Mobile Camera-based User Interaction

Antonio Haro, Koichi Mori, Tolga Capin, Stephen Wilkinson

Nokia Research Center, 6000 Connection Drive, Irving, TX 75039, USA
{antonio.haro, koichi.mori, tolga.capin, stephen.wilkinson}@nokia.com

Abstract. We present an approach for facilitating user interaction on mobile devices, focusing on camera-enabled mobile phones. A user interacts with an application by moving their device. An on-board camera is used to capture incoming video and the scrolling direction and magnitude are estimated using a feature-based tracking algorithm. The direction is used as the scroll direction in the application, and the magnitude is used to set the zoom level. The camera is treated as a pointing device and zoom level control in applications. Our approach generates mouse events, so any application that is mouse-driven can make use of this technique.

1 Introduction

Mobile devices currently support navigation through a joypad/direction keys or scroll bars on touch-sensitive screens using a stylus-based pen. Although these modes of interaction are sufficient for small sized content, more intuitive techniques are required for navigating more complex data. It is difficult to use these techniques to navigate a full-sized Web page or to select an item from dozens of choices in a list box of messages, photos, audio files, phonebook entries or other mobile content. On devices with larger form-factors, additional keys provide a better user experience since keys can be dedicated to specific tasks such as page up/down and zoom level. Smart phones cannot make use of such keys due to limited physical space. Stylus-based interaction for navigation is an alternative, but requires two-handed interaction and has been shown to cause additional attentional overhead in users [1]. Consequently, alternative interaction techniques are desired. Other sensors could be added to mobile devices such as accelerometers (*e.g.,* Samsung's SCH-S310 smartphone), but these can be difficult to integrate into existing consumer-level devices at both the software and hardware level. In addition, such sensors are known to have error buildup over time since some infinitesimal acceleration is always measured.

To address these problems, we propose using the camera sensor as the input device. Feature-based tracking of the incoming video is used to estimate both motion direction and magnitude. The direction estimates are used for scrolling while the magnitude of the physical movement can drive the current zoom level in an application or be used for shake detection. This approach provides a more natural user interaction maximizing the use of the display, minimizing attentional overhead to the user, and permitting one-handed interaction. This approach does not preclude the use of a joypad, and can be used as an extension of joypad-based interaction, where the joypad could be used for fine-grained selection. We tested our approach on an image-browsing task in a photo browsing

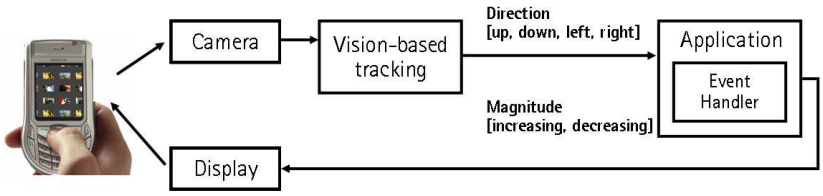


Fig. 1. A tracking algorithm is used to determine movement direction and magnitude. These are then treated as mouse events by the application's event handler, and the user's view is updated.

application, as well as in a document viewer and in games. In informal tests, users preferred our solution to a joypad-based navigation. Joypads and scroll buttons are adequate for navigation of small datasets on limited sized displays, but not for large or complex data.

2 Related Work

Mobile camera-based tracking has been researched by several groups. Rohs *et al.* [2] perform tracking based on dividing incoming camera frames into blocks and then determine how the blocks move given a set of discrete possible translations and rotations. Our algorithm is instead based on tracking individual corner-like features observed in the entire incoming camera frames. This allows our tracker to recognize sudden camera movements of arbitrary size, as long as at least some of the features from the previous frame are still visible, at the trade-off of not detecting rotations.

Augmented reality research on mobile camera-based tracking systems includes that of Möhring *et al.* [3], who track a color-coded 3D marker to estimate 3D camera pose, after an initial calibration step. Our work is instead focused on new user interfaces using computed 2D motion, so we do not require markers or user calibration for tracking. Drab *et al.* [4] present a computationally inexpensive tracking system, however their system has problems with repeating textures and requires scenes with high dynamic range which ours does not. Beier *et al.* [5] present a marker-less tracking algorithm, however it does not run on mobile devices as ours does and also requires matching with known 3D models, which we do not require.

Additional related work includes the Mosquito game available for the Siemens SX1 mobile phone, among others that have been created since then for many smart phone platforms. While camera motion is indeed estimated in these games to translate sprites accordingly, it should be noted that the detected motion does not need to be exact as the sprites are rendered on top of the video but not attached to any feature. As such, only approximate motion is required. Since our tracked motion needs to match the user's physical motion exactly, a higher degree of accuracy is required which from our testing is not present in current commercial camera motion tracking-based games. Recently [6] have created the first Kalman-based tracker for mobile devices. Kalman tracking yields higher

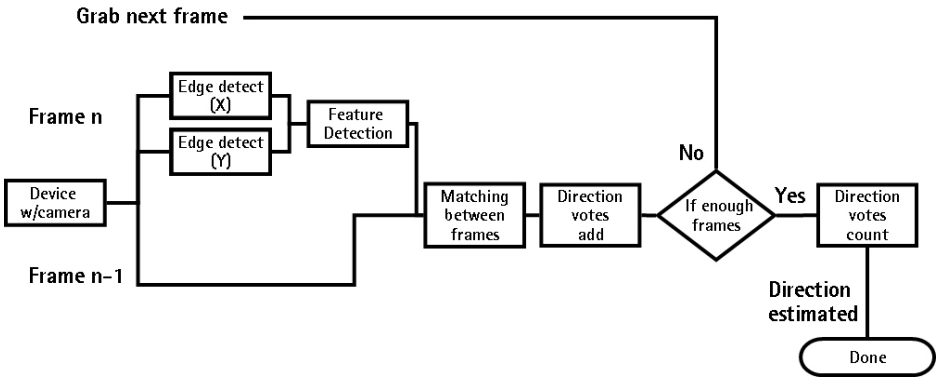


Fig. 2. Tracking algorithm and direction estimation flowchart.

quality motion estimates but has higher computational requirements and needs a more complex implementation than in our work.

Prior work on speed-dependent automatic zooming on mobile devices has focused on performing zoom and/or panning using either additional hardware or sensors. Igarashi and Hinckley [7] performed speed-dependent automatic zooming by creating equations based on mouse motion which determine whether to zoom in or out. Recently, Cockburn *et al.* [8] have performed extensive user studies to find empirical values for the equations relating speed and zoom levels for mouse motion. Other pointer device based scrolling techniques include the Alphaslider [9], the FineSlider [10] and the Popup Vernier [11]. Those works focused on how to effectively select an item from a list of a large number of items. An extended scroll bar component that allows the user to change the scrolling speed was used. Our approach can be used as an alternative in instances where a pointing device is not available, such as on a mobile device.

Our work is also similar to the scroll [12], and peephole [1] displays works and work on tilt-based interaction [13, 14]. In these works, the goal was to perform scrolling on mobile devices in a more intuitive fashion by using additional sensors. Scroll-detection sensors that were used included both mechanical and optical mouse sensors, position and orientation sensors, and ultrasonic transmitters/receivers. While additional sensors were required in those works, we use only the camera as the direction sensor instead of adding new sensors. Doing so allows for regular camera-equipped smart phones to have an additional interaction modality without modifying the phone.

Other hardware-based solutions to scrolling come from the commercial domain. Apple's iPod, while not performing zooming, makes use of a touch-sensitive scroll wheel whose scrolling speed depends on the number of songs in a play list, to maximize display usage. On other mobile devices such as cell phones, touch screens are commonly used to address display size limitations. Touch screens can allow users to interact and scroll through their data more effectively than using buttons as the stylus can just be dragged down a scrollbar. However, touch screens have the disadvantage of not permitting one-handed operation.

3 Camera-based Movement Estimation

Our approach is to use the mobile phone's onboard camera as the source of input. The user's physical movement of the device is captured in incoming video, which is analyzed to determine scroll direction and magnitude. The detected direction is sent to the event handler exactly as a corresponding mouse event, while the magnitude is used to specify the current scroll level. Figure 1 shows an overview of our system.

Correctly interpreting the observed motion from the camera's incoming video requires accurate tracking. To determine the motion direction, a feature-based tracking algorithm is used. To determine the magnitude of the physical movement, motion history images (MHI) [15] are used, which were originally used for performing action and gesture recognition. Our tracking algorithm provides four directions as application-level events, similar to mouse movement: up, down, left and right, in the camera plane. The magnitude is also passed as an event, where two states are possible: motion magnitude increasing or decreasing. The rest of the application remains the same as the only changes are the cause of the events passed to the event handler. This allows applications to use the camera easily, without any knowledge of the underlying tracking algorithms or camera hardware.

High-level Algorithm Description: The tracking system was implemented on the Symbian OS. The process diagram of the tracker is presented in Figure 2. Two frames are grabbed, n and $n-1$. Edge detection is performed on both frames using the Sobel filter. The thresholded absolute values of the x and y derivatives are used as features as they peak in corner-like regions. Feature matching is performed between frames using template matching with 15x15 search windows. Direction voting is performed using variables, and the final decision on motion estimation is performed every 4 frames. This allows several frames to 'vote' on the motion, keeping the scrolling from being incorrect due to any errors in other parts of the system.

Feature Detection: Traditional features include edges and corners. However, edges are not significantly temporally coherent and corner features are too computationally expensive to find at many image locations while retaining real-time performance. Instead, corner-like features are detected using image gradient information (Equation 1).

$$S(x,y) = (G_x^2 + G_y^2) \tag{1}$$

$$G_x(x,y) = \frac{\partial I}{\partial x} \approx sobel_x(x,y), \;\; G_y(x,y) = \frac{\partial I}{\partial y} \approx sobel_y(x,y) \tag{2}$$

$S(x,y)$ is the Sobel operator and the Sobel functions denote convolution with the x and y components of the Sobel kernel. All corners cannot be detected using the Sobel operator; however, it provides a useful first-step culling of pixels for additional processing. Frame n is filtered using the Sobel x and y filters. The Sobel operator is then applied to every pixel in scanline order. If $S(x,y)$ is

greater than an edge threshold, the pixel at (x, y) in frame n is labeled a feature. Once k features are detected the Sobel operator is no longer applied, with $k = 50$ providing good results. The list of detected features is then passed onto the next step, template matching.

Template Matching: Template matching alone is not reliable since only image pixel difference errors are used and neither sensor noise nor lighting variations are modeled. Template matching is used in this algorithm because it is computationally inexpensive and provides useful match estimates. Matching is performed for each feature detected in Frame n. For each feature, the 15x15 pixel neighborhood around the feature is tested for image similarity using the sum of squared differences (SSD). 15x15 sized features were chosen as this size is large enough to capture visually distinct regions and significant intra-frame physical motion. Let t_f denote a 15x15 pixel sized template image consisting of the pixel neighborhood at (i, j), where feature f was detected in Frame n. Then, to find the closest match in Frame $n - 1$, we can use the following equation (equivalent to SSD in the case of a non-changing image and template):

$$\min_{(x,y)\in N} M(x, y) = \sum_{k=-7}^{7} \sum_{l=-7}^{7} t_f(k + 7, l + 7) f(x + k, y + l) \tag{3}$$

where N is the 15x15 pixel neighborhood around (i, j). The location of the closest match is found by testing every offset around location (i, j) and comparing the 15x15 sub-image there with the 15x15 sub-image from the feature's pixel neighborhood. The matching is performed from the current frame to the previous frame instead of vice versa since a feature detected in Frame $n - 1$ may not be detected in Frame n.

Direction Estimation: The direction cannot be estimated by simply counting the most dominant template matching direction amongst all features. Such estimation would be temporally incoherent since neither the feature detection nor template matching component is perfectly coherent. To remove temporal incoherencies, the estimated directions of the matched feature locations are temporally filtered. For each frame, the most dominant direction is computed and a counter for that direction is incremented. For each direction, a counter is initialized at zero. After m frames, where m is typically between $3 - 5$ frames, the counter with highest count is chosen as the estimated direction with other counters reset to zero. Only a small amount of temporal filtering is needed since the features are individually robust. The direction estimation fails if the camera is moved largely between frames since at least one feature from the previous frame must be visible, as in other template matching based algorithms.

Determining Camera Motion Magnitude: The directions of dominant camera motion are computed using the tracking algorithm, but their magnitudes are not known accurately. Camera motion magnitude must be calculated accurately to determine how to adjust the scroll speed in applications that need zoom control. We use motion history images (MHI) [15] to estimate camera

motion magnitude. Motion histories are encoded in single images such that a single image can be used for simple, robust and computationally inexpensive gesture recognition. An MHI is computed by performing background subtraction between the current and previous frames. At locations where the pixel values change, the MHI is updated by decrementing by a pre-defined constant amount. By averaging the intensity values of the MHI, the average camera motion magnitude is estimated. The following equation calculates the MHI's value at position (x, y) in the camera image at time t:

$$H_T(x, y, t) = \begin{cases} \tau & \text{if } D(x, y, t) = 1, \\ \max(0, H_T(x, y, t-1) - 1) & \text{otherwise} \end{cases} \quad (4)$$

where H_T is initialized to be 0 in the first frame and $D(x, y, t)$ denotes an image difference between frame t and $t-1$, with τ being the number of frames of motion that the MHI should represent. In this manner, the MHI compactly represents the motion magnitude from the incoming video, with areas 'lighting up' when significant motion is detected and the whole MHI fading to black if no motion is detected. The simplicity of the MHI calculation makes it amenable for use in driving the scroll level and velocity as well as camera shake detection (Section 5).

4 Results

Our algorithm's frame rate is 10fps on our test hardware, a Nokia 6630 smartphone, with 1 motion magnitude and motion direction update every $3-5$ frames, with more frequent updates possible at the expense of accuracy. In our experimentation, only smooth walls result in complete tracking failure since temporally coherent features are not found. Otherwise, the tracker's performance matches users' physical motion at a responsive rate with no errors in typical indoor and outdoor environments.

Motion estimates computed are accurate for user interaction, never estimating the wrong motion direction even when the device is abruptly switched in directions. The motion magnitude is also accurate, mainly representing the immediate past since is is not recomputed often, which turns out to be suitable for automatic zooming. The biggest limitation of our algorithm is that detectable physical motion speed is limited since intra-frame matching is performed and part of the previous frame must be visible in the current frame to establish feature correspondences.

To measure the accuracy and performance of our algorithm, we compared our tracking algorithm with the Kalman filter-based tracker from [6]. The Kalman tracker has higher motion estimation accuracy, as expected, since the Kalman filter greatly improves the quality of intra-frame matching. However, the computational requirements are significantly greater since several matrices must be multiplied and inverted per frame. On devices with limited computational resources, our algorithm provides sufficient motion and velocity accuracy for user interaction with many leftover cycles for intense applications such as 3D games

Fig. 3. Picture browser application. The application automatically adjusts the zoom level to help the user browse.

(Section 5) at the trade-off of more limited accuracy since the temporal filtering in our algorithm cannot match a Kalman filter.

A general issue with camera-based mobile device user interfaces is that the user's physical environment may have very limited space. In such situations, it may be advantageous to provide a 'clutch' to turn the tracking on/off. This would emulate the act of lifting a mouse once the edge of a desk is reached in traditional desktop interaction. In our informal testing we did not provide a clutch, however in commercial implementations this is a consideration to keep in mind. Another general issue is that all camera-based user interfaces require adequate light for tracking. Problems can arise particularly for a mobile device in low lighting conditions if the device has an automatic 'night mode' as incoming images will already be processed and may be too noisy. In dark environments, applications should default to joypad-based interaction.

5 Applications

We implemented several test applications to demonstrate our algorithm's strengths and limitations. In general, any mobile application that requires scrolling and/or zooming could make use of our approach provided that an on-board camera is present.

Zooming Photo Browser: As cameras become more widespread on mobile phones and storage size increases, managing photos becomes a more difficult task for the user as large amounts of information must be viewed with limited input modalities. Current typical photo viewer applications show photo thumbnails as lists, grids, or 3D carousels. Since image selection and scrolling are done with the joystick, the amount of time a user needs to browse their images is directly related to the number of images that they are browsing.

Our photo browser test application (Figure 3) shows thumbnails of the user's photos in a grid layout. The user can scroll in four directions (up, down, left, right, in the camera plane) by physically moving the mobile device. In this case it is difficult to view all the images as some zoom control is required when looking for a particular image. If the zoom level is not properly set, it is difficult for a user to select a particular image from the set as the scrolling will be too fast. To address this problem, we used the technique introduced in [7]. Adaptive zooming based on the magnitude of the user's physical movement keeps the scroll speed

(a) **(b)**

Fig. 4. (a) Camera-based interaction in a document viewing application. (b) Mapping physical motion to viewing direction creates the illusion of a window into an environment.

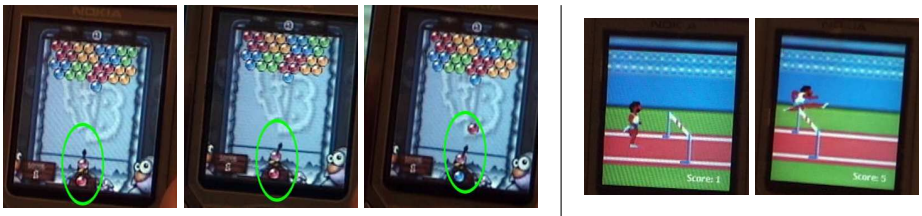
virtually consistent, allowing the user to browse more thumbnails by only moving the device faster.

Document Viewer: Scrolling a document is a commonly difficult task on mobile devices. For instance, web content designed for desktop computers is much vertically longer since mobile devices have narrower screens. In addition, joystick scrolling is especially difficult when scrolling line-by-line. An alternative is to add an extra hardware button for scrolling. However, an extra button is not a preferable solution for mobile device manufacturers due to the lack of extra physical space on the device along with additional manufacturing costs.

In the document viewer prototype application we implemented (Figure 4 (a)), the user can vertically scroll documents by moving the device. The scroll speed depends on how fast the user moves the device, which is much more intuitive than changing scrolling speed depending on how long the user presses the joystick or via menu options and settings. One issue we identified in this application is that at some point, the user has to move the device more than they can reach. For example, if the user is scrolling to the right, at some point they will reach the physical limit of their arm's motion. To address this problem, we use the joystick as a 'carriage return', which scrolls the document to the beginning of the next line and allows the user to move their arm back to the left again. After a carriage return, all tracked motion except movement to the right is ignored.

3D Game Interaction: Creating an immersive 3D experience is difficult on mobile devices due to the limited display size. The most immersive experiences are typically created using a combination of large displays reducing peripheral vision as much as possible and/or virtual environment navigation tied to the user's physical motion. In our prototype (Figure 4 (b)), we map the user's physical motion to the view-point to create the illusion of a window into a 3D world.

Our renderer loads standard Quake III^{TM} or Quake III Arena^{TM} maps. Textures, light maps, curved surfaces and lighting calculations are disabled for performance. The rendering is done using the OpenGL ES implementation available in the latest Symbian OS-based Series 60 SDK. Pre-computed vertex lighting and fixed point calculations are used to improve performance due to the lack of a floating-point unit on our test hardware. The renderer is able to realistically ren-



(a) **(b)**

Fig. 5. (a) Players move the device left and right to aim, and shake the device to launch a bubble. (b) A jump command is issued by shaking the device at the correct time to avoid tripping on the hurdle.

der lit virtual environments with several thousand polygons per scene at $3-10$ frames per second, depending on the environment that is chosen.

Navigation of the virtual environments is performed with a combination of physical motion and keypad presses. Actual movement in the environment is controlled by the keypad. The user looks around in the scene by physically moving the device around their body in the directions that they would like to look. We map the tracked camera motion directions to a trackball as in traditional mouse-based 3D interaction. The combination of detailed environments, camera-based control and interactive frame rate create a mobile user experience closer to that using additional hardware or larger displays.

2D Game Interaction: Camera-based user interaction can be used to enhance 2D games as well as those that are 3D. Camera motion can be used to add an additional element of interaction in games that require precise movements or very well-timed button presses. We created puzzle and action game prototypes to investigate these ideas using the camera motion and shake detection algorithms presented.

We modified the open source Series 60 port of the 'Frozen Bubble' puzzle game (http://fb–s60.sourceforge.net/), switching the game control from using the keypad to using the camera (Figure 5 (a)). In our version, the user moves their device left and right to aim and performs sudden shakes to launch their bubble. This has the effect of significantly changing gameplay as careful arm motions are now required to aim, instead of a number of button presses, which increases the excitement as the game is now more physically-based.

We created a camera-based action game prototype as well. Using sprites and artwork from Konami's 'Track and Field^{TM}' game for the Nintendo Entertainment System, a new game (Figure 5 (b)) was created. A runner must jump over a never-ending number of approaching hurdles. To jump, the player must time the shaking of their device correctly so that the character does not crash into hurdles. Relying on the camera exclusively for input results in a game that is very simple to learn and understand but difficult to master, providing a new type of game. Shake detection is performed by thresholding the average intensity of the computed MHI (Section 3).

6 Conclusions and Future Work

We introduced a new approach to improve the user experience on camera-equipped mobile devices. A feature-based tracking algorithm was presented to detect both physical motion direction and magnitude to permit one-handed physical movement based interaction. A camera was chosen since cameras are now widely available on mobile devices and are very powerful sensors that can be used without introducing new sensors. We demonstrated our approach in several applications including a document viewer, photo collection browser and some games. In the future, we would like to perform user studies to determine how to improve user interaction further using mobile cameras.

While our tracking algorithm is computationally efficient and works well in practice, some situations cannot be handled. Severe lighting differences will cause the template matching to stop working properly. Motion in front of the camera is ambiguous and can affect tracking results as it is impossible to tell whether the camera is moving or not. Shadows may confuse the tracking system, but there are known techniques for robust tracking in the presence of shadows that will be incorporated into the tracking algorithm once additional processing speed is available.

References

1. Yee, K.P.: Peephole displays: pen interaction on spatially aware handheld computers. In: Proceedings of the SIGCHI conference on Human factors in computing systems. (2003) 1–8
2. Rohs, M.: Real-world interaction with camera-phones. In: International Symposium on Ubiquitous Computing Systems. (2004)
3. Möhring, M., Lessig, C., Bimber, O.: Optical tracking and video see-through ar on consumer cell phones. In: Proceedings of Workshop on Virtual and Augmented Reality of the GI-Fachgruppe AR/VR. (2004) 193–204
4. Drab, S., Artner, N.: Motion detection as interaction technique for games & applications on mobile devices. In: Extended Abstracts of PERVASIVE: Workshop on Pervasive Mobile Interaction Devices. (2005) 48–51
5. Beier, D., Billert, R., Bruderlin, B., Stichling, D., Kleinjohann, B.: Marker-less vision based tracking for mobile augmented reality. In: Proceedings of the Second IEEE and ACM International Symposium on Mixed and Augmented Reality. (2003) 258–259
6. Hannuksela, J., Sangi, P., Heikkila, J.: A vision-based approach for controlling user interfaces of mobile devices. In: IEEE Workshop on Vision for Human-Computer Interaction. (2005)
7. Igarashi, T., Hinckley, K.: Speed-dependent automatic zooming for browsing large documents. In: Proceedings of the ACM symposium on User interface software and technology (UIST). (2000) 139–148
8. Cockburn, A., Savage, J., Wallace, A.: Tuning and testing scrolling interfaces that automatically zoom. In: Proceedings of the SIGCHI conference on Human factors in computing systems. (2005) 71–80
9. Ahlberg, C., Shneiderman, B.: The alphaslider: a compact and rapid selector. In: Proceedings of the SIGCHI conference on Human factors in computing systems. (1994) 365–371
10. Masui, T., Kashiwagi, K., George R. Borden, I.: Elastic graphical interfaces to precise data manipulation. In: CHI'95: Conference companion on Human factors in computing systems. (1995) 143–144
11. Ayatsuka, Y., Rekimoto, J., Matsuoka, S.: Popup vernier: a tool for sub-pixel-pitch dragging with smooth mode transition. In: Proceedings of the ACM symposium on User interface software and technology (UIST). (1998) 39–48
12. Siio, I.: Scroll display: Pointing device for palmtop computers. In: Asia Pacific Computer Human Interaction. (1998) 243–248
13. Bartlett, J.: Rock 'n' scroll is here to stay. IEEE Computer Graphics and Applications **20** (2000) 40–45
14. Rekimoto, J.: Tilting operations for small screen interfaces. In: Proceedings of the ACM symposium on User interface software and technology (UIST). (1996) 167–168
15. Davis, J., Bobick, A.: The representation and recognition of human movement using temporal templates. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). (1997) 928–934