Mobile Camera-Based Adaptive Viewing

Antonio Haro, Koichi Mori, Vidya Setlur, Tolga Capin*
Nokia Research Center
6000 Connection Drive
Irving, TX 75039, USA

Abstract

In this paper, 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 computer vision-based 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. The user is free to browse through large data sets on a limited size display with one hand, ideal for the mobile domain.

CR Categories: H.5.2 [Information Interfaces and Presentation]: User Interfaces—Interaction Styles; I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.4.8 [Image Processing and Computer Vision]: Scene Analysis—Motion

Keywords: camera-based interaction, zoom control, scroll control, adaptive views, mobile device, computer vision.

1 Introduction

Recent advances in mobile device hardware have made it possible to store and view large amounts of content. For example, it is now possible on a smart mobile phone to have personal media consisting of thousands of photos or complex documents and full sized web content. The key limitations remain in the small display size and lack of intuitive interaction techniques for navigation of large datasets.

Mobile devices currently support navigation through a joypad/direction keys or scroll bars on touch sensitive screens using stylus-based panning. Although these modes of interaction are sufficient for small sized content, more intuitive techniques are required for navigating larger amounts of information. 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, phone book 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 [Yee 2003]. Consequently, alternative interaction techniques are desired. Other sensors could be added to mobile devices such as accelerometers, but these can be difficult to integrate into existing consumer level devices in both the software and hardware level.

*e-mail:{antonio.haro,koichi.mori,vidya.setlur,tolga.capin}@nokia.com

Copyright is held by the HIT Lab NZ, University of Canterbury
MUM 2005 Christchurch, New Zealand
ISBN 0-473-10658-2

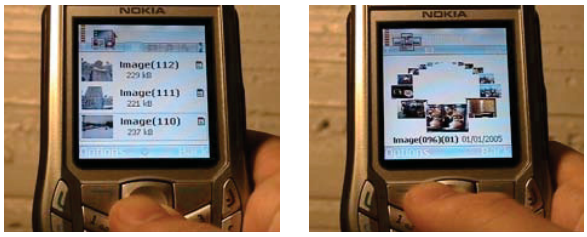

Figure 1: *Picture browsing task. Left: traditional list view, Right: 3D image carousel. Browsing a large collection of images is difficult using conventional approaches as the user's attention is focused on controlling views instead of their objectives.*

To address these problems, we propose a new navigation technique for camera-equipped mobile devices, specifically mobile phones, using the camera sensor as the input device. Our solution is based on analyzing the series of input images from the camera and estimating the motion of the device. Computer vision techniques are used to estimate both motion direction and magnitude. The direction estimates are used for scrolling while the magnitude of the physical movement drives the current zoom level in an application. 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 several tasks, including an image browsing task in a photo browsing application and a location finding task in a map viewer application. In informal tests, users preferred our solution to a joypad-based navigation, but further investigation is required to determine the instances where this holds. Joypads and scroll buttons are adequate for navigation of small data sets on limited sized displays, but perhaps not for large or complex data such as media collections or maps.

2 Related Work

Prior work on adaptive views on mobile devices has focused on performing zooming and/or panning using either additional hardware or sensors. The most relevant work to ours is the work done in [Igarashi and Hinckley 2000]. Igarashi and Hinckley performed adaptive zooming by creating equations based on mouse movement which determine whether to zoom in or out. Our work utilizes the same equations except that it is driven by camera motion computed by a tracking algorithm instead of provided by a mouse as there is

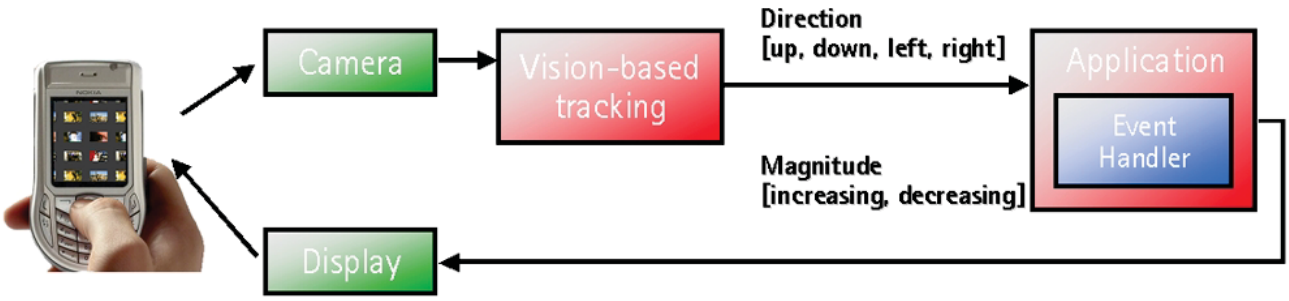

Figure 2: *The user moves their camera enabled mobile phone. A computer vision based tracking algorithm is used to determine movement direction and magnitude. These are then treated as mouse events by the application's event handler, and the user's view is updated.*

none on a mobile device.

Other pointer device-based scrolling techniques include the Alphaslider [Ahlberg and Shneiderman 1994], the FineSlider [Masui et al. 1995] and the Popup Vernier [Ayatsuka et al. 1998]. 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 [Siio 1998], and peephole [Yee 2003] displays works and work on tilt-based interaction [Bartlett 2000; Rekimoto 1996]. 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 scroll bar. However, touch screens have the disadvantage of not permitting one handed operation.

Related camera-based tracking work includes the Mozzies 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. It should also be noted that in these works, only mobile camera motion is used as input; in our work we use the magnitude of the motion as well to control zoom levels, which is novel.

Rohs *et al.* [2004] 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. Kalman filter-based camera motion estimation was demonstrated by Hannuksela *et al.* [2005]. The Kalman tracker has higher motion estimation accuracy, as expected, since Kalman filtering 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 at the trade-off of more limited accuracy since the temporal filtering in our algorithm cannot match a Kalman filter.

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 2 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) [Davis and Bobick 1997] 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.

3.1 High-level Algorithm Description

The tracking system was implemented on the Symbian OS. The process diagram of the tracker is presented in Figure 3. Two frames are grabbed, n and $n-1$, using Symbian's camera API providing

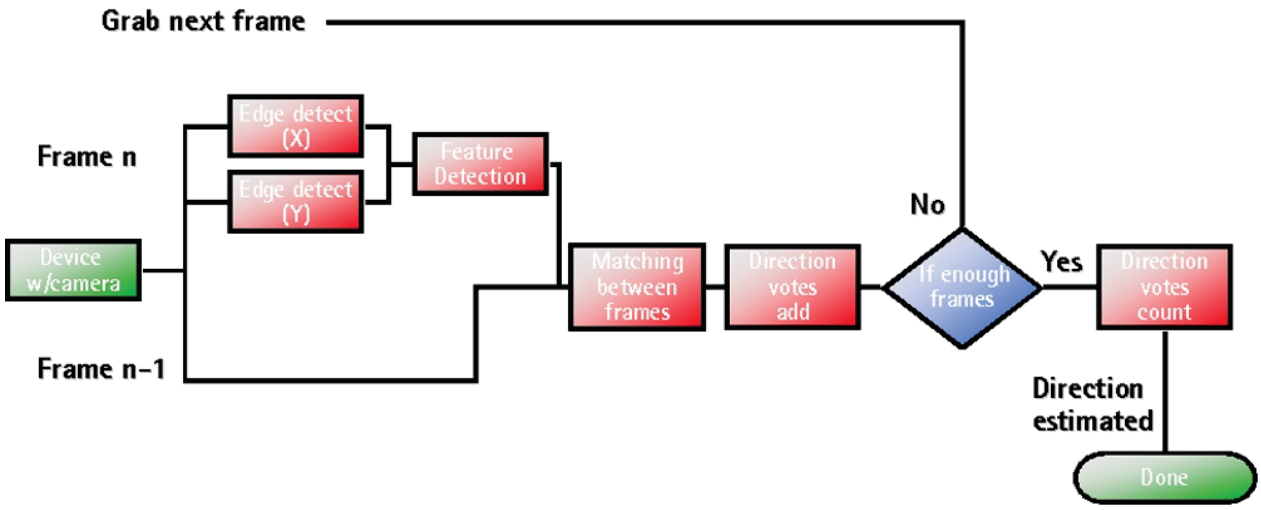

Figure 3: *Our tracker uses the current and previous frame captured by the camera for tracking. Corner like features are detected in the new frame which are matched with the features found in the prior frame so that the prior frame does not have to be reprocessed for features. Direction estimates are accumulated for a number of frames before a movement direction estimate is made.*

JPEG encoded frames. 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. We use a threshold of 50 on both derivatives for each pixel as this value results in a good number of feature candidates for typical scenes. Feature matching is performed between frames using template matching with 15x15 search windows, which we empirically found to be sufficient for our test hardware. With a higher framerate, the templates could be made even smaller making this step even faster, however at the 15 fps framerate of our hardware, this provides good matching performance. 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.

3.2 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).

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

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

$S(x,y)$ 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 (50 in our implementation), 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.

3.3 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. We ignore perspective effects and changes in appearance due to rotation in favor of pure 2D motion since we have limited processing speed. In practice, we have not found these assumptions to be problematic in terms of ease of use as the detected motion matches the user's motion.

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) on the captured images, not on the edge images. 15x15 sized features were chosen as this size is large enough to capture visually distinct regions and significant intra-frame physical motion on our test hardware. 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 that is faster to compute and 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.

3.4 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.

3.5 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) [Davis and Bobick 1997] 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 predefined 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:

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

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.

4 Mapping Motion to Interaction

In order to support intuitive and efficient user interaction, it is important to understand what kind of information is provided by the tracking algorithm, and what the limitations are given the output of the tracking algorithm. The most basic but potentially most important input that can be acquired from the tracking algorithm is the two dimensional movement of the mobile device on a plane parallel to the camera in 3D. With this type of data, the camera can be used as an input device to capture the device's movement in up/down, left/right directions, as well as its speed in each direction. Mobile camera-based input has restrictions, primarily due to limitations of mobile device hardware. Forward and backward motion cannot be detected with the current algorithm, so six degree of freedom movement is not supported. Forward/backward motion is possible to detect if the algorithm were extended, however this would increase computational demands and reduce the frame rate, impoverishing the user interaction.

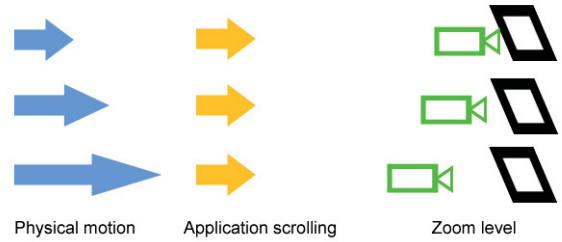

Figure 4: *For physical motion in a particular direction at different magnitudes, the direction and magnitude are decoupled. The scroll direction is purely direction based, while the zoom level is purely magnitude based.*

Physical movement speed is another challenge for camera-based interaction. The algorithm must perform all of its video analysis in the time between camera frames being captured to support real time interaction. As a result, there are implicit limits on the computational complexity of the tracking. In addition, there is a fundamental assumption in our algorithm that each frame contains some portion of the prior frame. This assumption is motivated by the observation that users will typically not move their phones erratically when focused on a task. We have also verified our tracking algorithm in informal experiments and found that the algorithm works well in practice. Users usually operate mobile phones with one hand. Mobile phones can also be used anywhere in an office, school, public location, home, etc. Considering these use environments, there are certain interactions which are not appropriate.

Precise tasks: Precise motion is very difficult holding a mobile device with one hand. Interaction should not require operations like 'move the device 2.5cm up', or 'move the device 34 degrees from the horizontal line.' As a result, camera-based interaction will probably be most useful when navigating large amounts of data, or zoom level dependent data.

Large motion: This restriction is more serious in some environments, such as in crowded public locations. 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.

Extended and/or frequent interaction: Using single handed operation, interactions that require extended time and/or frequent movement may fatigue users.

Our approach works best with coarse selections at different speeds and scales of data. It is critical that visual feedback follows physical motion and that the feedback differs according to motion speed, in order to provide an intuitive user experience. The most typical use case is moving the device to scroll UI content such as a list or a document. In the following applications section, we discuss some example applications using camera-based input.

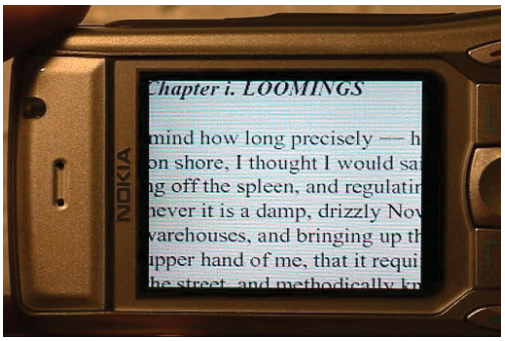

Figure 5: *Camera-based interaction in a document viewing application. The user's physical motion maps directly to the document's scrolling. The magnitude of their movement could be used for alternate document views.*

5 Applications

We have implemented several test applications using the proposed approach to clarify its strengths and limitations. In general, any mobile application that requires scrolling and/or zooming could make use of our approach provided that an onboard camera is present. We implemented the tracking algorithm and applications in C++ using the Series 60 second edition feature pack 2 SDK. Our test platform was a Nokia 6630 mobile phone, which features an ARM 9 220mhz processor, 10 megabytes of RAM, 176x208 screen resolution, and a 1.3 megapixel camera capable of capturing frames at 15 fps.

5.1 Document Viewer

Scrolling a document is a common task on mobile devices. For example, web content, especially when originally designed for desktop computers, typically becomes vertically long due to the narrow screen width on mobile devices. In addition, scrolling with the joystick is difficult especially 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, the user can vertically scroll documents by moving the device up and down. 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 and is moving the device slowly, the zoom level will be very high for maximum readability. However, at some point the user 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 left again. After a carriage return, all tracked motion except movement to the right is ignored. Applications can selectively ignore tracked motions in such a manner to create a robust physically-based interaction. Figure 5 demonstrates this application.

Figure 6: *Picture browser application. In this prototype, a large collection of photographs is presented. The user can move their phone to scroll through the collection. If they slow down their movement, the application automatically adjusts the zoom level to help the user browse.*

5.2 Zoomable Photo Browser

As cameras become more widespread on mobile phones and storage size increases, managing photos becomes a more difficult task for the user. Challenges include how to view a large amount of information on a small screen and how to browse with limited input modalities. Current typical photo viewer applications show photo thumbnails as lists, grids, or 3D carousels (Figure 1). As 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 shows thumbnails of the user's photos in a grid layout. The user can scroll in four directions (up, down, left, right) 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 [Igarashi and Hinckley 2000]. Adaptive zooming based on the magnitude of the user's physical movement keeps the scroll speed virtually consistent, allowing the user to browse more thumbnails by only moving the device faster. Figure 6 demonstrates this application.

5.3 Vector graphics-based map

Scalable vector graphics are becoming more widespread on mobile devices since they are amenable for limited display sizes. Vector graphics are infinitely scalable with no loss in quality, in contrast to raster graphics. We modified an implementation of the open source vector graphics language SVG for mobile devices [W3C] to utilize

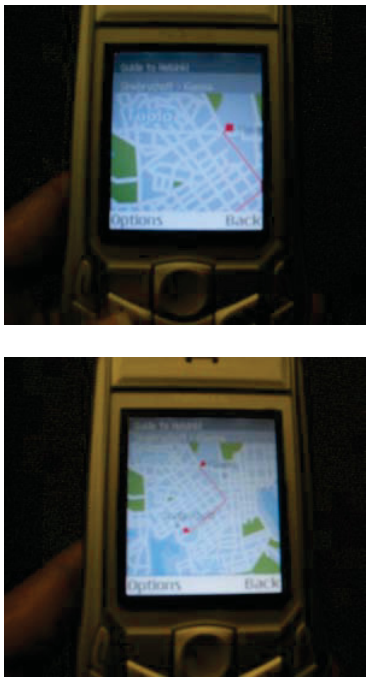

Figure 7: *Camera-based interaction in a map viewing application. The map is rendered using vector graphics, so it can be scaled with no loss in quality. Physically-based scrolling and zooming coupled with vector graphics creates a better navigation experience for users.*

camera-based input rather than keypad presses for scrolling. In particular, we experimented with SVG map content, since this would be one of the most common uses for it (Figure 7).

On a mobile device, a user would typically extensively pan and zoom a map while performing a navigation task. Panned and zoomed SVG content has no loss in quality due to its vector-based nature, but is difficult to navigate using only buttons and a joypad. We performed informal testing and users found zoomable SVG content often easier to navigate with camera-based panning and zooming. Further testing is required to determine how physically-based panning and zooming of content compares to traditional techniques. We believe camera-based interaction is better for getting to approximate map locations quicker, while joypads are better for fine-grained interactions such as selecting a particular street on the map.

6 Conclusion

We introduced a new approach to improve the user experience on mobile devices, specifically with mobile phones. A computer vision-based tracking algorithm was presented to detect both physical motion direction as well as 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, a document viewer, a photo collection browser, and a vector graphics based map. In the future, we would like to collect user feedback to determine how to improve user interaction further using mobile cameras.

While our tracking algorithm is very computationally efficient and works well in practice, there are some situations that 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 without either significantly more expensive computations or other sensors. Shadows may confuse the tracking system, but there are known computer vision techniques for robust tracking in the presence of shadows that will be incorporated into the tracking algorithm once additional processing speed is available.

References

AHLBERG, C., AND SHNEIDERMAN, B. 1994. The alphaslider: a compact and rapid selector. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 365–371.

AYATSUKA, Y., REKIMOTO, J., AND MATSUOKA, S. 1998. 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)*, 39–48.

BARTLETT, J. 2000. Rock 'n' scroll is here to stay. *IEEE Computer Graphics and Applications 20*, 3 (May/June), 40–45.

DAVIS, J., AND BOBICK, A. 1997. 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)*, 928–934.

HANNUKSELA, J., SANGI, P., AND HEIKKILA, J. 2005. A vision-based approach for controlling user interfaces of mobile devices. In *IEEE Workshop on Vision for Human-Computer Interaction*.

IGARASHI, T., AND HINCKLEY, K. 2000. Speed-dependent automatic zooming for browsing large documents. In *Proceedings of the ACM symposium on User interface software and technology (UIST)*, 139–148.

MASUI, T., KASHIWAGI, K., AND GEORGE R. BORDEN, I. 1995. Elastic graphical interfaces to precise data manipulation. In *CHI'95: Conference companion on Human factors in computing systems*, 143–144.

REKIMOTO, J. 1996. Tilting operations for small screen interfaces. In *Proceedings of the ACM symposium on User interface software and technology (UIST)*, 167–168.

ROHS, M. 2004. Real-world interaction with camera-phones. In *International Symposium on Ubiquitous Computing Systems*.

SIIO, I. 1998. Scroll display: Pointing device for palmtop computers. In *Asia Pacific Computer Human Interaction*, 243–248.

W3C. http://www.w3.org/TR/SVGMobile/. SVG Tiny Profile.

YEE, K.-P. 2003. Peephole displays: pen interaction on spatially aware handheld computers. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1–8.