# A Mobile Camera-Based Evaluation Method of Inertial Measurement Units on Smartphones

Lars Middendorf[1](✉), Rainer Dorsch[2], Rudolf Bichler[2], Christina Strohrmann[2], and Christian Haubelt[1]

[1] University of Rostock, Rostock, Germany
lars.middendorf@uni-rostock.de

[2] Bosch Sensortec GmbH, Reutlingen, Germany

**Abstract.** In order to support navigation, gesture detection, and augmented reality, modern smartphones contain inertial measurement units (IMU) consisting of accelerometers and gyroscopes. Although the accuracy of these sensors directly affects the soundness of mobile applications, no standardized tests exist to verify the correctness of the retrieved sensor data. For this purpose, we present a novel benchmark, which utilizes the camera of the phone as a reference to estimate the quality of its sensor data fusion. Our experiments do not require special equipment and reveal significant discrepancies between different phone models.

## 1 Introduction

A large number of mobile and ubiquitous applications for smartphones rely on a sound and stable orientation estimation. For example, a step counter [1] can improve the accuracy of indoor localization when the GPS signal becomes unavailable [2]. However, both the occurrence of a step and its direction must be measured precisely to track the relative movement of the user. Similarly, augmented reality applications [3] also require the exact orientation of the device to display context-dependent information on the screen, which should match the real environment as close as possible. For instance, the navigation application presented by [4] shows the direction to the destination as a perspective arrow within the camera image and counts the remaining number of steps. In addition, the detection of gestures for user authentication strongly depends on the accuracy of the IMU [5].

Inertial sensors are often built as micro-electro-mechanical systems (MEMS) to reduce size, power consumption, and production costs. According to [6], possible error sources are a constant bias, thermo-mechanical noise, bias instabilities due to flicker noise, temperature effects, and calibrations errors. Due to the integration, even small deviations can lead to significant differences. For example, a constant bias in the gyro causes a continuously growing angle error. As a result, the soundness of the sensor data at the application-level, is directly affected by the accuracy of the IMU.

As an introductory example for the relevance of an IMU benchmark, we show the results of a pedestrian tracking application running on three different but

B. Mandler et al. (Eds.): IoT 360° 2015, Part II, LNICST 170, pp. 362–372, 2016.
DOI: 10.1007/978-3-319-47075-7_41

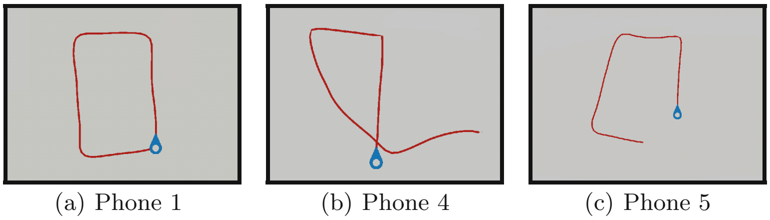


**Fig. 1.** Rectangular path measured on different phones.

anonymized phones (Fig. 1). Only *Phone 1* has correctly captured the rectangular shape, while the path of *Phone 5* is slightly deformed. On *Phone 4*, the second half of the path is flawed due to the integration of erroneous directions.

Consequently, the development of sensor-based applications requires extensive testing to deal with the large variations between different IMUs. As a possible solution, we present a benchmark for Android phones (Fig. 2), which measures both the static and dynamic accuracy of the sensor data fusion through the usage of the built-in camera and a specially developed reference pattern. For this purpose, the application simultaneously records sensor data and captures a video stream, which is then used to reconstruct the 3D orientation of the phone. Since the camera represents a potential error source, the latency between camera and sensor is compensated through a calibration procedure.

In particular, the innovations of our benchmark can be summarized as follows:

1. The camera of the smartphone is used to estimate a reference orientation. If the rotation vectors obtained from sensor and camera match, we can assume the correctness of the IMU.
2. Our benchmark computes an error metric, which permits the quantitative comparison of multiple test runs and different inertial sensors.
3. The test can be performed without additional equipment like an external camera system and requires only a sheet of paper containing the pattern.

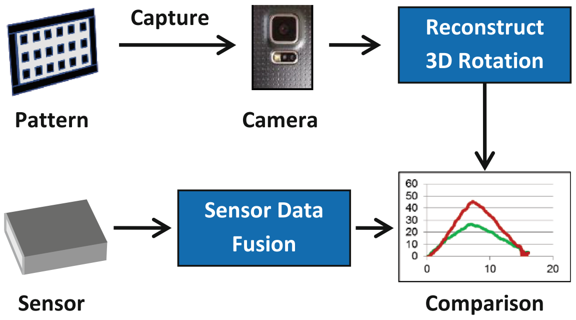


**Fig. 2.** Data flow model of our sensor benchmark.

The rest of this paper is organized as follows: In Sect. 2, we present related approaches for the evaluation of inertial sensors and techniques for optical tracking. The concept of our benchmark is specified in Sect. 3, Sect. 4 contains experimental results, and Sect. 5 is reserved for conclusion.

## 2 Related Work

Several methods for the calibration of IMUs through image processing already exist. For instance, the work presented in [7] utilizes an optical tracking system (*Optotrack*) consisting of three IR cameras and three LEDs, which are attached to the IMU in order to determine its absolute orientation. In addition, the relative orientation between a camera and an IMU can be estimated by capturing several images of a chessboard and correlating the vertical lines with the gravity vector [8]. Further, also the mirrored image of the camera is suitable for self-calibration of the sensor [9]. Our approach takes the opposite direction and assumes a fixed orientation between camera and IMU to evaluate its accuracy.

Determining the orientation of the device from a single camera image requires a set of *feature points* in screen space and their corresponding coordinates in world space. For this purpose, iterative algorithms like [10] provide a robust solution in case of inaccurate input data. Although four coplanar points are sufficient to compute the relative camera orientation [11], a larger number of features increase both the accuracy and the robustness of the result [12]. Our benchmark utilizes the algorithm presented by [13], which accepts a variable number of corresponding points for calculating the camera orientation and has been implemented in the *OpenCV* library [14].

The detection of feature points usually depends on distinct markers, which encode their world position in form of different shapes [15]. Alternatively, each marker can be uniquely identified by patterns of black and white blocks [16]. In comparison to a more regular chessboard [17], this technique permits extensive rotations of the camera because only a sub-set of the markers must be visible in each frame. However, for use cases like the video-based augmented reality conferencing system [18], six markers are adequate to locate a virtual shared whiteboard. Also, the *VideoMouse* supports six degrees of freedom [19] and contains a camera to record a regular pattern of circular markers. Due to the restricted viewing angle of the camera, only a small and quickly changing section of the grid can be observed, so that additional codes within the markers are necessary for global localization.

Since we are mainly interested in the relative rotation of the camera, we have chosen a regular pattern of uniform squares. It resembles both [13,19] but can be detected in a similar way to QR codes [20]. While one rectangle offers four feature points and is therefore sufficient to estimate the camera rotation, each additional marker further increases the accuracy.

## 3 Benchmark

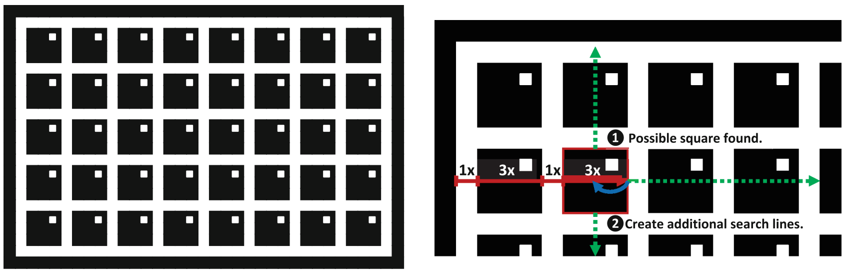
The test procedure (Fig. 2) consists of an image processing step, which determines the relative rotation of the phone, the recording of sensor data, and the final evaluation. In particular, the reconstruction of the rotation and the analysis of the results represent the main contributions of this paper, which are discussed in the following sections.

### 3.1 Overview

In order to compute the rotation of the phone from a camera frame, a set of corresponding feature points must be found in the image. For this purpose, we have designed a reference pattern (Fig. 3a), according to the following requirements:

– The benchmark is performed in office environments under different lighting conditions and backgrounds.
– The benchmark runs on a mobile phone with memory restrictions and limited computational power.
– Since the phone is rotated during the test, a small part of the pattern must be sufficient to compute its orientation.

Our pattern contains a regular grid of uniforms squares, which are enclosed by a thick border. All elements are either black or white to reduce memory usage as well as the complexity of image processing. As a consequence, the contrast is increased so that the quality of the camera becomes less important. In addition, the thick border helps to distinguish the squares of the pattern from the background. Since the rotation of a single square is ambiguous for steps of 90°, we include an internal marker to determine its actual rotation. Further, the edge length of each square is three times larger than the width of the border and the padding, which is used for identification. As a result, the pattern can be printed in any size and can contain an arbitrary number of squares as long as its relative proportions remain constant.



(a) Reference Pattern (b) Detection of Squares

**Fig. 3.** Reference pattern developed for the purpose of the tracking; it is used for measuring the orientation of the camera. (Color figure online)

The image processing algorithm first looks for the pattern and collects screen space coordinates of each square. Although the four corners of a square are sufficient to compute the rotation of the camera [21], a larger number of squares significantly increases the accuracy. Due to the regularity of the pattern, already detected squares are used as hints to guide the further search. In a second step, the relation between squares is examined to assign world space coordinates, which are then used for 3D reconstruction. In the next sections, these two steps are described in more detail.

## 3.2 Detecting Possible Squares

The goal of this first step is to compute a list of *possible squares* from the 8-bit greyscale image. We start by applying a binary threshold for simplification, so that the resulting pixels are either set to 0 or 255. Experiments have shown that a threshold of 128 is sufficient to highlight the pattern. Due to the white balancing in the camera of modern phones, an adaptive filtering is not necessary.

The next step utilizes the unique proportion between squares and spacing to separate the pattern from the background. For this purpose, we trace a set of random lines through the image and look for alternating black and white segments. According to the design of the pattern (Fig. 3b), the characteristic ratio between these segments corresponds to $\approx 3:1$ if the camera is aligned in parallel to the raster. However, in order to improve the detected of squares at steep angles, the ratio of $\approx 3:1$ is automatically adapted by our algorithm.

If a run of at least four consecutive segments has been found, the current position might be located at the border of square region (red). Hence, the center of the square can be reached by stepping back half of the distance (blue).

Since only lines, which are more or less aligned to the pattern, satisfy these conditions, we can speed up the search by creating more appropriate lines (green). After a square has been found, additional search lines are constructed through the midpoints of opposing sides. Consequently, they are automatically aligned to the grid and intersect most likely several other squares. As a result, the entire pattern can be retrieved from a single square in two iterations. Finally, each region is flood-filled with a different color and passed to the next stage, which extracts the contour and screen space coordinates of the square.

## 3.3 Extracting Valid Squares

Currently, each *possible square* corresponds to a set of pixels with a specific color. In order to derive its screen space coordinates, we have to ensure that the region is actually a valid square, discover its four corners and align its sides according to the internal marker (Fig. 4). For this purpose, the contour of the region is extracted (Fig. 4a) and iteratively simplified (Fig. 4b) until it consists of exactly four points (Fig. 4c). Invalid regions either vanish during this process or converge to a polygon with a different number of corners.

To compensate for inaccuracies during the simplification step, the corners of the remaining squares are adapted to match the color gradient in the original

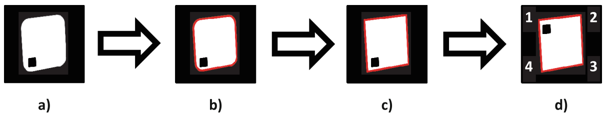


**Fig. 4.** Extraction and alignment of a square.

image (*cornerSubPix*). Since the 3D reconstruction requires consistent world space coordinates, the squares must be also aligned, so that the internal marker is located at the first point. Let $p_1, p_2, p_3, p_4 \in \mathbb{R}^2$ be the screen space coordinates of the square. Based on the construction of the pattern, the four possible locations of the marker $m_1, m_2, m_3, m_4 \in \mathbb{R}^2$ are given by the weighted sums:

$$\begin{aligned} m_1 &:= \tfrac{9}{16} \cdot p_1 + \tfrac{3}{16} \cdot p_2 + \tfrac{1}{16} \cdot p_3 + \tfrac{3}{16} \cdot p_4 \\ m_2 &:= \tfrac{3}{16} \cdot p_1 + \tfrac{9}{16} \cdot p_2 + \tfrac{3}{16} \cdot p_3 + \tfrac{1}{16} \cdot p_4 \\ m_3 &:= \tfrac{1}{16} \cdot p_1 + \tfrac{3}{16} \cdot p_2 + \tfrac{9}{16} \cdot p_3 + \tfrac{3}{16} \cdot p_4 \\ m_4 &:= \tfrac{3}{16} \cdot p_1 + \tfrac{1}{16} \cdot p_2 + \tfrac{3}{16} \cdot p_3 + \tfrac{9}{16} \cdot p_4 \end{aligned}$$

Since the square is printed in black and the marker is printed in white, it can be identified by choosing the brightest of all four possible locations. Finally, the points of the square are rotated until $p_1$ corresponds to the corner of the marker.

## 3.4 Collecting Squares

While the four screen space coordinates of each square are known at this point, we also require corresponding world space coordinates to compute the orientation of the camera. Actually, the square can be placed anywhere on a XY-plane in world space because relative distances are sufficient to determine the current rotation vector. Hence, the four coordinates $w_1, w_2, w_3, w_4 \in \mathbb{R}^3$ of a single square are defined as follows:

$$w_1 := (0,0,0) \qquad w_2 := \left(\tfrac{3}{4},0,0\right) \qquad w_3 := \left(\tfrac{3}{4},\tfrac{3}{4},0\right) \qquad w_4 := \left(0,\tfrac{3}{4},0\right)$$

The size of $\frac{3}{4}$ is specified according to the ratio of $(3:1)$ between squares and spacing in the pattern, which becomes especially important when using multiple squares. In this case, their relation must be considered as well to construct a global coordinate system. In particular, two squares are either neighbours in one of the four directions (*left*, *right*, *up*, *down*) or not directly connected. Each direction is associated with an offset in world space, so that one of the two squares can be placed at the origin, while the other is shifted accordingly.

By iteratively assigning coordinates of adjacent squares, we can successively construct a world space coordinate system as shown in Fig. 5. In this example, some of the squares are missing (*dark*) and the initial square (*grey*) is placed at the origin $(0,0)$. The arrows (*green*) illustrate the incremental expansion, so that the square at the right of $(0,0)$ gets an offset of $(1,0)$. In order to determine the

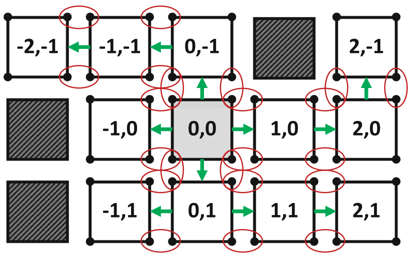


**Fig. 5.** Building the coordinate system. (Color figure online)

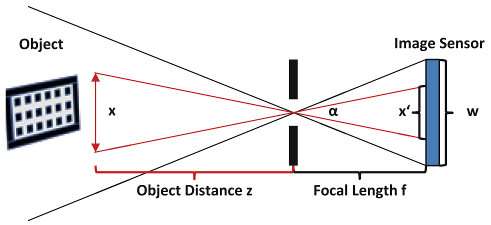


**Fig. 6.** Pinhole camera model.

relation of two arbitrary squares, the distances between the four pairs of opposing edges (*red*) are calculated. Due to the layout of the pattern, two edges can be considered as adjacent if their average distance is shorter than their length. At the end of this iterative process, all squares either belong to the connected subset containing the origin or correspond to separated isles, which are removed from the set. Hence, in case of $n$ valid squares, the result contains $4n$ pairs of screen space and world space coordinates.

## 3.5 Camera Model

Our camera model describes the relation between a point in homogeneous coordinates $(x, y, z)$ and its projection $(x', y', z')$ on the screen using the equation:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} := I \cdot R \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{1}$$

In particular, the rotation matrix $R$ is the unknown variable to be determined, while the intrinsic matrix $I$ can be derived from the camera model (Fig. 6). We utilize a pinhole camera, which is characterized by the focal length $f$ and the field-of-view $\alpha$. According to the intercept theorem, the relation between the width $x$ of an object and its projection $x'$ is given by:

$$\frac{x}{x'} = \frac{z}{f} \Leftrightarrow x' = \frac{x \cdot f}{z}$$

This relation can be also expressed in homogeneous coordinates using a projection matrix $P$:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} := P \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ width } P := \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The coordinates are scaled according to the size $w \times h$ (matrix $S$) of the image sensor and its resolution $u \times v$:

$$I := S \cdot P = \begin{pmatrix} \frac{u}{w} & 0 & \frac{u}{2} \\ 0 & \frac{v}{h} & \frac{v}{2} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{f \cdot u}{w} & 0 & \frac{u}{2} \\ 0 & \frac{f \cdot v}{h} & \frac{v}{2} \\ 0 & 0 & 1 \end{pmatrix}$$

The width $w$ of the image sensor is derived via trigonometry,

$$\tan\left(\tfrac{\alpha}{2}\right) = \tfrac{w}{2\cdot f} \Leftrightarrow w = 2 \cdot f \cdot \tan\left(\tfrac{\alpha}{2}\right)$$

while the height $h$ can be computed from the aspect ratio:

$$h = w \cdot \tfrac{v}{u}$$

Therefore, the intrinsic matrix $I$ of the camera is given by:

$$I := \begin{pmatrix} \lambda & 0 & \frac{u}{2} \\ 0 & \lambda & \frac{v}{2} \\ 0 & 0 & 1 \end{pmatrix} \text{ with } \lambda := \tfrac{f\cdot u}{w}$$

Beside the rotation matrix $R$, all variables of Eq. (1) are known. As a result, we can insert the $4n$ pairs of world space and screen space coordinates to compute an approximation of $R$ via [11–13] or the OpenCV method *solvePnP* [14].

### 3.6 Evaluation

Since the reference pattern (Fig. 3) can be placed anywhere for testing, it is impossible to determine the absolute orientation of the camera from the image. As a consequence, our benchmark compares relative rotations of camera and sensor with regard to a reference orientation, which is captured at the beginning of a test run.

Optimally, the relative rotation angles match at all times but since camera and sensor are usually implemented as separate hardware components, their time stamps are not synchronized. We have examined a latency of approximately one frame between camera and inertial sensor. Since the sensor usually produces a significant higher number of samples, both curves must be adjusted and cross-correlated to minimize this error.

For a test run of length $l$, the error $e(t)$ at time $t$ is defined as the shorted rotation angle in degrees between camera and sensor. As a result, we can define an error score $E$ as the weighted mean error over the time $t$ of the test.

$$E := \frac{1}{t}\int_0^t e(x)dx \tag{2}$$

The unit of $E$ is degrees and smaller values are better.

## 4 Results

Based on the concept of Sect. 3, we have developed a benchmarking application for *Android* phones, which records the video sequence of a reference gesture (Fig. 7) in order to compute the dynamic accuracy of the IMU. The error score is calculated in a post-processing step and defined by the integral of the mean deviation between camera and sensor angles (2). The application also shows a detailed presentation in form of graphs and statistics for every test run.

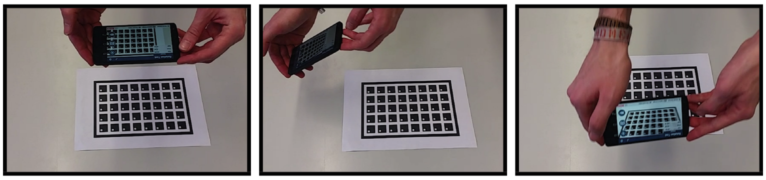


**Fig. 7.** Gestures performed for the rotation test.

**Table 1.** Results of the rotation test and the pedestrian tracker

| Phone | Rotation vector | | Game rotation vector | | Pedestrian tracker |
|---|---|---|---|---|---|
| | $\overline{x}$ | $\sigma$ | $\overline{x}$ | $\sigma$ | Score $s \in [-1, 1]$ |
| Phone 1 | 1.48 | 0.20 | 1.33 | 0.23 | 0.208 |
| Phone 2 | 2.35 | 0.29 | 1.37 | 0.13 | −0.278 |
| Phone 3 | 4.67 | 3.89 | 1.75 | 0.61 | 0.083 |
| Phone 4 | 6.79 | 3.58 | 7.89 | 0.96 | −0.333 |
| Phone 5 | 9.62 | 11.96 | n/a | n/a | −0.125 |

Our benchmark has been evaluated using the *LG G3*, the *Samsung S5*, the *LG Nexus 5*, the *Moto G2*, and the *HTC One*, which are anonymized in a different order as *Phone 1* to *Phone 5*. During a test run, the phone is tilted $\approx 40°$ forwards, backwards, and to both sides, while the camera is pointed at the pattern (Fig. 7). As a possible error source, blurring artefacts might prevent the correct detection of the pattern, so that the speed of the gesture must be adapted according to the quality and exposure time of the camera. Therefore, smaller resolutions like $800 \times 480$ pixels usually provide more accurate results than HD recordings due to higher frame rates.

The rotation gesture is performed manually but repeated ten times per phone to improve the significance of the benchmark. Table 1 lists the mean error ($\overline{x}$) as well as the standard deviation ($\sigma$). We are using both the default rotation vector and the game rotation vector, which are acquired using the *SensorManager* API of Android. While the default rotation vector is automatically recalibrated by a compass, the game rotation vector is better suited for fast movements and cannot be disturbed by magnetic influences. The results show significant differences between the evaluated phones. In particular, the average error is less than 3° for *Phone 1* and *Phone 2* but reaches almost 10° for *Phone 5*. For the three best phones, the game rotation vector (not available on Phone 5) offers even better results with an error score of less than 2°.

Our benchmark assumes that the reconstructed rotation vector from the camera is more accurate than the rotation obtained directly from the IMU. Since both rotations are computed using entirely different algorithms, a correlation between camera and sensor is a strong indicator for the correctness of the measurement. The reproducibility of our test is further emphasized by the fact that accurate results coincide with a small standard deviation of less than 1°. However, in case of discrepancies like *Phone 5*, it remains ambiguous whether the camera, the IMU, or both provide erroneous data.

For comparison, we have also evaluated the pedestrian tracker application using various paths, which results in a score $s \in [-1, 1]$ for each phone (Table 1). Contrary to the rotation test, larger values are better in this case and we can detect a coincidence between precise results. For instance, *Phone 1* produces sound results in both tests, while *Phone 4* and *Phone 5* are much less reliable with $\overline{x} > 6$ and $\sigma > 3$.

## 5 Conclusion

This paper describes the concept of a mobile and camera-based benchmarking method for inertial sensors and sensor fusion on smartphones. As a result, we can identify significant discrepancies between the accuracy of different IMUs, which also affect their usability at the application-level. In contrast to existing methods, our benchmark does not require an expensive test environment but can be performed using a recent *Android* phone and the reference pattern printed on a single sheet of paper. Future work includes improving the reproducibility of the test, supporting a wider range of movements, and detecting external error sources like magnetic fields.

## References

1. Mladenov, M., Mock, M.: A step counter service for Java-enabled devices using a built-in accelerometer. In: COMSWARE 2009. CAMS (2009)
2. Serra, A., Carboni, D., Marotto, V.: Indoor pedestrian navigation system using a modern smartphone. In: MobileHCI 2010 (2010)
3. Schmalstieg, D., Wagner, D.: Experiences with handheld augmented reality. In: Mixed and Augmented Reality, ISMAR 2007 (2007)
4. Mulloni, A., Seichter, H., Schmalstieg, D.: Handheld augmented reality indoor navigation with activity-based instructions. In: MobileHCI 2011 (2011)
5. Liu, J., Zhong, L., Wickramasuriya, J., Vasudevan, V.: uWave: accelerometer-based personalized gesture recognition and its applications. Pervasive Mob. Comput. **5**(6), 657–675 (2009)
6. Woodman, O.J.: An introduction to inertial navigation. University of Cambridge, Computer Laboratory, Technical report UCAMCL-TR-696, vol. 26(6), pp. 561–575 (2007)
7. Kim, A., Golnaraghi, M.: Initial calibration of an inertial measurement unit using an optical position tracking system. In: PLANS 2004 (2004)
8. Lobo, J., Dias, J.: Relative pose calibration between visual and inertial sensors. Int. J. Robot. Res. **26**(6), 561–575 (2007)
9. Panahandeh, G., Jansson, M.: IMU-camera self-calibration using planar mirror reflection. In: Indoor Positioning and Indoor Navigation (IPIN) (2011)
10. Oberkampf, D., DeMenthon, D., Davis, L.: Iterative pose estimation using coplanar points. In: Computer Vision and Pattern Recognition (1993)

11. Gao, X.S., Hou, X.-R., Tang, J., Cheng, H.-F.: Complete solutionclassification for the perspective-three-point problem. IEEE Trans. Pattern Anal. Mach. Intell. **25**(8), 930–943 (2003)
12. Lepetit, V., Moreno-Noguer, F., Fua, P.: EPnP: an accurate o(n) solution to the PnP problem. Int. J. Comput. Vis. **81**(2), 155–166 (2009)
13. Zhang, Z.: A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. **22**(11), 1330–1334 (2000)
14. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media Inc., Sebastopol (2008)
15. Zhang, X., Fronz, S., Navab, N.: Visual marker detection, decoding in AR systems: a comparative study. In: Mixed and Augmented Reality. ISMAR (2002)
16. Fiala, M., Shu, C.: Self-identifying patterns for plane-based camera calibration. Mach. Vis. Appl. **19**(4), 209–216 (2008)
17. Kelly, J., Sukhatme, G.S.: Visual-inertial sensor fusion: localization, mapping and sensor-to-sensor self-calibration. Int. J. Robot. Res. **30**, 56–79 (2011)
18. Kato, H., Billinghurst, M.: Marker tracking and HMD calibration for a video-based augmented reality conferencing system. In: IWAR 1999 (1999)
19. Hinckley, K., Sinclair, M., Hanson, E., Szeliski, R., Conway, M.: The videomouse: a camera-based multidegree-of-freedom input device. In: ACM UIST 1999 (1999)
20. Liu, Y., Yang, J., Liu, M.: Recognition of QR code with mobile phones. In: 2008 Chinese Control and Decision Conference, CCDC 2008, July 2008
21. Nister, D.: An efficient solution to the five-point relative pose problem. IEEE Trans. Pattern Anal. Mach. Intell. **26**(6), 756–770 (2004)