, even though the motion-based interface was felt to be more intuitive by many participants. Yet there were others who felt that human hands are not always steady, and that it was difficult initially when they had to scroll maps by moving the phone and at the same time hold it at a steady distance from the reference object in order to maintain the zoom level.

People have used keys on the phone all along; when switching to motion-based interfaces, the more time they spent with the new interfaces, the more they got adapted to using them. Hence, we believe that with adaptation and experience, people would find virtual map navigation and other similar applications easier to handle with a motion-based interaction than with keys on the phone.

5. CONCLUSIONS

In this paper, we proposed a intuitive motion-based interfaces for two common applications on the mobile phone, namely, a car-racing game and virtual map navigation. Intuitive phone movements are mapped to pre-defined controls on the interface that are used to control the applications. We used three techniques to implement our motion tracking system, namely marker-based, face tracking, and TinyMotion. We also conducted user studies to compare a key-based interface with the motion-based interface, evaluating their

accuracy, sensitivity and responsiveness. The results show that the motion-based interfaces are well appreciated for their intuitiveness and perform equally well when compared with a key-based interface even for a first trial.

6. ADDITIONAL AUTHORS

7. REFERENCES

[1] Bradski, G. R. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, Q2 (1998), 15.

[2] FotoNation. http://www.fotonation.com/index.php?module=company.news&id=43.

[3] Foxlin, E. Motion tracking requirements and technologies. In *Handbook of Virtual Environment Technology*, K. M. Stanney, Ed. Lawrence Erlbaum Associates, Hillsdale, N.J., USA, 2002, ch. 8, pp. 163–210.

[4] Hansen, T. R., Eriksson, E., and Lykke-Olesen, A. Use your head: exploring face tracking for mobile interaction. In *CHI Extended Abstracts on Human Factors in Computing Systems* (Montreal, Canada, 2006), pp. 845–850.

[5] Kato, H., and Billinghurst, M. Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In *Proc. IEEE and ACM International Workshop on Augmented Reality (IWAR)* (San Francisco, CA, 1999), pp. 85–94.

[6] Lienhart, R., Kuranov, A., and Pisarevsky, V. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Proc. 25th DAGM Symposium* (Magdeburg, Germany, September 2003), pp. 297–304.

[7] Viola, P., and Jones, M. Rapid object detection using a boosted cascade of simple features. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Kauai, HI, USA, December 2001), vol. 1, pp. 511–518.

[8] Wagner, D., and Schmalstieg, D. ARToolkit on the PocketPC platform. In *Proc. IEEE Intl. Augmented Reality Toolkit Workshop (ART032)* (Tokyo, Japan, October 2003), pp. 14–15.

[9] Wang, J., Zhai, S., and Canny, J. Camera phone based motion sensing: Interaction techniques, applications and performance study. In *ACM Symposium on User Interface Software and Technology (UIST)* (Montreux, Switzerland, October 2006).