

THE CASE FOR VISUAL-INERTIAL SLAM
OVER VISUAL SLAM

by

Jia-Shen Boon

A project submitted in partial fulfillment
of the requirements for the degree

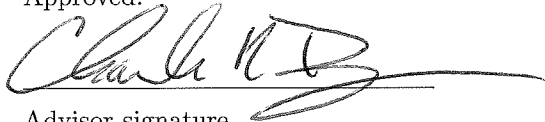
of

MASTER OF SCIENCE

in

Computer Sciences

Approved:

A handwritten signature in black ink, appearing to read 'Charles Dyer', with a long horizontal flourish extending to the right.

Advisor signature

Charles Dyer

UNIVERSITY OF WISCONSIN-MADISON

May 2016

The case for visual-inertial SLAM over visual SLAM

Jia-Shen Boon

May 2016

1 Introduction

The ability to map one’s environment and locate oneself within that map has always been important in robotics. This capability has applications in augmented reality as well. It is possible to realize this capability just with one or more cameras, but the resulting system has one serious flaw during a failure mode known as “track failure”. In this report, we argue that in the bid to achieve the aforementioned capability it is overwhelmingly beneficial to the system engineer to have a sensor suite consisting at least of a camera and an inertial measurement unit, instead of solely having a camera.

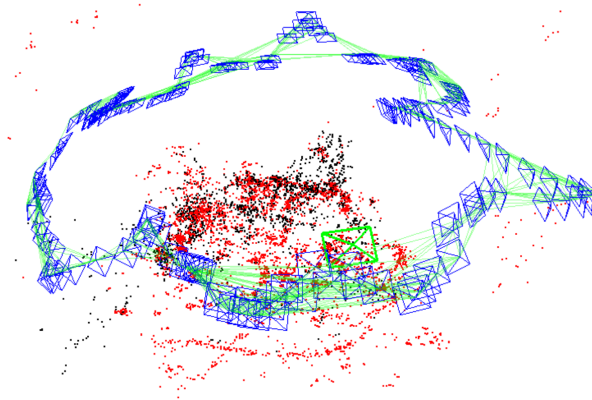


Figure 1: Example output of a SLAM system. Dots are mapping information. They are 3D points in the real world. Blue wireframe pyramids are localization information. They are the pose of the camera at select moments of the run-time.¹

¹http://wp.doc.ic.ac.uk/thefutureofslam/wp-content/uploads/sites/93/2015/12/ICCV15_SLAMWS_RaulMur.pdf

2 Why SLAM?

A **simultaneous localization and mapping** (SLAM) system simultaneously computes a representation of its environment (“mapping”), and the pose of its host platform (“localization”) [3]. See Figure 1 for an example. The pose of a rigid body refers to its location and orientation. This host platform, which we just call “platform” for brevity, is typically an autonomous vehicle such as a micro aerial vehicle (commonly known as a “drone”) or self-driving car, or a human-operated mobile device such as a smartphone or head-mounted device.

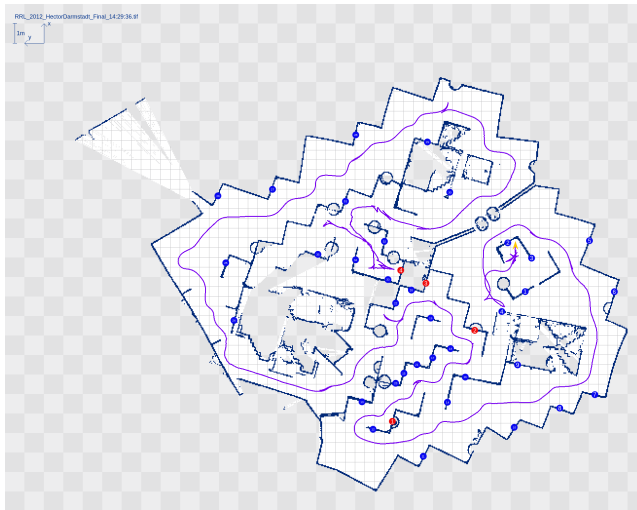


Figure 2: SLAM output of a robot in the RoboCup Rescue tournament. Dark blue squares are the mapping output; the purple line is the localization history of the robot. The robot acts as an emergency first responder that reports the location of casualties in a previously unmapped disaster site. The robot has to map the environment in order to report the casualties’ relative location.²

SLAM has applications in robotics [13] and augmented reality (AR) [6]. An autonomous vehicle needs to know its location to perform most robotic tasks, e.g., to move to point A , or to maintain a distance of 3 meters from body B (see Figure 2). An AR application needs to know the structure of its user’s environment to augment the scene (see Figure 3). In both contexts, both localization and mapping have to be performed due to their coupled nature.

The tasks of localization and mapping have long been known to be coupled [2]. You need to know your surroundings in order to know your location in those surroundings; you need to know your location in order to piece together new information about your surroundings. In some cases, the platform has a complete map before run-time, e.g., the indoor radio map of a building [12]. It

²<http://wiki.ssrrsummerschool.org/doku.php?id=rr1-rules-2014>

³<https://youtu.be/s4pICjMTKMs>

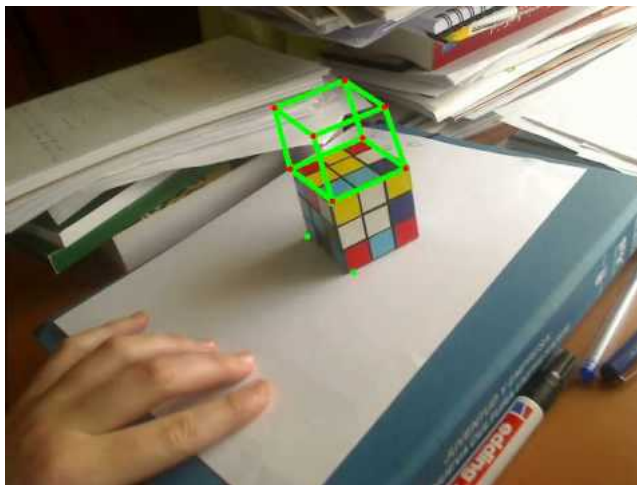


Figure 3: A simple AR application that draws a cubic wireframe on top of a cube in the environment. The cube may be drawn by inferring scene structure (from mapping) and projecting the points back into the camera (from localization).³

then just has to perform localization without the mapping. We make a simplifying assumption in this report that no map is available *a priori*. In practice, the platform may already have an incomplete map due to prior experience or human knowledge.

An implied property of SLAM systems is that they are *real-time*, which in the context of SLAM means that mapping and localization estimates are updated close to the rate of sensor readings. Real-time performance is critical for both robotics and AR. Robotics requires it for control stability. AR requires it for a pleasant user experience. In contrast, large scale maps such as Google Maps can be computed offline, via techniques that would *not* be considered SLAM due to their offline nature.

Next we describe how SLAM is achieved.

3 A baseline: visual SLAM

SLAM systems may be categorized by the types of sensors that the system employs. Common sensors in SLAM include LIDAR, RADAR, cameras, IMU, and GPS. LIDAR, RADAR, and cameras are *exteroceptive* sensors in that they sense properties of the environment; IMU and GPS are *proprioceptive* sensors in that they sense properties of the platform⁴. SLAM that uses one or more cameras as the only sensor(s) is known as **visual SLAM** (V-SLAM). Cameras

⁴Strictly speaking, GPS sensors are exteroceptive since they measure the time of flight between the sensor and satellites. However they are often classified as proprioceptive since these measurements are taken for the purpose of computing the global position of the sensor.

are particularly popular sensors because they are ubiquitous, comparatively cheap, and research in this imaging modality is comparatively mature. V-SLAM is the baseline method in this report, as implied the report title.

There are two major branches of V-SLAM - methods based on **probabilistic methods** and **Structure from Motion** (SfM). The interested reader may turn to Hartley and Zisserman [5], and Thrun et al. [13] respectively for more details.

Probabilistic methods are dominated by a recursive estimator known as the **extended Kalman Filter** (EKF). An EKF models a state as Gaussian distributed, where the state typically consists of the pose of the platform and the 3D location of all the landmarks observed up to the current time. The mean vector is the *maximum a posterior* estimate of the state, while the covariance matrix reflects the uncertainty of the estimate. Probabilistic methods conveniently characterize the performance of the filter by the covariance matrix, but is not robust to data association errors, and has poor time complexity [3].

SfM methods make heavy use of projective geometry developed by the computer vision community [5]. In particular, **bundle adjustment** (BA) formulates SLAM as an optimization problem, finding all 3D points of the map and all the poses in the platform’s history that minimize projection error between the observation and the estimates (see Figure 4). Concretely, the optimization problem is

$$\min_{\{X_j, P_t\}} \sum_{j,t} d(P_t X_j, x_{tj}) \quad (1)$$

where $d(\mathbf{x}, \mathbf{y})$ is a distance metric, such as Euclidean distance, between vectors \mathbf{x} and \mathbf{y} , and the other terms are consistent with Figure 4.

BA tends to produce accurate results, but it is not robust to poor initialization, and when naively applied to SLAM it runs too slowly. In a seminal paper, Klein and Murray [6] proposed a solution to this problem by having two threads. One thread’s input/output stream runs at frame rate to provide real time estimates; one thread’s input stream runs at lower than frame rate and continually runs bundle adjustment. Multiple threads enable SfM to run in real-time.

4 The fatal flaw of visual SLAM

When a V-SLAM system deems track quality to be poor [6, 11] (or equivalently, “when the track is lost” or “when **track failure** occurs”), it stops estimating the platform pose. Track quality can be poor for a multitude of reasons, e.g., motion in the scene or camera leading to image blur, a lack of features in the image, textures or repeated features in the image. Pose estimation only resumes when the camera returns to a previous explored environment [15, 10]. This loss of pose estimation is clearly bad for an autonomous robot, since it cannot, for

⁵http://cs.nyu.edu/~fergus/teaching/vision/11_12_multiview.pdf

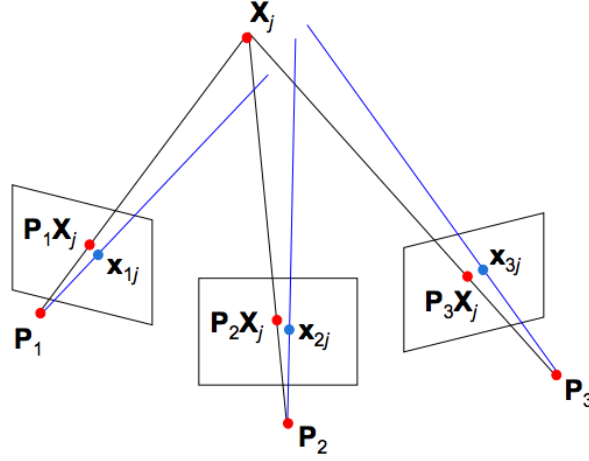


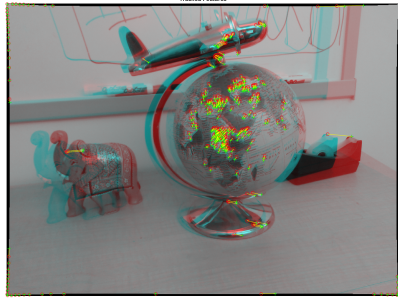
Figure 4: Illustration of bundle adjustment in the context of SLAM. X_j is an estimate of the 3D location of scene point j ; P_t is an estimate of the camera pose at time t ; x_{tj} is the observation of map point j at time t . The gaps between $P_t X_j$ and x_{tj} are the projection error. Bundle adjustment seeks to minimize these gaps by varying all P_t and X_j .⁵

example, execute a pose-dependent mission, or dynamically control its state. For an AR application, the loss is undesirable as well, since it represents downtime in the application. What we would like is a SLAM system with always-available pose estimation.

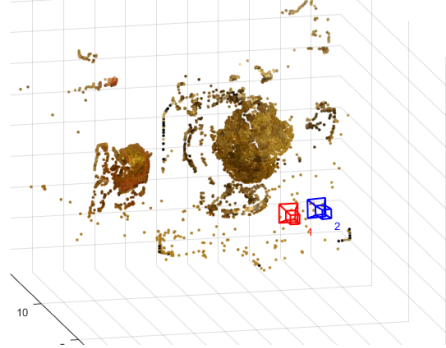
Track quality is a single number metric that is meant to capture how well a new image explains a geometric model of the scene, w.r.t. previous images. When a new camera image arrives, the SLAM system extracts features from the image and attempts to establish correspondence between features in this image and the previous one. The system then attempts to fit a geometric model from these putative correspondences, and along the way compute the track quality. See Figure 5. For example, ORB-SLAM [11] fits a geometric model via RANSAC, during which putative correspondences are split into two sets known as “inliers” and “outliers”, and track quality is defined as the ratio of inliers to the total number of putative correspondences.

The reason that we want pose estimation to stop during track failure is that the alternative - continuing to estimate pose - is likely to lead to poor results. If the geometric model is erroneous, the current estimated pose and new map points will be erroneous. Subsequent pose and map point estimates, being dependent on previous ones, are likely to be erroneous as well. We can see that poor image data can lead to a vicious cycle of poor estimates of both the platform’s pose and the map.

⁵<http://www.mathworks.com/help/vision/examples/structure-from-motion-from-two-views.html>



(a) Finding the correspondence between features of two images. The two images are made to overlap to ease human analysis. Lines indicate correspondence.



(b) Fitting a geometric model, known as the **fundamental matrix**, from putative correspondences. Camera-shaped wireframes represent the relative poses between two consecutive timesteps.

Figure 5: Prior steps to computing tracking quality in a SfM-based visual SLAM system⁶

Next, we propose a way to achieve always-available pose estimation.

5 Inertial-measurement unit as an odometer

Odometry is the estimation of the change in a platform’s position over time. Traditionally, the term is used in the context of wheeled robots equipped with rotary encoders. With some trigonometry and the odometry given by the robot’s wheels, we can compute an estimate of the robot’s pose w.r.t to its starting position. The key difference between such pose estimation and the localization in SLAM is that SLAM attempts to maintain global consistency in its estimates, while odometry does not.

A simple way to obtain odometry during track failure is by means of a sensor called the **inertial measurement unit** (IMU). An IMU is itself comprised of two sensors - a 3-axis **gyroscope**, which measures angular velocity, and a 3-axis **accelerometer**, which measures linear acceleration. Even on its own, an IMU can theoretically provide odometry with the following steps. First, compute the orientation of the IMU by integrating gyroscope readings over time. Second, transform accelerometer readings from the IMU frame to the world frame, using results from step one. Lastly, double-integrate results from step two over time to get linear position of the IMU in the world frame. In practice though, this naive approach’s performance worsens over time, as we will see below.

Among the various IMU types, the **strapdown MEMS IMU** is the most common [16] due to its low cost, weight, power and volume. To better understand its drawbacks, we briefly describe how such a IMU works and its characteristics.

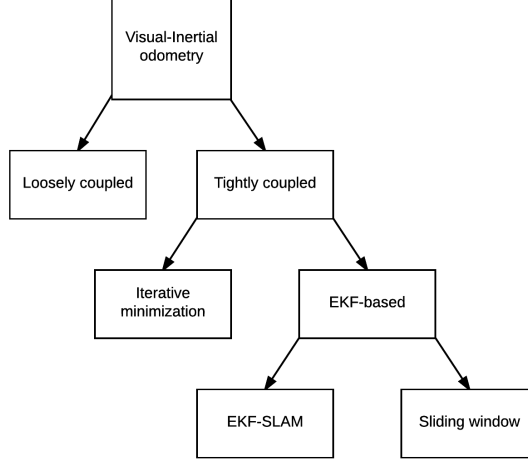


Figure 6: Taxonomy of visual-inertial odometry

5.1 Gyroscope modelling

A MEMS gyroscope measures **Coriolis force**, which is the apparent force experienced by a mass m that is moving in a rotating frame. Formally, this force is given by

$$\mathbf{F}_C = -2m\boldsymbol{\omega} \times \mathbf{v} \quad (2)$$

where $\boldsymbol{\omega}$ is the angular velocity of the frame (and is what we want the gyroscope to provide), \mathbf{v} is the velocity of the mass, \times is the cross product operator, and bold represents a vector quantity. A simple implementation of a MEMS gyroscope would consist of some force sensors together with a mass m actuated at velocity \mathbf{v} . When the IMU rotates, the mass experiences a Coriolis Force which is measured by the force sensors. The hardware computes $\boldsymbol{\omega}$, knowing \mathbf{F}_C , m , and \mathbf{v} .

A MEMS gyroscope reading can be modelled as

$$\boldsymbol{\omega}_m = \boldsymbol{\omega} + \mathbf{b}_g + \boldsymbol{\epsilon}_g \quad (3)$$

where $\boldsymbol{\omega}$ is the true angular velocity, \mathbf{b}_g is bias caused by noise in the electronics, and $\boldsymbol{\epsilon}_g$ is thermo-mechanical white noise. The bias term is usually modelled as a random walk over time, often characterized by $^\circ/\text{h}$ in manufacturer specifications, i.e., the standard deviation of \mathbf{b}_g grows linearly in time. The white noise term is usually modelled as zero-mean i.i.d. random variables, often characterized by $^\circ/\sqrt{\text{h}}$, since the integration of i.i.d. white noise has a standard deviation that grows to the square-root of time [7].

5.2 Accelerometer modelling

A mechanical MEMS accelerometer measures the displacement of a mass suspended by springs, all enclosed in the accelerometer. We can compute the linear acceleration of the accelerometer by measuring that displacement, knowing the spring coefficients and the mass, and applying Newton’s second law.

A simple implementation of such an accelerometer would consist of a mass m attached to a body by a spring of coefficient k , and a displacement sensor that measures x , the extension of the spring. Then the linear acceleration of the body is given by

$$a = -\frac{k}{m}x - g + \frac{d^2x}{dt^2} \quad (4)$$

where g is acceleration due to gravity.

A MEMS accelerometer reading can be modelled as

$$\mathbf{a}_m = \mathbf{R}(\mathbf{a} + \mathbf{g}) + \mathbf{b}_a + \boldsymbol{\epsilon}_a \quad (5)$$

where \mathbf{a} is the true linear acceleration in the world frame, \mathbf{g} is the gravity vector, \mathbf{R} is the rotational matrix from the world frame to the IMU frame, and \mathbf{b}_a and $\boldsymbol{\epsilon}_a$ are similar to their counterparts in the gyroscope model [7].

Given the characteristics of these sensors, it should be clear that integrating sensor readings will not provide accurate odometry. The more “well-behaved” terms are $\boldsymbol{\epsilon}_g$ and $\boldsymbol{\epsilon}_a$, due to their time-stationarity, i.e., their stochasticity does not change with time. But even these terms induce integrals that are a first order random walk. Such a random walk has a standard deviation which grows unbounded over time. This characteristic tells us that the IMU is a reliable odometer only for a short period of time.

6 Visual-inertial odometry

In the previous section, we considered IMU-only odometry. Since the baseline method in this report already utilizes a camera, it makes sense that in our efforts to compute odometry we incorporate camera readings as well! **Visual-inertial odometry** (VIO) is the estimation of the state of a platform using an IMU and a camera. The state vector typically includes rotation and translation terms, as well as higher order terms like linear velocity. As such, state estimation can be seen as a generalization of pose estimation.

We first describe the taxonomy of VIO [1, 8] before explaining how VIO can remedy the aforementioned flaw. See Figure 6 for a visual representation of the taxonomy.

VIO can be classified into **loosely-coupled methods** and **tightly-coupled methods** (Figure 7). Loosely-coupled methods process visual and IMU data in two separate filters that run at two different frequencies. The filtered IMU data narrows the search space of image features, while the filtered visual data

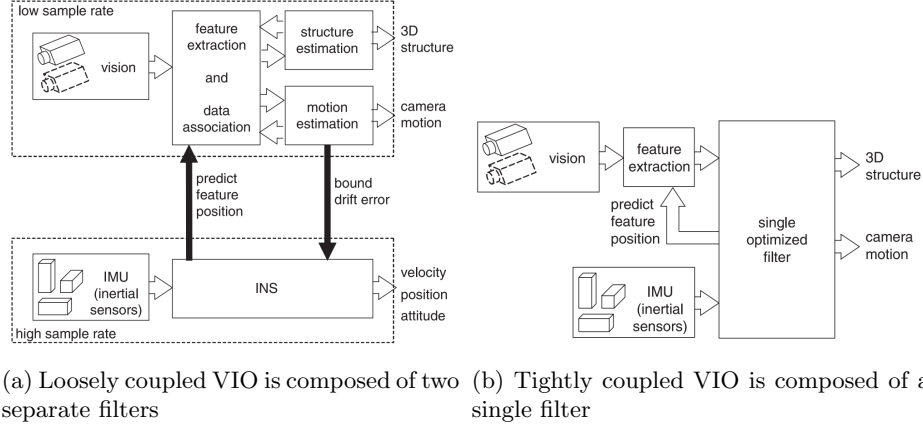


Figure 7: Difference between loosely and tightly coupled visual odometry [1]

provides an upper bound on the integration of IMU data [1]. Such methods tend to be cheaper to compute but may be sub-optimal estimators since sensor data is not fully exploited.

Tightly-coupled methods process visual and IMU data in a single filter. These methods can be further categorized into **iterative minimization methods**, which are basically bundle adjustment of a sliding window of sensor data, and **EKF-based methods**. Methods in the former category tend to be computationally expensive, being iterative methods, although expensive methods in the latter category take time quadratic in the number of tracked features [3].

EKF-based methods [9] assume that uncertainty in the platform’s state is Gaussian distributed. At each time step, the method performs a prediction step which incorporates IMU data, and a correction step which incorporates camera data. Methods primarily differ in the variables in the state vector, and consequently how camera data is incorporated in the correction step. EKF-based methods fall into two separate categories - **EKF SLAM methods** and **sliding window methods**.

In an EKF-SLAM method, the state vector consists of the current platform pose and real-world coordinates of all the features being tracked. In a sliding window method, the state vector consists of a sliding window of platform poses, from the current pose to some fixed time step into the past. Li and Mourikis [8] suggest that sliding window methods can be more accurate and consistent than iterative minimization methods, and at lower computational cost.

Coming back to our original discussion - we want always-available pose estimation, even during track failure due to poor camera data. We explained that an IMU is a reasonable odometer over a short period (e.g., during track failure, assuming the failure is short-lived), and in this section we implied that visual-inertial odometry should perform better than IMU-only odometry. But won’t the poor camera data worsen the odometry instead? It turns out that

some VIO methods are robust to such problems. For instance, Weiss et al. proposed a method called **inertial-optical flow** which only requires two features per image [14]. Such methods are a suitable stand-in during track failure, and hence are a remedy to our fatal flaw. A V-SLAM system that falls back on VIO during track failure may be termed **visual-inertial SLAM** (VI-SLAM).

7 Technical overhead of having an IMU

Adding an IMU to a platform has benefits, as we have seen, but incurs technical overhead as well. Here, we argue that the overhead is small.

Obviously, adding an IMU increases the cost, weight and power consumption of the platform. Power consumption includes the power to power the IMU, and the power to move the added weight of the platform in the case of a robotic platform. MEMS technology has enabled these increases to be small. For instance, Weiss et al. developed an IMU-and-camera chip that weighs 20 grams. Furthermore, most modern mobile devices include an IMU. As such, utilizing the IMU would not be an overhead in such cases.

Another overhead is the need to time-synchronize the IMU data and camera data. Ideally, we want the IMU and camera to capture their respective data simultaneously. Alternatively, if there is a time delay between the IMU data capture and image data capture, we should know the delay to take it into account. In other words, the time stamp of IMU and camera data should be w.r.t. a common clock. Time synchrony can be achieved through hardware or software [4].

8 Conclusion

We identify a flaw in visual SLAM which is detrimental in both the applications of autonomous robotics and AR. This flaw can be remedied by adding an IMU to the platform (if there wasn't one already) and running VIO during track failure. Since the technical overhead of this improvement is minimal, there is little reason in practice not to embrace VI-SLAM over V-SLAM.

References

- [1] Peter Corke, Jorge Lobo, and Jorge Dias. An introduction to inertial and visual sensing. *The International Journal of Robotics Research*, 26(6):519–535, 2007.
- [2] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.
- [3] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

- [4] Paul Furgale, Joern Rehder, and Roland Siegwart. Unified temporal and spatial calibration for multi-sensor systems. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1280–1286. IEEE, 2013.
- [5] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
- [6] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.
- [7] Mingyang Li and Anastasios I Mourikis. Consistency of ekf-based visual-inertial odometry. *University of California Riverside, Tech. Rep*, 2011.
- [8] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.
- [9] Peter S Maybeck. *Stochastic models, estimation, and control*, volume 2. Academic press, 1982.
- [10] Raúl Mur-Artal and Juan D Tardós. Fast relocalisation and loop closing in keyframe-based slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 846–853. IEEE, 2014.
- [11] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *Robotics, IEEE Transactions on*, 31(5):1147–1163, 2015.
- [12] Veljo Otsason, Alex Varshavsky, Anthony LaMarca, and Eyal De Lara. Accurate gsm indoor localization. In *UbiComp 2005: Ubiquitous Computing*, pages 141–158. Springer, 2005.
- [13] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.
- [14] Stephan Weiss, Markus W Achtelik, Simon Lynen, Margarita Chli, and Roland Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 957–964. IEEE, 2012.
- [15] Brian Williams, Georg Klein, and Ian Reid. Real-time slam relocalisation. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.
- [16] Oliver J Woodman. An introduction to inertial navigation. 2007.