Hybrid Tracking using Gravity Aligned Edges

Samuel Williams
Computer Science
University of Canterbury
Christchurch, New Zealand

Richard Green
Computer Science
University of Canterbury
Christchurch, New Zealand

Mark Billinghurst
The HIT Lab NZ
University of Canterbury
Christchurch, New Zealand

ABSTRACT

We have developed a hybrid tracking algorithm for mobile outdoor augmented reality (AR) applications. Our approach combines inertial sensors and camera video to improve global bearing calculations. Prior research in this area has focused on gravity aware feature descriptors, but we expand this to efficient full-frame vertical edge detection. We discuss our implementation and evaluate it's performance on an iPhone 5, which reveals that our approach is over 100 times faster than existing feature alignment algorithms and can improve tracking with only 2-4ms of additional processing per frame on current generation mobile phones.

Categories and Subject Descriptors: H.5.1 [Multimedia Information Systems]: Augmented Reality; I.4.8 [Scene Analysis]: Tracking

1. INTRODUCTION

Users have authentic augmented reality (AR) experiences when visual cohesion between the real world and virtual content is maintained precisely and correctly. Tracking and registration errors, including drop outs, misalignment, failed initialisation, and drift, have such an effect on the experience that they make AR systems unusable or unsuitable for practical deployment. Thus there has been a large amount of research with focus on tracking and registration in the last two decades with ongoing work still required to improve accuracy and efficiency of these systems.

Outdoor AR encompasses the problem of visualising geographically registered data sets with a mobile device that includes a camera, a set of inertial sensors and some kind of video output. The most challenging part of outdoor AR is tracking the camera pose with respect to the geographical frame of reference. Modern mobile devices provide us with a wide range of sensors, typically including gyroscopes, accelerometers, magnetometers, global position (GPS) and cameras. Yet, in practice, consumer level hardware has insufficient accuracy for precise tracking.

CHINZ 14th Annual Conference of the New Zealand Chapter of the ACM Special Interest Group on Computer-Human Interaction

Ideally, applications should provide reliable and unrestricted end-user experiences, but this usually requires feature-specific algorithms that give the best possible results for a specific interaction, and often fail or work poorly in other situations. Tracking algorithms that primarily focus on computing a global frame of reference (e.g. WGS84) are not suitable for applications that require precise visual alignment. This is especially true for consumer level hardware, because a computed global frame of reference does not provide sufficiently accurate coordinates for visual alignment. High quality hybrid approaches are therefore critical to applications that want to provide end-user interactions that depend on both accurate local and global registration.

We present details on the implementation of a hybrid vertical edge tracking algorithm and explain how we used the gravity vector and gyroscope to optimise visual processing. We show how the algorithm can be implemented efficiently and used to improve the accuracy of the global bearing.

2. BACKGROUND

It has been identified that tracking is one of the most important aspects of AR applications[18]. When presented with multiple options, users tend to avoid AR visualisations, and visual jitter has been identified as one possible reason for this[24]. Panorama tracking[23] improves visual cohesion significantly, but requires a significant amount of memory and per-frame processing which makes it unsuitable for high resolution real time applications.

Using vision analysis to reduce errors in inertial sensor measurements is a proven technique[15]. Sensor data is relatively efficient to compute and use, but suffers from a variety of issues[20]. Relative inertial sensors, such as the gyroscope, may drift over time or provide incorrect measurements. Absolute global sensors such as the magnetometer and GPS may provide incorrect measurements due to local interference and usually have a high latency on consumer grade hardware. Image based methods are generally robust in such circumstances, but suffer in cases of visual occlusion, motion blur and computational expense. To resolve these issues, it is possible to exploit the complementary nature of these different inputs and combine them to produce a robust output[29].

The gravity vector provides a locally consistent frame of reference in which visual information can be processed. Gravity aligned feature descriptors have been shown to improve the reliability and performance of existing feature matching algorithms, and can be used to extract and track

gravity-rectified planar surfaces[8, 9]. However, while the gravity vector improves reliability of matching, it does not reduce the geometric efficiency of the feature extraction process.

Modern mobile platforms (including iOS and Android) provide at least a basic level of integrated sensor fusion which includes the gravity vector as an output. Tracking systems which depend on custom hardware or platforms usually implement custom low level sensor fusion algorithms[17]. By depending on the platform's native low level sensor fusion, we can improve performance; dedicated hardware which offloads low-level sensor fusion is already available in consumer level hardware and we expect this trend to continue.

3. FAST VERTICAL EDGE ALIGNMENT

We have developed a hybrid tracking algorithm, and a high level overview is shown in Figure 1. The vertical edge alignment algorithm is composed of the steps in the green boxes.

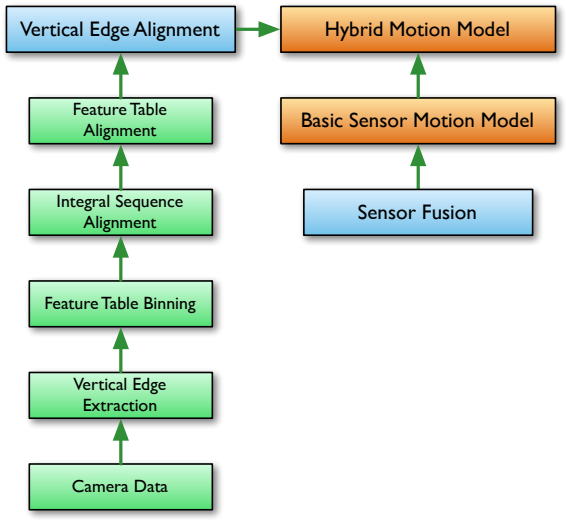

Figure 1. The structure of our proposed hybrid tracking algorithm.

Our proposed vertical edge alignment algorithm is designed to improve the accuracy of global alignment by combining sensor fusion with image processing to produce a robust bearing measurement. Inter-frame alignment issues are the most noticeable for users of outdoor AR applications[1] and a purely sensor based approach may suffer from jitter, latency and drift. Our implementation focuses on fixing these issues and has been carefully designed to be both efficient and scalable on current generation mobile hardware.

Our approach to visual alignment is conceptually different from many existing computer vision techniques that essentially depend on the image component as the ground truth[7, 3]. By relying *mostly* on sensor data, we minimise the amount of image processing required for visual alignment. Our approach is validated by our performance results; a high level performance comparison (in Section 3.2.3) of our proposed algorithm with the ORB[19]/Lucas-Kanade Optical Flow[12, 22, 21] implementation in OpenCV shows that we are almost 100 times faster with sufficient accuracy.

3.0.1 Vertical Edges

Users of outdoor AR are typically pointing their devices towards the horizon and in such video frames we expect a large number of vertical edges, especially in urban environments where there are buildings and other artificial structures. By tracking vertical lines rather than specific feature points, we can reduce the computational costs involved significantly. Vertical lines can be easily identified using a memory-efficient scan line based search, unlike feature extraction algorithms that must process large amounts of pixel data to extract good edges for tracking purposes.

In addition, vertical edges, parallel to gravity, are the best features to track when measuring translations and rotations perpendicular to gravity. For our proposed algorithm, a good vertical edge is one that has a large luminance gradient perpendicular to gravity and ideally is part of a long continuous edge parallel to gravity. This allows our algorithm to identify the edge easily over several frames.

3.1 Implementation

The vertical edge alignment algorithm computes the precise pixel offset between two video frames, such that vertical edges overlap as precisely as possible. As input, it requires two images, i_1, i_2, the gravity vector at the time each image was captured g_1, g_2, and the estimated translation between the two frames in pixels e, normally computed using the gyroscope. The output sub-pixel offset includes a confidence value which is the number of vertical features aligned correctly.

The resulting alignment can be combined with existing sensor fusion based estimates. In our implementation, we used a weighted combination of the sensor fusion input with the image alignment, but we mostly depend on the image alignment (usually more than a 0.95 weighting). In the case that the confidence of the match was low (e.g. less than 3-5 bins matched), we would defer completely to sensor fusion. This worked sufficiently well in our testing, because the image alignment process is typically more robust and as accurate than the fusion of the gyroscope and magnetometer, but in difficult cases, e.g. extreme motion blur, we would defer entirely back compass/gyroscope.

3.1.1 Gravity Vector

The gravity vector is sampled at a rate of 60Hz to 120Hz, to ensure that the vector is accurate enough for frames captured at 30Hz. For each frame, the last value for the gravity vector is used to compute the tilt angle, as this is the most accurate and up to date reading.

We have found that the accuracy of the gravity vector with respect to the image plane is excellent, typically on the order of $\pm 0.1^\circ$. In addition, for many typical outdoor AR applications, the user is unlikely to rotate the phone in a way which significantly affects the gravity vector. This makes it an ideal sensor for input to a computer vision algorithm.

The gravity vector can be computed by combining the accelerometer, gyroscope and magnetometer[17]. However, modern mobile platforms include the gravity vector as part of the inertial sensor data. This vector is usually very accurate and the sensor fusion computation may be accelerated using hardware (e.g. the iPhone M7 motion co-processor), so we use this vector directly rather than calculating it ourselves.

3.1.2 Tilt Calculation

The tilt is the rotation of the image frame such that gravity is aligned with the Y axis (see Figure 2). The tilt is only valid for gravity vectors that are not parallel to the camera axis (e.g. not looking directly up or down), and it's computation is dependant on the various coordinate systems of the device's hardware configuration.

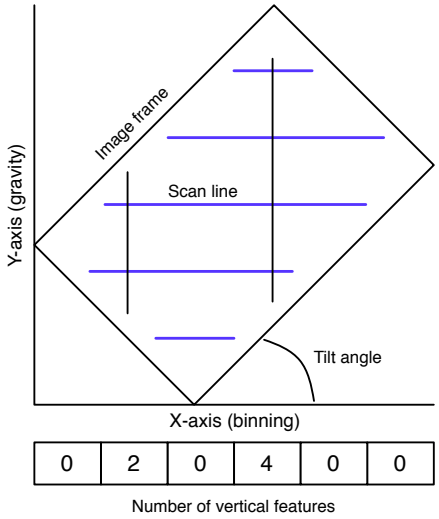

Figure 2. Vertical edges are extracted and binned relative to gravity.

We define a gravity local coordinate system such that gravity points down $-Z$ and $+X$ points right. This cylindrical mapping is the same as is used in other panoramic tracking algorithms[4] and is useful for most typical outdoor AR applications[1].

3.1.3 Scan Line Extraction

Using the tilt angle, we can compute a set of scan lines perpendicular to the gravity vector (the blue lines in figure 2). We compute a rotated bounding box for the image frame and use this to clip a set of horizontal scan lines. We rotate these scan lines back into image space to extract features. The distance between scan lines can be specified and this influences the number of feature points extracted and the amount of pixel data processed.

We use Bresenham's line drawing algorithm[2] to trace these scan lines efficiently (see Figure 3) and apply the Laplacian of Gaussian operator sequentially to pixels. The Laplacian operator approximates the 2^{nd} derivative and we extract the coordinates of the zero crossings with approximate sub-pixel precision using a fast mid-point calculation.

The Laplacian operator allows us to calculate the mid-points precisely even when edges are not perfectly visible. This allows for accurate key-point detection even when the image includes a large amount of motion blur and/or unfocused regions, which is common for cameras with relatively small sensors and contrast based focusing.

[1] Another option we have considered involves using globally registered scan lines radiating around the user, such that in our gravity local coordinate $Z = 0$ would be the horizon. This may increase the quality of tracking during vertical motion.

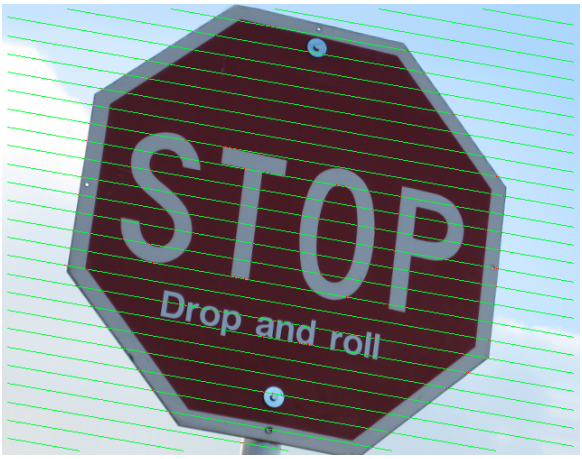

Figure 3. Scan lines overlaid on an image rotated by 10°.

3.1.4 Approximate Zero Crossings

Our original algorithm used only adjacent pixels to detect edges - if the difference between the two pixels was above a certain threshold, we would use it for tracking. Despite being very fast, we found this approach failed to generate many feature points in frames with motion blur.

To alleviate this problem, we implemented an efficient Laplacian of Gaussian (`LoG`) kernel to detect approximate 2^{nd} derivative zero crossings. We experimented with a number of kernel sizes and variations, but eventually found that $\texttt{LoG}_5$ gave good results both in terms of accuracy and performance:

$$\texttt{LoG}_5(I, x) = \begin{bmatrix} -1 \\ -1 \\ 4 \\ -1 \\ -1 \end{bmatrix} \cdot \begin{bmatrix} I(x-2) \\ I(x-1) \\ I(x) \\ I(x+1) \\ I(x+2) \end{bmatrix}$$

where I is an intensity function, e.g. a sequence of pixels, and x is the offset at which we are sampling.

In addition, we improved our feature point extraction to be approximately sub-pixel accurate by calculating the approximate zero crossing. Traditional curve fitting algorithms for finding the precise zero crossing are inefficient[6]. We propose a simple mid-point calculation which computes the sub-pixel zero-crossing with at most and error of 0.5 pixels, and does not require additional sampling or image processing. Due to the speed of our approach, simply increasing the resolution of the input video frames increases the accuracy of the gradient calculations; in effect, the smaller pixel width reduces the error in our approximation significantly, and is possible because to do so because our approximation can be implemented efficiently.

Given an intensity function for a scan line, $I(x)$, we evaluate the Laplacian of the Gaussian (`LoG`) kernel at x and $x+1$. This gives us a value for the approximate 2^{nd} derivative curve at x and $x+1$. If $I''(x)$ is positive and $I''(x+1)$ is negative (or the opposite), there is a zero-crossing point between those two samples[2]. We can compute the approximate

[2] We don't consider the case that $I''(x+1) = 0$ because in that situation the mid point can be computed precisely.

value for x such that $I''(x) = 0$ using a function *midpoint*:

$$\texttt{zerocrossing}(x_1, x_2) = x_1 + (x_2 - x_1) \times \texttt{midpoint}(x_1, x_2)$$

One way to approximate the zero crossing is to take the average value of x_1, x_2:

$$\texttt{midpoint}(a, b) = 0.5 \tag{1}$$

However, if the discrete sampling of I'' is not equally balanced around the zero-crossing, which we expect is the most common case, this formulation will produce incorrect results. By modelling the curve as a straight line between the sampled points $\langle x, I''(x)\rangle \to \langle x+1, I''(x+1)\rangle$, we can compute a more accurate approximation. The following computation calculates the point where this line crosses zero as a relative factor in the range $(0.0 \to 1.0)$:

$$\texttt{midpoint}(a, b) = \frac{-a}{b-a} \tag{2}$$

We used the 2013B data set[26], to evaluate the average midpoint function in equation 1 and the approximate midpoint function in equation 2. We ran the tests once for each algorithm.

Table 1. Midpoint Calculation Accuracy

	Error (°N)			
	Average		Approximate	
Data Set	**S.D.**	**S.E.**	**S.D.**	**S.E.**
2013B0	0.716274	0.102325	**0.647186**	0.0924552
2013B1	1.63702	0.187779	**1.35561**	0.155499
2013B2	1.31868	0.203477	**1.17316**	0.181022

The results in Table 1 show a consistent improvement of between $10 - 20\%$.

3.1.5 Feature Thresholding

Once a zero crossing has been detected, we calculate the local variance and compare it to a threshold. Because the `LoG` function is sensitive to noise, a considerable effort is required to filter good features. We assume that the zero crossing represents a pixel, typically on one side of a gradient, and so we look at the variance between the left side, the centre, and the right side.

$$\texttt{left(I, x)} = \left[\frac{I(x-2) + I(x-1)}{2} - I(x)\right]^2$$

$$\texttt{right(I, x)} = \left[\frac{I(x+2) + I(x+1)}{2} - I(x)\right]^2$$

$$\texttt{variance}(I, x) = left(I, x) + right(I, x)$$

This function tries to analyse the local structure of the gradient. It takes a small 2 pixel sample on the left of the zero crossing, and a small 2 pixel sample on the right of the zero crossing. It then compares these samples to the value at the zero crossing and looks at the difference. For a smooth gradient, we'd expect a similar value for both left and right, but for a step gradient, one side will be big and the other side will be small. This function should compute the same value for the same step size whether or not it is blurred or sharp - in practice it appears to do a good job as seen in Figure 4.

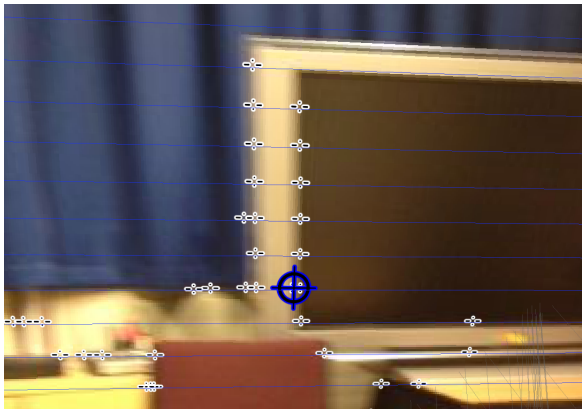

Figure 4. Features are still detected with good accuracy despite significant motion blur.

We tried a number of different variance functions, and found that the sum of squared difference gave the best accuracy and good performance. It requires no additional sampling of the input image as we use the greyscale values computed for the Laplacian. The amount of noise is tolerable in practice.

3.1.6 Feature Alignment

Our initial feature table implementation tried to extract and match vertical lines efficiently. The green edges in Figure 5 show the structure that could be extracted from the feature points. However, after analysing a number of data sets, we decided against this approach. Extracting complete edges robustly across different lighting conditions and motion blur is difficult. Missing feature points could break connected edges incorrectly, and matching up edges in these conditions would require a complex heuristic.

Rather than trying to discretely track connected vertical edges, we implemented a statistical model for alignment. The proposed algorithm considers all identified vertical features, and effectively computes a histogram (an example is given in Figure 5) of this data perpendicular to the gravity vector. By aligning the histograms, we compute the global alignment.

3.1.7 Feature Table Binning

After extracting the vertical edges, we distribute them into a series of bins (referred to as a feature table), where each bin covers a fixed portion of the X axis in the gravity local coordinate system, as shown in Figure 2. Sequential items in a single bin often represent connected vertical lines in the source image.

The size of the bins is flexible; by increasing the bin width, we reduce the total number of bins. This can improve efficiency at the cost of precision. However, if the bin width is too small, visual noise could become problem. In practice, we've identified that a width of 1px or 2px is ideal.

In order to reduce aliasing issues, we ensure that all feature tables have an even number of bins. Different tilt rotations may require a different number of bins, depending on the width of the rotated bounding box. Feature tables with an even number of bins will always align such that the centre of rotation falls precisely between two bins, so our proposed algorithm ensures this quality for all feature tables.

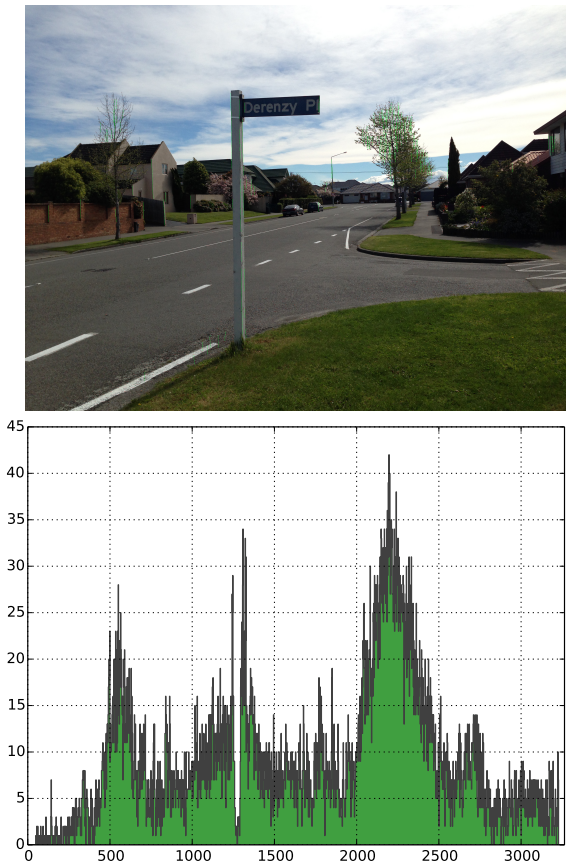

Figure 5. A histogram of vertical edges in 2px wide bins. The three main peaks from left to right correspond to the house, the road sign, and the tree. Approximate vertical lines overlaid in green on the source image. Individual red pixels mark precise feature points.

3.1.8 Table Alignment

We can align two feature tables with bin-level accuracy using the Fast Integral Sequence Alignment algorithm, as discussed in Section 4. We use the number of vertical edges in each bin to compute an integral sequence u and v for each feature table. To compute the estimate bin offset, we compute the relative rotation between two frames using the gyroscope and convert this into a bin offset estimate e.

Once we have calculated the bin alignment, $\texttt{FISA}(u, v, e)$, we compute the precise sub-pixel alignment of individual vertical features. For corresponding bins in each feature table, we compute average X position in gravity local coordinates. The difference between the average vertical feature position is computed for all corresponding bins, and used to compute an average displacement in pixels. This displacement is then converted back into a rotation and used as input into the sensor fusion computation.

An alternative approach is to compare the bins in the feature table sequentially. As the feature points are ordered vertically along the Y axis (i.e. gravity), we can compute a precise correspondence between features with similar Y value efficiently. We hoped that this approach would be more tolerant to noise, but instead found that in practice it was not a systematic improvement.

When computing the relationship between two bins, ideally we would like the bins to have a similar and significant number of feature points. This helps to reduce noise. Our proposed algorithm therefore only aligns individual bins that have a fixed minimum number of vertical features.

In practice, this isolates noise from sequential vertical lines, but in difficult tracking situations (e.g. significant motion blur), it might be the best match available, so fine tuning is required. Our confidence of a good match is therefore directly related to the number of bins we could use to compute the alignment. If we have less than a moderate number of corresponding bins, our hybrid tracking algorithm reverts back to a purely sensor based approach.

3.2 Evaluation

We evaluate our algorithm on an iPhone 5 running iOS 7, using the stop sign image from the metaio data set[10], as shown in Figure 3. We tested a number of different scenarios to examine the performance of the proposed algorithm and its tolerance to erroneous input.

We also took this opportunity to compare our approach with an existing feature extraction and alignment algorithm to assess the viability of our implementation.

3.2.1 Rotational Alignment

We modified the sample image so that it could be systematically rotated, and generated rotations in 1° increments from −20° to +20° using a script. We used the 0° rotation as the basis and calculated the offset of every other rotation using our proposed algorithm. As the source images only have rotations applied, the translation is expected to be 0 in all cases. As such, the translation estimate supplied to the vertical feature alignment algorithm was set to 0 for these tests.

We varied the distance between scan lines, which directly affects the number of feature points detected, to look at the effect on performance and accuracy. The scan lines are distributed vertically every dy pixels. We give the time for feature extraction and alignment processing separately, and the alignment error for all images from −20° to +20°.

Table 2. Rotation Performance and Accuracy.

			Output Error (px)		
dy	**Features**	**Alignment**	**Mean**	**S.D.**	**S.E.**
5px	12.13ms	303.9µs	-0.0026	0.019	0.003
10px	6.29ms	143.3µs	-0.0045	0.028	0.0044
15px	4.24ms	109.9µs	-0.0082	0.04	0.0062
20px	3.27ms	119.5µs	-0.023	0.046	0.0071
25px	2.57ms	125.8µs	-0.0099	0.066	0.01
30px	2.08ms	126.5µs	0.16	0.64	0.099

The results in Table 2 show that there is a good balance between performance and accuracy when $10 <= dy <= 20$. The number of samples that matched up is dependant on the source image, but we want to ensure a reasonable number of good quality matches. As dy is increased, the accuracy is diminished, as expected.

3.2.2 Rotation Noise

Under normal circumstances it would be typical to have a small amount of error in the measured rotation from the

gyroscope. We simulate this by adding gaussian noise to the tilt estimate and passing this incorrect data into the alignment algorithm. We fix $dy = 10$ and the bin size is $2px$ for this evaluation, and use the same data set as described in Section 3.2.1.

Table 3. Rotation with noise.

Input Error	Output Error (px)			
S.D.	**Samples**	**Mean**	**S.D.**	**S.E.**
0.0°	49.2	-0.0049	0.045	0.00071
0.2°	48.3	0.0025	0.044	0.00069
0.4°	46.6	0.013	0.12	0.0019
0.6°	44.8	0.012	0.29	0.0045
0.8°	43.4	-0.032	0.47	0.0073
1.0°	42.3	-0.077	0.58	0.0091

From the results in Table 3, we can see that despite significant error in the tilt, the standard deviation is about half a pixel of error. This is fairly reasonable, as the alignment of the feature tables will become progressively worse, proportional to the amount of error in the tilt angle. We also note that the number of matching samples was reduced too, indicating a reduced confidence of a good match, which is also expected.

3.2.3 Translation Comparison

We compared our proposed algorithm with an alignment algorithm implemented using optical flow. We were primarily interested in the efficiency of our approach in comparison to existing feature point extraction and correspondence algorithms. We generated a set of test images with fixed X offsets from -20px to +20px in 10px increments, and processed them using both implementations.

Table 4. Image Alignment Performance Comparison on iPhone 5.

	ORB/LK Optical Flow		Proposed Implementation	
X	**Features**	**Alignment**	**Features**	**Alignment**
-20px	112ms	309ms	3.84ms	55.0µs
-10px	107ms	318ms	3.62ms	54.2µs
0px	112ms	310ms	3.60ms	55.1µs
10px	107ms	314ms	3.77ms	54.6µs
20px	108ms	322ms	3.60ms	53.9µs
	Error: $\pm 0.00005px$		Error: $\pm 0.05px$	

The results in Table 4 show that our proposed algorithm is significantly faster in practice than existing feature extraction and optical flow alignment algorithms. In particular, ORB is considered to be a reasonably fast feature extraction algorithm, and yet it performed relatively poorly in comparison. Lukas-Kanade optical flow is generally considered a robust and efficient method for calculating the relative motion of feature points. Our results confirmed that it produces highly accurate alignment, but at a significant cost. In addition, while this implementation of optical flow has a fixed window size of 21px, our algorithm has no such limitation, provided the estimate is reasonable.

3.2.4 Translation Noise

We also look at how a poor estimate affects the quality of the computed alignment. This is particularly important as some of the biggest alignment issues are caused by errors in the gyroscope. The estimate is used directly as an input into the `FISA` algorithm and thus it is critical that we find the correct offset even if the input estimate contains significant error. We fix $dy = 10$ and the bin size is $2px$ for this evaluation, and use the same data set as described in Section 3.2.3.

Table 5. Translation with noise.

Input Error	Output Error (px)			
S.D.	**Samples**	**Mean**	**S.D.**	**S.E.**
0px	58.1	0.32	0.36	0.012
5px	58.1	0.32	0.36	0.012
10px	58.1	0.32	0.36	0.012
15px	58.1	0.32	0.36	0.012
20px	58.1	0.32	0.36	0.012

From the results in Table 5, we can see that despite significant error in the estimated translation, the alignment is still computed accurately. The results are identical each time which indicates that all features were matched in the same way.

4. FAST INTEGRAL SEQUENCE ALIGNMENT

Finding the correspondence between two sequences of numbers can help us align images. We looked at statistical methods for computing the correlations between two sequences[14], and in particular, how this has been applied to computer vision problems in the past[11]. As a result, we developed a correlation function with a formulation that allows for efficient implementation, and we show how it performs directly on the iPhone 5.

The Fast Integral Sequence Alignment is essentially a peak matching algorithm. It works by looking at "peaks" in the given sequences and measuring how they match up. When the sequences are matching up sufficiently well with a given offset, such that the error between individual corresponding peaks is low, we say that the sequences are aligned. As an example, here, u and v are aligned with an offset of 3:

$$u = [3, 7, 8, 7, 6, 0, 0, 7, 5, 3]$$
$$v = [7, 7, 0, 0, 7, 5, 4, 0, 1, 5]$$

The definition is similar in principle to convolution, and practically speaking, the result is similar to FFT based image alignment[16]. However, FFT based approaches are generally not fast for small data sets, such as the ones we are dealing with, despite having a better computational efficiency[5].

4.1 Definition

Given two sequences of positive integers, u and v, of length n, the following sum of squared errors correlation:

$$(u * v)(k) = \sum_{i=-\infty}^{\infty} [u(i) - v(i-k)]^2$$

should yield minima when the two arrays u and v are aligned. In cases where $u(i)$ or $v(i-k)$ are undefined, the difference is 0. For a general pair of sequences, there is the

potential that there are multiple values of k that produce good alignments, especially in the case that the arrays contain noise.

Error calculations are inherently based on the size of the overlap between u and v. Therefore ideally, $-n/2 < k < n/2$. Our confidence of a good match when k is outside these bounds is reduced, as the the minima will naturally be lower as less items overlap[3].

We adapt this function to include an initial estimation parameter e:

$$(u * v)(k, e) = (k - e)^E + \sum_{i=-\infty}^{\infty} [u(i) - v(i - k)]^2$$

where e is assumed to be close to the actual value of k. The exponent E in the bias should be adjusted so that it approximates a gaussian distribution on the same magnitude as the actual error. This can be pre-selected or computed dynamically; in our implementation we use 2 for efficiency.

This estimation bias serves two important purposes, it reduces the ambiguity in the case that there are multiple values of k that give a good alignment, and it provides several opportunities to improve the performance of the implementation. In practice, the estimation bias is how we leverage the sensor data - the more accurate the initial estimate, the more efficient and accurate the result.

4.1.1 Reducing Ambiguity

In the case that there are multiple minima, we need some way to distinguish between them. The initial estimate bias solves this by increasing the error for solutions further away from e. For example in the following,

$$u = [0, 5, 0, 4, 0, 5, 0, 4, 0, 5]$$
$$v = [4, 0, 5, 0, 4, 0, 5, 0, 5, 0]$$

an offset of $k = -3$ and $k = +1$ give the same minima 0. If we estimate that the alignment $e = 2$, we reduce this to a single minima at $k = 1$.

In the very rare case that we *still* have ambiguity, we can often pick the minima closest to e. However, in practical data sets, this event has never been observed.

4.1.2 Improving Performance

We can avoid evaluating $(u * v)(k, e)$ for values of k that are likely to give high error. We use an exponential initial error based on the distance of $k - e$ and if we incrementally evaluate $(u * v)(k, e)$, we can ignore values of k that are unlikely to yield good results.

The naive implementation has a best case $O(kn)$ because for all valid k, we must evaluate n multiplications and select the minimum. Leveraging the estimation, we can avoid computing k that are bigger than the currently found minimum.

We can improve the linear implementation by incrementally evaluating for k expanding outwards from e, avoiding computations for k that are bigger than the currently found minimum.

[3]The mean squared error is another statistical formulation which effectively normalises the error by the size of the overlap, but makes our approach hard to optimise and doesn't make a significant different in real usage.

Alternatively, we could use a min-heap and incrementally evaluate the summation over i to track the current minima for a given offset k. In particular, it is common to find a good value for k within a few iterations. By specially crafting the heap `siftdown` function[13] we can leverage this condition to minimise the amount of cache thrashing.

Finally, in order to maximise the benefit of the heap, we should avoid evaluating i sequentially. Doing so will often compute many uninteresting cases where $[u(i) - v(i - k)]^2$ is relatively small and cause the heap to reorganise itself repeatedly. In contrast, the biggest peaks will often lead to the biggest errors when data is misaligned. By evaluating these first, we push unlikely k to the bottom of the heap, and we reduce the chance that we continue to evaluate incorrect alignments. We compute the peaks of u such that $u[peaks[i]] >= u[peaks[i + 1]]$ and use this to incrementally evaluate $[u(peaks[i]) - v(peaks[i] - k)]^2$.

4.2 Implementation

We define the function `FISA`, Fast Integral Sequence Alignment, for arrays u and v of length n as follows:

$$\texttt{FISA}(u, v, e) = \min_{k=-n/2}^{n/2} (u * v)(k, e)$$

This function, in practice, returns a $(k, error)$ pair such that error is minimised. The range of k can be adjusted depending on the data sets we are dealing with. We implement this function in C++11 with two main variations, a linear search method that computes for all k but is bounded by the worst error found thus far, and a heap method that incrementally updates based on the lowest error.

4.3 Evaluation

We evaluated the above algorithm on an iPhone 5 running iOS 7, compiled with `clang++-3.3 -O3`. We vary the size of n using randomly generated sequences with values between 0 and 50. We add several large values in the data set between 0 and 250 and include up to 10% noise in all values. We compare 4 variations of the algorithm, including a linear search from left to right, the linear search expanding outwards from the initial estimate, the heap search with incremental evaluation and the heap search with peak-order evaluation. See Table 6 for the results.

Table 6. FISA Performance Results.

	Linear (µs)		Heap (µs)	
n	**Left-Right**	**Outward**	**Left-Right**	**Peaks**
8	0.552	**0.484**	3.628	6.629
16	1.716	**1.357**	10.721	10.489
32	4.914	**2.878**	22.536	20.126
64	16.560	**9.496**	46.687	34.780
128	56.323	**32.793**	94.913	60.527
256	176.742	112.915	201.484	**103.439**
512	733.457	430.844	492.797	**205.429**
1024	2987.51	1713.51	1253.59	**365.713**

For small problems $n < 256$, the outward expanding linear scan is the most efficient choice. This is likely due to the fact that it uses the L1 cache more effectively than the heap implementation which copies a significant amount of data during `siftup` and `siftdown` operations.

The cost of the initial sort in the heap+peaks implementation is moderate but pays off significantly for $n >= 512$. For $n = 1024$, it is almost 35 times faster.

These results suggest that for a general algorithm, a hybrid between the two algorithms would be appropriate, however in our case we are mostly concerned with $n < 256$ so we have chosen to use the linear+outward search.

5. SOURCE CODE

The source code for this research has been published as part of the Transform Flow Visualisation and Evaluation Toolkit[27, 28], and is compatible with the Transform Flow Browser for iOS[25].

6. CONCLUSION

We have presented an efficient method to reduce jitter and improve the AR experience on mobile consumer level hardware. Our method combines sensor data and video frames to ensure reliable tracking, and is tolerant to both inertial and visual tracking failures. The proposed scan line based approach can be tuned for real time performance on a variety of different input resolutions and hardware levels, and the fast integral sequence alignment algorithm can correct errors in the inertial sensor measurements efficiently.

References

[1] R. T. Azuma. The challenge of making augmented reality work outdoors. *Mixed reality: Merging real and virtual worlds*, 1:379–390, 1999.

[2] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems journal*, 4(1):25–30, 1965.

[3] A. Clarke. *OPIRA: The Optical-flow Perspective Invariant Registration Augmentation and other improvements for Natural Feature Registration*. PhD thesis, University of Canterbury, 2009.

[4] T. L. Daniel Wagner, Alessandro Mulloni and D. Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. *IEEE Virtual Reality*, pages 211–218, 2010.

[5] R. J. Fateman. When is FFT multiplication of arbitrary-precision polynomials practical? *University of California, Berkeley*, 2006.

[6] A. Huertas and G. Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):651–664, 1986.

[7] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, ISMAR '07, pages 1–10, Washington, DC, USA, 2007. IEEE Computer Society.

[8] D. Kurz and S. Benhimane. Gravity-aware handheld augmented reality. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '11, pages 111–120, Washington, DC, USA, 2011. IEEE Computer Society.

[9] D. Kurz and S. Benhimane. Handheld augmented reality involving gravity measurements. *Computers & Graphics*, 36(7):866–883, 2012.

[10] S. Lieberknecht, S. Benhimane, P. Meier, and N. Navab. A dataset and evaluation methodology for template-based tracking algorithms. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 145–151, Washington, DC, USA, 2009. IEEE Computer Society.

[11] U. Lieneweg. Automated optical extraction from line arrays of the alignment between microfabricated layers. *JPL Technical Report Server 1992+*, 1997.

[12] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th international joint conference on Artificial intelligence - Volume 2*, IJCAI'81, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[13] L. Mao and S. Lang. An empirical study of heap traversal and its applications. In *Proceedings of the 7th World Multi Conference on Systematic, Cybernatics and Informatics, July*, pages 27–30, 2003.

[14] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[15] O. Nestares, Y. Gat, H. Haussecker, and I. Kozintsev. Video stabilization to a global 3D frame of reference by fusing orientation sensor and image alignment data. In *Mixed and Augmented Reality (ISMAR), 2010 9th IEEE International Symposium on*, pages 257–258, 2010.

[16] B. S. Reddy and B. N. Chatterji. An FFT-based technique for translation, rotation, and scale-invariant image registration. *Image Processing, IEEE Transactions on*, 5(8):1266–1271, 1996.

[17] G. Reitmayr and T. W. Drummond. Going out: Robust tracking for outdoor augmented reality. In *Proc. ISMAR 2006*, pages 109–118, Santa Barbara, CA, USA, October 22–25 2006. IEEE and ACM, IEEE CS.

[18] T. Rossler, S. Rogge, and C. Hentschel. A case study: Mobile augmented reality system for visualization of large buildings. In *Consumer Electronics - Berlin (ICCE-Berlin), 2011 IEEE International Conference on*, pages 311–314, 2011.

[19] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[20] G. Schall, D. Wagner, G. Reitmayr, E. Taichmann, M. Wieser, D. Schmalstieg, and B. Hofmann-Wellenhof. Global pose estimation using multi-sensor fusion for outdoor augmented reality. In *Proceedings of the 2009 8th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '09, pages 153–162, Washington, DC, USA, 2009. IEEE Computer Society.

[21] J. Shi and C. Tomasi. Good features to track. Technical report, Ithaca, NY, USA, 1993.

[22] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[23] D. Wagner, A. Mulloni, T. Langlotz, and D. Schmalstieg. Real-time panoramic mapping and tracking on mobile phones. In *Proceedings of the 2010 IEEE Virtual Reality Conference*, VR '10, pages 211–218, Washington, DC, USA, 2010. IEEE Computer Society.

[24] J. Wen, W. Helton, and M. Billinghurst. A study of user perception, interface performance, and actual usage of mobile pedestrian navigation aides. In *The 57th Annual Meeting of the Human Factors and Ergonomics Society*, 2013.

[25] S. Williams. Tranform Flow browser application for iOS, 2013. `https://github.com/HITLabNZ/transform-flow-browser-ios`.

[26] S. Williams. Tranform Flow data sets, 2013. `https://github.com/HITLabNZ/transform-flow-data`.

[27] S. Williams. Tranform Flow library, 2013. `https://github.com/HITLabNZ/transform-flow`.

[28] S. Williams, R. Green, and M. Billinghurst. Transform flow: A mobile augmented reality visualisation and evaluation toolkit. In *Proceedings of the 28th International Conference of Image and Vision Computing New Zealand*, IVCNZ 2013. IEEE, 2013.

[29] S. You, U. Neumann, and R. Azuma. Hybrid inertial and vision tracking for augmented reality registration. In *Virtual Reality, 1999. Proceedings., IEEE*, pages 260–267. IEEE, 1999.