EUROGRAPHICS 2008 / G. Drettakis and R. Scopigno
(Guest Editors)

Volume 27 (2008), Number 2

Viewfinder Alignment

Andrew Adams[1], Natasha Gelfand[2], and Kari Pulli[2]

[1] Stanford University [2] Nokia Research Center Palo Alto

Abstract

The viewfinder of a digital camera has traditionally been used for one purpose: to display to the user a preview of what is seen through the camera's lens. High quality cameras are now available on devices such as mobile phones and PDAs, which provide a platform where the camera is a programmable device, enabling applications such as online computational photography, computer vision-based interactive gaming, and augmented reality. For such online applications, the camera viewfinder provides the user's main interaction with the environment. In this paper, we describe an algorithm for aligning successive viewfinder frames. First, an estimate of inter-frame translation is computed by aligning integral projections of edges in two images. The estimate is then refined to compute a full 2D similarity transformation by aligning point features. Our algorithm is robust to noise, never requires storing more than one viewfinder frame in memory, and runs at 30 frames per second on standard smartphone hardware. We use viewfinder alignment for panorama capture, low-light photography, and a camera-based game controller.

Categories and Subject Descriptors (according to ACM CCS): I.4.3 [Image Processing and Computer Vision]: Enhancement/Registration

1. Introduction

Motivation

Computational photography thus far has mostly been concerned with offline techniques. Data is acquired with a (possibly modified) camera and then post processed to produce a result. This in part is due to the computational requirements of many algorithms, and in part due to the closed nature of most camera platforms. High-end mobile phones, however, contain increasingly high-quality cameras with faster CPUs. They also have the benefit of being more open software development systems, allowing photographers write programs to directly control the behavior of the camera.

Mobile phones and similar programmable devices enable *online* computational photography. One aspect unique to online computational photography is the treatment of the viewfinder. Offline computational photography ignores the viewfinder, as it has no direct effect on the captured data, while for online computational photography the viewfinder is the photographer's main interaction with the data, and the photos themselves are secondary. We describe an algorithm for aligning successive viewfinder frames, and present three example applications of the technique. While we describe a specific algorithm and specific applications, our argument is more general: Viewfinder alignment can be implemented in real time on low-power devices, and it has high utility for a wide variety of applications.

Contributions

In a handheld camera, subsequent viewfinder frames are typically related by a small translation and a very small rotation. Our algorithm, described in Section 2, extracts edges and corners from a viewfinder frame to form a low-memory *digest*. It then aligns two digests by first estimating a translation from their edges, matching corners, and finally calculating rotation and translation that best aligns the corners. This algorithm is robust to noise, never requires more than one viewfinder frame in memory, and runs at 30 frames per second at 320×240 (QVGA) on a Nokia N95, which has a 330 MHz ARM11 CPU.

The first problem we apply this algorithm to is low-noise low-light viewfinding (described in Section 3.1). The standard approach to low-light situations is to increase analog gain a little, introducing noise, and to increase exposure time, which lowers the viewfinder frame rate. We instead increase the gain to maximum, maintain a short exposure time, and align and average subsequent frames to produce a motion-compensated temporal filter. This dramatically reduces noise, at the price of motion trails on moving objects.

Our second application is panorama capture. Standard panorama capture is much more time consuming than necessary, mostly because the photographer is never sure which areas of the desired view have been well covered. This results in either massively redundant sets of photos, or running the risk of an incomplete panorama. Panorama assistance programs currently implemented on cameras typically only help with simple 1D linear sets of photos, by presenting the right half of the previous image in the left half of the viewfinder.

We instead use viewfinder alignment to both display a map of captured areas, and to trigger the camera when the view is of an area not already captured. This allows the photographer to simply sweep the camera back and forth over the scene. The saved frames can then be used as input to a conventional panorama stitcher. Our capture method is described in detail in Section 3.2.

Finally, viewfinder alignment is also useful for non-photographic applications. It provides an analog input on devices that typically only have buttons. We describe using viewfinder alignment to control a video game in Section 3.3.

Prior Work

There is a vast amount of work on image alignment; a comprehensive review can be found in [Sze06]. The methods that have proven to be most successful are based on feature point matching using gradient-based descriptors [Low04]. These methods are accurate, able to deal with large inter-frame transformations, and are relatively robust to noise. However, they require expensive computations and therefore are too complicated to run at viewfinder frame rates on a low-power device.

An application closely related to our problem is software video stabilization, which requires very fast performance, but not pixel exact warps. For simply stabilizing a jerky video, only rough translations are required [Ova, HLL05]. More complicated alignment is usually done offline by post-processing the video sequence [MOTS05] to compute exact inter-frame alignment. Our algorithm exists somewhere along the continuum between high-quality slow image alignment and low-quality fast video stabilization. We desire pixel exact warps of whole images, albeit at a low resolution, and we need to compute them in real time on a memory and compute limited device.

The real-time requirement of viewfinder alignment makes it similar to the camera motion estimation problem in computer vision. We can think, therefore, of applying one of the well-known tracking methods [LF05] to the problem. However, the severe processing and memory constraints of our target device (discussed in detail in Section 2) rule out many of the existing tracking methods.

Much of the previous work on real-time tracking on camera phones or PDAs involves using markers [MLB04, HPG05]. Since the focus of our paper is on photographic applications, we do not wish to modify our scene, so we consider only markerless techniques.

Optical flow [BB95] analyzes how each pixel moves from one frame to the next. High-quality optical flow has been demonstrated at interactive rates [BW05] (6-7 fps on 160×120 images with a 3GHz machine), but we don't need a motion vector per pixel, only one for the whole image, and we target 30 fps on 320×240 images with a 0.3GHz machine, so we need an algorithm that is at least 200 times faster. Optical flow operates on entire images which need to be stored in memory if one wants to align several viewfinder frames; we would like to avoid storing those frames.

Multi-scale alignment algorithms (described in [Sze06]) use image pyramids to quickly compute global motion. Unfortunately, as was the case with optical flow, our target device is sufficiently limited that we cannot afford alignment algorithms that consider the entirety of the two images to be aligned.

Feature-based correspondence estimation methods operate only on small portion of each frame [TZ99], making it possible to store and align multiple frames at once. However, point features are often unstable in the presence of noise, requiring over-sampling and robust matching methods such as RANSAC [FB81], which, depending on the noise levels and problem dimensionality, may be too expensive for us. On the desktop impressive speed-ups have been obtained using GPUs [SFPG06]. However, current mobile graphics hardware only supports fixed function pipelines, which rules out most GPGPU techniques.

Area-matching methods operate by extracting a set of point features from an image and performing local motion estimation on patches centered on these feature points [ST94]. A patch-based tracking method running on a mobile phone has been implemented in [HSH07]. The disadvantage of patch-based methods is their requirement to perform pixel-wise comparison operations. This limits how many features can be tracked on a limited device. In [HSH07] only 16 patches were tracked to achieve the tracking rate of 10fps.

A compromise between matching individual points, which requires little memory, and tracking patches, which is more robust, is to perform edge-based tracking [Har92]. Tracking edges in the viewfinder frames is the basis of our method. We make edge-based tracking even more robust to noise by integrating edges along several directions, and more accurate by aligning a few well-chosen point features.

Our applications relate to recent work in computational photography, and indeed our intended audience is photographers who think they should be able to program their cameras. We will discuss the prior work relevant to noise reduction, panorama capture, and tracking in their corresponding sections.

Our argument is not for the specifics of our algorithm or applications, but rather that viewfinder alignment can be done with low computational cost, can run in real time on hardware typical of mobile devices, and enables a wonderful variety of novel ways to use your camera.

2. Alignment

Requirements

Previous markerless camera trackers [CHSW06, WZC06, HSH07] typically have managed to track roughly 160×120 images at about 10-12 frames per second. However, we wish to accurately align noisy viewfinder frames at 30 frames per second, at 320×240 resolution, on a Nokia N95 mobile phone, which has a 330MHz ARM11 CPU, and typically around 10 MB of free RAM. This introduces some very strict performance requirements on the algorithm. The timing requirement only affords us 150 cycles per pixel per frame, and the space restriction allows us to store about 30 viewfinder frames at once. We can't even make full use of the available memory; with so few cycles to spare cache misses are disastrous.

Many interesting applications require more than a single alignment computation per frame. For example, we may wish to align to many recent frames to increase confidence or reduce error. We may also wish to attempt to align to older frames to make any tracking robust to temporary occluders or sudden lighting changes. With this in mind, we would like to be able to compute alignments to about 10 other frames, thirty times a second. This immediately rules out methods that consider every pixel of the images to be aligned such as optical flow [BB95], as we would only have time for 15 instructions per pixel per frame. The same holds for patch-based methods such as [HSH07], and hierarchical methods described in [Sze06]. Instead we must extract some kind of low-memory feature set, or *digest*, from each frame, and use those to compute an interframe alignment. Given the memory limit, we would like to be able to forget frames and store only their digests, so the alignment must work without reference to the original images. Finally, there are cases where the tracker will fail, for example if the photographer moves the camera very quickly, or covers the lens. Our algorithm should provide a confidence value, in order to detect and handle such failures.

Fortunately, the requirement to run in real time places strong limits on the amount of motion expected between two frames. Consider a photographer turning on the spot through 90 degrees. If she takes at least one second to do this, then there is a three degree translation between subsequent frames. This represents about a twentieth of the camera's 50+ degree field of view. The photographer may also rotate the camera about its optical axis. If she rotates camera 90 degrees in three seconds, switching from landscape to portrait mode, then the rotation is one degree per frame. Unintentional rotation from hand shake will typically be much less, as hand shake produces primarily translation [FSH*06].

We assume that the scene is far enough away that parallax is insignificant, and that the difference between frames is small enough that we can model frame to frame warps as a similarity transform (rotation, scale, and translation), ignoring any perspective effects. Rotating the camera around its vertical or horizontal axis will cause a translation of the viewfinder frame, while rotating the camera around its optical axis will cause a rotation.

The Algorithm

We now need two algorithms: a method for producing a digest from an image, and a method for aligning two digests. We expect primarily translation, so the digest need not be invariant to large rotations or more general perspective warps, however the digest must be resistant to sensor noise. Our digest has two parts, as illustrated in Figure 1: four arrays recording edges and a set of point features.

To align two digests, we first align the image edges using the edge arrays, which gives us a translation between the two images. We then look for strong corners from one image that the translation places very near to a strong corner from the other. To reduce the likelihood of spurious matches we only use the k strongest corners (currently we use $k = 32$). Each of these point pairs forms a correspondence, which are used to compute a least-squares rigid transform. If the original translation was incorrect, we're unlikely to find more than one or two correspondences, so the number of correspondences acts as a confidence value. The rest of this section describes in more detail the extraction and alignment of our edge and point features.

Edge Detection. To extract edges we take the squared gradient of the image in four equally spaced directions: horizontal, vertical, and the two diagonal directions. We then perform an integral projection of each gradient image in the direction perpendicular to the direction of the gradient. This is a low angular resolution edge-detecting transform, which integrates the energy of edges into a few entries in one of the four output arrays. Edges that align with one of the four projection directions produce a sharp spike (e.g., the edges of the dog's leftmost ear in Figure 1), while edges that lie in between projection directions produce a broader hump (e.g., the edges of the dog's rightmost ear). Four directions are usually enough to record an edge in any direction as some kind of hump in one of the arrays.

These projections are fairly invariant under noise because they average many gradient values, and they vary only slowly under rotation. They transform in the desired fashion

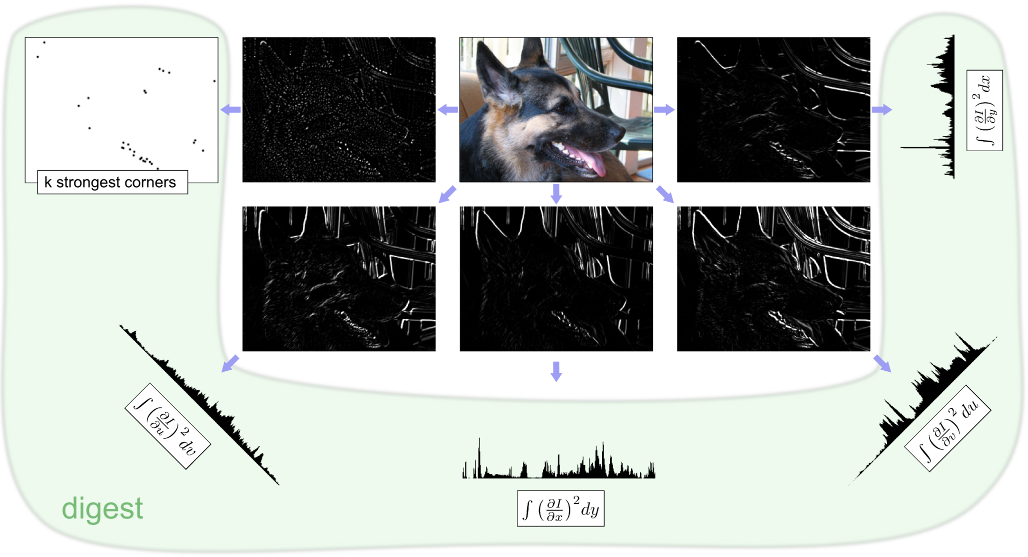

k strongest corners
$\int \left(\frac{\partial I}{\partial y}\right)^2 dx$
$\int \left(\frac{\partial I}{\partial u}\right)^2 dv$
$\int \left(\frac{\partial I}{\partial x}\right)^2 dy$
$\int \left(\frac{\partial I}{\partial v}\right)^2 du$
digest

Figure 1: *A digest is produced from a viewfinder frame. Gradients are taken in four different directions, and then projected in the respective perpendicular directions. The resulting arrays form a kind of Hough transform, which records image edges as spikes. Secondly, a corner detection filter is used to extract the* k *strongest corners. The resulting digest takes very little memory, and can be very quickly compared against another digest, without reference to the original images, to obtain a warp.*

under translation. For example, under horizontal translation the projection in the vertical direction translates, while the projection in the horizontal direction changes slowly as gradients enter and leave the frame. Under this translation, the two other projections would exhibit a combination of the two effects.

Edge Alignment. Alignment between two sets of projections is simple to compute. Independently for each direction, we do a brute force search over the range of expected translations (± 40 pixels) for the translation that maximizes the correlation of the projections. ± 40 pixels on a 320×240 image is sufficient to catch all motions slow enough that the whole image is not motion blurred beyond recognition. Multigrid methods [Ter86] could be applied here to accelerate the 1D alignment, but as this stage takes only about 5% of the total processing time, the potential for speed-up is small.

We choose the translation that minimizes a weighted sum of absolute differences. This gives us a proposed shift in each of the four directions, which we denote x, y, u, and v, where $u = \frac{x+y}{2}$ (the positive diagonal) and $v = \frac{x-y}{2}$ (the negative diagonal). The shift in x paired with the shift in y gives us a 2D translation. The shift in u paired with the shift in v gives us another 2D translation. We use the average of these two translations.

Alignment of the arrays in the diagonal directions is complicated by the fact that a different number of pixels were summed up at each array entry. We accommodate for this by accumulating a homogeneous value at each array entry. Formally, given an image I of size $w \times h$, our projections are given by:

$$\begin{bmatrix} p_x[i]_0 \\ p_x[i]_1 \end{bmatrix} = \sum_x \begin{bmatrix} (I(x,i) - I(x,i-1))^2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} p_y[i]_0 \\ p_y[i]_1 \end{bmatrix} = \sum_y \begin{bmatrix} (I(i,y) - I(i-1,y))^2 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} p_u[i]_0 \\ p_u[i]_1 \end{bmatrix} = \sum_{x,y} \begin{bmatrix} (I(x,y) - I(x-1,y-1))^2 \\ 1 \end{bmatrix} \quad \textit{where } \left\lfloor \tfrac{x+y}{2} \right\rfloor = i$$

$$\begin{bmatrix} p_v[i]_0 \\ p_v[i]_1 \end{bmatrix} = \sum_{x,y} \begin{bmatrix} (I(x,y) - I(x+1,y-1))^2 \\ 1 \end{bmatrix} \quad \textit{where } \left\lfloor \tfrac{x+h-y}{2} \right\rfloor = i$$

To compute the translation, we need to line up the corresponding arrays from two frames, p and q, and calculate the best shift for each projection direction: δ_x, δ_y, δ_u, and δ_v. One could naively compute the best shift in a particular direction d by dividing by the homogeneous coordinate and hence comparing the average square gradient values:

$$\delta_d = \arg\min_{\delta} \sum_i \left| \frac{p_d[i]_0}{p_d[i]_1} - \frac{q_d[i+\delta]_0}{q_d[i+\delta]_1} \right|$$

For the diagonal projections, this treats values at either end of the arrays (resulting from the average of a few gradient values) with equal importance to values in the middle of the arrays (which are the averages of a large number of gradient values). We would like to weight our distance by some function of the reliability of each value instead. We do this using a function of the number of samples taken, i.e., a function of $p_d[i]_1$ and $q_d[i+\delta]_1$. We weight by their product. This choice has the benefit of removing the need for a division, which is typically slow on mobile phone hardware. The new equation is as follows:

$$\delta_d = \arg\min_{\delta} \sum_i \left| p_d[i]_0 \cdot q_d[i+\delta]_1 - q_d[i+\delta]_0 \cdot p_d[i]_1 \right|$$

We treat values beyond the edges of each array as the zero vector. Unfortunately, this expression is then also minimized by very large values of δ, but this has little effect over our ±40 pixel search range. The ARM11 instruction set has a special instruction for calculations in the form of a 2×2 matrix determinant, so we are able to compute everything inside the absolute value signs in a single operation. This is one of the several design decisions directly influenced by the hardware, and a different choice could be made when adapting this algorithm to a different instruction set.

The shifts δ_u and δ_v combine to produce one estimated translation (t'_x, t'_y):

$$(t'_x,\ t'_y) = (\delta_u + \delta_v,\ \delta_u - \delta_v)$$

The shifts δ_x and δ_y act as another. We average the two estimates to produce the output translation (t_x, t_y) for this first phase of the algorithm.

$$(t_x,\ t_y) = ((\delta_x + t'_x)/2,\ (\delta_y + t'_y)/2)$$

This translation is accurate to within one pixel and extremely robust to noise, but does not take into account any rotation, and provides no measure of confidence. For these tasks we use the point features.

Corner Detection. Point features are extracted by searching for the k strongest local maxima of a corner detection filter applied over the image. Our corner detection filter is the minimum magnitude of the second derivatives in the four directions x, y, u, and v. An edge has a zero second derivative in at least one direction, while a corner or isolated point has a large second derivative filter response in any direction. Other corner detection filters, such as the determinant of the Hessian, or the minimum eigenvalue of the Hessian, are better known and perform well in general. However, for our application, this filter has the advantage that it can be calculated with comparisons, additions, and subtractions only. On the ARM11 processor, this can be parallelized with SIMD instructions that operate on each byte of a 32-bit word in parallel. This is another example of a hardware-specific design decision, and this corner detector could easily be swapped out for another.

Corner Alignment. To align two frames, A and B, we take the translation from A to B estimated from the projected gradients, and apply it to the corner features from frame A. For each corner from A, we find the nearest corner from B. If the two points are sufficiently close (within 3 pixels after translation), we count this as a correspondence. The number of correspondences is our confidence value. We found that in the case of an incorrect alignment of edges (for example if the frames are entirely unrelated), we typically get a confidence value of 0-3, while a correct alignment of edges typically gives a confidence above $\frac{k}{2}$. For the final refinement, we compute the best similarity transform (rotation, uniform scale, and translation) from A to B in the least-squares sense. With only four parameters, least-squares similarity transforms require O(1) memory to compute, and have a simple closed form solution [ELF97]. Note that the edge alignment stage only needs to provide accurate enough estimates so that the detected corners can be paired correctly, as aligning the corner features provides the final transformation sub-pixel accuracy.

This phase of our algorithm has two key parameters. Firstly, we call two points a correspondence if they are within 3 pixels. This choice is motivated by the amount of rotation we expect between two frames. If we expect at most one degree of rotation, and assume the translation was correctly estimated, then at 320×240 matching corners should land within $\lceil sin(1^\circ) * 160 \rceil = 3$ pixels of each other.

The other parameter of the algorithm is k, the number of corner features. We use $k = 32$. A large k will include increasingly poor corners and will increase the probability of false correspondences, while a small k may not provide enough correspondences to comfortably overconstrain the least-squares solution, and will not provide a clear distinction in confidence between correct and incorrect alignments.

Performance

To run at 30 frames per second, we have a time budget of about 33 ms per frame. On the N95 at 320×240 the digest construction (edge and corner extraction) takes about 20 ms. Alignment of two digests takes around 1 ms. This is precisely the type of timing we designed the algorithm to have. We never require more than one digest construction per incoming frame, so 20 ms is acceptable, and the time required to align two digests is low enough that we can perform many alignments per frame. The digest construction scales with the number of pixels per frame, while the alignment step scales with the width of each frame (and hence the length of the edge arrays), so operating the viewfinder at different resolutions changes the timing accordingly.

Testing Results

In order to test the limits of our alignment algorithm, we constructed a pair of motorized mounts. One spun the camera around its vertical axis, producing a horizontal translation in the viewfinder images. The second spun the camera around its optical axis, producing a rotation in the viewfinder

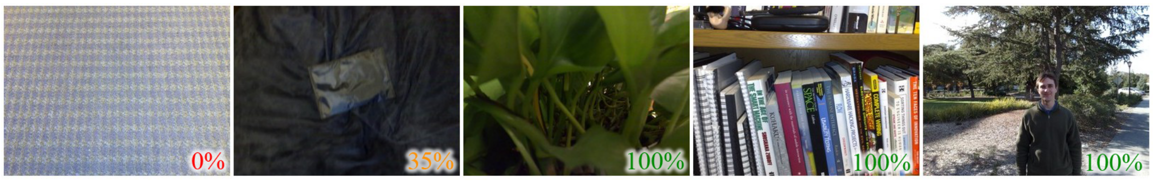


Figure 2: *The percentage of alignments that succeed for a variety of scenes when using the algorithm to combat hand-shake. From left to right we see: textured carpet, an off-angle black vinyl bag on a black suede background, the inside of a potted plant, a bookshelf, and a typical outdoor scene on a windy day.*

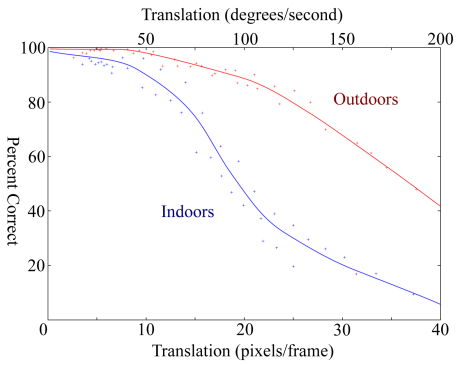


Figure 3: *The percentage of alignments that succeed when panning at different speeds. Motion blur is the limiting factor, as the indoor light-starved result (blue) fails with slower motions than the outdoor result (red).*

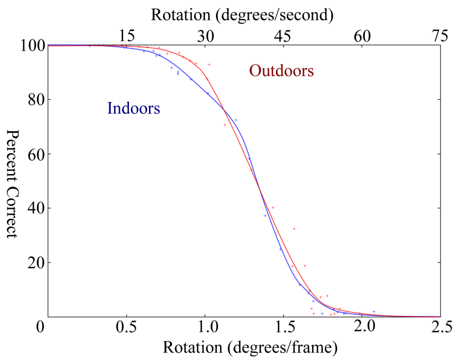


Figure 4: *The percentage of alignments that succeed when rotating the camera about its view direction at different speeds. The algorithm does not fail before its built-in limit of one degree per frame. Available light, and hence motion blur, is not the limiting factor in this case.*

images. We spun the camera at various speeds on the two mounts, attempting to align each frame to the next. We recorded what percentage of these alignments was successful, where success was defined as finding at least 10 correspondences between the frames and computing from them a plausible transform. In order to estimate the effect of motion blur and noise on the results, we performed the experiments under two lighting conditions: indoors in dim lighting, and outdoors on a bright sunny day. The indoor scene was a furnished living room, with lights off and curtains closed. The outdoor scene was a typical apartment complex courtyard.

The translation results (Figure 3) show that the algorithm copes well with rapid translations, and the large difference between the indoor results and the outdoor results suggests that the algorithm is more likely to fail due to motion blur than any inherent limitation. The rotation results (Figure 4) show that the algorithm starts to fail above one degree per frame, regardless of available light. This is expected; one degree per frame was the limit we used to select the algorithm parameters. The region between half a degree per frame and one degree per frame shows that we do gain some benefit from less noise and motion blur for moderate rotations.

Clearly, the alignment results depend on the content of the scene, in particular on the presence of strong edges. Figure 2 shows the performance of our method on a variety of test scenes. Typical indoor and outdoor scenes contain enough edge information for successful alignment, and performance degrades, as expected, on images which contain only soft edges.

Limitations and Future Work

The main two advantages of our alignment algorithm are that it is extremely fast, and extremely robust to noise. The main disadvantage is that it can't handle large warps from one frame to the next. Specifically, it does not tolerate translations greater than about 10% of the frame (especially if strong gradients enter or leave), rotations greater than about one and a half degrees, or the perspective effects that occur when panning with a wide angle lens. This disadvantage does not pose a serious problem when the algorithm is used for viewfinder alignment. If the photographer moves the camera quickly, the algorithm is more likely to fail due to motion blur, and indeed our only application that requires large movements (Section 3.2) is motion blur limited.

Given more processing power, there are a few ways the algorithm could be scaled up. First, the number of gradient projections can be scaled all the way up to a full Hough transform, which would allow detection of both translation and rotation based on edges. A lighter weight approach to also detect rotation would be to record a histogram of gradient directions as a fifth array.

Second, our corner features are locations only with no descriptors. We can't tell two points apart. Calculating a descriptor such as SIFT [Low04] would allow us to better distinguish which points really match each other, reducing the probability of false matches. Unfortunately, effective descriptors are expensive to compute.

Third, we need not compute a simple least-squares similarity transform. Our algorithm works in principle for any warp which is close to a translation. Point features could be used to compute a full homography, using RANSAC to reject outliers, which would be more robust than simple thresholding. The reason we use such a simple model for aligning the point features is both due to computational cost and because the extra degrees of freedom introduced by more complicated models are usually unneccessary.

Finally, one could attempt to break the motion blur limit on the algorithm by explicitly estimating blur kernels and using those as motion tracks. However, current methods of estimating blur kernels from a single image (such as [FSH*06]) are nowhere near real time.

Even with a growing amount of computing power there will be a need for light-weight alignment methods for a long time. Real-time alignment is usually used as a secondary mechanism for some other application, and should ideally use only a small fraction of CPU time. Additionally, mobile phones are battery-powered devices, and CPU and memory usage consume valuable power.

3. Applications

3.1. Low-Noise Low-Light Viewfinding

Running a viewfinder at 30 fps necessarily limits the exposure time of each frame to at most 1/30s. In dark environments, where one would normally use a flash to take a casual photo, this requires a large analog gain, and hence a very noisy viewfinder. Many cameras have a 'night' viewfinder mode, which doubles the exposure time and halves the frame rate. This reduces noise a little, but reduces responsiveness and increases motion blur from handshake. One could imagine increasing exposure time without sacrificing frame rate by maintaining a ring buffer of the last *n* frames, and displaying the average of them. However, this kind of temporal filter increases motion blur too much to be useful.

We instead propose using an *aligned* temporal filter, which computes an average of earlier frames *after alignment*. The advantage over regular viewfinding is dramatically reduced noise (see Figure 5). The only disadvantage is motion blur on objects moving within the scene (but not due to handshake or panning). If the alignment fails (for example due to sudden movement or overwhelming noise), then the viewfinder merely displays the most recent viewfinder frame, so our worst case scenario is the regular viewfinder.

Prior Work. Image and video denoising is a fundamental problem in image processing. An excellent survey of denoising methods can be found in [BCM05]. Video-specific denoising methods greatly benefit from combining information from adjacent frames to remove noise [DS84, JASS98] and to enhance dynamic range [BM05].

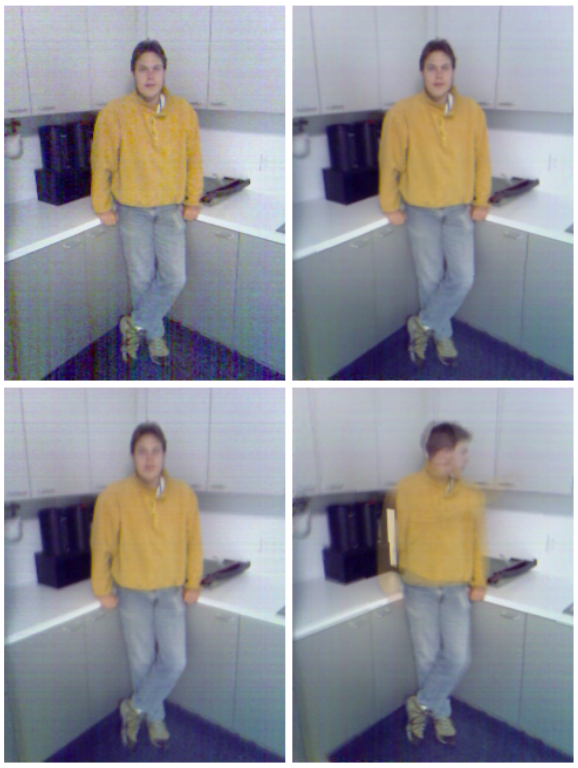


Figure 5: *The regular viewfinder (top left) suffers from noise in low-light situations. To reduce this, we take an aligned temporal filter of the previous few frames (top right). Without alignment, such a temporal filter introduces blur due to hand shake (bottom left). A downside of using this technique indiscriminately is that the filter causes moving subjects to exhibit motion blur (bottom right). If noise were preferred to motion blur, one could detect such motion with aligned differencing and selectively turn off the temporal filter at those pixels.*

Implementation. Our application maintains a single accumulation buffer. Every frame, the buffer is aligned and warped to match the incoming frame. The incoming frame is then multiplied by α and added to $(1-\alpha)$ times the accumulation buffer. This produces an aligned exponentially decaying temporal filter. Small values of α produce very low noise and much motion blur, while larger values produce less motion blur and more noise. The right value to use is just enough to remove visible noise. We use $\alpha = \frac{1}{8}$. In addition, we calibrate for fixed pattern noise by running the camera at the same settings with the lens cover closed and averaging a long sequence of frames. We subtract this average fixed pattern noise from each incoming frame, to ensure the aver-

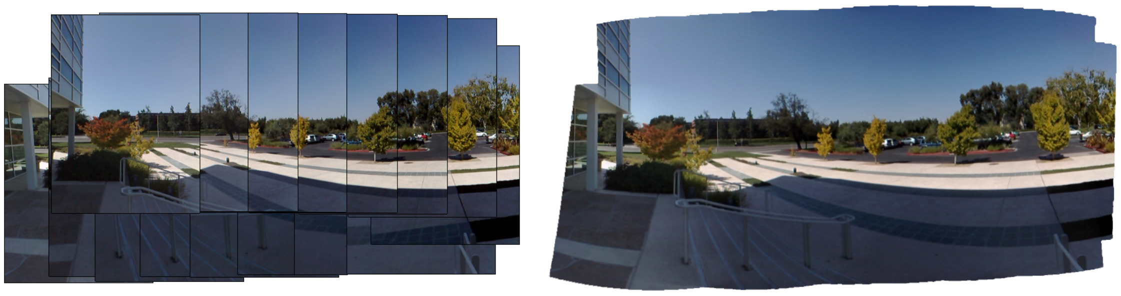


Figure 6: *The photographer sweeps the camera back and forth. On the left we see the frames automatically captured by the panorama assistance program, translated according to the translation estimates from viewfinder alignment. Note the accumulated global error especially visible in the posts of the stair railing. On the right, we see the result after stitching the frames in a conventional panorama stitcher [BL03]. These frames took less than 15 seconds to acquire.*

aging truly removes the noise, rather than just presenting us with the average noise image.

Warping the accumulation buffer to match the incoming frame leaves blank areas at the edges where no earlier information is available. In these areas we simply use the values from the incoming frame. When the photographer pans, the edge of the frame will be slightly noisier than the center, as fewer samples have been recorded there.

One unexpected benefit of the algorithm we observed was the removal of aliasing on image edges (caused by sensor subsampling). Hand shake corrected with alignment serves as a stochastic sampling mechanism.

Limitations and Future Work. The concept of an aligned temporal filter can lend itself to uses other than noise reduction. For example, differences between subsequent aligned frames could be computed to use as a trigger based on scene motion that ignores hand shake. Alternatively, if a photographer wishes to view a scene ignoring transient occluders, an online median of the last n aligned frames may be useful. This, for example, can be used to record a busy street as if it were deserted. Approximations to the online median could circumvent the otherwise heavy computational and memory requirements.

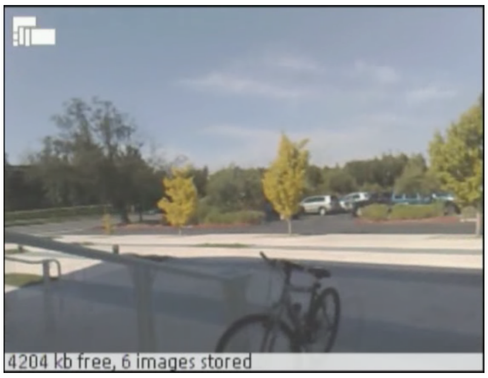
More robust statistics than the average may also be useful for noise reduction. For example, a bilateral filter in time (as in [BM05]), would reduce noise without causing motion blur. A real-time approximation could average the incoming frame with the accumulation buffer where the values are within some threshold distance, and use values from the incoming frame where they are not.

3.2. Panorama Capture

Taking the source images to make a panorama is slow and error prone. A photographer is likely to either miss part of the scene, or more commonly, waste time and memory by capturing a massively redundant set of photos. Some cameras include panorama assist modes, which show a ghost of the right half of the previously captured frame in the left half of the viewfinder. These modes are very helpful for 1D panoramas but not for more general 2D panoramas.

Viewfinder alignment can help by triggering the camera whenever it is pointed at a previously uncaptured part of the scene. In addition a map of previously captured areas can be presented to the photographer and displayed in a corner of the screen, as in Figure 8, to avoid common panorama capture mistakes like missing the corners of the scene. This alignment-based triggering lets the photographer simply pan over the scene to be captured, pressing no buttons at all, without having to worry about wasting memory or missing any part of the scene. It is a very fast and easy way to capture a panorama (see Figure 6).

Prior Work. There are many methods that deal with alignment and stitching of collections of images to create panoramas [Sze06]. Methods such as [BTS*06] can incrementally compute a panorama at near real-time speed on a PC, but are still too heavy duty to be able to run on our target device. In contrast, our method is fast since it does not seek to compute the exact panorama on the fly. Instead we only



4204 kb free, 6 images stored

Figure 8: *A screenshot from the panorama capture program. The top left shows a map of the relative locations of the 6 frames captured so far.*

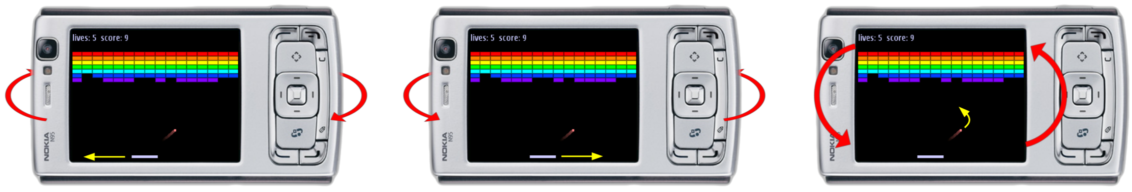


Figure 7: *Viewfinder alignment acts as a continuous input on devices which otherwise lack them. This figure illustrates how horizontal translation and camera rotation are used to control a game of Breakout. Horizontal translation moves the paddle left and right, while rotation steers the ball.*

keep track of approximate locations of all previously taken images using viewfinder alignment. By being conservative about the amount of displacement allowed between pairs of images, we can be sure that the captured frames will be able to be aligned into a panorama later.

Implementation. To compute a global position for the current viewfinder frame, we accumulate local frame to frame warps. We resist the accumulation of small local errors into a large global error by aligning each viewfinder frame to 5 previous viewfinder frames, or rather, their digests. Recall that our alignment algorithm was specifically designed to allow many alignments per frame in this way. We use the confidence values from each alignment to compute a weighted average global position for the new frame, based on the global positions of the previous frames, and the relative alignments from the new frame to the previous frames. If the computed global position is greater than some minimum distance from *any* previously saved frame, we save this frame to disk to use in our panorama.

Limitations and Future Work. Even using measures to resist global error accumulation, the final set of saved frames cannot be simply blended together after applying their respective warps. It is impossible to lay out all the frames in a panorama on a plane and have them align without first warping to a cylindrical or spherical projection surface. The set of saved frames, however, is ideal for stitching by a conventional panorama stitcher, and the computed warps could be used as good initial estimates of frame position.

It is possible, if the camera is moved rapidly, for all alignments to the 5 previous frames to fail. In this case the UI notifies the photographer, and she can return the camera to the most recently captured area to reacquire the tracking. In practice the speed of camera motion is limited by the desire to avoid motion blur in the captured frames, rather than by any limitation of the tracking.

This is an application where the camera decides for itself when to save a frame, so the work could be extended to select frames not only based on their location, but also their inherent quality. For example an application could reject blurry or poorly exposed frames. In applications where motion blur is unavoidable, such as bad lighting conditions, we can also instruct the user to hold the camera still when the algorithm detects that a new frame needs to be acquired. We can then use viewfinder alignment to determine when the user has complied with the request, and only then take a photo.

Digests take very little memory, so there is no reason we can't store hundreds of them at once. We chose to align to the most recent five, but one could imagine more sophisticated schemes that attempted to align to very old frames in an attempt to reduce global error. This task is a two-dimensional analogue to SLAM [TBF05]—we wish to simultaneously construct an environment map and localize the viewfinder within it. An algorithm along these lines would become more accurate at estimating camera pose the longer it ran, and could be very useful for augmented reality applications.

3.3. Camera Pose as a Continuous Input Device

Mobile phones typically do not have a continuous input device, such as a mouse. When they do it is usually a touch screen, which, while excellent for interacting with a user interface, has several disadvantages: it ties up one of the user's hands, obscures the display, and it either has low resolution or requires use of a stylus. Viewfinder alignment does not tie up an extra hand, leaves the display free, and has high resolution. In addition, it offers an extra degree of freedom (camera roll), and is unbounded. These properties make it excellent for use as a game controller, and as a proof of concept, we have implemented the well-known arcade game *Breakout*, as illustrated in Figure 7.

Prior Work. As more processing power becomes available, video games have started to include ideas from computer vision. A camera, known as the 'EyeToy' was sold as a peripheral for the PlayStation 2 to be used with games based on simple motion detection. The 'Eye of Judgment' for the PlayStation 3 uses more sophisticated vision techniques, recognizing cards in a physical card game, and augmenting them with graphics displayed on the screen. Motion-based controllers have also become more popular, with the PlayStation 3 using accelerometers in the controller, and the Nintendo Wii controller using a combination of accelerometers and vision.

Computer vision techniques have also been used for browsing of 3D models and maps on a small screen device [HPG05], as a game controller on mobile phones [Sie03, WZC06], for scrolling through large documents and interactive games on a camera phone [CHSW06], selecting menu items and input characters [WZC06], and even painting the strokes of input characters [HSH07].

Implementation. For use as an input device, we compute alignment between each frame and the most recent frame. The resulting similarity transform gives us three useful degrees of freedom: translation in x and y, and rotation. Scale is not useful in practice, because it either requires a close planar target, or a large back and forth motion. If the confidence value is low, we can either return a zero result, or we can return the last known good result. Returning a zero result works well, because it correctly freezes the cursor if the mobile phone is dropped, passed to another person, or put down on a table.

For our version of Breakout we use the estimated horizontal translation to move the paddle back and forth, and, in a literal twist to regular Breakout, we use rotation to affect the heading of the ball.

Limitations and Future Work. While viewfinder alignment allows for high accuracy control at low speeds, it can't handle large movements. It could be coupled with a device that provides a coarser estimate but can handle higher speeds, such as an accelerometer. Some high end mobile phones have accelerometers, and we expect them to become more widespread and more easily accessible to the programmer in the future.

Viewfinder alignment could also find use as an input device on dedicated cameras. Pointing with the camera is a natural way to specify things such as autofocus or light metering points.

4. Conclusion

In this paper, we have argued that viewfinder alignment is a useful technique for photography and general interaction with camera devices. We do this by presenting an alignment algorithm that runs in real time at full viewfinder resolution on a mobile phone camera, along with three sample applications of viewfinder alignment.

This paper is also a demonstration of the benefit in making camera devices programmable by third parties. As a camera, the mobile phone is merely adequate, but the advantage obtained from being able to directly program the device is significant. A programmable high quality consumer camera or DSLR could be instrumental in bringing computational photography to a broader audience.

Acknowledgements. We'd like to thank Marc Levoy for his many useful comments while writing this paper, as well as our canine collaborators: Belinda and Tony.

References

[BB95] BEAUCHEMIN S. S., BARRON J. L.: The Computation of Optical Flow. *ACM Computing Surveys 27*, 3 (1995), 433–466.

[BCM05] BUADES A., COLL B., MOREL J.: A Review of Image Denoising Algorithms, with a New One. *Multiscale Modeling and Simulation 4* (2005).

[BL03] BROWN M., LOWE D.: Recognizing Panoramas. In *Proc. ICCV* (2003), pp. 1218–1225.

[BM05] BENNETT E. P., MCMILLAN L.: Video Enhancement Using Per-Pixel Virtual Exposures. In *Proc. SIGGRAPH* (2005).

[BTS*06] BAUDISCH P., TAN D., STEEDLY D., RUDOLPH E., UYTTENDAELE M., PAL C., SZELISKI R.: An Exploration of User Interface Designs for Real-time Panoramic Photography. *Australian Journal of Information Systems 13*, 2 (2006).

[BW05] BRUHN A., WEICKERT J.: Towards Ultimate Motion Estimation: Combining Highest Accuracy with Real-Time Performance. In *Proc. ICCV* (2005).

[CHSW06] CAPIN T., HARO A., SETLUR V., WILKINSON S.: Camera-Based Virtual Environment Interaction of Mobile Devices. *21st International Symposium on Computer and Information Sciences (ISCIS)* (2006).

[DS84] DUBOIS E., SABRI S.: Noise Reduction in Image Sequences Using Motion-Compensated Temporal Filtering. *IEEE Transactions on Communication 32*, 7 (1984), 826–831.

[ELF97] EGGERT D., LORUSSO A., FISHER R.: Estimating 3-D rigid body transformations: A comparison of four major algorithms. *Machine Vision Applications 9* (1997), 272–290.

[FB81] FISCHLER M., BOLLES R.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. ACM 24* (1981), 381–395.

[FSH*06] FERGUS R., SINGH B., HERTZMANN A., ROWEIS S. T., FREEMAN W. T.: Removing Camera Shake from a Single Photograph. *ACM Trans. on Graphics (Proc. SIGGRAPH) 25*, 3 (2006), 787–794.

[Har92] HARRIS C.: *Tracking with Rigid Objects*. MIT Press, 1992.

[HLL05] HSU S.-C., LIANG S.-F., LIN C.-T.: A Robust Digital Image Stabilization Technique Based on Inverse Triangle Method and Background Detection. *IEEE Trans. on Consumer Electronics 51* (2005), 335–345.

[HPG05] HACHET M., POUDEROUX J., GUITTON P.: A Camera-Based Interface for Interaction with Mobile Handheld Computers. In *Proc. I3D* (2005).

[HSH07] HANNUKSELA J., SANGI P., HEIKKILÄ J.: Vision-based Motion Estimation for Interaction with Mobile Devices. *Computer Vision and Image Understanding 108* (2007), 188–195.

[JASS98] JOSTSCHULTE K., AMER A., SCHU M., SCHRÖDER H.: Perception Adaptive Temporal TV-Noise Reduction using Contour Preserving Prefilter Techniques. *IEEE Transactions on Consumer Electronics 44*, 3 (1998).

[LF05] LEPETIT V., FUA P.: Monocular Model-Based 3D Tracking of Rigid Objects: A Survey. *Foundations and Trends in Computer Graphics and Computer Vision 1*, 1 (2005), 1–89.

[Low04] LOWE D.: Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[MLB04] MÖHRING M., LESSIG C., BIMBER O.: Video See-Through AR on Consumer Cell-Phones. In *Int. Symp. on Mixed and Augmented Reality* (2004), pp. 252–253.

[MOTS05] MATSUSHITA Y., OFEK E., TANG X., SHUM H.-Y.: Full-frame Video Stabilization. In *Proc. CVPR* (2005), pp. 50–57.

[Ova] OVATION SYSTEMS: Stable eyes video stabilization system. `http://www.ovation.co.uk/Video-Stabilization.html`.

[SFPG06] SINHA S. N., FRAHM J.-M., POLLEFEYS M., GENC Y.: *GPU-Based Video Feature Tracking and Matching*. Tech. Rep. 06-012, UNC Chapel Hill, 2006.

[Sie03] SIEMENS: Mosquito Hunt Game. http://w4.siemens.de/en2/html/press/ newsdesk_archive/2003/foe03111.html, 2003.

[ST94] SHI J., TOMASI C.: Good Features to Track. In *Proc. CVPR* (1994), pp. 593–600.

[Sze06] SZELISKI R.: Image Alignment and Stitching: A Tutorial. *Foundations and Trends in Computer Graphics and Computer Vision 2*, 1 (2006), 1–104.

[TBF05] THRUN S., BURGARD W., FOX D.: *Probabilistic Robotics*. MIT Press, 2005.

[Ter86] TERZOPOULOS D.: Image Analysis using Multigrid Relaxation. *IEEE Trans. Pattern Analysis and Machine Intelligence 8*, 2 (1986), 129–139.

[TZ99] TORR P., ZISSERMAN A.: Feature Based Methods for Structure and Motion Estimation. *International Workshop on Vision Algorithms* (1999), 278–295.

[WZC06] WANG J., ZHAI S., CANNY J.: Camera Phone Based Motion Sensing: Interaction Techniques, Applications and Performance Study. In *Proc. UIST* (2006), pp. 101–110.