# CrossMotion: Fusing Device and Image Motion for User Identification, Tracking and Device Association

Andrew D. Wilson and Hrvoje Benko
Microsoft Research
Redmond, WA 98052 USA
awilson@microsoft.com, benko@microsoft.com

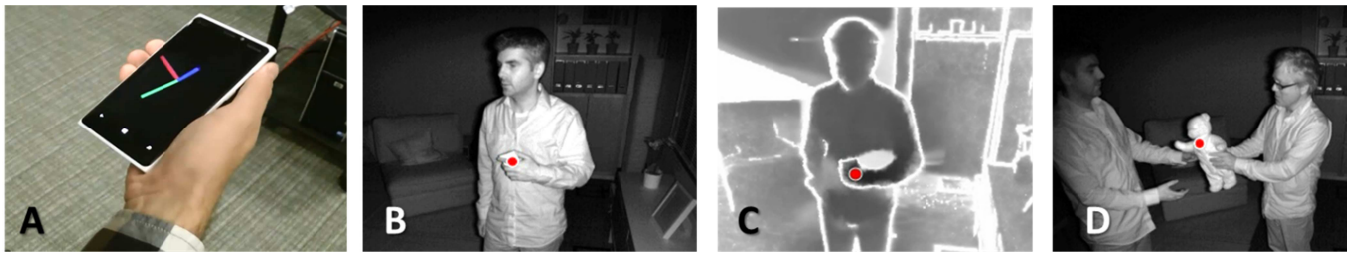


**Figure 1. CrossMotion matches phone device acceleration with accelerations observed in the infrared and depth images of the Kinect v2 camera. (a): the phone's acceleration is calculated in the Earth's coordinate frame, shown by the axis; (b) CrossMotion finds the phone held by the user (red marker); (c) image and device acceleration match image (darker is better); (d) toys and other objects can be tracked by embedding or attaching the device (the device itself does not need to be directly visible to the camera).**

## ABSTRACT
Identifying and tracking people and mobile devices indoors has many applications, but is still a challenging problem. We introduce a cross-modal sensor fusion approach to track mobile devices and the users carrying them. The CrossMotion technique matches the acceleration of a mobile device, as measured by an onboard internal measurement unit, to similar acceleration observed in the infrared and depth images of a Microsoft Kinect v2 camera. This matching process is conceptually simple and avoids many of the difficulties typical of more common appearance-based approaches. In particular, CrossMotion does not require a model of the appearance of either the user or the device, nor in many cases a direct line of sight to the device. We demonstrate a real time implementation that can be applied to many ubiquitous computing scenarios. In our experiments, CrossMotion found the person's body 99% of the time, on average within 7cm of a reference device position.

## Categories and Subject Descriptors
I.4.8 Image Processing and Computer Vision: Scene Analysis – *Sensor Fusion, Tracking*

## Keywords
Sensor fusion; depth cameras; inertial measurement units

## 1. INTRODUCTION
Tracking a mobile device and its owner is useful for a number of ubiquitous computing scenarios that rely on identifying and tracking the device's owner to provide location-based services, such as connecting the smartphone with nearby infrastructure such as a wall display. In this paper we consider the problem of tracking a mobile device user with a video camera.


*ICMI '14,* November 12 - 16 2014, Istanbul, Turkey

http://dx.doi.org/10.1145/2663204.2663270

Recognizing and tracking mobile devices from video is difficult for a number of reasons. Many smartphones are small, shiny, and dark, making them difficult to image clearly. It might be impossible to differentiate two devices of the same model. Handheld devices may be partially occluded by the hand, while those kept in a purse or clothes pocket can't be seen at all. Active markers such as infrared LEDs can assist in tracking and identification [16]. For example, Xbox One and Sony PS4 controllers use infrared and visible LEDs to assist in tracking and associating controllers with players. However, such active markers are rare, and require a line of sight.

We present a sensor fusion approach to locating and tracking a mobile device and its user in video. CrossMotion matches device acceleration with acceleration observed in the infrared and depth images of the Kinect camera (Figure 1). It uses the inertial sensors common to many mobile devices to find device acceleration. Device and image accelerations are compared in the 3D coordinate frame of the environment, thanks to the absolute orientation sensing capabilities common in today's smartphones, as well as the range sensing capability of commodity depth cameras.

Previous works ([4], [13], [15], [11]) explore the fusion of device sensors and visual features to find the user carrying the device. These rely on external means of determining candidate objects in the video. For example, ShakeID considers which of up to four tracked hands holds the device. Rather than compare the motion of a small number of candidate objects in the video, CrossMotion sensor fusion is performed *at every pixel in the video image* and requires no separate process to suggest candidate objects to track. The technique requires no knowledge of the appearance of the device or the user, thereby avoiding many of the difficulties of traditional appearance-based approaches, and allows for a wide range of camera placement options and applications. An interesting and powerful consequence of the technique is that the device user, and in many cases the device itself, may be reliably tracked even if the device is in the user's pocket (Figure 4), or embedded in another object (Figure 1d).

In this paper, we review related work, detail the CrossMotion algorithm, discuss our implementation, present the results of two evaluations of the algorithm's performance, and discuss considerations and limitations in applying the technique.

## 2. RELATED WORK

### 2.1 Device Association Using Sensor Fusion

"Sensor fusion" refers to the combination of multiple disparate sensors to obtain a more useful signal. Of particular relevance to the present work are fusion techniques that seek to associate two devices by finding correlation among sensor values taken from both. For example, when two mobile devices are held together and shaken, accelerometer readings from both devices will be highly correlated ([3], [7]). Detecting such correlation can cause application software to pair or connect the devices in some useful way. Similarly, when a unique event is observed to happen at the same time at both devices, various pairings may be established. Perhaps the simplest example is connecting two devices by pressing buttons on both devices simultaneously [10], but the same idea can be applied across a variety of sensors. For example, two devices that are physically bumped together will measure acceleration peaks at the same moment in time. Hinckley et al. [2] refers to these interactions as "synchronous gestures."

It can be particularly useful to establish correlations across very different modalities, since often such modalities complement each other. We mention just a few of these "cross-modal" approaches: a mobile phone may be located and paired with an interactive surface by correlating an acceleration peak in the device with the appearance of a touch contact [12], or when the surface detects the visible flashing of a phone at the precise moment it is triggered [18]. An object tagged with an RFID chip can be detected and located as it is placed on an interactive surface by correlating the appearance of a new surface contact with the appearance of an RFID tag [8].

### 2.2 Correlating Image and Device Motion

A small number of previous works investigate the idea of correlating mobile device inertial sensor readings with movement observed in a video camera.

Kawai et al. [4] propose correlating accelerometers worn at the waist with visual features to track young children in school. They consider tracking head-worn red LEDs, as well as tracking the position of motion blobs. For the accelerometer measurements, they consider integrating to obtain position for direct comparison with the visual tracking data, as well as deriving pedometer-like features. While the paper lacks specifics, the authors favor pedometer features in combination with markerless motion blob visual features.

Shigeta et al. [13] propose computing normalized cross-correlation between the motion trajectory of an object and device accelerometer readings to determine which of several tracked objects contains the device. Their approach requires a window of many samples to perform correlation and relies on an external process to find and track objects from monocular video. Plötz et al. [9] use a similar approach to synchronize inertial sensors and video cameras.

Teixeria et al. [15] propose identifying and tracking people across multiple existing security cameras by correlating mobile device accelerometer and magnetometer readings. They describe a hidden Markov model-based approach to find the best assignment of sensed devices to tracked people. As with Shigeta et al., they rely on an external process to generate tracked objects and use a large matching window, though they demonstrate how their approach can recover from some common tracking failures.

Most closely related to the present work is Rofouei et al.'s ShakeID system, which matches smartphone accelerometer values with the acceleration of up to four hands tracked by the Microsoft Kinect sensor [11]. The hand holding the phone is inferred by matching the device acceleration with acceleration of hand positions over a short window of time (1s). A Kalman filter is used to estimate the acceleration of each hand. The hand with the most similar pattern of acceleration is determined to be holding the device. This previous work further studies the correlation of contacts on a touch screen by the opposite hand. Ultimately touch contacts are associated with the held device by way of the Kinect tracked skeleton that is seen to be holding the device.

All of the above previous works require that a small number of candidate objects are first tracked. The subsequent correlation process involves determining which of these object's motion most closely matches that of the device. The step of generating candidate objects can be prone to failure. For example, ShakeID compares the motion of the tracked hands of the one or two users detected by the Kinect sensor skeletal tracking process. If the device is not held in the hand, or if the Kinect skeletal tracking fails, the device cannot be tracked. Furthermore, holding a mobile device can impact the hand tracking process to an extent that estimating hand acceleration robustly is difficult. Kinect skeletal tracking requires a fronto-parallel view of the users. Thus relying on Kinect skeletal tracking constrains where the camera may be placed. For example, skeletal tracking fails when the camera is mounted in the ceiling for an unobstructed top-down view of the room.

In comparison to previous work, CrossMotion avoids the difficulty of choosing candidate objects by matching low level motion features throughout the image. It may be used in many situations where skeletal tracking is noisy or fails outright and thus can be used in a wide variety of application scenarios. Whereas most of the related work performs matching over a significant window in time, CrossMotion uses a fully recursive formulation that relies on storing only the previous frame's results, not a buffer of motion history. In fact, the recursive nature of the computation allows it to be applied everywhere in the image in real time, avoiding the need to track discrete objects.

We argue that to correlate image and device motion for the purposes of locating the device or the user carrying it, the best approach is to match image motion directly, since as with "synchronous gestures" the pattern of image motion will provide the discriminative power to robustly detect the device or its user. Making fewer assumptions about the appearance of the device or user extends the range of applicability of the approach, and makes the technique less complex, more robust, and ultimately more useful.

We take some inspiration from models of visual search which feature a pre-attentive, massively parallel processing stage in which low level features are computed across the visual field, to be selected by further higher-level top-down processes [19]. For example, a pedestrian can detect interesting motion in the periphery, such as a car approaching a crosswalk, even on a windy day where the visual scene is full of motion.

Maki et al. [5] and Stein et al. [14] similarly propose matching trajectories of a set of features tracked using optical flow. Trajectory acceleration magnitude is compared to device acceleration magnitude to find the closest matching trajectory. These previous works require that the tracked points be periodically resampled to maintain even distribution throughout the image. Maki et al. demonstrates tracking 512 points, while our recursive approach uses optical flow rather differently, considering all pixel locations in the image without relying on a set of proposed features, effectively tracking a few hundred thousand points. Furthermore, these previous works compare acceleration *magnitude* which is invariant to orientation. CrossMotion instead uses the full 3D acceleration in a known coordinate frame, and thus avoids many simple scenarios that lead to false matches when using only acceleration magnitude

(e.g., a motion to the left will have the same acceleration magnitude as a similar motion to the right).

# 3. CROSSMOTION ALGORITHM

The CrossMotion algorithm matches device (e.g., smartphone) acceleration with acceleration observed in the infrared and depth images of the Kinect camera (Figure 1). This matching process is performed at each pixel in the infrared image (Figure 1c). By virtue of the absolute orientation sensing available on the smartphone and the ability to determine the 3D position of an observed point in the Kinect color image, the match is performed in a common 3D coordinate frame (world reference frame).

In this paper, we present results using the Kinect for Windows v2 sensor, which includes a time of flight depth camera. Other depth sensors may also be used. For example, we have a CrossMotion implementation that uses the first Kinect for Windows sensor which is based on a structured light depth sensor.

The algorithm may be summarized as:

1. Find device acceleration: During runtime, mobile device acceleration is continually transmitted to a host PC (Figure 1a).
2. Find image motion: Simultaneously, dense optical flow is computed on the Kinect infrared image (Figure 2a). Each flow vector is converted to a 3D motion using the depth image, and transformed to the coordinate frame of the mobile device.
3. Estimate image acceleration: 3D acceleration is estimated by a Kalman filter at each point of the image, with the 3D flow at the point provided as input (Figure 2b).
4. Match device and image accelerations: The difference between image and device acceleration is computed at each pixel in the Kinect infrared image (Figure 2cd). Small values indicate the possible presence of the device at those pixel locations.

We next describe each of the above steps in detail.

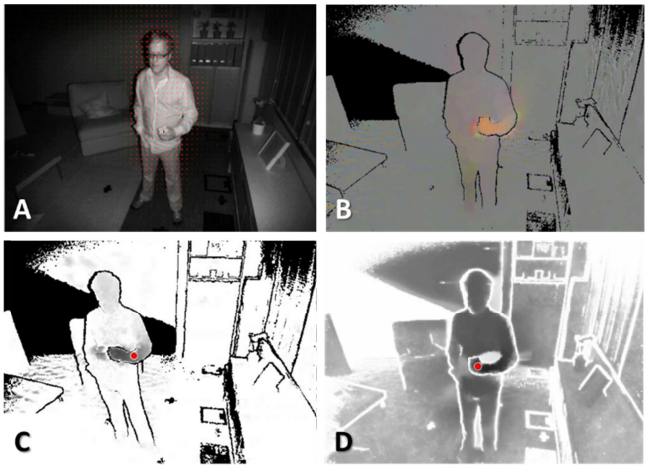


**Figure 2. Stages of the CrossMotion algorithm: (a) dense optical flow is computed from the infrared image; (b) per-pixel acceleration estimates using Kalman filter; (c) instantaneous match image (darker is better) and (d) filtered match image.**

## 3.1 Device Motion

Many mobile device APIs offer real time device orientation information. Today orientation is computed by combining information from the onboard accelerometers, gyroscopes and magnetometers. Because this orientation is with respect to magnetic north (as measured by the magnetometer) and gravity (as measured by the accelerometer, when the device is not moving), it is often considered an "absolute" orientation. In the work reported in this paper, the mobile device reports orientation in a standard "ENU" (east, north, up) coordinate system. While magnetic north is disturbed by the presence of metal and other magnetic fields present in indoor environments, in practice it tends to be constant in a given room. For our purposes it is only important that magnetic north not change dramatically as the device moves about the area imaged by the Kinect sensor.

Mobile device accelerometers report device acceleration in the 3D coordinate frame of the device. Having computed absolute orientation using the magnetometers, gyros and accelerometers, it is easy to transform the accelerometer outputs to the ENU coordinate frame and subtract acceleration due to gravity. Of course, the accuracy of this estimate of device acceleration in the ENU coordinate frame is only as good as that of the orientation estimate. While many smartphone SDKs include functions to report absolute orientation, for reasons explained later we implement our own algorithm.

Our prototype implementation of CrossMotion transmits this device acceleration (ENU coordinates, gravity removed) over WiFi to the host PC that performs sensor fusion.

## 3.2 Image Motion

The CrossMotion algorithm compares the 3D acceleration of the mobile device with 3D acceleration observed in video. Our approach to find acceleration in video is to first compute the velocity of movement at every pixel in the Kinect infrared image using a standard optical flow technique. This 2D image-space velocity is augmented with depth information and converted to velocity in real world 3D coordinates (meters per second). Acceleration is estimated at each point in the image using a Kalman filter. We next describe each of these steps in detail.

### *3.2.1 Finding 2D Velocity with Optical Flow*

Rather than track the position of a discrete set of known objects in the scene, image motion is found by computing dense optical flow on the entire Kinect infrared image. Dense optical flow algorithms model the motion observed in a pair of images as a displacement $u, v$ at each pixel. There are a variety of optical flow algorithms. Our prototype implementation uses the algorithm proposed by Brox et al. [1], which is known for its accuracy. While optical flow is typically used to compute the motion forward from time $t-1$ to the frame at time $t$, for reasons explained later CrossMotion computes flow from the current frame at time $t$ to the frame at time $t-1$. The velocity $u, v$ at each point $x, y$ we denote as $u_{x,y}$ and $v_{x,y}$. We note that $x, y$ are integer-valued, while $u, v$ are real-valued.

### *3.2.2 Converting to 3D Motion*

Depth cameras such as the Microsoft Kinect sensor report distance to the nearest surface at every point in its depth image. The Microsoft Kinect for Windows v2 SDK provides an API to compute the 3D position of a point in the depth camera in real world units (meters). We denote the 3D position corresponding to a 2D point $x, y$ in the infrared image at time $t$ as $\mathbf{z}_{x,y,t}$.

Rather than convert 2D velocities (as computed by optical flow) to 3D quantities directly, CrossMotion uses a Kalman filter-based technique that estimates velocity and acceleration at each pixel.

### *3.2.3 Estimating Acceleration*

CrossMotion uses a Kalman filter to estimate acceleration of moving objects in the image. The Kalman filter incorporates our knowledge of sensor noise and is recursive (that is, it incorporates all previous observations). The technique thus allows much better estimates of acceleration compared to the approach of using finite

differences. While it is beyond the scope of the paper to fully explain the Kalman filter (see [17] for a good introduction), we attempt to describe the basics of the technique by way of explaining the particular formulation used in CrossMotion.

The Kalman filter is closely related to the simpler "exponential" filter which computes a smoothed estimate $x_t$ of a scalar $z_t$ using the recursive relation:

$$x_t = x_{t-1} + \alpha(z_t - x_{t-1})$$

where the gain $\alpha \in (0,1)$ controls the degree to which the filter incorporates the "innovation" $z_t - x_{t-1}$. The smaller the gain, the less the filter follows the observation $z_t$, and the more the signal is smoothed. An improved version of the exponential filter is

$$x_t = x_t^* + \alpha(z_t - x_t^*)$$

where $x_t^*$ is a prediction of $x_t$ given $x_{t-1}$ (for example, by assuming constant velocity). The Kalman filter is essentially this "improved" exponential filter, and moreover includes a principled means to set the value of the gain given our uncertainty in both the prediction $x_t^*$ and observation $z_t$.

For our problem of estimating acceleration from image motion, we first consider the motion of a single object in 3D. The familiar equations of motion predict the object's position $\mathbf{x}_t^*$, velocity $\mathbf{v}_t^*$ and acceleration $\mathbf{a}_t^*$ from previous values, $\mathbf{x}_{t-1}$, $\mathbf{v}_{t-1}$, and $\mathbf{a}_{t-1}$:

$$\mathbf{x}_t^* = \mathbf{x}_{t-1} + \mathbf{v}_{t-1}\Delta t + \frac{1}{2}\mathbf{a}_{t-1}\Delta t^2$$

$$\mathbf{v}_t^* = \mathbf{v}_{t-1} + \mathbf{a}_{t-1}\Delta t$$

$$\mathbf{a}_t^* = \mathbf{a}_{t-1}$$

Given observation $\mathbf{z}_t$ of the 3D position of a tracked object, we correct the predictions of position, velocity and acceleration with

$$\mathbf{x}_t = \mathbf{x}_t^* + \mathbf{k}_x * (\mathbf{z}_t - \mathbf{x}_t^*)$$

$$\mathbf{v}_t = \mathbf{v}_t^* + \mathbf{k}_v * (\mathbf{z}_t - \mathbf{x}_t^*)$$

$$\mathbf{a}_t = \mathbf{a}_t^* + \mathbf{k}_a * (\mathbf{z}_t - \mathbf{x}_t^*)$$

where $*$ denotes element-wise multiplication. Kalman gains $\mathbf{k}_x$, $\mathbf{k}_v$, $\mathbf{k}_a$ relate the innovation, or error in the prediction of position, to changes in each of our estimates of position, velocity and acceleration. Kalman gain is computed as described in [17], and is related to our uncertainty in our predictive model $\mathbf{x}_t^*$ and observations $\mathbf{z}_t$. In particular, it is crucial to assign a high uncertainty to our estimate of acceleration $\mathbf{a}_t$ to reflect our belief that acceleration of the object varies over time (indeed, this is the quantity we wish to estimate). Similarly, our uncertainty in $\mathbf{z}_t$ is related to the noise of our sensor.

Finally, we note that the usual formulation of Kalman gain is time-varying. However, if the uncertainty of our predictive model and observations is constant, Kalman gain converges to a constant value [17], as presented above. This leads to a simplified implementation of the update equations, and further underscores the relationship between the Kalman filter and the simpler exponential filter.

### *3.2.4 Incorporating Flow*

CrossMotion maintains a Kalman filter of the form described above to estimate 3D acceleration at each pixel location in the image. We denote our estimated position, velocity and acceleration at each pixel location $x, y$ as $\mathbf{x}_{x,y,t}$, $\mathbf{v}_{x,y,t}$ and $\mathbf{a}_{x,y,t}$ respectively.

Optical flow information is used in two ways: first, the flow at a point in the image is a measurement of the velocity of the object under that point. It thus acts as input to our estimate of acceleration using the Kalman filter. Second, we can use flow to propagate motion estimates spatially, along patches of the image whose motion is being estimated. In this way the Kalman filter can use many observations to accurately estimate the acceleration of a given patch of an object as it moves about the image.

Flow quantities $u_{x,y}$ and $v_{x,y}$ (which we abbreviate as $u$ and $v$) are incorporated by predicting $\mathbf{x}_{x,y,t}^*$, $\mathbf{v}_{x,y,t}^*$, and $\mathbf{a}_{x,y,t}^*$ from $\mathbf{x}_{x+u,y+v,t-1}$, $\mathbf{v}_{x+u,y+v,t-1}$, and $\mathbf{a}_{x+u,y+v,t-1}$ using the equations of motion as above. In practice, $\mathbf{x}_{x,y,t}$, $\mathbf{v}_{x,y,t}$ and $\mathbf{a}_{x,y,t}$ are stored as a 2D array the same dimension as the Kinect infrared image, but because $x+u$ and $y+v$ are real valued, quantities $\mathbf{x}_{x+u,y+v,t-1}$, $\mathbf{v}_{x+u,y+v,t-1}$, and $\mathbf{a}_{x+u,y+v,t-1}$ are best computed by bilinear interpolation on the 2D array. Finally, observation $\mathbf{z}_{x,y,t}$ is simply the 3D world coordinate position at image coordinates $x, y$. In this process, the Kalman filter at $x, y$ updates motion estimates found at $x+u, y+v$ in the previous time step, and motion estimates follow along or "track" the objects whose motion is being estimated.

This interpolation finally motivates computing optical flow in reverse fashion, from time $t$ to time $t-1$: $u_{x,y}$ and $v_{x,y}$ are defined for all integer values $x, y$. Computing flow in the usual fashion from time $t-1$ to time $t$ might leave some pixels without "predecessors" from the previous frame, even if previous motion estimates are distributed across multiple pixels using bilinear interpolation. Computing flow from time $t$ to time $t-1$ avoids this problem.

## 3.3 Sensor Fusion

### *3.3.1 Common Coordinate System*

In the following, we describe a one-time calibration procedure which obtains the camera's orientation with respect to the ENU coordinate frame of the mobile device. Motion observed in the camera may then be transformed to ENU coordinates and compared to device accelerations directly.

While there are many ways to compute the relative orientation of the Kinect camera to the coordinate system used by our mobile device, we adopt a straightforward semi-automatic procedure that is easy to implement and gives good results. First the mobile device is placed display-side down on a plane that is easily observed by the Kinect camera, such as a wall or desk. Viewing the color video stream of the camera, the user clicks on three or more points on the plane.

The 3D unit normal $\mathbf{n}_k$ of the plane in Kinect coordinates is computed by first calculating the 3D position of each clicked point and fitting a plane by a least-squares procedure. The same normal $\mathbf{n}_w$ in ENU coordinates is computed by rotating the unit vector $\mathbf{z}$ (out of the display of the device) by the device orientation. Similarly, gravity unit vector $\mathbf{g}_k$ in camera coordinates is taken from the 3-axis accelerometer built in to the Kinect sensor. Gravity $\mathbf{g}_w$ in the ENU coordinate frame is by definition $-\mathbf{z}$.

The 3×3 rotation matrix $\mathrm{M}_{\mathrm{kinect}\rightarrow\mathrm{world}}$ that brings a 3D camera point to the ENU coordinate frame is calculated by matching the normals $\mathbf{n}_k$ and $\mathbf{n}_w$, as well as gravity vectors $\mathbf{g}_k$ and $\mathbf{g}_w$, and forming orthonormal bases K and W by successive cross products:

$$\mathbf{k}_1 = \mathbf{n}_k,\ \mathbf{k}_2 = \frac{\mathbf{n}_k \times \mathbf{g}_k}{\|\mathbf{n}_k \times \mathbf{g}_k\|},\ \mathbf{k}_3 = \mathbf{n}_k \times \mathbf{k}_2,\ \mathrm{K} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \mathbf{k}_3 \end{bmatrix}$$

$$\mathbf{w}_1 = \mathbf{n}_w,\ \mathbf{w}_2 = \frac{\mathbf{n}_w \times \mathbf{g}_w}{\|\mathbf{n}_w \times \mathbf{g}_w\|},\ \mathbf{w}_3 = \mathbf{n}_w \times \mathbf{w}_2,\ \mathrm{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{bmatrix}$$

$$\mathrm{M}_{\mathrm{kinect}\rightarrow\mathrm{world}} = \mathrm{K}^{-1}\mathrm{W}$$

While this procedure uses a mobile device to place the Kinect camera in ENU coordinates, we note that this calibration need only be performed once when the camera is mounted. An unfamiliar device will work with the system without further calibration as long it also reports orientation in ENU coordinates.

### *3.3.2 Matching*

3D image accelerations are estimated at each pixel and transformed to the ENU coordinate system as described above. The acceleration observed at each pixel may be compared directly to the device acceleration $\mathbf{d}_t$:

$$r_{x,y,t} = \sqrt{\left\|\mathbf{a}_{x,y,t} - \mathbf{d}_t\right\|^2}$$

Regions of the image that move with the device will give small values of $r_{x,y,t}$. In particular, the hope is that pixels that lie on the device will give the smallest values (Figure 2c). If we assume that the device is present in the scene, it may suffice to locate its position in the image by finding $x^*, y^*$ that minimizes $r_{x,y,t}$. However, other objects that momentarily move with the device, such as those rigidly attached (e.g., the hand holding the device and the arm) may also match well.

In practice, locating the device by computing the instantaneous minimum over $r_{x,y,t}$ will fail to find the device when it is momentarily still or moving with constant velocity. In these cases device acceleration may be near zero and so matches many parts of the scene that are not moving, such as the background. We address this by smoothing $r_{x,y,t}$ with an exponential filter to obtain $s_{x,y,t}$. This smoothed value is "tracked" using optical flow and bilinear interpolation, in the same manner as the Kalman motion estimates (Figure 2d). Small values over the smoothed value $s_{x,y,t}$ will pick out objects that match device acceleration over the recent past (depending on smoothing parameter $\alpha$) and "remember" the moments when some non-zero device acceleration uniquely identified it in the image. In the case where the device stops moving, the small values $s_{x,y,t}$ will stay with the device for some time, hopefully until the device moves again.

Our current implementation takes a further optional step to avoid the problem of matching static backgrounds by adding a small penalty to pixel locations that exhibit little motion.

An important consideration in performing the above matching process is that the latency of the Kinect sensor is much greater than that of the mobile device, including WiFi communications. Without accounting for this difference, the measure of similarity $r_{x,y,t}$ will be inaccurate. CrossMotion accounts for the relative latency of the Kinect sensor by artificially lagging the mobile device readings by some small number of frames. In our prototype implementation this lag is tuned empirically to five frames, approximately 80ms.

Figure 3 shows a typical trace of device acceleration and image acceleration at $x^*, y^*$. Considering that these values are computed in very different ways, they track each other surprisingly well.

In some applications it may not be appropriate to assume that the device is in the scene. For example, the user holding the device may leave the camera's field of view. In this case the minimum value over $s_{x,y,t}$ can be checked against a threshold to reject matches of poor quality. We denote the minimum value at $x^*, y^*$ as $s^*$.

## 4. IMPLEMENTATION

Our prototype implementation of CrossMotion uses a Microsoft Kinect for Windows v2 sensor and a Nokia Lumia 920 running Windows Phone 8. Device acceleration and orientation information is transmitted to a host PC over WiFi at a rate of approximately 50Hz. The Kinect camera is configured using the Microsoft Kinect for Windows v2 Developer Preview SDK to acquire infrared and depth images at resolution 512×424 at 30Hz. CrossMotion uses the Brox optical flow API in OpenCV 2.4.7. This optical flow implementation uses the GPU to achieve real time performance, and is written in Nvidia's CUDA GPU programming language. Our host PC runs Windows and includes an Nvidia GeForce GTX 660Ti graphics card hosting CUDA 5.5.

All image processing runs at 30Hz. Optical flow calculations are performed on the full resolution infrared image. Per-pixel Kalman filter updates and sensor fusion matching is implemented in CUDA. Optical flow parameters (e.g., smoothness) and Kalman filter parameters (sensor noise and process noise) are derived empirically.

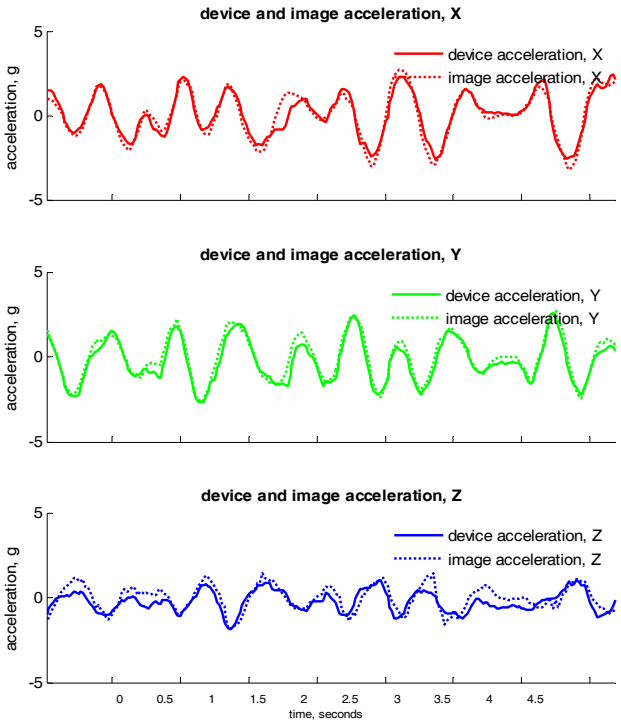


**Figure 3. Device acceleration compared to image acceleration at the point $x^*, y^*$ in each of the coordinate axes. The device was held in by the user and moved back and forth. Note how device and image acceleration track each other closely.**

### *4.1.1 Computing Orientation*

Today's mobile devices combine gyros, accelerometers and magnetometers to compute absolute orientation. Gyros measure angular velocity and are used to provide fast, accurate updates to orientation, while the noisier magnetometers and accelerometers are used to take observations of magnetic north and gravity. The absolute orientation information in the observations of magnetic north and gravity is applied to gradually remove the inevitable drift encountered when using gyros alone. Accelerometers measure the direction of gravity only when the device is motionless. Thus, if the device is undergoing significant motion, the orientation estimate may be distorted for some time. We find that when using the built-in Windows Phone 8 API for orientation, orientation estimates are disrupted by vigorous motion, and several seconds are required to recover. Because CrossMotion relies on accurate orientation information to transform accelerometer outputs to the ENU

coordinate frame, when using the built-in API vigorous motion causes CrossMotion fusion to fail for a few seconds.

Our own implementation of an algorithm to compute absolute orientation allows us to make trade-offs that differ from the built-in implementation. Our implementation opts for near instantaneous drift correction when the device is deemed motionless, and otherwise performs updates to the orientation estimate using the gyro running at 250Hz.

# 5. EVALUATION

CrossMotion may be useful in applications where the user interacts with a large display using speech and gesture, or with a phone application that is connected with the display. In these settings it may be valuable to know the identity of the user, their location in front of the display, and possibly the location of the device.

To evaluate CrossMotion's performance we considered two particular modes of use that may be useful in interacting with large displays by mobile device: gesturing towards the display while holding the device in the hand, and wearing the device in a pocket while moving about in front of the display.

## 5.1 Experiment 1: Gesturing with the Device

In our first experiment, we tested CrossMotion's ability to track the mobile device as the user gestures in a front display, holding the device in the moving hand. This configuration approximates how a user might interact with a remote display application by gesturing with the device. As we are interested to see if CrossMotion performance is impacted by increasing distance from the camera, we varied the participants' distance to the display. In a second set of trials we introduced a distractor user (one of the experimenters) which did their best to mimic the motion of the participant. This was repeated for each distance condition, giving a 2×2 design with distance and the presence of a distractor as experimental conditions. For the purposes of this study we take as ground truth the output of the Kinect SDK body tracking, which includes the 3D position of the hands, hip and middle of the spine.

### *5.1.1 Procedure*

We solicited five adult participants (one female, four male) from the authors' institution. All were right-handed and were familiar with using smartphones.

Participants were instructed to stand on a marked spot on the floor 1.5m from the display (near condition) or 2.5m from the display (far condition), and were directed to attend to the display, a 24" LCD. The Kinect camera was mounted above the display, about 180 cm above the floor, to observe the area in front of the display. An experimenter instructed the participant to hold the device in their right hand, orienting it so that it is approximately vertical, with the smartphone's screen facing them.

The experimenter explained the following set of trials, repeated for each distance condition: a letter was shown on the display for four seconds. During that time, the participant was instructed to "draw" the letter with the device by moving the device in the space in front of their body, and to finish approximately in four seconds. This duration was conveyed by a progress bar animation which finished at the end of four seconds. At the end of four seconds the screen was blanked. After a three second pause, during which time the participant was instructed to return to a comfortable center position in front of their body, another letter was presented. After a few practice trials, the participant performed the same task for each of the unistroke letters in the alphabet (17 letters, e.g., B, C, D, etc.). We note the letter stimulus was employed to merely cause the user to exhibit a variety of gestures, not to test letter recognition. Participants were told that the order of strokes in performing the letter gestures was unimportant.

This set of trials was repeated but with the introduction of a distractor user (one of the experimenters) who did their best to mimic the precise motion of the participant. This distractor stood next to the participant and performed the same letter in the same stroke order, and at the same pace. This second set of trials was included to test the performance of CrossMotion in the presence other people, as well as give an initial indication of how easy it is to "spoof" a user's motion.

During every CrossMotion frame, including pauses, software logged the 3D position corresponding with the algorithm's best match as well as the right hand joint position returned by the Kinect SDK body tracker. Software also logged whether the position of CrossMotion's solution $x^*, y^*$ lies on the participant's body as determined by the Kinect SDK's "body index" image.

### *5.1.2 Results*

Across all trials, of the 76,296 video frames (about 41 min), 1,629 frames (2.1%) occurred when the WiFi connection from the mobile device dropped temporarily. Of the rest, 98.9% correctly placed $x^*, y^*$ on the participant's body. Taking the right hand position as ground truth, average error in 3D was 6.9cm (s.d. = 11.6cm). Position error in the "near" condition was 6.5cm (s.d. = 8.7cm), while error in the "far" condition was 7.4cm (s.d. = 14cm). The increase in error in the "far" condition may be attributed to one participant in that condition. We suspect that in that trial the magnetometer had fallen out of calibration. In any case, there is no reason to suspect that the difference in near and far is statistically significant. CrossMotion never placed its solution on the distractor user, thus the distractor condition had no effect on the results, and we do not report them separately.

## 5.2 Experiment 2: Wearing the Device

In the second experiment, we tested CrossMotion's ability to reliably pick out the person carrying the phone; i.e., does $x^*, y^*$ lie on the user's body. We are also interested to see if CrossMotion can further locate the device on the person. In many circumstances we might not expect to be able to locate the device, since, as noted earlier, CrossMotion does not track the device, but instead finds regions of the image which are consistent the motion of the device. Thus parts of the user's body which are rigidly attached to the device may match as well as locations on the device itself.

We consider two locations of the device on the body: hanging from a lanyard hung around the user's neck, and placed in the right front pocket of a jacket. These positions were selected because they both have good analogues in Kinect's body tracking outputs: the lanyard position matches well with the "middle spine" position, while the jacket pocket corresponds well with the "right hip" position. We used the same participants as in the first experiment.

### *5.2.1 Procedure*

The placement of camera and display was the same as in the first experiment. In the "middle spine" condition, the phone was placed in a wearable lanyard-style badge holder, of the kind sometimes used at conferences. In the "right hip" condition, the phone was placed in the right front pocket of a light fleece jacket. In both conditions, the participant was instructed to move to any one of four markers on the floor when the remote display indicated, but to do so in way that they continually faced the remote display. Two of the markers were the same used in the first experiment, while two more were located at distance of 2m from the display, but at a distance of 0.5m on either side of the line connecting the first two points. The

four markers thus formed a diamond shape. The participant moved a total of 34 times in a session (4 seconds between each move).

Software logged the same information as in the first experiment, except rather than logging the right hand position returned by the Kinect SDK, the right hip and middle spine positions were logged.

### *5.2.2 Results*

Across all trials, of the 43,060 video frames (about 24 min) analyzed, 1,352 frames (3.1%) occurred when the WiFi connection from the mobile device dropped temporarily. These frames can be considered invalid. Of the rest, 99.0% correctly placed $x^*, y^*$ on the participant's body. Considering "middle spine" and "right hip" positions together, average error in 3D was 20.5cm (s.d. = 14.3cm). Position error in the "middle spine" condition was 17.1cm (s.d. = 7.8cm), while error in the "right hip" condition was 23.9cm (s.d. = 20.7cm). The results suggest a trend of larger error for the right hip, which is consistent with informal observation of the system in observation. In part this may be explained by the inherent lack of agreement between our choice of ground truth (Kinect SDK tracked body points) and the actual position of the device. In fact, if we examine the average error as a 3D vector quantity we see that much of the error is in the z coordinate, consistent with the fact that the device is worn outside the body, while the body skeleton reported by Kinect typically lies inside the body.

Comparing average error in position across both experiments, we note that error in the second experiment is greater than that of the first. This is unsurprising, since (as discussed in the next section) a smaller moving object more precisely indicates position than a large rigid moving object. In fact, we are pleasantly surprised at how well the device is tracked when it is worn.

## 5.3 Discussion

While CrossMotion does not require an appearance model of the device, in order to detect and track the user it requires that the camera have a line of sight to some part of the user that moves with the device. Contrast this with other vision-based approaches that require observing the device directly. Our example sequences demonstrate that the technique will often track the person holding the device if the device is in the user's shirt pocket or pants pocket (Figure 4). In these cases CrossMotion finds objects that move precisely with the device. This behavior may be appropriate for applications that require tracking at the level of the user. In those cases placing the point $x^*, y^*$ anywhere on the user would be a positive match.

Furthermore, our examples show that CrossMotion often finds the position of the device itself, even if it is not directly in view. This ability requires that the device motion is unique against the background of the rest of the motion in scene, at least periodically. While our initial experiments are promising examples of where this is often true, it will require more work to show that this works generally since it depends on the surrounding context of motion. For example, the ability to localize the device on the user's leg when it is in their pants pocket depends in part on the leg's occasionally moving differently than the torso. Conversely, when holding the device it is not uncommon for it to find some part of the arm holding the device (however, we are surprised by the system's ability to find the phone in the user's shirt pocket, since it would seem that any point on the torso should match equally as well).

By the same token, because it matches device motion to image motion, CrossMotion cannot initially detect the mobile device if it does not move. If the device is being held by its user, it is likely that even a slight motion of the user is enough to establish reliable tracking, but if the device is lying on a desk, CrossMotion will not be able to detect it against a static background.

Our initial experiments show that a simple threshold on the value of $s^*$ is not enough to reliably determine if the smartphone is in the view of the camera or off camera. By inspecting some test sequences, it seems that this is primarily because during moments when the user is not moving, static parts of the physical environment imaged by the camera can sometimes give good matches. In our own experiments we have been reluctant to add application-specific heuristics to the basic localization algorithm to address this problem. Partly, this is to investigate the power of the simplest version of the algorithm. However, we note that there are a number of pre-processing steps that would be reasonable to incorporate in a production version of CrossMotion. For example, the static background could be modeled and removed from consideration. Similarly, in many applications it may be possible to use the "body index" segmentation made available by the Kinect SDK. This coarse segmentation assigns each pixel of the depth map to one of up to six people or the background, and could be used as a mask or given some hysteresis at the level of the user.

An interesting question is whether it is possible for one mobile device user to "spoof" another mobile device user by mimicking their motion simultaneously. While our first experiment suggests that it is very difficult to do, in theory it is possible. In such cases it should be possible to at least detect that two devices appear to match the same image motion, or themselves, and block further action. Another approach is to use CrossMotion's results to bootstrap further vision-based analysis to achieve a two-factor authorization. For example, device or user identity and location within the image could seed a targeted face recognition process.

Finally, we note that the CrossMotion fusion described in this paper assumes that the device motion and corresponding image motion consists only of translation, and not rotation. Consider, for example, spinning the mobile device in place. In this case CrossMotion may not find the device, since the device accelerometers will give very low values, while there may be extensive motion observed about the device. In practice this rarely is an issue, as the rotation of the device and user is often accompanied by significant translation. We envision extending the model to more completely model device motion by including angular velocity as reported by device gyros and observed by rotation in the optical flow field about the device. Including angular velocity may improve the precision and robustness of the matching process.

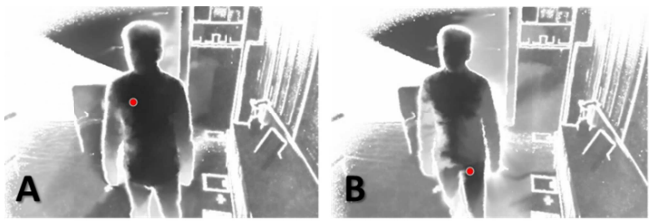


**Figure 4. Image of $s_{x,y,t}$ for phone located in (a) shirt pocket and (b) pants pocket. Darker pixels indicate better match.**

# 6. APPLICATIONS

Because it makes few assumptions about the appearance of the mobile device or the user carrying it, CrossMotion can be used in a wide variety of application settings.

Robust indoor person tracking is useful in providing location-based services to individuals. Way-finding and delivering targeted advertising are two often-cited example applications. CrossMotion can be used in many indoor person tracking applications where a

computer vision-based approach would be acceptable, and where users are likely carrying a mobile device such as a smartphone or tablet. Computer vision-based approaches to person tracking are attractive because they can yield fine-grained position information but they typically either rely on visible markers (as in [16]) or complex models of the user's appearance. Other approaches require the user to carry new devices and therefore are unlikely to be adopted outside of critical applications. As an example way-finding application, consider providing directions to a visitor as they approach key hallway intersections, or when they leave an elevator. A CrossMotion-equipped depth camera could be installed at these locations to direct the visitor.

CrossMotion's ability to track and identify users by way of their mobile device makes it particularly appropriate for various device association problems. For example, a connection between a mobile device user and a wall display could be automatically established as the user approaches the display. Knowing the precise position with respect to the display can be used in number of ways. For example, when there are multiple simultaneous users, the wall display can render each user's own set of virtual objects nearby. Users can "flick" objects from their device onto the display. Fine-grained person tracking and device association can also be used to detect when users bring their devices near to each other, which can used for a variety of applications [6].

CrossMotion may be useful in various settings where it is valuable to track objects of interest as they move about the environment, particularly when it is undesirable to attach visual markers to those objects. For example, young children wearing small wrist worn sensor packages could be tracked in school [4], a child's stuffed animal could be tracked throughout the house, while real animals could be tracked to study their patterns of movement. The Xbox One Skype app performs an automatic digital pan and zoom to capture the active participants in the room, based on simple image motion processing techniques. CrossMotion might be used to limit this selection to particular users. Similarly, given a future long-range depth camera, a calibrated pan/tilt/zoom camera might follow a particular soccer player on the field.

# 7. CONCLUSION

We introduce CrossMotion, a cross-modal sensor fusion technique to detect and track mobile devices and the people carrying them. Because it matches inertial device motion with motion observed in video, it makes very few assumptions about the appearance of either the device or the user. This paper details a real time implementation of the technique based on estimating image acceleration from optical flow. Our initial experiments with the technique demonstrate its ability to find the user reliably (99% of the time). In many cases it can find the device itself even when it is not in direct view of the camera. While there are a number of considerations in applying the technique, we believe it is a unique and potentially useful option in many ubiquitous computing scenarios.

# 8. REFERENCES


[1] Brox, T., Bruhn, A., Papenberg, N., and Weikert, J. High accuracy optical flow estimation based on a theory for warping. In *Proc. 8th European Conference on Computer Vision,* vol. 4. 2004. 25-36.

[2] Hinckley, K., Ramos, G., Guimbretiere, F., Baudisch, P., and Smith, M. Synchronous gestures for multiple persons and computers. In *Proc. UIST 2003*. 149-158.

[3] Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., and Gellersen, H. Smart-Its friends: A technique for users to easily establish connections between smart artefacts. In *Proc. Ubicomp 2001*. 116-122.

[4] Kawai, J., Shintani, K., Haga, H., and Kaneda, S. Identification and positioning based on motion sensors and a video camera. In *Proc. 4th IASTED Int. Conf. on Web-Based Education,* 2005.

[5] Maki, Y., Kagami, S., and Hashimoto, K. Accelerometer detection in camera view based on feature point tracking. In *Proc. IEEE/SICE Int'l Symp. On System Integration.* 2010. 448-453.

[6] Marquardt, N., Hinckley, K., and Greenberg, S. Cross-device interaction via micro-mobility and F-formations. In *Proc. UIST 2012*. 13-22.

[7] Mayrhofer, R., and Gellersen, H. Shake well before use: intuitive and secure pairing of mobile devices. *IEEE Transactions on Mobile Computing*, 8(6). 2009. 792-806.

[8] Olwal, A., and Wilson, A.D. SurfaceFusion: Unobtrusive tracking of everyday objects in tangible interfaces. In *Proc. Graphics Interface 2008.* 235-242.

[9] Plötz, T., Chen, C., Hammerla, N. Y., and Abowd, G.A. Automatic synchronization of wearable sensors and video-cameras for ground truth annotation- a practical approach. In *Proc. 16th Int. Symp. on Wearable Computers,* 2012.

[10] Rekimoto, J., Ayatsuka, Y., and Kohno, M. SyncTap: An interaction technique for mobile networking. In *Proc. Mobile CHI 2003.* 104-115.

[11] Rofouei, M., Wilson, A.D., and Brush, A.J. Your phone or mine?: Fusing body, touch, and device sensing for multi-user device-display interaction. In *Proc. ACM SIGCHI.* 2012. 1915-1918.

[12] Schmidt, D., Chehimi, F., Rukzio, E., and Gellersen, H. PhoneTouch: A technique for direct phone interaction on surfaces. In *Proc. UIST 2010.* 13-16.

[13] Shigeta, O., Kagami, S., and Hashimoto, K. Identifying a moving object with an accelerometer in a camera view. In *Proc. Int. Conf. Intelligent Robots and Systems,* 2008.

[14] Stein, S., and McKenna, S.J. Accelerometer localization in the view of a stationary camera. In *Proc. Ninth Conference on Computer and Robot Vision,* 2012. 109-116.

[15] Teixeira, T., Jung, D., and Savvides, A. Tasking networked CCTV cameras and mobile phones to identify and localize multiple people. In *Proc. Ubicomp,* 2010. 213-222.

[16] Want, R., Hopper, A., Falcão, V., and Gibbons, J. The active badge system. *ACM Transactions on Information Systems,* 10(1), 1992. 91-102.

[17] Welch, G., and Bishop, G. An introduction to the Kalman filter. Technical Report TR 95-041. University of North Carolina at Chapel Hill Dept. of Computer Science. 1995.

[18] Wilson, A.D., and Sarin, R. BlueTable: connecting wireless mobile devices on interactive surfaces using vision-based handshaking. In *Proc. Graphics Interface 2007.* 119-1.

[19] Wolfe, J.M. Guided search 2.0: a revised model of visual search. *Psychonomic Bulletin & Review 1, 2* (1994), 202-238.