ISSN 2464-4617 (print)
ISSN 2464-4625 (DVD)
Computer Science Research Notes
CSRN 2901
WSCG Proceedings Part II

Porting A Visual Inertial SLAM Algorithm To Android Devices

Jannis Möller
Technische Hochschule Köln
Steinmüllerallee 1
51643, Gummersbach,
Germany
jmoeller@th-koeln.de

Benjamin Meyer
Technische Hochschule Köln
Steinmüllerallee 1
51643, Gummersbach,
Germany
benjamin.meyer@th-koeln.de

Martin Eisemann
Technische Hochschule Köln
Steinmüllerallee 1
51643, Gummersbach,
Germany
martin.eisemann@th-koeln.de

ABSTRACT

Simultaneous Localization and Mapping aims to identify the current position of an agent and to map his surroundings at the same time. Visual inertial SLAM algorithms use input from visual and motion sensors for this task. Since modern smartphones are equipped with both needed sensors, using VI-SLAM applications becomes feasible, with Augmented Reality being one of the most promising application areas. Android, having the largest market share of all mobile operating systems, is of special interest as the target platform. For iOS there already exists a high-quality open source implementation for VI-SLAM: The framework VINS-Mobile. In this work we discuss what steps are necessary for porting it to the Android operating system. We provide a practical guide to the main challenge: The correct calibration of device specific parameters for any Android smartphone. We present our results using the Samsung Galaxy S7 and show further improvement possibilities.

Keywords

Augmented Reality; Visual Inertial SLAM; Smartphones; Camera calibration

1 INTRODUCTION

Visual inertial SLAM describes a class of algorithms that use both visual and inertial measurements to build a map of the environment and at the same time locate an agent in it. Besides the research field of robotics, they also are of interest for Augmented Reality (AR) applications. In this application context, the goal is the perspectively correct rendering of virtual 3D content for a given viewing position, either using see-through devices like the Microsoft HoloLens or on top of a live recorded camera image.

Recently, computing hardware has been getting more and more powerful. The smartphone market in particular is developing very fast. This rapid progress makes algorithms that required powerful desktop hardware a few years ago now suitable for use on mobile devices.

In order to enable a robust and therefore immersive augmented reality experience on the users' devices, three-dimensional localization within the environment is required. However, most freely available augmented reality frameworks only offer tracking of specific markers. Since all popular smartphones are equipped with a camera and an inertial measurement unit (IMU) these days, visual inertial odometry and SLAM poses a suitable solution for this problem. To utilize these algorithms a set of device specific parameters has to be known beforehand.

The intention behind this work is to give a simple explanation of the steps that are required to port an existing framework to Android. We focus especially on providing a comprehensible and practical guide for determining these parameters. Less relevant parts, like translating the code base, are omitted for the sake of clarity as they are highly depended on the individual framework. The procedures are exemplarily explained on the basis of the framework *VINS Mobile* [LQH*17], which is originally only available for iOS and has been ported to Android in the course of this work. However, the procedure of porting and calibrating the parameters are of relevance for other algorithms or frameworks as well.

This work is divided into six sections:
Section 2 gives an overview over the recent development in the field of Augmented Reality frameworks as well as the research field of visual inertial SLAM algorithms. It also provides a brief introduction into the algorithm behind the VINS-Mobile framework. In Section 3 we show the different steps that are required to calibrate the device specific parameters. Section 4 ex-

plains the main problems we encountered in regards to Android hard- and software. We present and analyze the results of our quantitative and qualitative tests in Section 5. Finally, in Section 6 we list options of further improvements for future work.

2 RELATED WORK

There are many different ready-to-use solutions in the field of augmented reality applications and frameworks. The favored open source candidate for arbitrary AR-projects seems to be the AR framework *ARToolKit* [ARTa]. Its compatibility with Unity and the capability to develop a single application for multiple platforms simultaneously surpass the capabilities of most other open source frameworks. In the current version, however, it offers no possibility to recognize or track a 3D environment using feature matching. It is limited exclusively to 2D images or markers. Future versions of this framework are said to improve feature tracking and add area recognition. But the official development seems to have come to a standstill, after it was bought by *DAQRI*. In the future, development is planned to be continued as the new project *artoolkitX* [ARTb] by the open source community. Compared to this framework other open source solutions are even more limited in scope and function; some offer nothing more than a simple abstraction layer around the image processing library *OpenCV* [OPE].

Furthermore, some proprietary solutions are available, but they are not completely open-sourced and often come at a high financial price in later use. Some of these are *Vuforia* [VUF], *Kudan* [KUD], *MAXST* [MAX] and *Wikitude* [WIK], as well as the free to use *ARKit* [ARK] from Apple and *ARCore* [ARC] from Google. It has to be mentioned that recent advancements in these proprietary frameworks included the implementation of more complex localization and tracking algorithms. These features are often called *markerless tracking*, or just *motion tracking*. In addition, some of the frameworks like ARKit and ARCore also support basic understanding of the scene through plane detection. Even though being rich in features, the fact that they are either very costly or not completely open-source is a knockout criterion for when adaptability is a high priority like in a research context.

In the general field of visual inertial odometry and SLAM algorithms, there are many approaches often coming with open source implementations. They can be categorized into filter-based approaches, such as [LM13, HKBR14, BOHS15] and optimization-based approaches such as [IWKD13, SMK15, LLB*15]. Filter-based ones usually need less computational resources and optimization-based ones might provide greater accuracy. However, most of these are designed or implemented for other platforms with different hardware requirements, the calculations are often done offline, not in real time or on powerful desktop hardware.

One of the few mobile solutions is *VINS-Mobile* [LQH*17], the iOS port of the *VINS-Mono* project [QLS17]. The source code and the underlying scientific papers are publicly available [VIN]. It fuses visual and inertial measurements in a tightly-coupled, optimization-based approach. The position and orientation for successive selected camera frames (keyframes) are stored in a global pose graph. Several performance improving measures are utilized to enable usage of mobile hardware. The main one being the limitation of the global pose graph optimization to a sliding window of 10 keyframes. Another important feature is the initialization step, which automatically determines the metric scale. There is no need for assumptions or knowledge about the environment, but the device-specific parameters have to be known. To correct the occurring position drift a simple bag of words based loop closure procedure is integrated into the global pose optimization.

3 CALIBRATION

To port a framework many different steps are required. Besides the translation of operating system specific code to Android, a few parameters have to be adjusted to the new hardware platform. The visual inertial SLAM algorithm assumes that these parameters are known before run-time. They include the intrinsic camera parameters, focal length and center of the image needed for visual reconstruction and the extrinsic parameters that describe the transformation from the IMU frame to the camera frame for correctly combining inertial information with the camera motion.
This section exemplary shows the different steps that are required to determine these parameters. It can be used as a guideline to port and use this framework on any Android smartphone. We tested the Android port on a Samsung Galaxy S7.

3.1 Intrinsic Parameters

The required intrinsic parameters are the focal length and the center of the image (principal point), both in pixels in X and Y direction. The focal length specifies the distance between the lens and the focal point, while the principle point specifies the image center in the pinhole camera model. It usually is close to the center of the output image. Pixels as a unit of length are used in relation to the photo sensor pixel size. A visualization of the parameters based on the pinhole camera model is shown in Figure 1.

To determine the intrinsic parameters, we took a video of a checkerboard pattern at a video resolution of 480×640, manually extracted 10 to 20 frames

 ISBN 978-80-86943-38-1

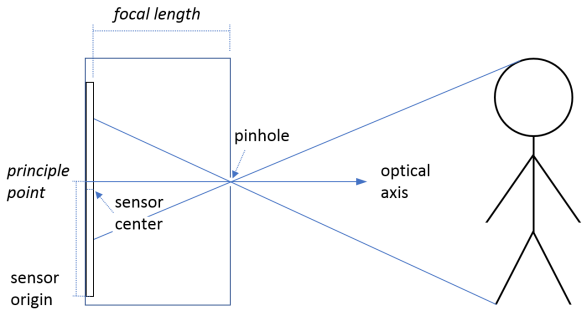


Figure 1: Pinhole camera model with the parameters *focal length* and *principle point*

spanning a variety of different camera positions and reconstructed the intrinsic parameters using *Camera Calibrator* from the software suite *MatLab* [MAT].

The reconstruction of the intrinsic parameters is normally only computable up to a scaling factor. This limitation can be circumvented by manually providing the length of a checkerboard square. Figure 2 shows the mean reprojection error we were able to achieve during the calibration process. Apart from the focal length and principal point this procedure also reconstructs the radial distortion of the lens, which, however, is being ignored in the VINS-Mobile framework.

Instead of using standard checkerboards, it has been proven advantageous to resort to special calibration checkerboards with an odd number of rows and an even number of columns. Unlike with standard checkerboard patterns, orientation is always unique in this combination of row and column count. This will prevent most calibration algorithms from selecting the wrong corner when placing the pattern origin.

During our work, we tested two different setups for this calibration step. The first video was taken with a checkerboard pattern printed on a DinA4 sized sheet of paper. The individual squares had a side length of exactly 30 mm. The mean reprojection error in this setup can be seen in Figure 2(a). For the second video, the filmed pattern was displayed on a flat computer monitor. The size of the squares in this setup was slightly larger at 33.8 mm. The resulting mean reprojection error is shown in Figure 2(b). These two different inputs led to the calibration results printed in Table 1 & 2. As can be seen, the reprojection error from the printed checkerboard is quite large compared to the one from the screen checkerboard. The reason for that is presumably that the printed paper had built up slightly uneven waves due to the large amount of printer ink. The surface not being completely flat caused the error to be more than twice as big. Modern flat-panel displays serve the purpose of a perfect plane reasonably well.

FocalLength:	[498.3953 495.7647]
PrincipalPoint:	[226.4976 319.1671]
RadialDistortion:	[0.2526 -0.5535]
ImageSize:	[640 480]
MeanReprojectionError:	0.4699

Table 1: Camera intrinsics from printed checkerboard

FocalLength:	[478.7620 478.6281]
PrincipalPoint:	[231.9420 319.5268]
RadialDistortion:	[0.2962 -0.6360]
ImageSize:	[640 480]
MeanReprojectionError:	0.2147

Table 2: Camera intrinsics from screen checkerboard

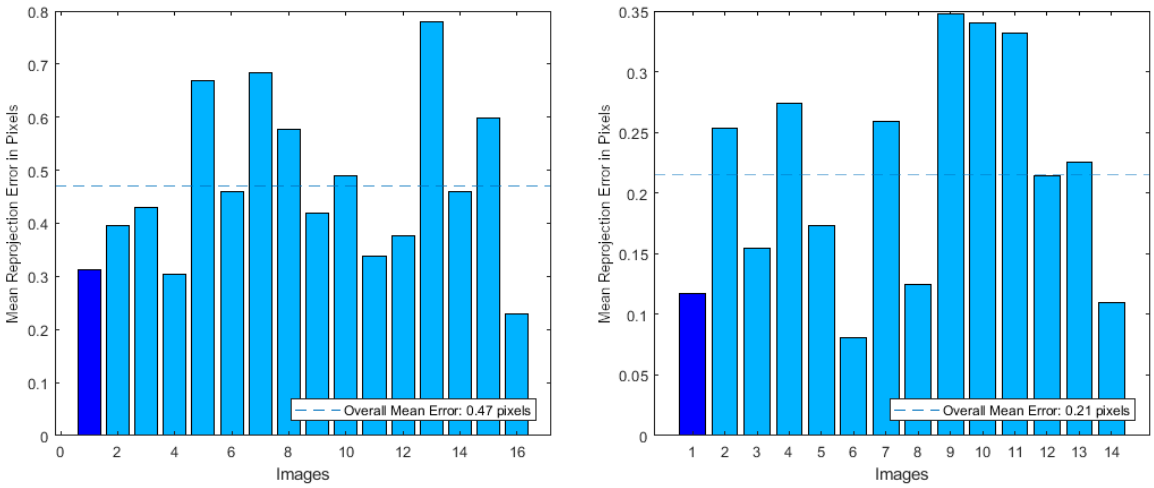
3.2 Extrinsic Parameters

In addition to the described intrinsic parameters, the visual inertial SLAM-algorithm needs specific knowledge about the relative positioning of the inertial measurement unit (IMU) to the camera. The relative orientation is being ignored as the camera and the IMU are expected to be axis-aligned. The extrinsic parameters are highly dependent on the target hardware. In this work, we focus on the Samsung Galaxy S7 as target mobile phone - however, the necessary steps for porting the iOS VINS framework remain the same for arbitrary Android phones.

Usually, the extrinsic parameters are irrelevant in the development of normal smartphone applications, so it is not surprising that the manufacturers did not publish the necessary parameters required for the calibration. Even internet sources that deal with the hardware of smartphones more meticulously have not specified this information at all. As official information about the IMU and camera locations are unavailable, we gathered the needed information from available frontal straight X-ray images [iFi16a] for the Apple iPhone 7 Plus and an open device image with visible PCB [iFi16b] for the Samsung Galaxy S7. Outer dimensions of the smart phones are provided by the manufacturers and can be used as a reference of scale.

In order to make the measurement of the dimensions easier and more precise, the images of the devices were rectified and scaled. The camera respectively the IMU-axis should be the frame of reference for this rectification. A visualization of the results can be seen in Figure 3. Displayed are the iPhone 7 Plus on the left and the Galaxy S7 on the right. Both are frontal views looking straight at the screen. The measurements from the method described above are drawn in green. Blue indicates the values found in the frameworks source code for the iPhone 7 Plus. The translation in direction of the visual axis of the camera is neglected because it cannot be determined easily, is usually very small and thus has only very little impact on the resulting accuracy.

ISBN 978-80-86943-38-1



(a) Printed checkerboard (b) Screen checkerboard

Figure 2: Mean reprojection error from the camera calibration process. The individual bars represent the mean reprojection error of the corresponding images and the dotted line the overall mean reprojection error.

4 PROBLEMS

In the following we describe further hard- and software-posed challenges we encountered.

4.1 Synchronization

On the hardware side of Android, the synchronization of the sensor events needs special attention. It is not guaranteed that acceleration and gyroscope events are always processed alternating, even if set to the exact same frequency. For example, it can happen that ten acceleration events occur at first and the ten corresponding gyroscope events afterwards. Therefore in the porting process, it was necessary to rewrite the buffer system of the IMU measurements. It now uses a separate queue for both sensor event types. The IMU measurements are processed and inserted into the algorithm only once both corresponding measurements occurred. It is also ensured that if one event type skips a time step, the buffer system does not lock forever.

4.2 Unsolved Problems

It was observed that the time difference between the timestamps of the camera and the IMU can be shifted by up to 300 ms. The reason is not obvious and might lie in the hardware and OS-software architecture. At this point it should be noted that the camera image time interval at 30 Hz is just $33.\overline{3}$ ms and the IMU measurement interval at 100 Hz is 10 ms. This shows how severe this desynchronization issue is on Android. A possible explanation for this observation might be that the image buffer of the camera cannot be processed fast enough due to the increasing processing time as the run progresses.

Due to the processing order of the camera and IMU data, the consequences are limited to a general delay in processing. For the preprocessing of the IMU data, it is important that the processing of camera images is deferred longer than the measurement processing of the IMU. If this was not the case, some measurements of the IMU would be lost because they have not yet been inserted into the event queue before the latest image has been processed. For the next picture they would be neglected because their timestamp is older than the one of the last picture. To reliably fix this desynchronization issue the system could be improved by building another buffer system, that fuses both IMU and camera data, before processing them.

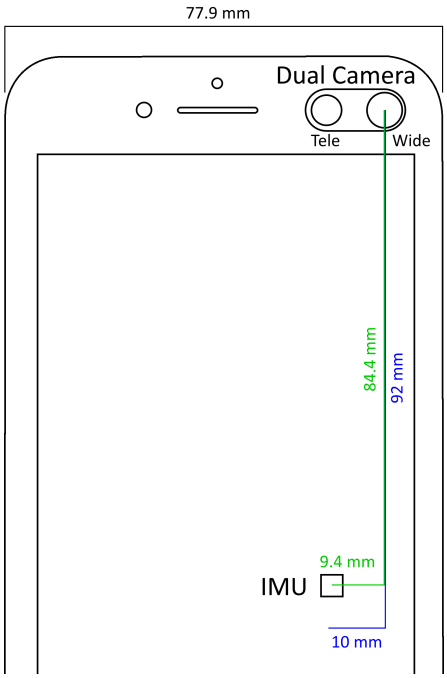
5 RESULTS

After the porting of the framework was completed we compared the Android port with the original iOS version in performance, robustness and accuracy. The results of the quantitative performance study are presented in section 5.1. In section 5.2 we discuss the qualitative differences in the form of an experimental study.

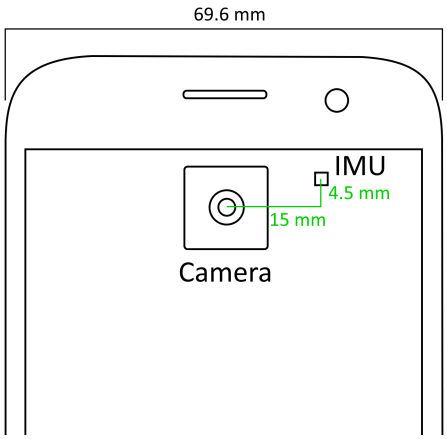
For Android the Samsung Galaxy S7 was used as the test device. It features an Exynos 8890 64-Bit Octa-core processor clocked at 4 x 2.3 GHz and 4 x 1.6 GHz. It has 4 GB of RAM and a 12 MP front-facing camera (f/1.7, 26mm) with optical image stabilization [GSMb]. Thus it can be classified in the middle to upper performance class of Android smartphones.

For iOS the Apple iPad Pro 12.9 (2015) was used for comparison. This tablet features an Apple A9X 64-Bit Dual-core processor clocked at 2.26 GHz. It has 4 GB of RAM too and a 8 MP front-facing camera (f/2.4, 31mm) [GSMa].

ISBN 978-80-86943-38-1



(a) Apple iPhone 7 Plus



(b) Samsung Galaxy S7

Figure 3: Extrinsic parameters: Translation from IMU to Camera. Drawn in blue are the values found in the frameworks source and drawn in green the values determined by our method.

5.1 Quantitative Performance

This section presents the statistical results of our performance benchmark. We gathered timings for each of the individual steps of the algorithm. The resulting statistical observations are listed in Table 3. The structure and values for iOS are inferred from [LQH*17]. Android Studio in combination with the native developement kit (ndk) and its GNU Compiler Collection (GCC) variant was utilized for porting and compiling the framework on Android. To reach comparable levels of performance on Android it is necessary to enable the highest possible compiler optimization level. Without these optimizations the modules relying on the Ceres Solver library are more than a magnitude slower.

T	Module	Freq.	iOS	Android
1	Feature Detection	10 Hz	17 ms	16 ms
	Feature Tracking	30 Hz	3 ms	8 ms
	Visualization	30 Hz		6 ms
2	Initialization	once	80 ms	268 ms
	Nonlinear Opt.	10 Hz	60 ms	113 ms
3	Loop Detection	3 Hz	60 ms	34 ms
	Feature Retrieval	3 Hz	10 ms	44 ms
4	Global Pose Opt.	marg.		29 ms

Table 3: Performance comparison. Modules are grouped by threads (T) they run in.

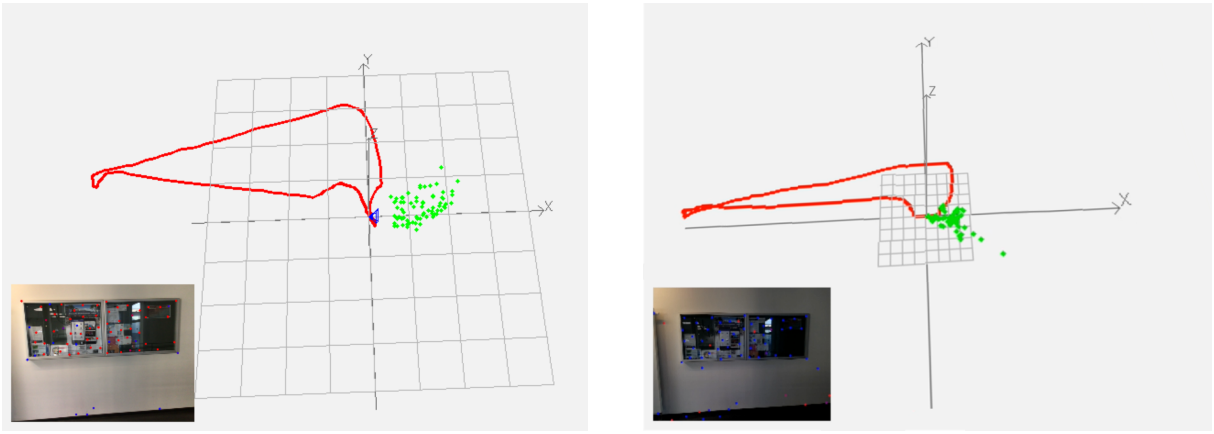
For the measurements of the optimized version we chose two different approaches:
First, we ran 10 tests which we canceled immediately after initialization. These were used solely to gather information about the average initialization time. Second, we carried out 5 more test runs, each limited to 20 seconds. In each of them the initialization was completed and a 360 degree turn performed. In 4 out of 5 of them the previously visited regions were recognized and the loops in the pose graph successfully closed. Like in the first approach, we used the collected data to extract a meaningful average of the other algorithm steps over these runs.

Not specified in table 3 is the total time of the `onImageAvailable` function, which is executed for each camera input frame. It includes the necessary image format conversion as well as the feature detection and tracking and the visualization. As it takes up 37.25 ms on average, the processing isn't able to keep up with the image input rate of 30 Hz and thus causes skipping of some camera frames. The resulting delay and skipping of frames is noticeable to the naked eye, but partly affects the iOS version as well, though not as strongly.

The observed value for loop detection is surprisingly low in comparison to the iOS measurements. This could be attributed to the short duration of the test. As the pose graph continues to grow over time, this module will take more time.

Overall, the results indicate that the interaction between software and hardware on iOS is optimized better. This is particularly noticeable in the measurements of initialization and nonlinear optimization, as both heavily rely on the Ceres Solver library. The Android version would potentially benefit a lot from a better parallelization implementation of the applied algorithms, due to the great multithreading performance of most Android smartphones.

ISBN 978-80-86943-38-1



(a) iOS - Apple iPad Pro (b) Android - Samsung Galaxy S7

Figure 4: Successful loop closure

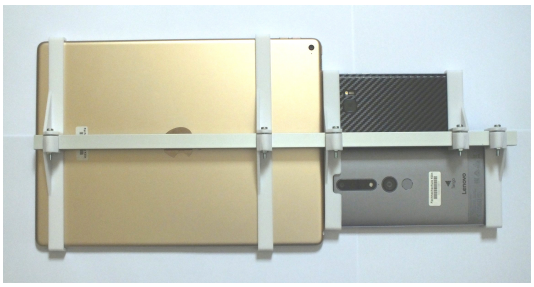


Figure 5: Test setup for fixing the devices. For testing purposes we added a Google Tango device, a now abandoned AR-project by Google that uses specialized hardware for tracking movement and the environment.

5.2 Experimental Performance

This section presents the results of the experimental comparison between the Android port and the iOS version. The two platforms are very different in some regards, especially in the sensor inputs, which have a direct impact on the quality and performance of the framework. Therefore, it makes little sense to use a publicly available dataset for the comparison, such as MH_03_medium used in [LQH*17] or a similar one. Instead, a test under practical conditions was chosen, exposing the different platforms to the same environmental conditions. Of course, physically it is impossible to produce the exact same conditions, since the cameras cannot all be at the same position. In order to provide the best possible approximation, a test apparatus was designed on which the devices can be firmly fixed. A picture of this setup can be seen in Figure 5.

Due to the lack of a possibility to acquire the ground truth of the traveled route, the result could only be evaluated manually. For that purpose the screen of each device was recorded. Since it is not possible to record a video of the screen on iOS version 10, we took screenshots at key points of the route on the iPad.

For the test, as the starting point we chose a feature rich object, which stood out from the rather repetitive surroundings. We then took a walk inside our university building. The covered distance is approximately 80 m. At the end of the test run we returned to the starting point, so that the algorithm had a chance to detect and close the occurring loop.

Both the iOS version and the Android port are able to recognize the previously visited location and adjust their pose accordingly, which can be subject to a significant drift, caused e.g. by barren corridors or fast movements. The screenshots in Figure 4 visualize this loop closure. As can be seen both show roughly the same quality in tracking. To enable measuring the occurring error, an additional online or offline system for recording the groundtruth would have to be installed. Furthermore the framework would need to be modified in a way that allows logging the estimated current pose over the course of the testing. After testing, the error could then be calculated and visualized.

6 CONCLUSION AND FUTURE WORK

We describe the process of porting a visual inertial SLAM algorithm from iOS to Android and discuss how to solve the according challenges. At the example of the VINS Mobile Framework we explain the necessary steps to calibrate the device specific parameters correctly and analyze its performance both quantitatively and qualitatively.

In future work, the complex calibration process could be partially automated, so that a broader range of android smartphones could be used by the framework. To further expand the functionality of the framework, techniques of dense environment reconstruction could be used to enable occlusion of AR-content by the environment.

Our framework is publicly available at [GIT].

ISBN 978-80-86943-38-1

7 REFERENCES

[ARC] ARCore - Google. URL: https://developers.google.com/ar [cited 29.04.2019].

[ARK] ARKit - Apple. URL: https://developer.apple.com/arkit [cited 29.04.2019].

[ARTa] ARToolKit. URL: https://github.com/artoolkit [cited 29.04.2019].

[ARTb] artoolkitX. URL: http://www.artoolkitx.org [cited 29.04.2019].

[BOHS15] BLOESCH M., OMARI S., HUTTER M., SIEGWART R.: Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sep. 2015), pp. 298–304. doi:10.1109/IROS.2015.7353389.

[GIT] VINS-Mobile-Android. URL: https://github.com/jannismoeller/VINS-Mobile-Android [cited 27.04.2019].

[GSMa] GSMARENA.COM: Apple iPad Pro 12.9 (2015) - Full tablet specifications. URL: https://gsmarena.com/apple_ipad_pro_12_9_(2015)-7562.php [cited 29.04.2019].

[GSMb] GSMARENA.COM: Samsung Galaxy S7 - Full phone specifications. URL: https://gsmarena.com/samsung_galaxy_s7-7821.php#g930f [cited 29.04.2019].

[HKBR14] HESCH J. A., KOTTAS D. G., BOWMAN S. L., ROUMELIOTIS S. I.: Consistency analysis and improvement of vision-aided inertial navigation. *IEEE Transactions on Robotics 30*, 1 (Feb 2014), 158–176. doi:10.1109/TRO.2013.2277549.

[iFi16a] IFIXIT: iPhone 7 Plus Teardown, Sept. 2016. URL: https://de.ifixit.com/Teardown/iPhone+7+Plus+Teardown/67384#s136470 [cited 29.04.2019].

[iFi16b] IFIXIT: Samsung Galaxy S7 Teardown, Mar. 2016. URL: https://de.ifixit.com/Teardown/Samsung+Galaxy+S7+Teardown/56686#s122920 [cited 29.04.2019].

[IWKD13] INDELMAN V., WILLIAMS S., KAESS M., DELLAERT F.: Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems 61*, 8 (2013), 721 – 738. doi:10.1016/j.robot.2013.05.001.

[KUD] Kudan. URL: https://kudan.io [cited 29.04.2019].

[LLB*15] LEUTENEGGER S., LYNEN S., BOSSE M., SIEGWART R., FURGALE P.: Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research 34*, 3 (2015), 314–334. doi:10.1177/0278364914554813.

[LM13] LI M., MOURIKIS A. I.: High-precision, consistent EKF-based visual-inertial odometry. *The International Journal of Robotics Research 32*, 6 (2013), 690–711. doi:10.1177/0278364913481251.

[LQH*17] LI P., QIN T., HU B., ZHU F., SHEN S.: Monocular visual-inertial state estimation for mobile augmented reality. In *2017 IEEE International Symposium on Mixed and Augmented Reality* (Piscataway, NJ, 2017), Broll W., Regenbrecht H., Swan J. E., (Eds.), IEEE, pp. 11–21. doi:10.1109/ISMAR.2017.18.

[MAT] MatLab. URL: https://mathworks.com/products/matlab.html [cited 29.04.2019].

[MAX] MAXST AR SDK. URL: https://developer.maxst.com/ [cited 29.04.2019].

[OPE] OpenCV. URL: https://opencv.org [cited 29.04.2019].

[QLS17] QIN T., LI P., SHEN S.: VINS-Mono: A robust and versatile monocular visual-inertial state estimator, 2017. URL: http://arxiv.org/pdf/1708.03852.

[SMK15] SHEN S., MICHAEL N., KUMAR V.: Tightly-coupled monocular visual-inertial fusion for autonomous flight of rotorcraft MAVs. In *2015 IEEE International Conference on Robotics and Automation (ICRA)* (May 2015), pp. 5303–5310. doi:10.1109/ICRA.2015.7139939.

[VIN] VINS-Mobile - GitHub. URL: https://github.com/HKUST-Aerial-Robotics/VINS-Mobile [cited 29.04.2019].

[VUF] Vuforia. URL: https://engine.vuforia.com [cited 29.04.2019].

[WIK] Wikitude. URL: https://wikitude.com [cited 29.04.2019].

 ISBN 978-80-86943-38-1