Camera-based Scrolling Interface for Hand-held Devices

Giorgio Davanzo, Eric Medvet, Alberto Bartoli
DEEI - University of Trieste
giorgio.davanzo@deei.units.it, emedvet@units.it, bartoli.alberto@units.it

Abstract

Hand-held devices have become widespread and provided with significant computing capabilities, which results in an increasing pressure for using these devices to perform tasks formerly limited to notebooks, like web browsing. Due to their small screens, however, hand-held devices cannot visualize directly documents that were not designed explicitly for small-screen rendering. Such documents may be rendered either at a scale too small to be useful, or at a scale that requires intensive scrolling operations by the user. Unfortunately, scrolling a small window across a large document with a hand-held device is quite cumbersome. In this paper we propose a scrolling system much simpler and more natural to use, based on the embedded camera—a component available in every modern hand-held device. We detect device motion by analyzing the video stream generated by the camera and then we transform the motion in a scrolling of the content rendered on the screen. This way, the user experiences the device screen like a small movable window on a larger virtual view, without requiring any dedicated motion-detection hardware. We performed an experimental evaluation aimed at assessing the effectiveness of the proposed system in the considered scenario, characterized by low image quality, unpredictable framed scene and so on. We performed an objective benchmark quantifying the accuracy of the detected trajectory and a subjective benchmark examining users' confidence with the proposed system. For the latter evaluation, we involved a panel of 20 subjects that executed a trajectory with our system and, as a comparison, with keyboard, mouse and touchpad. The results demonstrate that our approach is indeed practical.

Keywords—**Hand-held Devices, Camera Phones, Scrolling, Computer-Human Interaction, Motion Detection**

1 Introduction

Hand-held devices, and in particular smart phones, have become more and more widespread thanks to their powerful capabilities and affordable prices. They are increasingly used to perform tasks that were usually constrained to desktop computers, like web browsing or exploring hi-res photo collections. Due to the small screen size of such devices, however, users experience in these tasks is often not fully satisfactory.

Concerning web browsing in particular, both literature [11, 2] and common sense suggest that, when a content version specifically designed to be accessed by small-screen devices is not available, the content should be rendered as originally designed, i.e., as rendered on a desktop computer. Since the original content cannot fit the device screen at a magnification level comfortable to the user, intensive scrolling operations become necessary. A scrolling facility must be provided in order to enable the user to (virtually) move the screen around the full content. Such facility should allow the user to move comfortably over the full content in order to rapidly focus on those parts of the content he wants to see. Scrolling a small window across a large document with a hand-held device, on the other hand, is usually quite cumbersome.

In this paper we propose a scrolling system much simpler and more natural to use, based on the *embedded camera*—a component available in every modern hand-held device. We detect device motion by analyzing the video stream generated by the camera and then we transform the motion in a scrolling of the content rendered on the screen. We believe that our proposal could improve the user experience significantly, including in circumstances when the device can be used only with one hand— e.g., while eating meals or carrying a bag. More broadly, the ability to detect motion without requiring any additional dedicated hardware could enable novel interaction schemes for hand-held devices.

We consider a case in which a user is viewing some content on its hand-held device screen—e.g., a web page, an eBook page, a picture or technical diagram—and in particular a (perhaps magnified) portion of the original page. By enabling the user to scroll by simply *moving* the device, he will perceive the device screen like a small movable window on a larger virtual view. This goal can be achieved in two steps. First, the device movement has to be detected and quantified; second, it has to be transformed in a corresponding movement—both in direction and quantity terms—of the content displayed on the device screen. For the first step, we use the device embedded camera and

1550-6037/08 $25.00 © 2008 IEEE
DOI 10.1109/IV.2008.92

extract a *movement vector* from the current video streaming in real time. In the second step, we simply transform that movement vector to perform a scrolling of the content currently displayed on the screen device. In this work we are concerned with step one, which is the most relevant and technically challenging part of the problem.

We remark that while the basic idea underlying our proposal is simple, it is not clear whether it can be indeed implemented in practice with sufficient accuracy and performance. We need to cope with the image quality produced by typical embedded camera, which involves noise, low sharpness and so on. We need to cope with unpredictability of the scene that is being viewed by the camera: it can be composed mainly by far objects which move slowly, or by nearby objects moving fast, or by a combination of the two which, moreover, may also vary over time.

2 Related work

The problem of visualizing information—and, in particular, web pages—on the small screen of hand-held devices is gaining attention in literature. Due to the increasing computing and networking capacity of such devices, users expect to be able to browse the web on such devices as comfortably as they do on their desktop systems. Web browsing on small screen devices has been analyzed from both the interaction point of view [4, 2], and the visualization point of view [11]. The authors of the latter work identify two key points of web browsing on mobile devices (more precisely, mobile phones) that constitute an important motivation for our work: (i) content should be readable but, at the same time, rendered as originally designed; and (ii) such devices usually lack a pointing tool. Despite recent advances resolved in the presence of a pointing tool even on smaller devices (e.g., “touch screen”-based interfaces with or without using stylus), we think that our proposal may constitute a practical and powerful alternative.

Concerning specifically human-computer interaction, the use of the device movement itself as a way to interact with applications has been proposed by several works. The authors of [10] propose a “tilt to scroll” approach: differently from our proposal, they detect the device movement using an accelerometer. A phone prototype equipped with a 2D acceleration sensor is also used by [7], together with a vibration motor, to propose an novel interaction scheme with the device which is then used in a sample bouncing ball game. These works are based on components that usually are not present in hand-held devices.

An approach closer to ours is taken in [6]. The authors of this work investigate about the use of small devices as Virtual Reality (VR) platforms and find that, thanks to the physical interaction with the device, the user experience may still be satisfactory in spite of the small size of the screen being used. Besides, they propose the use of the camera as a motion tracking facility, together with a 3D accelerometer. In particular, they use an implementation of the Lukas-Kanade method—as we do, see next section—as an optical flow estimator to track forward/backward motion and rotation around Y axis. The authors do not provide an evaluation of the motion detection algorithm alone, since they evaluate its effectiveness, as perceived by the user, as a part of a more complex framework that involves other input means (buttons and accelerometers). As pointed out already, our proposal is completely based on the embedded camera and does not require any further hardware component.

3 Our approach

We want to convert the device current movement in a corresponding scrolling of the content displayed on the screen. We detect that movement by looking at the video stream generated by the embedded camera. We consider the camera as a discrete-time source of image frames and we extract a *movement vector* from each image frame, which is obtained by comparing the current frame against the previous one. The movement vector defines a movement in a 2-D space (because we intend to use it for driving a scrolling interface) although the device usually moves in a 3-D space.

Broadly speaking, *optical flow algorithms* [5, 9, 3] allow estimating complex deformations between two consecutive image frames. Nevertheless, although the actual movement might be a more complex transformation, we chose to elaborate the optical flow outcome in order to obtain always a simple *translation*, which is what we need in order to scroll the screen content.

We used the Lucas-Kanade [9] method for optical flow estimation: such approach has been used in different areas, e.g., scene recognition in sport video sequences [1] or head tracking for hands-free mouse-pointer manipulation [8]. In particular, we used the Kanade-Lucas-Tomasi (KLT) implementation of the method [12]. The KLT algorithm selects a number n of salient points (*good features* or briefly features) from an image frame and then tracks these features one-by-one on the following frames. The number n of features constitutes a parameter for the KLT algorithm. Good features are usually high contrast points in the image. The lack of a sufficient number of good features in a frame—e.g., when a uniform opaque surface is framed—results in the impossibility of detecting motion. This is an intrinsic limitation of the proposed approach and is similar to what happens with optic mice when they are used on unsuitable surfaces. As an aside, we are interested to see whether this limitation may constitute a practical impediment for the proposed approach or not.

More in detail, the algorithm works as follows. For each frame of the video stream:

1. Generate a set of $n' \leq n$ movement vectors $\vec{m}^i = (m_x^i, m_y^i)$. Each vector quantifies the movement of a feature between the previous frame and the current one. It will be $n' = n$ when all features of the previous frame have been found in the current frame and $n' < n$ otherwise. The case $n' < n$ is when one or more features have been lost between two consecutive frames, for example because the device movement caused some object to go out of the camera view. If $n' < n$ the algorithm performs a new selection of features.

2. Compute a single movement vector $\vec{m}$ averaging all the $\vec{m}^i$ obtained for that frame, hence obtaining $\vec{m} = (m_x, m_y) = (\frac{1}{n'}\sum_i m_x^i, \frac{1}{n'}\sum_i m_y^i)$. Note that if the device movement is purely translational *and* its camera is viewing a flat object which is parallel to the movement plane, then all the $\vec{m}^i$ will be identical.

3. Compute a scroll vector $\vec{f} = (f_x, f_y)$ that will be actually used to scroll the content on the device screen; $\vec{f}$ is computed from $\vec{m}$ based on a predefined threshold t and scaling factor λ as follows: $f_x = \lambda m_x$ if $m_x > \delta$ and $f_x = 0$ otherwise; the same for f_y.

The scaling factor λ plays a role analogous to the speed of a mouse. The threshold δ is a sensitivity parameter: movements so small to generate movement vectors that differ by less than δ are not detected.

4 Experimental evaluation

In order to verify whether our proposal is indeed feasible, we performed two kinds of experimental evaluations. First, we defined and performed an *objective* benchmark aimed at assessing the effectiveness of the motion detection algorithm we adopted. To this end we used a video sequence obtained by simulating the camera movement inside a computer rendered 3D scene. The results are presented in Section 4.1. Second, we defined and performed a *subjective* benchmark involving 20 subjects in a usability test. In this experiment we compared a laptop-based prototype implementation of our proposal against more traditional devices: mouse, keyboard and touchpad. The results are presented in Section 4.2. We first performed some preliminary experiments in order to find out acceptable values for the number n of features and other internal parameters of the KLT algorithm: we found that we could achieve good performances with $n = 40$.

4.1 Objective benchmark

We defined and performed an objective benchmark aimed at assessing the effectiveness of the motion detection algorithm we adopted. We built a video sequence using an artificial (computer rendered) 3D scene in which the view point has been moved along a predefined trajectory T_1. We input this sequence to the algorithm described in the previous section and determined the corresponding detected trajectory T_2. Finally we compared quantitatively the difference between T_1 and T_2.

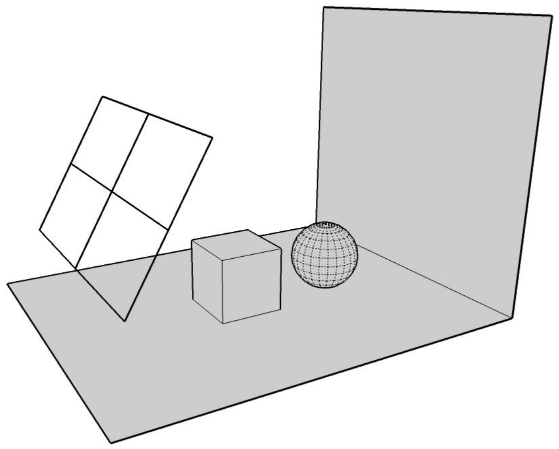


Figure 1: The test scene. The camera moves on the wireframed tilted rectangle on the left, pointed towards a direction perpendicular to the same square.

We composed the scene and defined the trajectory in order to simulate the movement of the device inside a real scenario composed by both near and far objects, as follows. The scene is composed by two objects (a cube and a sphere) posed on a solid floor and a background, as visible in Figure 1. The two objects are rendered with a diffuse color surface; two photo-realistic textures of a real laboratory are applied to floor and background.

We defined a closed trajectory T_1 on which the camera (i.e., the view point in the 3D scene) moved at constant speed. The trajectory is shown in Figure 2(a) and lies on the wireframed rectangle of Figure 1. The point of view is always towards the populated part of the scene, perpendicularly to the wireframed rectangle, hence leading to a purely transitional movement of camera.

In order to simulate typical embedded camera limitations, we repeated the experiment by varying the resolution of the video obtained by rendering the 3D scene. We also varied the speed of the movement along the trajectory, by decimating the video sequence frames and hence obtaining sequences composed of different numbers of frames. The latter parameter is important since the more "distant" two consecutive image frames are, the more difficult is to track features with KLT method.

We assessed the effectiveness of our approach based on two numerical indexes calculated on each T_1, T_2 pair, ϵ_{module} and $\epsilon_{\text{direction}}$ that capture respectively the imprecision in movement module and direction, as follows. For each frame of the video sequence, we computed the corre-

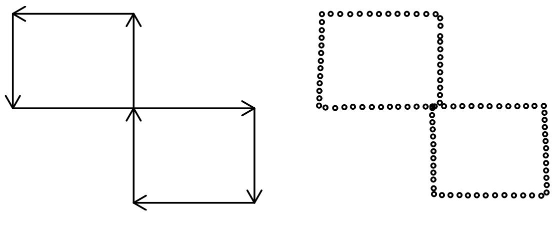


Figure 2: The trajectory along which the camera moves: on the left, as defined; on the right, as obtained with the detection algorithm (with $r = 1$ and $s = 1$, see the text). Note that these trajectories lie on the wireframed one shown in Figure 1.

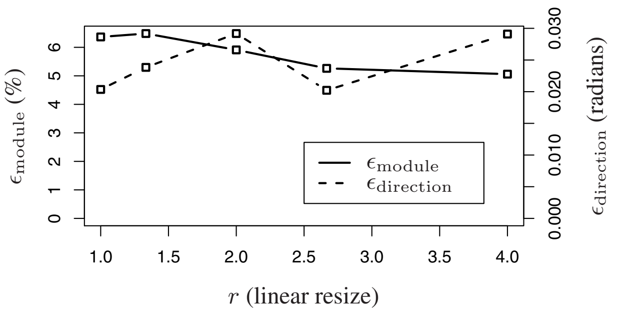


Figure 3: Indexes vs. resolution, at slow movement speed ($s = 1$)

sponding scroll vector $\vec{f_k} = (f_{x,k}, f_{y,k})$, k being the k-th frame. We computed ϵ_{module} as the average relative error of scroll vectors *modules* $|\vec{f_k}| = \sqrt{f_{x,k}^2 + f_{y,k}^2}$, with respect to the expected (and constant) movement module. We computed $\epsilon_{\text{direction}}$ as the average absolute error of scroll vectors *directions* $\angle\vec{f_k} = \arctan(\frac{f_{y,k}}{f_{x,k}})$, compared against the expected directions (i.e., against $\pi/2$ in the first segment of the trajectory, π in the second, and so on). Clearly, for both indexes, the lower the index value, the better the motion detection.

Figure 3 shows the values of the benchmark indexes ϵ_{module} and $\epsilon_{\text{direction}}$ against the resolution r: we experimented with 320×240 ($r = 1$), 240×180 ($r = 1.33$), 160×120 ($r = 2$), 120×90 ($r = 2.66$) and 80×60 ($r = 4$). It can be seen that both indexes are largely independent of resolution: ϵ_{module} varies from 5% to 7% and $\epsilon_{\text{direction}}$ varies from 0.02 to 0.03 radians. The obtained values, in particular for $\epsilon_{\text{direction}}$, confirm that the motion detection algorithm performances are satisfactory. These rather small errors, moreover, can be mitigated further by the user during real usage, since the user would receive from the device screen an immediate feedback of the movement.

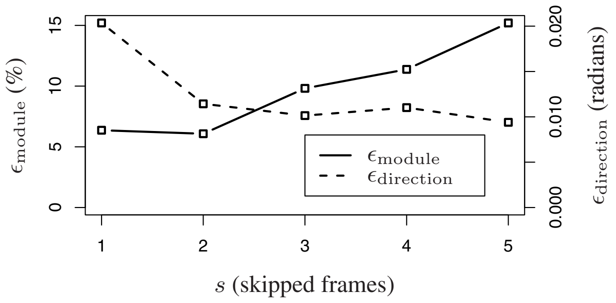


Figure 4: Indexes vs. movement speed at 320×240 ($r = 1$)

Figure 4 shows the values of the benchmark indexes against the movement speed s. We started from a video sequence of 100 frames and we experimented with shorter sequences obtained using only 1 over s frames of the original sequences: the higher s, the faster the movement as perceived by the prototype. We experimented with sequences of 100 ($s = 1$), 50 ($s = 2$), 33 ($s = 3$), 25 ($s = 4$) and 20 frames ($s = 5$). The figure shows that, while increasing the movement speed (i.e., while increasing s), ϵ_{module} increases almost linearly. Hence, as expected, accuracy of motion detection decreases at high speeds. Concerning $\epsilon_{\text{direction}}$, it decreases with low values of s and then remains almost constant. This result is justified by the fact that when the movement speed increases, being larger the vector modules, errors on vector x and y components have a marginal influence on the direction.

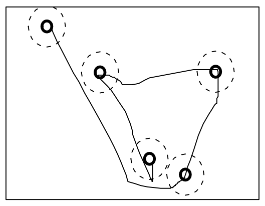
We visually inspected the trajectories obtained in the experiments which confirmed the good results given by the benchmark indexes: Figure 2(b) shows the one corresponding to $r = 1$ and $s = 1$.

In conclusion, the objective benchmark experimentation confirms that the chosen algorithm is indeed capable of detecting motion with a small device camera, and it is sufficiently robust in respect to resolution and movement speed.

4.2 Usability test

We defined and performed a subjective benchmark involving 20 subjects in a usability test. The benchmark consisted of a number of tests. In each test a testing software on a laptop computer generated a random trajectory to be followed by the user, first with an USB webcam, then with mouse, keyboard, touchpad and camera again, in this order. We chose to have two sessions with the camera to allow users to benefit of some form of training. The webcam was easily movable with a single hand. We configured the speed of the input devices so that they could be used "comfortably" by the authors of this paper, and then used the same configuration across all tests (we further elaborate on this point below).

Each trajectory consisted of a sequence of 20 targets (small red circles) generated at random on the laptop


(a) Touchpad

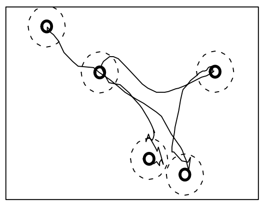

(b) Camera-2

Figure 5: The trajectory recorded for subject n. 1 with the touchpad and with the keyboard. Thick circles represent the targets; dashed circles represent the proximity zones.

Table 1: Index t^{hit} for the 5 sessions. Column 3 shows the variation of the index in respect to the keyboard, which we took as a baseline (the lower, the better). Columns 4 and 5 show minimum and maximum variation among users.

Session	t^{hit} (secs)	Variation	Min	Max
Keyboard	54	0%	0%	0%
Mouse	14	−75%	−81%	−68%
Touchpad	19	−65%	−80%	−51%
Camera-1	43	−21%	−46%	1%
Camera-2	41	−24%	−48%	9%

screen. The user had to hit the first target appearing on the screen. Once a target was hit, the next one appeared at a random position. The sequence of target positions was the same among the 5 sessions but varied among different subjects.

Figure 5 shows two portions (first 5 targets over 20) of the trajectory recorded in the Touchpad and Camera-2 sessions for one of the subjects. It can be seen that the trajectory is slightly more erratic nearby the targets—the proximity zones delimited by dashed circles—using the camera (Figure 5(b)); however, the two trajectories are substantially similar.

For each session and for each user, we recorded the trajectory followed by the cursor. Based on these data, we generated two quantitative benchmark indexes for each <session, user> pair, as follows. First, we computed the time t^{hit} the user took to hit all the 20 targets.

Then, we observed that in many cases the cursor arrives quickly in the neighborhood of a target and then spends some time wandering around in the attempt to actually hit the target. We also observed that this effect tends to depend on the input device. For this reason, we defined and evaluated an additional benchmark index, as follows. We defined a *proximity zone* around each target, i.e., an invisible circle larger than the target itself and centered on it. The size of the proximity zone is the same for all targets. We measured, for each target j, the time t_j^{hit} the user took to hit the target and the time $t_j^{\text{prox}} < t_j^{\text{hit}}$ elapsed in between entering the proximity zone for the first time and hitting the target: t_j^{prox} hence represents the time the user took to refine his positioning around the target (usually with small and slow cursor movements). Finally, we computed the ratio $\frac{t^{\text{prox}}}{t^{\text{hit}}}$ between the time $t^{\text{prox}} = \sum_j t_j^{\text{prox}}$ spent refining positioning around targets and the total session time $t^{\text{hit}} = \sum_j t_j^{\text{hit}}$. In other words, $\frac{t^{\text{prox}}}{t^{\text{hit}}}$ quantifies the inaccuracy associated with the corresponding input device: the greater $\frac{t^{\text{prox}}}{t^{\text{hit}}}$, the longer the portion of time the user spends wandering around the target trying to hit it.

Tables 1 and 2 show the indexes obtained in the 5 sessions and averaged across all subjects. We took the keyboard as the baseline and highlighted variation of the indexes in respect to that input device. Recall that the lower t^{hit}, the faster the user, and the lower $\frac{t^{\text{prox}}}{t^{\text{hit}}}$, the more accurate the user.

In order to interpret these results correctly, it is worthwhile pointing out what follows. The speed of the movement provoked by a given input device is heavily dependent on the configuration parameters of that device—following a given trajectory with a mouse configured for slow motion speed will take much longer than with the same mouse configured for fast motion speed. Moreover, it is not possible to configure different input devices for working at the same speed—the repeat rate of a keyboard cannot be compared to the motion speed of a mouse, for example. It follows that this benchmark is *not* meant to provide an absolute ranking among the various devices. The aim is assessing whether our proposal may allow defining trajectories in a way that is at least "comparable" to that of more traditional pointing devices.

Said this, it can be seen that the proposed approach performs better than the keyboard in terms of time t^{hit} and comparably to touchpad in terms of inaccuracy $\frac{t^{\text{prox}}}{t^{\text{hit}}}$. As expected, mouse is the most effective input device, followed by touchpad. Such devices are usually not included in mobile devices, however—with the notable exception of those coming with a touchscreen. Looking at raw results, we found that all subjects but two completed the task in a slightly shorter time with the camera than with the keyboard.

Table 1 also shows that there was an improvement between the time t^{hit} obtained in the first and second sessions with the camera. Interestingly, this corresponds to a small worsening of achieved accuracy. We observed the subjects while performing the test and we think that this is due to the fact that users, thanks to a perceived increase of confidence in the use of the novel pointing device, often gave up accuracy in favor of time, aiming at accomplishing the task sooner. We remark that all subjects were novel to the use

Table 2: Index $\frac{t^{\text{prox}}}{t^{\text{hit}}}$ for the 5 sessions. Column 3 shows the variation of the index in respect to the keyboard (the lower, the better). Columns 4 and 5 show minimum and maximum variation among users.

Session	$\frac{t^{\text{prox}}}{t^{\text{hit}}}$ (%)	Variation	Min	Max
Keyboard	18	0%	0%	0%
Mouse	30	68%	10%	200%
Touchpad	34	90%	−14%	263%
Camera-1	33	84%	20%	273%
Camera-2	35	99%	21%	339%

of the proposed pointing approach and were not given any chance to train with, except of the first session (denoted by Camera-1 in Tables 1 and 2).

It seems fair to claim that the usability experiment demonstrates that our proposal allows defining trajectories with speed and accuracy similar to those of more traditional input devices—which would be more cumbersome or even impossible to use in the envisioned scenario. With the configuration settings of our experiment, our approach is faster than the keyboard and provides an accuracy very close to that obtained with the touchpad, even with untrained users.

In summary, we believe that our proposal may represent a powerful method for fine-grained motion detection on hand-held devices. It does not require any additional dedicated hardware and appears promising, in particular, for scrolling large contents on the device screen.

References

[1] R. Ando, K. Shinoda, S. Furui, and T. Mochizuki. A robust scene recognition system for baseball broadcast using data-driven approach. In *CIVR '07: Proceedings of the 6th ACM international conference on Image and video retrieval*, pages 186–193, New York, NY, USA, 2007. ACM.

[2] Y. Arase, T. Hara, T. Uemukai, and S. Nishio. OPA browser: a web browser for cellular phone users. In *UIST '07: Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 71–80, New York, NY, USA, 2007. ACM.

[3] T. Camus. Real-time quantized optical flow. *Real-Time Imaging*, 3:71–86, 1997.

[4] O. de Bruijn, R. Spence, and C. H. Tong. Movement in the web. In *CHI '01: CHI '01 extended abstracts on Human factors in computing systems*, pages 209–210, New York, NY, USA, 2001. ACM.

[5] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.

[6] J. Hwang, J. Jung, and G. J. Kim. Hand-held virtual reality: a feasibility study. In *VRST '06: Proceedings of the ACM symposium on Virtual reality software and technology*, pages 356–363, New York, NY, USA, 2006. ACM.

[7] J. Linjama and T. Kaaresoja. Novel, minimalist haptic gesture interaction for mobile devices. In *NordiCHI '04: Proceedings of the third Nordic conference on Human-computer interaction*, pages 457–458, New York, NY, USA, 2004. ACM.

[8] F. Loewenich and F. Maire. Hands-free mouse-pointer manipulation using motion-tracking and speech recognition. In *OZCHI '07: Proceedings of the 2007 conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artifacts and environments*, pages 295–302, New York, NY, USA, 2007. ACM.

[9] B. D. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 121–130, Vancouver, Canada, 1981.

[10] I. Oakley and S. O'Modhrain. Tilt to scroll: evaluating a motion based vibrotactile mobile interface. *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pages 40–49, 18-20 March 2005.

[11] V. Roto, A. Popescu, A. Koivisto, and E. Vartiainen. Minimap: a web page visualization method for mobile phones. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 35–44, New York, NY, USA, 2006. ACM.

[12] C. Tomasi and T. Kanade. Detection and Tracking of Point Features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.