# Inertial Sensors Aided Image Alignment and Stitching for Panorama on Mobile Phones

**Qingxuan Yang**
Google Research
Beijing, China
qingxuan@google.com

**Chengu Wang**
IIIS, Tsinghua University
Beijing, China
wangchengu@gmail.com

**Yuan Gao**
Department of Computer Science and Technology
Tsinghua University
Beijing, China
gaoy03@gmail.com

**Hang Qu**
Google Research
Beijing, China
quhang@google.com

**Edward Y. Chang**
Google Research
Beijing, China
edchang@google.com

## ABSTRACT
In this paper, we propose using signals collected from inertial sensors on cameras to speed up image alignment for panorama construction. Inertial sensors including accelerometers and gyroscopes are first calibrated to improve sensing accuracy. These sensors are then used to estimate the position and orientation of each captured image frame. By knowing the relative displacement of image frames, alignment can be performed with good accuracy and computational efficiency. Through examples we illustrate the effectiveness of inertial-sensor assisted panorama.

## ACM Classification Keywords
I.3.3 Computer Graphics: Picture/Image Generation.

## General Terms
Algorithms, Experimentation, Performance.

## Author Keywords
image alignment, panorama, inertial navigation systems, image stitching

## INTRODUCTION
While mobile phones are usually equipped with cameras, due to style and cost considerations, very few are equipped with an advanced lens. However, new generation smart phones such as the iPhone and Nexus come with high-capacity computational/graphical processing units and inertial navigation systems. In this work, we show that these CPUs/GPUs and inertial navigation systems (INS) can be utilized to improve the quality of images taken by inexpensive lenses of mobile phones.

One useful imaging application on mobile phones is panorama, which aligns and stitches frames of limited field of view into ones with a larger field of view. With such capabilities, a mobile-phone camera could "scan" a document, a room, or an outdoor wide-angle scene for example. Traditional panorama support requires a user to steadily hold a camera and then slowly pan the camera from one side to the other, sequentially snapping photos. Such requirements ensure that each frame quality is of good enough quality and consistent enough with other frames such that frame alignment can be performed effectively. However, such stringent usage requirements makes for poor user experience. Furthermore, an undesirable camera movement, such as a camera tilt/rotation or uneven-speed panning, can degrade stitching quality.

In this work, we show that using inertial sensors (accelerometers and gyroscopes) on mobile phones to register camera position, orientation, and moving trajectory can provide valuable information for the alignment, and subsequent stitching of captured frames. In particular, this allows a user the ability to scan a scene with any camera movement trajectory. Each frame can then be processed and adjusted based on the camera's orientation when the photos were taken. Next, alignment of adjacent frames can be performed using INS provided information. The inter-frame displacement information provided by the sensors reduces the search space for finding matching inter-frame features. Since the search space is largely reduced and denser sampling can be conducted to ensure high-quality alignment and stitching, both stitching quality and speed can be improved.

This paper makes two key contributions. First, we propose non-intrusive calibration methods to improve sensor accuracy. Second, we show how collected signals from inertial sensors can be used to obtain speed enhancements in panorama construction. In the following sections we first describe detailed techniques in Section II, and then show the results in Section III. Section IV introduces future work.

## RELATED WORK

The alignment and subsequent stitching together of image are among the widely studied areas in computer vision. Image alignment and stitching can be mostly divided in three steps, namely, motion model registration, global alignment, and compostion. In the motion model registration step, planar perspective motion, cylindrical coordinates and spherical coordinates are the three most common motion models. The motion model used is either specified by the user or selected through the Bayesian algorithm. In the global alignment step, images are aligned using either a pixel-based or feature-based methods to detect the overlapping area in each pair of images. In the composition step, images that have been aligned in the last step are stitched together.

Although the main aspects required to create panorama can be clearly divided into three steps, there still remain several open issues. Firstly, almost all the motion models mentioned assume the camera is fixed in one place, and successively rotated to capture all the scenes, specifically; the cylindrical coordinates require the camera to be level. Secondly, the method used to recognize the subset of images used to compose a panorama either relies on artificial input or depends on a time intensive feature-based alignment between each pair of images. We believe that the incorporation of inertial sensors will solve the above two problems in an easy way.

Beyond the aforementioned benefits to using inertial sensors in the construction of a panoramic image, state-of-the-art research projects focus on how to transplant current panorama techniques to mobile phones. This is a difficult task since to precisely align adjacent images and stitch them seamlessly using traditional methods requires high computational resources. Xiong and Pulli [10] designed a dynamic programming seam search algorithm to realize fast image stitching and editing for panorama painting on mobile phones. However, their algorithm still relies on registration transformation to extract overlapping areas between images. This is undesirable because it has been verified to be an overly complicated procedure no matter whether feature-based or pixel-based approaches are used, unless the inertial sensors provide the displacement information. Ha et al. [5] proposed a computationally efficient and performance improved image mosaic algorithm via integer arithmetic for mobile camera systems. The algorithm works only on the assumption that users move the cameras in one direction when capturing overlapped scenes, which results in a pool user experience.

## INERTIAL SENSORS AIDED IMAGES ALIGNMENT

When using a mobile phone to take several successive photos, the relative displacement among the photos can be derived from the position and orientation (angular position, or attitude) of the mobile phone. If one knew the exact position and orientation of each photo, one could stitch them directly without any knowledge of the content of the images. However, the sensors embedded in mobile phones suffer from errors due to low manufacturing quality that originate from cost constraints. To combat this, we need to first calibrate the sensors before use in order to estimate the position and orientation of the mobile phone.

In this section, we first introduce both the model and the method for calibrating the phone-equipped accelerometer and gyroscope. We then present a pipeline including high pass filter, zero velocity revision and noise gate to track both the orientation and displacement of the camera. Third, we expose how to use camera path tracking to predict the displacement between photos. Finally, we show that displacement prediction can speed up image alignment.

### Sensor Calibration

There are two opportunities in which the inertial units in a mobile phone may be calibrated: at the manufacturer and at home. The calibration process at the manufacturer can rely on external, expensive devices. However, once a user has purchased a phone, the calibration process at home should be one-time, non-intrusive, and certainly cannot rely on external devices such as a turn table.

#### *Preliminaries*

The inertial sensors give us a measured value of the real world. The aim of calibration is to ensure the measured value coincides with the real world value as much as possible. We denote $a(t)$ and $\tilde{a}(t)$ the real acceleration and the acceleration measured by the accelerometer, respectively; $\omega(t)$ and $\tilde{\omega}(t)$ the real angular velocity and the angular velocity measured by the gyroscope, respectively; $m(t)$ and $\tilde{m}(t)$ the real magnetic field and the magnetic field measured by the digital compass, respectively. All these real and measured values are taken in reference to the phone coordinate reference system at time $t$.

Since the measured values are all taken in the phone coordinate reference system, we need to acquire the orientation of the phone when receiving each value. A rotation matrix is an orthogonal matrix with determinant of one. We represent the orientation of the phone by a $3 \times 3$ rotation matrix $R$ if left-multiplying $R$ can move the phone from the reference placement to its current placement. Therefore, left-multiplying $R$ to the measured values can convert them from values in the phone coordinate reference system to those in the absolute coordinate reference system.

Besides the rotation matrix, quaternion is another convenient notation to represent rotation and orientation, which is more numerically stable and efficient. Quaternion is a unit vector $(a, b, c, d)^T \in \mathbb{R}^4$. The relationships between the quaternion $(a, b, c, d)^T$ and the rotation matrix $\mathbf{R} = (r_{ij})_{ij}$ are given below:

$$\mathbf{R} = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2(bc-ad) & 2(bd+ac) \\ 2(bc+ad) & a^2-b^2+c^2-d^2 & 2(cd-ab) \\ 2(bd-ac) & 2(cd+ab) & a^2-b^2-c^2+d^2 \end{pmatrix}$$

$$\begin{aligned} a &= \sqrt{1+r_{00}+r_{11}+r_{22}}/2 \\ b &= (r_{21}-r_{12})/4a \\ c &= (r_{02}-r_{20})/4a \\ d &= (r_{10}-r_{01})/4a. \end{aligned}$$

The quaternion can also be continuously updated by angular velocity. Suppose a rigid body is rotated at a uniform angular velocity $\omega$ from an orientation represented by $q_0$ to that represented by $q_1$ during a time span $\Delta t$. $q_1$ can then be derived from $q_0$ using Equation 1.

$$q_1 = \left(\cos(|\omega|\Delta t/2)\mathbf{I} + \frac{2\sin(|\omega|\Delta t/2)}{|\omega|}\mathbf{F}_q(\omega)\right) q_0, \quad (1)$$

where $|\cdot|$ is the $L^2$ norm of a vector and

$$\mathbf{F}_q(\omega) = \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix}.$$

Furthermore, if a rotation from some orientation $o_1$ to orientation $o_2$ is represented by $q = (a, b, c, d)^T$, the *inverse* of $q$, i.e. the rotation from orientation $o_2$ to $o_1$, is $q^{-1} = (a, -b, -c, -d)^T$. The *composition* of two rotations $q = (a, b, c, d)^T$ and
$q' = (a', b', c', d')^T$ (first rotate by $q'$ and then by $q$) is given by $qq' = (aa'-bb'-cc'-dd', ab'+a'b+cd'-c'd, ac'+a'c+db'-d'b, ad'+a'd+bc'-b'c)^T$. Also, the ratation quaternion maintains the information of rotation axis and angle. The ratation axis is given by the normailization of $(b, c, d)^T$. The rotation angle is given by $2\cos^{-1} a$.

### *Error Model*

For the accelerometer, we assume a linear error on each axis, i.e. $\tilde{a}(t) = \mathbf{S}a(t) + \gamma + \epsilon$, where $\mathbf{S}$ is a $3 \times 3$ diagonal matrix, $\gamma$ is a 3 dimensional vector, and $\epsilon$ is a zero-mean 3 dimensional Gaussian random vector.

For the gyroscope, we assume the error model is linear. Formally speaking, $\tilde{\omega}(t) = \mathbf{K}\omega(t) + \beta + \eta$, where $\mathbf{K}$ is a $3 \times 3$ matrix, $\beta$ is a 3 dimensional vector, and $\eta$ is a zero-mean 3 dimensional Gaussian random vector.

For the digital compass, we assume there are no random errors, but allow deterministic errors. Formally, $\tilde{m}(t) = f(m(t))$ for a deterministic continuous function $f$. In other words, $\tilde{m}(t') = \tilde{m}(t'')$ if we can rotate the phone from the orientation at $t'$ to the orientation at $t''$ about the axis in the direction of north.

### *Accelerometer Calibration*

We use the method in [4] for the calibration of the accelerometer. The random error $\epsilon$ can be eliminated by averaging the accelerometer readings spanning a time window. Allan Variance [2] is utilized to find a shortest length time window such that we can eliminate the random error. After the removal of the random error, we use the constant acceleration of gravity to help calibrate the accelerometer, detailed in [4].

### *Topology Calibration for Gyroscope*

Allan Variance is also used to eliminate the random error $\eta$ in the gyroscope. We design a brand new scheme called topology calibration to determine the scale factor errors $K$ as well as the biases $\beta$ of the gyroscope.

When we rotate the phone in a space freely, $m(t)$ runs on an sphere $\mathbb{S}$, and $\tilde{m}$ runs on a closed surface $f(\mathbb{S})$. We find all points of intersection, i.e all pairs $(t', t'')$ such that $m(t') = m(t'')$. For each pair $(t', t'')$, we compute the rotation $R$ between them by $\omega$. Using the fact that the rotation axis of $R$ is north for each pair $(t', t'')$, we want to find $K$ and $\beta$ that concentrate all the $R$'s. The details are given presently.

First, we compute all points of intersection of $m$. Because values of $\tilde{m}$ are given as discrete samples, we use linear interpolation to convert the function $\tilde{m}$ into a continuous one. We take $m(t') = m(t'')$ if $\tilde{m}(t') = \alpha \cdot \tilde{m}(t'')$, where $\alpha$ is a positive real number. More specifically, we want to find all pairs $t' = \beta t_i + (1-\beta)t_{i+1}$ and $t'' = \gamma t_j + (1-\gamma)t_{j+1}$ such that $\beta m(t_i) + (1-\beta)m(t_{i+1}) = \alpha\left(\gamma m(t_j) + (1-\gamma)m(t_{j+1})\right)$, where $\alpha > 0$, $0 \le \beta \le 1$, $0 \le \gamma \le 1$, and $t_i, t_{i+1}, t_j, t_{j+1}$ are the times of the samples.

Then, we compute the orientation represented by quaternion. We initialize the quaternion $q(t_0) = (1, 0, 0, 0)^T$, and update the quaternion for each gyroscope sample $\omega(t_i)$ sequentially. Here, since the time between two consecutive gyroscope samples is quite small, the rotation is treated as a uniform angular velocity $\omega(t_i)$. In this approximation, Equation 1 is refined to Equation 2.

$$\begin{aligned} q(t_{i+1}) = \big(&\cos(|\omega(t_i)|\Delta t/2)\mathbf{I} \\ &+ \frac{2\sin(|\omega(t_i)|\Delta t/2)}{|\omega(t_i)|}\mathbf{F}_q(\omega(t_i))\big)q(t_i), \end{aligned} \quad (2)$$

where $\Delta t = t_{i+1} - t_i$.

Next, for the $i$-th intersection $(t'_i, t''_i)$, we compute the rotation $q(t''_i)q(t'_i)^{-1}$ between them, and find the rotation axis $u_i$ and the rotation angle $\theta_i$. Ideally, $u_i$ should point north in the absence of error. Then, we compute the average rotation axis $\bar{u}$ of all $u_i$'s, weighted by $(1 - \cos\theta_i)$. Finally we normalize $\bar{u}$. The variation of the all the $u$'s is defined by $\sum_u (1 - u \cdot \bar{u})$, where $(1 - u \cdot \bar{u})$ is the square of the distance between $u$ and $\bar{u}$ on the unit sphere.

Finally, we find $K$ and $\beta$ which minimize the variation by using the Nelder–Mead method (or downhill simplex method), starting with $K = I$ and $\beta = (0, 0, 0)^T$. In each iteration of the Nelder-Mead method, we have to recompute the orientation, but do not need to recompute the intersections.

## Camera Path Tracking

In this part, we utilize the sensors to track the movement of the camera. The position of a rigid body is represented by the combination of its linear position (or displacement) and its angular position (or orientation).

### *Orientation*

The angular velocity $\omega = (\omega_x, \omega_y, \omega_z)^T$ measured by gyroscope of the mobile phone is used to track the orientation, which can be represented in several ways, e.g., via rotation matrix, Euler angle, quaternion and so on, as discussed in Section . We use quaternion in this work because it is simpler to compose and avoids the problem of gimbal lock compared to the Euler angle, and is more numerically stable compared to the rotation matrix. The quaternion is also updated by the angular velocity given by calibrated gyroscope as stated in Equation 2.

In order to determine the initial quaternion, the phone is required to be static for a fixed period of time. In the static state, the accelerometer can derive the direction of gravity and magnetometer can derive the north, which can therefore provide the initial orientation of the phone.

### *Displacement*

Accelerometers in mobile phones give the acceleration coordinate in the phone reference coordinate system (denoted as $a_p$). However, the calculation of displacement requires acceleration in an absolute reference coordinate system (denoted as $a_a$). $a_a$ is obtained from $a_p$ using Equation 3:

$$a_a = Ra_p, \tag{3}$$

where $R$ is the rotation matrix. Gravity can be directly substracted from $a_a$ to isolate only that acceleration caused by camera motion. Hereafter, $a$ is used to represent the acceleration without gravity in the absolute reference coordinate system.

Generally speaking, displacement can be calculated by applying a second-order numerical integration algorithm to $a$. However, calibration is only able to decrease bias and scale factor errors, leaving other errors such as nonlinear ones unresolved. These small unaccounted errors in the measurement of acceleration and angular velocity are integrated into progressively larger errors in velocity, which are then further compounded into even greater errors in displacement. Therefore, we design a pipeline of three steps including a *high pass filter*, a *zero velocity revision* and a *noise gate* which we employ during the integration process to finally reduce the calibration-avoiding errors in the displacement.

### *Step 1. High Pass Filter*

Because the biases of the gyroscope are relative to temperature, the calibration procedure cannot totally eliminate them. This makes the rotation matrix tilt the phone's orientation. As a result, gravity still influences the acceleration measured along the horizontal plane (i.e., along the x-y axes). The high pass filter aims at decreasing such influences. If the acceleration $a$ is fed to the filter as a sequence, we let $a_i$ denote the $i^{th}$ acceleration update. The output of the high pass filter $\mathrm{HPF}(a_i)$ is calculated using Equation 4:

$$\begin{aligned}\mathrm{HPF}(a_i) &= a_i - \mathrm{LPF}(a_i)\\ &= a_i - [(1-e^{-\frac{\Delta t}{\beta}})a_i - e^{-\frac{\Delta t}{\beta}}\,\mathrm{LPF}(a_{i-1})] \\ &= e^{-\frac{\Delta t}{\beta}}a_i + e^{-\frac{\Delta t}{\beta}}\,\mathrm{LPF}(a_{i-1})\end{aligned} \tag{4}$$

where $\mathrm{LPF}(a_i)$ is the result of $a_i$ passing through a low pass filter determined by an attenuation factor $\beta$, and $\Delta t$ is the time interval between $a_i$ and $a_{i-1}$.

Figure 1 shows the effect of the high pass filter defined in Equation 4, where $\beta$ is set to be 1 second. Obviously, the

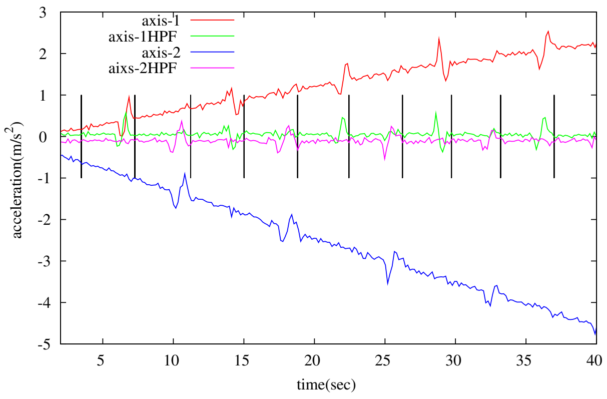


**Figure 1. The effect of a high pass filter acting on the acceleration of Figure 8's camera taking a 10 picture photo shoot. The vertical black lines mark the time the photos were taken. The green line shows the red line signal passing through the high pass filter, and the pink line shows the blue line signal passing through the high pass filter.**

accelerations along the two orthogonal horizontal axes are both drawn back to near zero.

### *Step 2. Zero Velocity Revision*

The output of the high pass filter still has errors. As shown in Figure 1, the adjusted acceleration is still a approximately non-zero constant even if the camera is at rest. This calculated velocity results in deviating away from zero with time as shown in Figure 2. In this step, we remove this kind of error based on the assumption that the velocity of the camera is small when the user takes photos. This assumption is resealable for cell-phones cameras because the photo will be blurry if the phone takes it with a fast movement.

We denote the moment when the shutter opens on the $i$-th photo by $t^{(i)}$, and assume the velocity at $t^{(i)}$ is small. We want to add a constant $c^{(i)}$ to the accelerations between $t^{(i)}$ and $t^{(i+1)}$ such that the velocity at $t^{(i+1)}$ is zero. This will represent the error in the acceleration that propagates into calculating the velocity gives a linear drift (see Figure 2). In this constant error assumption, we have the equation:

$$\int_{t^{(i)}}^{t^{(i+1)}} (a_H(t) + c^{(i)})dt = 0,$$

where $a_H(t)$ is the output acceleration of the high pass filter. When we solve the equation, we have

$$c^{(i)} = -\frac{\int_{t^{(i)}}^{t^{(i+1)}} a_H(t)dt}{t^{(i+1)} - t^{(i)}}.$$

Therefore, our algorithm is as follows: we integrate the accelerations to compute the $v^{(i+1)}$ as before (possibly non-zero). Then we subtract $-c^{(i)} = v^{(i+1)}/(t^{(i+1)} - t^{(i)})$ from the accelerations between $t^{(i)}$ and $t^{(i+1)}$. Finally, we compute the velocity again.

Figure 2 shows the velocities. The green and pink lines are much more accurate than the the original red and blue lines.

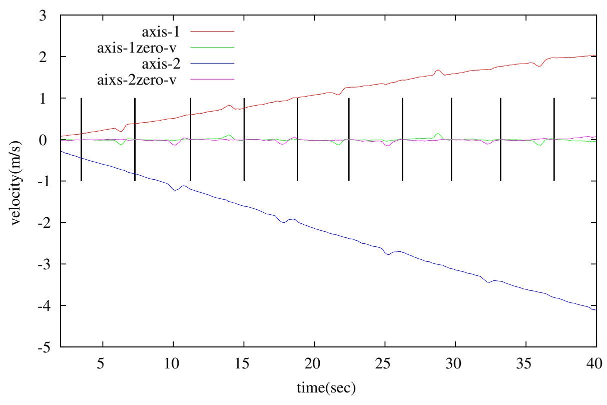


**Figure 2. The effect of zero velocity revision in experiment shown in Figure 8. The red and blue lines are the velocities in two axis output by the high pass filter, corresponding to the red and blue lines in Figure 1. We reset them when taking photos, and show them by the green and pink lines respectively. The vertical black lines mark the time points of taking photos.**

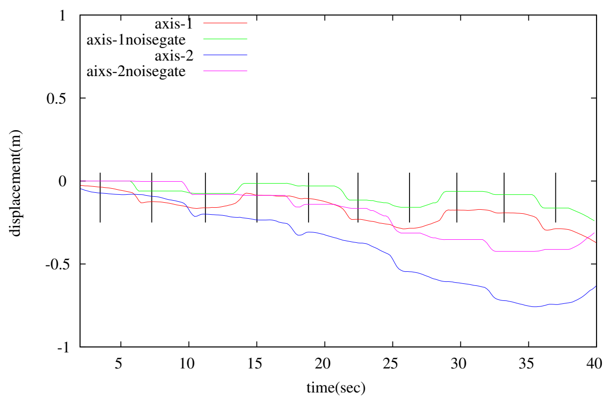


**Figure 3. The effect of applying the noise gate. The red and blue lines are the displacements along the two non-vertical axes. We apply the noise gate on the velocity and recompute the displacement, shown by the green and pink lines. We choose the high threshold to be $0.04$m/s, the low threshold to be $0.01$m/s, and the hold time to be $0.5$s.**

### *Step 3. Noise Gate*

In the beginning, the measured acceleration $a(t)$ is corrupted with errors, so we apply a high pass filter on it. The resulting acceleration $a_H(t)$ seems good, but the velocity derived from it deviates from zero. As a result, we “reset” the velocity at snapshot points produce a more accurate velocity approximation. Now, we come to final step of calculating the displacement via the integration of velocity, illustrated by the red and blue lines in Figure 3. The approximated displacements stray away from zero with time because, just like the non-zero acceleration causes a deviation of the velocity in Section , the velocity is non-zero when the camera is actually still. From Figure 2, we see that the velocity is almost zero most of the time. So, in this step, we set the velocity to zero when it is sufficiently small.

A noise gate is parameterized by an open threshold, a close threshold and a hold time. The noise gate opens when the input exceeds the open threshold, and it closes when the input is smaller than the close threshold or the input runs below the open threshold for hold time.

In Figure 3, we can see that the noise gate approximated displacement (shown as the green and pink lines) keeps when the noise gate is closed. This results in a more authentic approximation.

## Prediction of Displacement between Photos

In this part, we make use of our approximations about the movement of the camera to compute the displacement of the photo in pixels.

When the $i$-th photo is taken, we denote the position of the phone by $p_i$, the orientation matrix by $A_i$, and the distance between the scenery and the phone by $d_i$. Without loss of generality, we assume the camera points at $r = (0, 0, 1)^T$ in the reference system of the phone.

The position of the object the camera points to is $A_i d_i r + p_i$. If we take two successive photos, which are very similar but have an offset, we call this offset the *displacement* between the two photos. Formally speaking, assume a real-world object is projected onto $(x_1, y_1)$ in the first image, and onto $(x_2, y_2)$ in the second image. This insinuates that the object lies somewhere in the overlapped area of the two images. We define the displacement of the two successive images as $(x1 - x2, y1 - y2)$.

We choose the placement of the phone during the first photo as the starting reference, so $A_1 = I$ and $p_1 = 0$. We assume the two photos overlaps and the movement of the phone is small. When the object is far from the camera, we can assume that $d_1 = d_2$. Denote the horizontal view angle of the camera by $\theta$ and the width of the photo by $w$. Consider a simple case where the phone does not rotate around $r$, i.e. the eigenvector of $A_2$ is orthogonal to $r$. In the scenario, the displacement of the two photos can be estimated by the $x$ and $y$ coordinates of the vector

$$\frac{A_2 d_2 r + p_2}{d_2} \cdot \frac{w}{2 \tan \frac{\theta}{2}}.$$

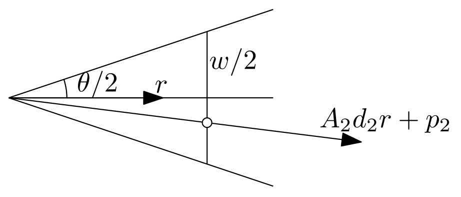


**Figure 4. Displacement between photos**

## Alignment Speed Up

Image alignment is used to discover the correspondence relationship among overlapped images. Pixel-based alignment and feature-based alignment are two popular of approaches.

Pixel-based alignment attempts to minimize the distance between two images, where the distance is defined to be the summation of the distances between each pair of pixels. Formally speaking, for two images $I_1$ and $I_2$ of size $w \times h$, let $I_i(x, y)$ be the color of the $(x, y)$ pixel. The distance between the two images is

$$\mathrm{dist}(I_1, I_2) = \sum_{(x,y)\in I_1 \cap I_2} \mathrm{dist}(I_1(x, y), I_2(x, y)).$$

The distance between two colors can be defined in many ways. For example, for gray images, we can define the distance between two colors $c_1$ and $c_2$ in the following ways:

$$\mathrm{dist}(c_1, c_2) = |c_1 - c_2|,$$

$$\mathrm{dist}(c_1, c_2) = (c_1 - c_2)^2,$$

$$\mathrm{dist}(c_1, c_2) = \begin{cases} 1 & \text{if } |c_1 - c_2| > a \\ 0 & \text{otherwise} \end{cases},$$

or

$$\mathrm{dist}(c_1, c_2) = \frac{(c_1 - c_2)^2}{1 + (c_1 - c_2)^2/a^2}.$$

A typical algorithm is to try every offset $(u, v)$ between the two images, compute the distance, then choose the best one. More specifically, we want to minimize the average distance

$$\frac{\sum_{(x,y)\in J} \mathrm{dist}(I_1(x, y), I_2(x - u, y - v))}{|J|},$$

where $J$ is the intersection between $I_1$ and $I_2$ moved by $(u, v)$, and $-w \leq u \leq w$, $-h \leq v \leq h$.

We can speed up the search process if we have a good prediction of the displacement. We first search $(u, v)$ around our prediction. If our prediction is not far from the real displacement, we will find the best alignment after only a short time.

Feature-based alignment is more robust than pixel-base ones. There are many detectors and descriptions of the features, e.g. Harris [6], FAST [8, 9], SIFT [7] and SURF [3]. All of these employ two steps: feature detection and feature matching. Feature detection is to find the key points and compute their descriptions. One surmises the description of a key point does not change too much between two images that both contain the key point. In the feature matching step, we search for pairs of key points which have similar descriptions.

With a prediction of the displacements, both feature detection and feature matching can be performed faster. In the feature detection step, we do not need to find key points over the whole image if we have a prediction of the alignment. Instead we can focus on finding only those key points that lay in the overlapped regions. The less the images overlap, the faster feature detection can be accomplished. In the matching step, we can speed up once again. We do not compare each pair of key points. Instead, within the overlapping region, we only try to match those pairs within the overlapping region that lie near our prediction of their displacement. This allows us to reduce the comparisons of key points significantly.

Furthermore, if we have more images to align, the displacement given by sensors can give more help. The predicted displacements tell us which pairs of images overlap, and which does not. Typically, the images are sparse (each image intersects with a constant number of other images), which allows us to reduce the number of pairs of images to compare from $\Omega(n^2)$ to $O(n)$, where $n$ is the number of images. Using INS, we can speed up the alignment progress and augment the correctness of the panorama, especially in the pixel-based approach.

## EXPERIMENT RESULTS

In the experiments, we use Nexus S[1], an Android mobile phone equipped with a 3-axis gyroscope, an accelerometer, a magnetometer, and a front and rear facing camera. The Android API gives the view angle by
Camera.Parameters.getHorizontalViewAngle() or
Camera.Parameters.getVerticalViewAngle().

We not only show the results of distant views (Figures 5, 6, and 7), but also those of close shots (Figures 8 and 9). Using sensors to predict movement does indeed speed the alignment in both the SIFT featured-based approach (Figures 5, 6, 7, and 8) and the pixel-based approach (Figure 9).

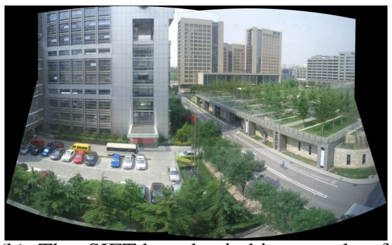
### Accuracy of Prediction of Movement

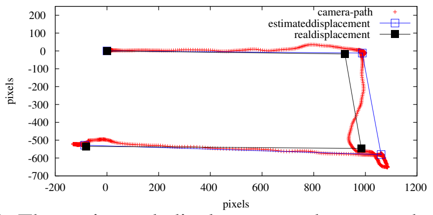
Figure 5(c), Figure 6(b) and Figure 7(b) show the camera paths estimated by sensors when compared to the real paths. The red crosses mark the estimated camera path, the blue squares are the estimated displacements on this path, and the filled squares are the "real" displacement of the camera from the stitched image, as computed by the alignment algorithm. All of these paths start at $(0, 0)$. Figures 8(c) and 9(b) show the estimated camera path when taking close shots. Figure 8(c) shows the estimated displacement of the camera in meters. Figure 9(b) shows the displacement in pixels. We first calculate the displacement in meters and convert it into pixels by using the estimated distance between the scenery and the camera.

Because Figures 5(b), 6(a), and 7(a) are distant views, the moving paths in these views are more sensitive to camera rotation as opposed to the translational motion. On the other hand, Figures 8(b) and 9(a) are close shots, and camera rotation is minimal (otherwise there is a large change in the depth of field, the processing of which is left as future work.) As a result, the moving path for close panoramas is sensitive to the translational motion of the camera. The accelerometer plays an important roll in these both cases.

In Figures 7(b) and 8(c), the moving paths of the camera are both long. Predicted displacement errors grow with time, because of cumulative effects in the integration process. These cumulative errors are manageable, because photos are aligned sequentially, and only the displacement between two successive photos is used. Also, the displacement calculated by alignment can help to adjust the estimated displacement.

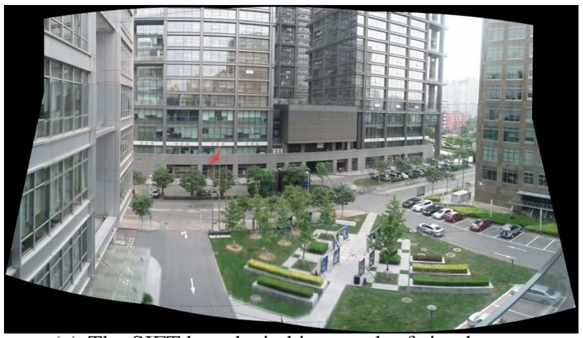
(a) The original four photos.

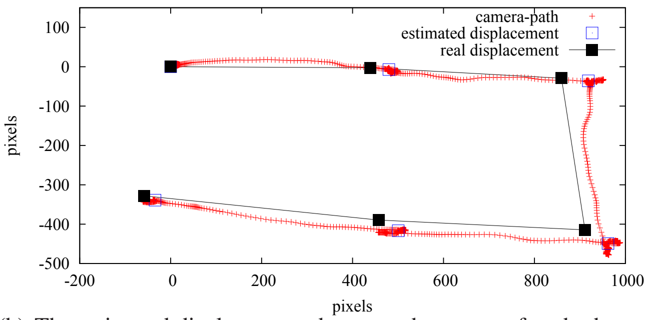

(b) The SIFT-based stitching result of four photos.



(c) The estimated displacements between the center of each photo, starting from (0,0). The blue empty squares and the black filled squares mark the estimated displacements via signals from INS, and the "real" displacement calculated by SIFT-based alignment, respectively. The red crosses show the estimated path of the whole procedure while taking the four photos.

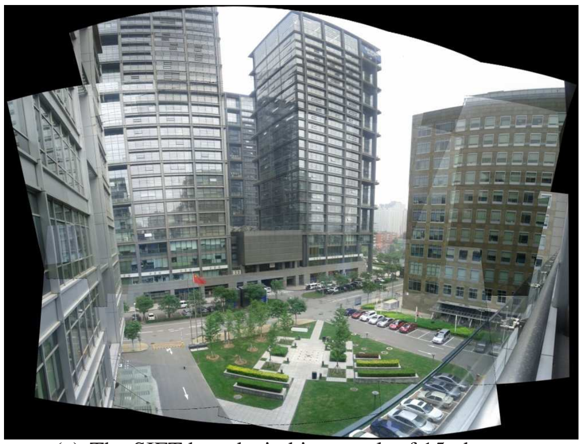
**Figure 5. Four photos about distance views.**

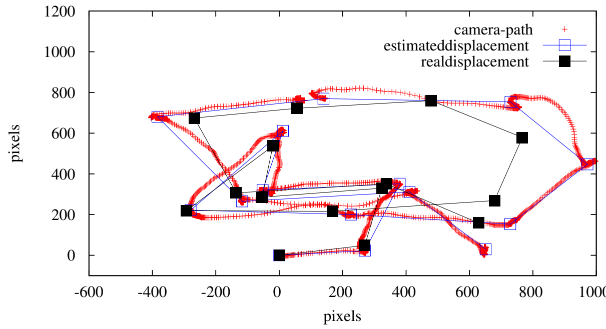

(a) The SIFT-based stitching result of six photos.



(b) The estimated displacements between the center of each photo, starting from (0,0). The legends have the same meaning as Figure 5(c).

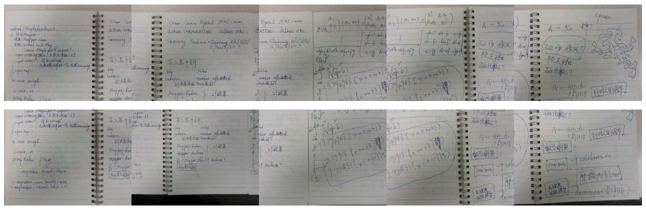
**Figure 6. Six photos about distance views.**

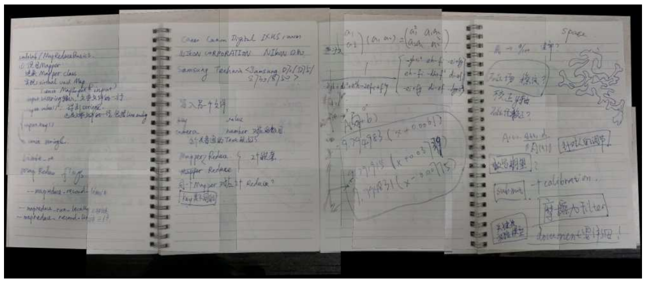

(a) The SIFT-based stitching result of 15 photos.

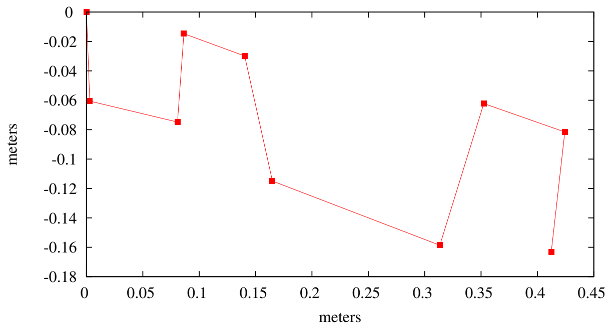


(b) The estimated displacements between the center of each photo, starting from (0,0). The legends have the same meaning as Figure 5(c). Different from Figure 5(c) and Figure 6(b), the 15 photos are taken in a free-hand-moving way.

**Figure 7. Fifteen photos about distance views.**


(a) The original ten photos.


(b) The SIFT-based stitching result of 10 photos.



(c) The estimated movement path of the camera, starting from (0,0). The squares marks the positions of the camera while taking photos.

**Figure 8. Ten close shots photos.**

### Speed Improvement

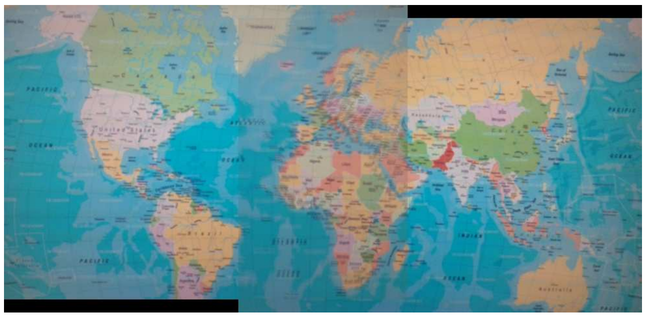
Table 1 shows that we can save time when we utilized the information provided by the sensors in the phone.

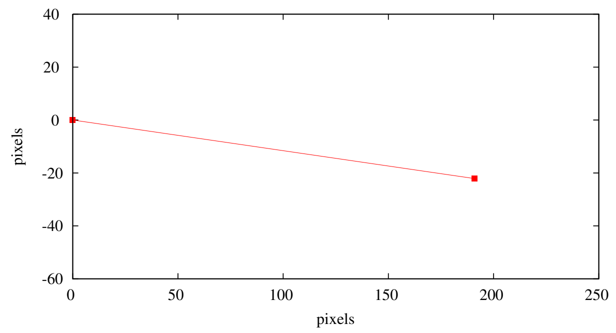
However, the largest amount of time is saved in Figure 9(a) with the pixel-based method. This is mainly due to search space being dramatically decreases because of our accurate displacement prediction, as shown in Figure 9(b).

The speed in computing Figure 5(b) is greatly improved as well. In the SIFT feature-based method, searching for key points and computing their descriptions are the most time consumption steps in the whole procedure. In this example, the overlapped area is small, so we only need to find the key points in a markedly reduced area.

## REMARKS AND FUTURE WORK

In Table 1, the running time of our program is still slow because the users have to wait several seconds for taking one photo. In the future we can improve the running time if we choose smaller image sizes, adjust parameters of SIFT, or change from SIFT to other features-based methods such as SURF. We believe that no matter what the alignment method used, the predicted moving path given by the sensors can speed up the alignment process.


(a) The pixel-based stitching result of 2 photos.



(b) The estimated displacements between the center of each photo, starting from (0,0). The squares marks the center of photos.

**Figure 9. Two close shots photos.**

In the Experiment Results section, we only test the translational motion of camera in close shots. If the camera is rotated when taking consecutive pictures at close range, it is hard for us to maintain the orientation of the camera precisely and in real time, so we cannot isolate and remove the gravity from the acceleration readings very easily. Since the gravity is much greater than the acceleration of the motion of the camera, the result is a large error in predicted displacement. Therefore, we wish to find better approaches to compute the orientation and displacement of the camera in cases that involve complicated motion.

| Figure number | Method | Time without sensors | Time with sensors | Rate of decrease |
|---|---|---|---|---|
| 6(a) | feature-based | 19.9s | 18.4s | 7.5% |
| 5(b) | feature-based | 11s | 6.3s | 43% |
| 7(a) | feature-based | 69s | 50s | 28% |
| 8(b) | feature-based | 49s | 43s | 12% |
| 9(a) | pixel-based | 7.3s | 1.7s | 77% |

**Table 1. Comparision of creating panorama with/without sensors.**

## REFERENCES

1. Nexus S. `http://www.google.com/nexus/#`.
2. D. Allan and J. Barnes. A modified Allan variance with increased oscillator characterization ability. In *Thirty Fifth Annual Frequency Control Symposium*, pages 470–475. IEEE, 1981.
3. H. Bay, T. Tuytelaars, and L. J. V. Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *ECCV (1)*, volume 3951 of *Lecture Notes in Computer Science*, pages 404–417. Springer, 2006.
4. Y. Gao, Q. Yang, G. Li, E. Chang, D. Wang, C. Wang, H. Qu, P. Dong, and F. Zhang. XINS: The Anatomy of an Indoor Positioning and Navigation Architecture (unpublished). In *The First International Workshop on Mobile Location-Based Service*. ACM, 2011.
5. S. Ha, H. Koo, S. Lee, N. Cho, and S. Kim. Panorama mosaic optimization for mobile camera systems. *Consumer Electronics, IEEE Transactions on*, 53(4):1217–1225, 2007.
6. C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.
7. D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.
8. E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005.
9. E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.
10. Y. Xiong and K. Pulli. Fast image stitching and editing for panorama painting on mobile phones. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, pages 47–52. IEEE, 2010.