

FAST CAMERA MOTION ESTIMATION FOR HAND-HELD DEVICES AND APPLICATIONS

Xu Liu¹, David Doermann, Huiping Li

Applied Media Analysis, Inc.

387 Technology Drive, Suite 3110, College Park, MD, 20742

{liuxu, doermann, huiping}@appliedmediaanalysis.com

ABSTRACT

In this paper we present an efficient motion estimation algorithm for camera-enabled handheld devices, such as mobile phones and PDAs. Compared to general camera motion estimation, the estimation of ego-motion of handheld devices presents unique challenges because most devices are limited in resources (processing power, memory and battery). The algorithm must be lightweight so it can be efficiently embedded and run fast enough to produce smooth seamless translation. Our solution includes a multi-resolution scheme which searches the match from coarse to fine; and optimization of the search space. As a demonstration, we implemented the algorithm on the Symbian based Nokia 3650/6600 camera phone, and explored several interesting applications, including using camera motion for browsing documents and as a pointing device. A cross-platform implementation is also discussed.

Keywords: camera motion, hand-held device, multi resolution

1. INTRODUCTION

Increasingly hand-held devices, such as smart phones and PDAs, provide imaging capabilities through built-in or attached cameras. Currently these cameras are used primarily to take pictures. As the processing power of handheld devices increase, more advanced applications of these cameras can be expected. In this paper we present an efficient algorithm for estimating the ego-motion of the device. Given the estimated motion, the device can be used as a “mouse” to control the cursor or make gestures as device input.

The challenge is that the estimation of motion must be efficient and sensitive enough so it can be embedded into devices with limited resources. Previous work on camera motion (also known as ego-motion) estimation has been presented in [1][2][15]. Most focus on omni-directional motion and are computationally expensive. A more efficient way to compute the ego-motion is to directly compute the planar motion from the image sequences captured by the device. Existing planar motion estimation algorithms can be categorized as feature-based [3, 4, 11] or region-based [12, 13]. Feature based approaches depend on the detection and matching of salient features such as corners and edges. While these feature-based

methods show promise for some applications, the computational complexity of feature extraction and matching (which usually employs a RANSAC [14] algorithm) is very high, and therefore, prohibitive. Based on these considerations, we use a region based motion estimation algorithm that has linear complexity in the size of the image. The complexity is further reduced by a hierarchical matching scheme.

The remainder of the paper is organized as follows: In Section 2 we describe our motion estimation algorithm in detail, and in Section 3 we briefly discuss an implementation suitable for cross platform use. Section 4 investigates the sensitivity of the algorithm, and some applications are addressed in Section 5.

2. MOTION ESTIMATION ALGORITHM

Generally, the imaging process of a camera can be modeled as a perspective mapping $M : \mathbb{Z}^3 \rightarrow \mathbb{Z}^2 :: X \rightarrow x$ that can be represented by a 3x4 matrix $[A | 0]$ where

$$A = \begin{pmatrix} f_{sx} & 0 & -i_0 \\ 0 & f_{sy} & -j_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (1)$$

A is defined as an internal parameter matrix, f is the focal length, (i_0, j_0) represents the coordinate of the optical center and (sx, sy) is the pixel size in image plane coordinates. Most omni-directional camera motion estimation approaches start from this model and estimate camera motion in 3D space from captured 2D image sequences. For our application, we simplify the model and directly compute the 2D motion of the camera from the captured image sequences rather than projecting the 3D motion to 2D. In our algorithm the motion is estimated by hierarchical matching in an optimized search space.

¹ Graduate Student at the University of Maryland, College Park on internship with Applied Media Analysis.

2.1 Hierarchical Matching

The idea of hierarchical matching originates from MPEG [6] encoding, which encodes the motion of a block from frame to frame to compress the video data. The situation for camera motion is different: the scene is fixed and the motion is caused by the camera. Consequently, blocks in a frame should generally move in the same direction, which we try to estimate.

We treat the motion estimation between frames X_0 and X_1 as an optimization problem, where the estimated motion \vec{m} is defined by (2):

$$\vec{m} = \arg \min_{(dx, dy)} \left(\sum_{(x, y)} |X_0(x, y) - X_1(x + dx, y + dy)| \right) \quad (2)$$

To estimate \vec{m} we populate a pyramid of images from coarse to fine and perform the estimation on each layer of the pyramid. The finer level uses the result from the coarser level as a rough estimate to limit the search space.



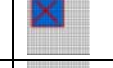

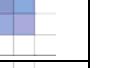
				
Original image	Layer4 243x243	Layer3 81x81	Layer2 9x9	Layer1 3x3

Figure 1- synthetic images at 5 resolutions

Figure 1 shows the hierarchical matching procedures using synthetic images. From the top level (Layer1) we estimate the motion to $(1,1)$ super pixels. When we go down to Layer 2, the motion is estimated to $(4,4)$. As we go down to the finer levels, we get a more accurate estimate of the motion based on an estimate of the previous layer. Equation (2) is applied to each layer. From a signal processing point of view, we first estimate the motion of the low frequency composition of the image and then estimate motion of the higher frequency composition. The motion of the higher frequency composition depends on the motion of lower composition. When we down sample the image, we convolve the image with a rectangular signal to weaken the high frequency components. In practice, we construct a hierarchical image sequence from the original image, each of which is a smoothed version of the previous one. The pyramid algorithm for MPEG [6] encoding is a mature implementation of this technique. However, for video encoding, the motion is focused on the content inside the image rather than on the motion of the camera itself. Here we merely use the casual multi grid, e.g. a low pass filter. An alternative is the algebraic multi grid (AMG). Sharon

[7] introduced AMG for image segmentation and proved its efficiency and accuracy, but the computational complexity is far beyond the requirements of today's hand-held devices. When the segmentation is done, motion estimation is no longer a problem.

2.2 Optimizing The Search Space

The main reason for estimating motion is to enable real time control of the interface. Speed is crucial to real time motion estimation because, when the frame rate is low, matching points may not exist on successive frames. When the device is moved, the motion must generally be continuous, sharp turns or rotation should be rare and the trace of the device should be smooth rather than zigzag. The movement of the camera can thus be regarded as a Markov random walk because the next movement relies only on the recent previous steps. A rough approximation of the device movement is a Markov random walk with a diagonal transition matrix, since the motion tends to repeat. This is the reason why a Kalman Filter can be applied for post processing to smooth the trajectory [16].

In our approach, the motion is estimated by hierarchical matching. This is essentially a search problem, so the speed is directly related to the dimension of the search space. We have included a branch cut mechanism in our algorithm: the current best match is used as the lower bound. When a motion direction is obviously wrong (worse than the lower bound) we will stop matching in this direction. The earlier we reach the best match the more wrong directions can be cut.

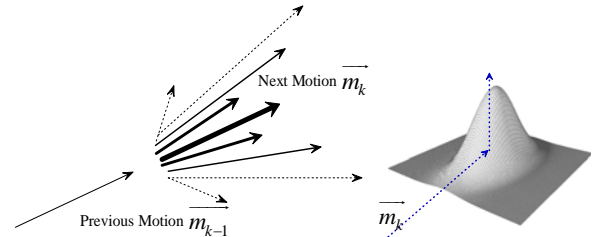


Figure 2 – Motion Predication
Left: next motion predicated by previous motion, bold indicating high possibility. Right: possibility of next motion modeled as a Gaussian distribution

The next motion vector is predicted by the Gaussian distribution centered at the previous motion vector (Figure 2). When we search for the next motion vector on the plane, we first sort the possible motions vectors by their distance from the previous motion vector. Those vectors near the previous motion are checked first because the motion vectors have higher possibility of remaining constant.

3. IMPLEMENTATION

Although the estimation requires a hierarchical sequence of images, we do not have to generate the image at each resolution, one by one. Instead, we generate all of them in one loop as follows:

1. Build the hierarchical image sequence
 - a) Clear the memory block which stores the image pyramid.
 - b) For each pixel $I(x, y)$ for each scale

$$\text{coefficient } t: I\left(t, \frac{x}{s^t}, \frac{y}{s^t}\right) = \sum I(x, y)$$

In our implementation we chose $s = 3$ and t vary from 1 to the scale that would not exceed the size of the image.

2. Estimate the motion from the coarsest to finest level
At each level we estimate the motion using equation (2) and pass the result $\vec{m} = (\overline{dx}, \overline{dy})$ to the finer level. Each time we search \vec{m} on the plane, we follow the order in a pre-computed look-up table which is indexed by the previous motion vector. The look-up table contains the sorted next motion vectors for every possible previous motion vector. Thus we avoid extra cost of computing a better guess of the next motion vector.

The algorithm is attempting to simulate the behavior of a human when matching two images. First it roughly overlaps two images then adjusts the image on top to find a perfect match. When the match is found, the motion is estimated.

Consequently the computational complexity consists of building the hierarchical image sequence plus the coarse-to-fine match. The former is $O(n^2)$ where n is the width of the image. The latter has a constant upper bound which can be omitted in practice. Thus it can perform fast enough on slow hand-held devices.

After simulating these algorithms on the desktop, one challenge is how to rapidly embed them and subsequent applications in different mobile devices. The range of development platforms are shown in Table 1. The

Table 1: The main-stream smart phone devices as well as the development platforms we are going to target in Phase II.

Device Type	Operation System	Development	Representative Device
PDA	Windows Mobile (Pocket PCs)	MS embedded visual tools	Dell Axim series
	Palm	Palm OS SDK/Developer Suite	Palm, Treo
Camera Phone	Symbian	EPOC/Symbian SDK S60/S90	Nokia 3650,6600,7610

software modules we will develop include the user-

interface, image acquisition and display, motion estimation, probably with a client/server module if the handheld is used as pointing device.

To achieve this we use a modular programming architecture so each of the modules is self-contained, and can be re-used, enhanced and modified for different platforms. The motion estimation algorithms are implemented in C or C++ first, and then transcoded into different embedded platforms. Most of the components will be identical, but some may be designed and optimized specifically to take advantage of device hardware. For example, for Symbian-based Nokia phones the processors are fixed-point. Therefore, we removed all floating point calculation so the best performance could be achieved.

4. PERFORMANCE

In the following sections we use the Nokia 3650/6600 camera phone as an example to examine the performance.

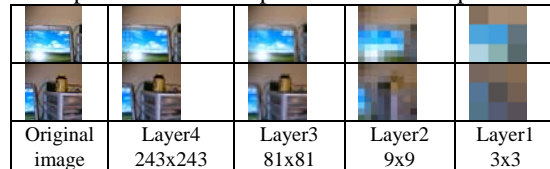


Figure 3- real image at 5 different resolutions

4.1 Speed

Speed is the key metric for evaluating real time processing, so we need to determine the level of which the real time performance can be achieved.

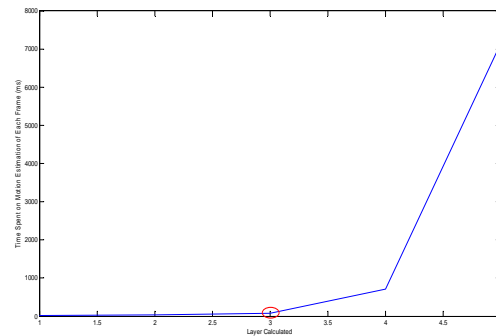


Figure 4- Speed- Layer Reached

Layer Calculated	1	2	3*	4	5
Time Spent on Estimation for Each Frame (ms)	19.5	38.3	81.3	714.3	7143

The test is on NOKIA 3650 Symbian Phone with ARM9 104MHz CPU

Figure 4 shows the speed for each layer processed. We can see that the time spent on motion estimation of each frame grows rapidly when motion estimation is conducted beyond layer 3. The reason is that the time complexity varies with the square of the image resolution. Since it is supposed to be a real time system, the speed becomes an important issue. If we spend too much time on each frame, the frame rate falls below a given value (typically 10 frames per second), image sequences captured may have little overlap, making the matching between two successive images impossible. Yet, when the resolution is too low (e.g. 3x3, 5x5 or 9x9), the estimated motion is no longer reliable because of the limited search space. When the estimated motion is not reliable, we observed the cursor jumps on the screen. As shown on the previous graph, Layer 3 is the ideal choice, and is stable in practice for maintaining both frame rate and accuracy.

4.2 Controllability

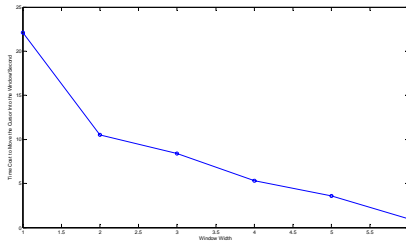


Figure 5—time spent on moving the cursor into the window

Window Size	1x1	2x2	3x3	4x4	5x5	6x6
Avg. Time (second)	22.1	10.5	8.4	5.3	3.6	1.0

The test is on NOKIA 3650 Symbian Phone with ARM9 104MHz CPU

To test the how stable the cursor is with the motion of the camera, we put a rectangular window in the center of the screen and ask three users to move the cursor into the window by moving the phone. **Figure 5** is the average time spent to move the cursor into the window as a function of the size of the window. We can see that when the width of the window is less than 4 (1x1 is actually a dot) the user spent considerably more time on the task. But when the window is large enough (>4x4) it becomes a relatively easy task, and an accuracy of 5x5 is enough for most of the cursor operations.

Another test we conducted was a drawing test. We asked the user to write “This is a Test” by moving the handheld (assume the handheld device is the pen!). The trace of the cursor controlled by the camera is displayed on the screen. Figure 6 shows an example. All these experiments indicate that the system can be easily controlled.

This is a Test

Figure 6-Drawing by Camera Controlled Cursor

5. APPLICATIONS

5.1 Optical Mouse

One problem with handheld devices is that they often lack ways to locate a position in the screen. The Pocket PC relies on a stylus and touch screen, but this is not yet available for most smart phones. Some phones have a “joystick” but it can be difficult to use. Our camera based cursor makes the handheld device itself as a “mouse” – Users can move handheld devices to locate the position they want to pinpoint.



Figure 7—Camera Controlled Cursor on Symbian Phone (Nokia 3650)

5.2 PC Control

The estimated motion can be sent to a laptop or desktop computer through a wireless communication channel such as Bluetooth. This allows the cursor on computers to be fully controlled by hand-held devices. This feature is useful, for example, when giving a presentation. The “Pebble” Project [8] [9] at Carnegie Mellon University established an excellent framework for interaction between hand-held devices and desktops. Presentations have been given using Pebble system. However, it has not yet utilized the internal camera on hand-held device to control the cursor, which would be a nice additional feature. The open source project Bemused [10] employed Symbian Phones to control a media player as well as Power Point on PC via Bluetooth. To verify this idea we did an experiment using a camera enabled Nokia phone to operate a desktop computer. We asked 3 users to

complete the following three tasks that are common in desktop computers:

- Task 1: open a text file through explorer;
 - Task 2: draw a horizontal line using PAINT;
 - Task 3: type A through E on the soft-keyboard;
- Here is the average time they spent on each task:

Task No.	Task1	Task2	Task3
Time Spent Using Camera Controlled Cursor	15.1s	2.1s	6.3s
Time Spent Using Regular Mouse on PC	3.3s	1.7s	4.2s

The test is on NOKIA 3650 Symbian Phone with Bluetooth Communication Channel

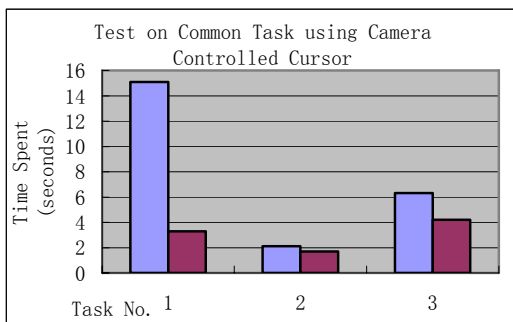


Figure 8—Comparison of “cell phone” and “real mouse”

From the experiment we can see that the handheld controlled mouse is competitive when dealing with subtle tasks. However, for the general task which always requires global movement of the cursor, the handheld controlled mouse takes longer since it is hard to maintain constant movement.

5.3 Document and Image Browser

Another shortcoming of the handheld devices is they have a relatively small display and therefore no convenient way for users to browse the large documents and images. The estimated camera motion can be used to control the scroll bar directly to browse images. We implemented a viewer on the Nokia Phone which can let users read documents which are significantly larger than the display size of the phone. As shown in Figure 9, a user can move the device up or down to update the page, and right or left to scroll the text. We used a double buffering technique to make the scrolling smooth.

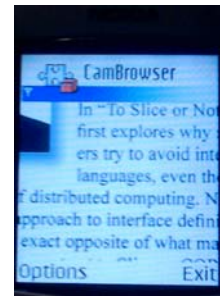


Figure 9— Document browsing based on camera moving (Nokia 6600)

6. CONCLUSION

In this paper we presented an efficient approach for motion estimation on hand-held devices with embedded or attached cameras. The approach is based on a coarse-to-fine image matching in optimized search space that significantly reduces the computational cost, and is suitable for mobile devices with limited resources. We have demonstrated numerous potential applications. With an integrated network capability (Bluetooth, for example), we successfully use a NOKIA camera phone as a controller for a desktop PC. We have also demonstrated that the estimated motion can be used to enable a “mouse” for the camera phone and for browsing large images and documents. We further tested the performance by measuring frame rates and sensitivity, and demonstrated the efficiency and accuracy in real environments with different users. We will conduct more extensive user studies and pursue more advanced applications in our future work as the optical sensor and processing power continue to improve.

7. REFERENCES

- [1] T. Tian, C. Tomasi, and D. Heeger, Comparison of approaches to egomotion computation. In CVPR96, pages 315–320, 1996
- [2] O. Shakernia, R. Vidal, and S. Sastry. Omnidirectional egomotion estimation from back-projection flow. 2003 CVPR Workshop - Volume 7 p. 82
- [3] N. Molton and M. Brady, “Practical structure and motion from stereo when motion is unconstrained,” International Journal of Computer Vision, vol. 39, pp. 5–23, August 2000.
- [4] A. Mallat, S. Lacroix, and L. Gallo, “Position estimation in outdoor environments using pixel tracking and stereo vision,” in Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 4, pp. 3519–3524, April 2000.

- [5] A. Fitzgibbon. Simultaneous linear estimation of multiple view geometry and lens distortion. In *Proc. CVPR*, volume 1, pages 125–132, 2001.
- [6] P. Burt and E.H. Adelson, “The Laplacian pyramid as compact image code.” *IEEE Trans. on Communication*; Vol. 31, No. 4, April 1983.
- [7] E. Sharon, A. Brandt, and R. Basri, “Fast multiscale image segmentation,” *CVPR*, 1:70–77, South Carolina, 2000.
- [8] Brad Myers, "The Pebbles Project: Using PCs and Hand-held Computers Together; Demonstration Extended Abstract." Adjunct Proceedings CHI'2000: Human Factors in Computing Systems. April 1-6, 2000. The Hague, The Netherlands. pp. 14-15.
- [9] Website <http://www.pebbles.hcii.cmu.edu/>
- [10] Website <http://bemused.sourceforge.net/>
- [11] Marco Grimm and Rolf-Rainer Grigat. Real-time hybrid pose estimation from vision and inertial data. Proceedings of the 1st IEEE Canadian Conference on Computer and Robot Vision (CRV), London, Canada, pages 480–486, May 2004
- [12] Ryan C. Jones, D. DeMenthon, and D. S. Doermann. Building mosaics from video using MPEG motion vectors. University of Maryland, College Park, Language and Media Processing Laboratory No. 035, 1999.
- [13] Jae-Choon Chon, Hyong-Suk Kim, Chung-Hyun Ahn, and Kyung-Ok Kim. The computation of the real-time camera motion through a correlation and a dynamic model of the system. 8th IEEE conference on Mechatronics and Machine Vision in practice, Hong Kong, 2001
- [14] R. M. A. Fischler and C. Bolles. Random sample consensus: A paradigm for model fitting with application to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981
- [15] Q. Ke and T. Kanade, Transforming Camera Geometry to A Virtual Downward-Looking Camera: Robust Ego-Motion Estimation and Ground-Layer Detection (*CVPR 2003*), June, 2003.
- [16] Hannuksela J, Sangi P & Heikkilä J , A vision-based approach for controlling user interfaces of mobile devices, *IEEE Workshop on Vision for Human-Computer Interaction (V4HCI)*, San Diego, CA, 2005