Noname manuscript No.
(will be inserted by the editor)

A SLAM Overview from a User's Perspective

Udo Frese · René Wagner · Thomas Röfer

Received: date / Accepted: date

Abstract This paper gives a brief overview on the Simultaneous Localization and Mapping (SLAM) problem from the perspective of using SLAM for an application as opposed to the common view in SLAM research papers that focus on investigating SLAM itself.

We discuss different ways of using SLAM with increasing difficulty: for creating a map prior to operation, as a black-box localization system, and for providing a growing online map during operation.

We also discuss the common variants of SLAM based on 2-D evidence grids, 2-D pose graphs, 2-D features, 3-D visual features, and 3-D pose graphs together with their pros and cons for applications. We point to implementations available on the Internet and give advice on which approach suits which application from our experience.

Keywords SLAM · localization · navigation

1 Introduction

Navigation is the "*science of getting ships, aircraft, or spacecraft from place to place*" (Merriam-Webster's Collegiate Dictionary). It is also the science of getting mobile robots from place to place, a problem that is central to mobile robotics and has been subject to extensive research. According to Leonard and Durrant-Whyte [22] this involves answering the three questions "*Where am I*", "*Where am I going?*" and "*How do I get there?*".

SLAM, i.e., the Simultaneous Localization and Mapping problem, aims at a fully autonomous answer to the question "Where am I?" by providing an autonomously built map. Figure 1 illustrates what SLAM is about. A robot observes the environment relative to itself. We call the system robot for simplicity, in principle it could also be a sensor that is somehow moved around. If the

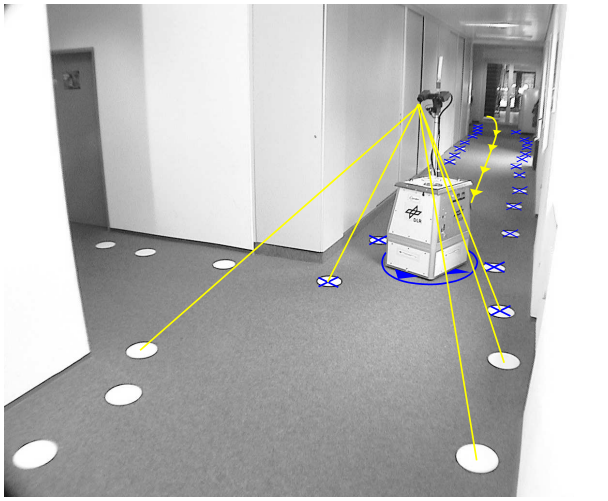

Fig. 1 *What is Simultaneous Localization and Mapping (SLAM)?* A robot observes the environment relative to its own unknown pose. Also, the relative motion of the robot is measured. From this input, a SLAM algorithm computes estimates of the robot's pose and of the geometry of the environment. In the example illustrated here, a camera on a robot measures the relative position of artificial features on the floor (light lines), while the sensor's motion is provided by the robot's odometry (light arrows). The output is the robot's pose (dark, circled arrow below the robot) and the global position of each feature (dark crosses).

This work has been partly supported under DFG grant SFB/TR 8 Spatial Cognition and BMBF grant 01IS09044B.

Deutsches Forschungszentrum für Künstliche Intelligenz GmbH,
Enrique-Schmidt-Str. 5, 28359 Bremen
Tel.: +49-421-218-64207
E-mail: udo.frese@dfki.de

Universität Bremen
Enrique-Schmidt-Str. 5, 28359 Bremen

environment was known the robot's pose (position and orientation) could be derived from this information. This is called *localization*. Conversely, if the robot's pose was known one could aggregate sensor readings corresponding to features in the environment in a global reference frame. This is called *mapping*. Now, in general, neither the environment nor the robot's pose is known but *both* must be estimated from the same data. This is called *Simultaneous Localization and Mapping*, i.e., SLAM and turned out to be much more difficult.

This problem has been explored since the late 1980s. In recent years, it has received enormous attention and has reached a certain level of maturity. Hence this paper takes the perspective of someone who wants to *use* SLAM rather than *investigate* SLAM. For a general overview we refer the reader to the textbooks [33, 34] and the articles [8, 2]. We follow the argument by Press et al [28], §1.0 and offer practical judgment on the different approaches because we believe this kind of advice is helpful to the practitioner. It is not infallible though and readers are invited to form an opinion of their own.

The paper is structured as follows: Section 2 explains loop closing, the probably most important phenomenon in SLAM. In Section 3 we discuss ways of using SLAM in applications and in Section 4 different SLAM variants regarding sensor and map representation.

2 Loop Closing

To understand SLAM, one has to realize that the input data for SLAM is relative information. Imagine a room at the end of a long winded corridor and the coordinate origin at its start (Fig. 2). Then the pose of that room is (implicitly) derived by the SLAM algorithm by concatenating many observed relations along the corridor. Each has some error and hence the global pose of the room can accumulate an arbitrarily large error. In contrast to that, the local shape of the room is only affected by observations inside the room and relatively precise. This uncertainty structure has been paraphrased by one of us as *"Certainty of relations despite uncertainty of positions."* [10].

Now, assume the robot moves along a closed loop, returns to the start of the loop, but has not realized this yet. Then the loop is not closed in the map due to the error accumulated (Fig. 2, left). Once the start of the loop is re-identified and the corresponding measurement is integrated into the map, the SLAM algorithm will close the loop. To achieve this, it has to "deform" the whole loop to incorporate the information of a connection between both ends of the loop without introducing a break somewhere else (Fig. 2, right).

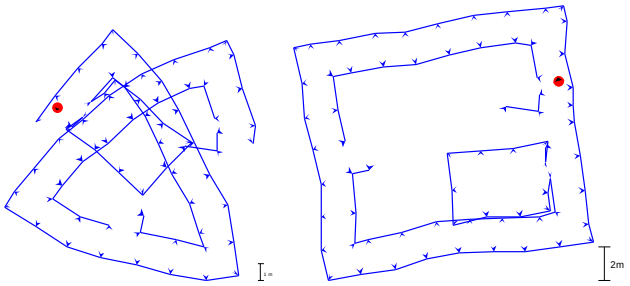

Fig. 2 The map before (left) and after (right) loop-closing. An animation of several random outcomes of this mapping process shows the structure of the uncertainty and can be downloaded from `www.informatik.uni-bremen.de/agebv/en/SlamDiscussion`.

Loop closing is a special situation, because a single observation changes the whole map estimate significantly. It is also the most challenging situation, for the following reasons:

- The system has to detect that it is at the same place again (data-association). Depending on the sensors used and the uncertainty accumulated, this is still an open research question and a major failure mode.
- The incorporation of the loop-closure into the map estimate has to be efficient enough. This has long been a central question but is nowadays solved [11].
- Any application that uses the SLAM map or pose estimate must handle the sudden jump when closing a loop. To avoid this problem, all places in the application, e.g., objects or way-points, must be defined relative to the pose when the robot was at these places. Then they will jump with the map. However, this behavior is difficult to implement (Sec. 3.4).

3 Different Ways of Using SLAM

An application can use SLAM for different purposes. This is seldom mentioned beyond the general motivation of a map for navigation. As the difference is important for the application, we discuss it here briefly.

3.1 Offline-SLAM for Map Acquisition

The most frequent and easiest way to use SLAM is map acquisition. During a map learning phase, the robot is manually steered through the environment. Sensor data is recorded and annotations, such as place labels, are given. Later, in a post-processing step, SLAM computes a map from the sensor data that is henceforth kept fixed and used for localization, path-planning, etc. during the actual operation of the system.

A prominent example of this approach is the robotic museum tour guide Minerva [35], where a grid-map (Sec. 4.1) is acquired with offline-SLAM during the setup in the museum, and then used for localization and navigation.

Similarly, the shopping companion robot TOOMAS [14] acquires a grid-map with a laser rangefinder. In practice, it was essential to align that map with a floor plan to link shop items to their respective shelves [9]. Therefor, the robot was driven to marker points with known coordinates in the floor plan. Graph-based SLAM (Sec. 4.2) can integrate both types of information. However, in practice, the pose estimate was already precise enough with odometry and marker points alone. This is a very minimalistic SLAM algorithm, where at every marker point the accumulated error is linearly distributed on the trajectory since the last marker point. The lessons learned were: floor plans are not only helpful but often necessary for a useful map and a simple algorithm can do the job with some additional information.

Another example is [21], where 3-D SLAM creates a map of a multi-story parking garage that is then later used for autonomously driving a car through the garage.

Overall, mapping prior to operation has advantages although or because it is scientifically less ambitious:

- All other modules of the system can use coordinates with respect to the map as usual without taking care of map changes during operation.
- Slow but reliable offline SLAM algorithms can be used. Indeed, many algorithms are efficient approximations to non-linear least-squares. If time is available, a textbook least-squares algorithm such as the Levenberg-Marquardt-algorithm[28, §15.7] is simpler and better.
- The human operator can check the map visually after it is computed.
- The operator can help building the map interactively by providing a few additional links for closing loops and for aligning with a floor plan. These can be precise marker points or rough links where the metric information is obtained from sensor data. Closing loops manually will hardly take a minute, but simplify the problem greatly for the SLAM algorithm. This procedure works well with graph-based SLAM (Sec. 4.2).

3.2 Online-SLAM for Localization

A more complex mode of operation is using SLAM as a black-box localization system. SLAM performs both localization and mapping online during operation, but the application uses only the pose information and the map is just for the SLAM algorithm itself. Sometimes, a part of the map is given a-priori so the localization operates in an application specific coordinate system and error accumulation is limited. Unlike with incremental sensors, e.g., odometry or inertial sensing, the error does not grow as long as the robot operates in a known part of the environment. It only grows when extending the map.

In particular, when the robot returns to a known place, the system knows this fact with the precision of the environment sensor. This allows the robot for instance to travel back or repeat a path taken before with great precision. There is one caveat however: all estimates can change, e.g., because of loop-closure. Then a different pose estimate is reported at a place visited before. So trajectories cannot simply be stored as lists of coordinates. This problem is negligible if the error is small or it can be solved by a more elaborate handling of stored coordinates (Sec. 3.4).

An example of this approach is the MALTA project, where a forklift truck navigates autonomously in a paper reel storage depot [1] (Sec. 4.3). The positions of some pillars are given a-priori to anchor the map and limit error accumulation. SLAM estimates the vehicle pose and the position of paper reels. Since these are moved continuously into and out of the depot, this map changes during operation and cannot be acquired a-priori. The SLAM pose estimate is directly fed into the motion controller driving the forklift truck, which is only possible since the error is limited by giving pillar positions in advance.

An alternative is "sensor odometry", e.g., visual or laser odometry. This means estimating the motion from sensor data but without building a map. It is often much more precise than an incremental motion sensor such as odometry or inertial sensing. Unlike in real SLAM, the error accumulates even when moving through the same environment. Sensor odometry can be implemented by a SLAM algorithm that forgets parts of the map having moved by, avoiding efficiency problems.

Konolige et al. [20] realize visual odometry, i.e., estimating motion from the images of a stereo-camera. Visually distinct points, so-called center-surround-extrema are detected in the images and fed into a visual SLAM algorithm (Sec. 4.4). The motion during a sequence of images can be obtained by estimating the end-pose relative to the start-pose, thus providing visual odometry.

Visual odometry has also been used on the Mars Exploration Rovers Spirit and Opportunity [7] with a simplified approach matching successive images. Mathematically this is SLAM with only one unknown pose and hence it did run on the rover's 20 MHz CPU.

Going beyond odometry, these local relative poses can be used as links in graph-based SLAM (Sec. 4.2) if not only successive poses but all spatially close poses are matched. The resulting two-stage SLAM algorithm can handle very large maps, because it avoids estimating millions of distinct points in a single computation.

A different application is augmented reality [18], where 3-D graphics is overlaid on a real camera image. Therefor, the camera's pose is needed that can be provided by visual SLAM. Again, the map is only used by SLAM and (mostly) not by the application itself. If a separate sensor was used for tracking the camera, this sensor would have to be calibrated and synchronized with the camera very precisely, since humans are very sensitive to erroneous relative motion between real and virtual objects. By using the same images for SLAM and for overlaying, these problems are avoided. The videos in [18] show such a system in operation, which at the moment is limited to a desktop-size workspace.

3.3 Online-SLAM for a Continuously Updated Map

The most complex way of using SLAM and also the main motivation for SLAM research is to continuously build a growing map, localize in that map, and use both results for robot control and the overall application. The challenge here is that the map changes during operation, and in case of loop-closure it changes even abruptly.

Tele-operating mobile robots for instance is difficult because the operator only sees a small fraction of the environment in the robot's cameras and thus has to memorize and grasp the overall spatial structure of the environment. This is most notable in the area of Safety, Security, and Rescue Robotics (SSRR), where the environment is often complex and difficult to grasp. Birk et al. [4] as well as Kleiner and Dornhege [19] have shown 2-D SLAM in harsh realistic field experiments and reported that the maps were helpful to the operator.

It is tempting to propose that a robot should autonomously explore its environment instead of being manually controlled as in Section 3.1, in particular in service robotics. Indeed, Stachniss et al. have shown autonomous exploration even with a team of robots [32] in a lab setting. However, we would argue against this idea, unless there is compelling reason for it in a specific application. Autonomous exploration gives rise to a set of very difficult problems beyond the SLAM and exploration algorithm itself: obstacles or hazards invisible to the sensor; forbidden areas for the robot; how to tell the robot where to stop exploring; and generally the increased chance of failure. In our opinion, all these problems more than outweigh the convenience of not having to manually steer the robot. Indeed, this is one example that for an application a clever combination of human and robot capabilities is a better design paradigm than the ultimate challenge of full autonomy.

In contrast to that, abandoned mine mapping is an application, where autonomy is essential. The underground environment is too dangerous for people and also prevents wire-less teleoperation. One of the most impressive systems that uses SLAM in this way is the Groundhog robot by Thrun et al. [36]. In an autonomous experiment, it went 308m into an abandoned mine, where it detected that the way was blocked by a broken beam and went back. On the way back software problems required manual intervention, but still this is an impressive real-world exploration experiment.

3.4 Handling Changing Map Estimates in Applications

In the discussion above, we have mentioned repeatedly that changing estimates are difficult for the application to use. We will now discuss approaches to solve these problems.

Usually, the robot stores local obstacles, intermediate way-points or planned trajectories as part of its motion controller for some meters of travelling. Now, imagine the pose estimate obtained from SLAM changes significantly: these stored coordinates are suddenly far away from the robot and motion control fails.

For such temporary spatial data, i.e., coordinates only kept for a few meters, this problem can be solved by storing them in odometry coordinates. This is the coordinate system, where the robot pose is defined by the uncorrected odometry. Its origin slowly drifts due to odometry error but never jumps. So, all local control modules can operate in odometry coordinates. Spatial data from the map should be converted into odometry coordinates at controlled points in time keeping in mind that there might be a jump since the last conversion.

Long term spatial data, e.g., global way-points, object locations, or application specific places, however, must be represented in a way that they consistently change with the SLAM map estimate. Formally, this is clear as these locations are usually obtained relative to a robot pose during teach-in. Hence, the most elegant way is to represent them in the SLAM algorithm as part of the map, where they are updated when the map changes. Alternatively, they can be stored relative to the respective robot pose used for defining them and SLAM updates these old robot poses. This is convenient for graph-based SLAM which updates all past poses anyway.

On the downside, this approach is often incompatible with an existing software architecture since it re-

quires every module to refer to the SLAM module for storing long term spatial data. So, in summary, it is possible to represent spatial data compatible to the SLAM philosophy of a changing map estimate but it requires considerable effort. This, again, underlines the practical advantage of a prior mapping phase, where during operation global coordinates can be used as usual.

4 Different Variants of SLAM

In this section, we will briefly discuss the most common variants of SLAM regarding the sensor input and the map representation. Following a user's perspective, we will stress the assumptions made on the environment and point to implementations available on the Internet. Unless otherwise noted implementations are in C/C++.

4.1 2-D Evidence Grid-based SLAM

In this SLAM variant a laser rangefinder is used to obtain a bitmap-like map where white corresponds to free space, black to obstacles, and gray to unknown space. Odometry is (usually) needed as a second input. A particle filter based solution called *GMapping* has been developed by Grisetti et al [12] and evaluated on a broad range of data-sets. Recently, it has been integrated in the *Robot Operating System* (ROS) [29] where it is complemented by a module based on the Adaptive Monte Carlo Localization (AMCL) algorithm [34, §8] that localizes on the maps generated by GMapping.

The interface in ROS is clear and well documented. While GMapping itself is research-code, we know of several groups that have used it successfully and we believe it is the most practical approach for indoor service robot navigation or similar applications. The laser rangefinder input also serves for obstacle avoidance and the obtained map models free-space, i.e., it can be used for navigation, path-planning, or exploration. For indoor navigation, the fact that a laser rangefinder only sees a horizontal slice of the environment usually causes only small problems. Sometimes, however, overhanging obstacles, e.g., tables and chairs, need to be marked, either physically (e.g., with a string) or in the map.

4.2 2-D Graph-based SLAM

This SLAM variant maintains a map consisting of a graph of 2-D poses, usually robot poses at regular intervals. The input are links with uncertain spatial relations between these poses. Links between successive poses are directly obtained from odometry, most links are generated by scan-matching, i.e., by an algorithm that aligns two laserscans thereby computing their relative pose [6]. Additionally, a policy must be implemented that controls which scans to match, because matching all overlapping pairs of scans is too slow. If feasible (Sec. 3.1), additional links can be manually added by the user. Beyond SLAM, scan-matching can correct odometry and localize in a map represented by scans.

Graph-based maps are slightly more difficult to use than grid maps: In offline SLAM (Sec. 3.1), one can easily compute a grid map by overlaying all laserscans according to the estimated poses. When operating online, the poses can be used as reference frames as described in Section 3.4. Further, they constitute a navigation graph for course path planning. However, free-space information for metrical path planning is only available in the original laserscans, not in the graph-based map.

To our knowledge, no full graph-based SLAM implementation is available, but the essential components are. Censi [6] has implemented several scan matchers and Grisetti et al [13] provide a SLAM algorithm, *TORO*, operating on a graph of pose relations. Both are well documented and easy to use. The above mentioned control policy needs to be implemented by the user. To increase precision, scans can be aligned globally by matching all points from all scans simultaneously [27].

Overall, graph-based SLAM with scan-matching is a slightly more complicated alternative to grid-based SLAM. It handles larger maps and allows the user to incorporate manual loop-closing information easily. While a laser rangefinder is the most popular sensor, in principle graph-based approaches can use any sensor-data that can be matched, i.e., where a relative pose can be obtained from comparing the sensor data at two points in time. In particular, cameras can be used in 3-D.

4.3 2-D Feature-based SLAM

By far the largest fraction of the SLAM literature considers a variant of SLAM, where the map consists of 2-D point features which are observed relative to the robot (Fig. 1). For an application with feature-based SLAM, the key question is what type of features to use and how to detect and identify them reliably. Options include fiducials (Fig. 1), trees which make good outdoor features [15], the paper reels reported in [1], and walls or corners in buildings. Observations of a wall without its end-corners need special treatment [30].

The most popular example of feature-based SLAM is mapping the Victoria Park in Sydney [15] where trees are detected with a laser rangefinder. Feature-based

SLAM is the preferred variant for research on the core algorithm. Hence most articles use this dataset and simulations to evaluate their algorithms.

For every feature observation the identity of the observed feature needs to be obtained, which is called data-association. The simplest solution is nearest-neighbor association, where within a given threshold an observation is associated with the nearest neighbor already inside the map. This is simple but works only if the map uncertainty is smaller than the separation between features. Some algorithms compare a group of features with the map taking map uncertainty into account [26], however, this sometimes fails due to consistency problems [17]. Some algorithms even require the application to provide data-association. In our opinion, unless the map uncertainty is low enough for nearest-neighbor, data-association is a major hurdle that requires a deeper understanding of SLAM to master.

Another issue is that a set of point features is useful for localization but does not give any free-space or connectivity information. It is advisable to retain old robot-poses at regular intervals in the map so a navigation-graph can be obtained from the trajectory used during mapping. This is possible with most algorithms, but also surprising as many algorithm take quite an effort to be able to forget (marginalize out) old robot poses.

As an example of a feature-based SLAM algorithm, a well-documented implementation of the treemap algorithm is available [11]. The implementation clearly separates the core computational engine from the front-end that defines different quantities to be estimated and different measurement equations. This simplifies incorporating various kinds of geometric objects, such as corners, walls, and poses. However, it provides no data-association leaving the user with nearest-neighbor. A MATLAB implementation of the I-SLSJF provided by Huang et al [17] is also well documented but also only uses nearest-neighbor data-association.

Overall, to non-experts, we can recommend feature-based 2D- SLAM only if reliable features can be obtained in the target application and the uncertainty is small enough for nearest-neighbor data-association.

4.4 3-D Visual SLAM

Visual SLAM is similar to feature-based SLAM as the map is a set of visually distinct points and the observations are image positions of these points. The offline version of this problem has been studied in photogrammetry as "bundle adjustment" [37] and in computer vision as "structure from motion" [16, §18, App. 6.3] for a long time and can be considered a mature technology. The canonical procedure consists of the following steps:

- Detect interesting feature points, e.g., with the so-called SIFT [25] or SURF [3] detector.
- Compute a descriptor vector for each feature point describing its visual appearance, e.g., with the SIFT or SURF descriptor.
- Match features in each pair of images by finding nearest neighbors in the descriptor space.
- Use Random Sample Consensus (RANSAC) [16] with the 5/7 point algorithm on the pairs found: RANSAC repeatedly draws 5 or 7 pairs from the feature matching result, therefrom computes a relative camera pose using the so-called 5/7 point algorithm [23, 16], and counts how many pairs are compatible with that pose. It then chooses the repetition with the highest count. If it is above a threshold these pairs are accepted as observations of the same feature.
- Run least-squares estimation (bundle adjustment) on all observations of all images accepted by RANSAC.

All building blocks for visual SLAM are available: The SURF detector and descriptor, RANSAC with the 7-point algorithm and nearest neighbor computation are included in the well documented *OpenCV* computer vision library [5]. Lourakis and Argyros [24] provide *sba*, a well established and documented implementation of bundle adjustment using sparse matrix methods. The package *bundler* [31] implements the full procedure.

There are some caveats for applications. First, with a single camera the overall scale of the scene is undefined. Even if one distance is known, e.g., by looking at a known object, the scale can drift over time. A stereo-camera solves this problem and simplifies data-association. When all features are first matched between left and right image, only a 3-point algorithm is needed later. Second, cameras have a smaller field of view (45-100°) than laser rangefinders making it difficult to have enough overlapping features between images, in particular when viewing the same place from reverse directions. One or two sideways looking cameras alleviate this problem but this is often undesirable.

The *sba* package needs some seconds of computation time for 50 images and 5000 features, which is enough for offline SLAM but not for processing every frame of a video sequence online. Klein and Murray [18] address this challenge with their Parallel Tracking and Mapping framework (PTAM), where one thread tracks the camera motion relative to a fixed map and another thread updates the map at larger intervals with sparse bundle adjustment. The implementation is available, but should be considered research code.

If the application requires free motion in 3-D, as with aerial vehicles or in rough-terrain navigation, an inertial sensor can complement computer vision.

Overall, using images to compute a feature map prior to operation is an established technique, online visual SLAM is still an area of active research.

4.5 3-D Graph-based SLAM

This variant is the 3-D analogon to Section 4.2. 3-D point clouds are obtained at certain robot poses and pair wise aligned with a scan-matching algorithm. The results form a graph of uncertain pose relations from which the most likely poses are computed. As in 2-D, finally all scans can be matched globally to increase precision.

A 3-D scan is usually obtained by tilting or rotating a conventional 2-D laser rangefinder while the robot is at a temporary halt. Taking the scans while moving would require motion tracking, e.g., by an inertial sensor which is a precision challenge. While stop-and-go during operation could be problematic, we believe it is feasible to map the environment this way before operation (Sec. 3.1). Localization with scans in motion is much easier than full SLAM in motion.

Nüchter [27] provides *SLAM6D*, a well documented implementation of 3-D scan matching and 3-D graph-based SLAM. Considerable effort went into making it efficient enough for practical applications, because the amount of data is much higher in 3-D.

A point cloud representation is not very convenient for most tasks, in particular motion planning, so one can convert it into a 3-D voxel representation. Wurm et al [38] implemented *octomap*, a library for maintaining and updating a 3-D voxel-map in a memory efficient oct-tree representation.

Overall, 3-D SLAM with tilted laser rangefinders is feasible whenever a 3-D map is needed, e.g., often outdoors. We recommend being more careful than with 2-D SLAM when planning an application, because the environments under scope of 3-D SLAM are more diverse and involved than the 2-D office environments.

5 Conclusions

In this article, we have discussed different ways of using SLAM in an application and different variants of SLAM, pointing to available implementations where possible. To conclude with an overall recommendation: Indoor navigation is a mature technology. We recommend the use of a laser rangefinder as the primary sensor, acquiring a map with the robot prior to operation, and 2-D evidence-grid based SLAM. If interactive mapping is possible, 2-D graph-based SLAM will be more reliable but needs more custom implementation work.

In 3-D, the situation is more challenging. If stop-and-go is feasible, 3-D graph-based SLAM with a tilted laser rangefinder is a good choice. For fast motion, as often seen in aerial applications, we recommend computer vision with a stereo camera as the method of choice (perhaps combined with inertial sensing). However, concerning visual SLAM, only visual odometry can be considered mature enough to be used without a deep understanding of SLAM itself.

References

1. Andreasson H (2010) An application of SLAM to localize AGVs. Tech. Rep. urn:nbn:se:oru:diva-10393, Örebro University, project web page: `aass.oru.se/Research/Learning/malta`
2. Bailey T, Durrant-Whyte H (2006) Simultaneous localisation and mapping (SLAM): Part II state of the art. Robotics and Automation Magazine 13(3):108–117
3. Bay H, Tuytelaars T, Gool LV (2006) SURF: Speeded up robust features. In: Ninth European Conference on Computer Vision, software: included in the *OpenCV* library.
4. Birk A, Schwertfeger S, Pathak K (2009) A networking framework for teleoperation in safety, security, and rescue robotics (SSRR). IEEE Wireless Communications, Special Issue on Wireless Communications in Networked Robotics 6(13):6–13
5. Bradski G, Kaehler A (2008) Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly, software: `opencv.willowgarage.com`
6. Censi A (2008) An ICP variant using a point-to-line metric. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Pasadena, software: `purl.org/censi/2007/csm`
7. Cheng Y, Maimone MW, Matthies L (2006) Visual odometry on the mars exploration rovers. IEEE Robotics and Automation Magazine pp 54–62
8. Durrant-Whyte H, Bailey T (2006) Simultaneous localisation and mapping (SLAM): Part I. Robotics and Automation Magazine 13(2):99–110
9. Einhorn E (2009) Personal communication
10. Frese U (2006) A discussion of simultaneous localization and mapping. Autonomous Robots 20(1):25–42
11. Frese U, Schröder L (2006) Closing a million-landmarks loop. In: Proceedings of the IEEE/RSJ International Conf. on Intelligent Robots and Systems, Beijing, pp 5032–5039, software: `www.openslam.org/treemap.html`
12. Grisetti G, Stachniss C, Burgard W (2007) Improved techniques for grid mapping with rao-

blackwellized particle filters. IEEE Transactions on Robotics 23(1), software: www.openslam.org/gmapping.html
13. Grisetti G, Stachniss C, Burgard W (2009) Nonlinear constraint network optimization for efficient map learning. IEEE Trans on Intelligent Transportation Systems 10(3):428–439, software: www.openslam.org/toro.html
14. Gross, et al (2009) TOOMAS: Interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In: Proc. IEEE/RJS International Conference on Intelligent Robots and Systems, St. Louis, pp 2005–2012
15. Guivant JE, Nebot EM (2001) Optimization of the simultaneous localization and map building (SLAM) algorithm for real time implementation. IEEE Trans on Robotics and Automation 17:242–257, dataset: services.eng.uts.edu.au/~sdhuang/research.htm
16. Hartley RI, Zisserman A (2004) Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press
17. Huang S, Wang Z, Dissanayake G, Frese U (2009) Iterated D-SLAM map joining: Evaluating its performance in terms of consistency, accuracy and efficiency. Autonomous Robots 27(4), software: www.openslam.org/2d-i-slsjf.html
18. Klein G, Murray D (2007) Parallel tracking and mapping for small AR workspaces. In: Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07), Nara, software: www.robots.ox.ac.uk/~gk/PTAM
19. Kleiner A, Dornhege C (2009) Operator-assistive mapping in harsh environments. In: Proc. of the IEEE Int. Workshop on Safety, Security and Rescue Robotics (SSRR), Denver
20. Konolige K, Agrawal M, Sola J (2007) Large-scale visual odometry for rough terrain. International Symposium on Research in Robotics
21. Kümmerle R, Hähnel D, Dolgov D, Thrun S, Burgard W (2009) Autonomous driving in a multi-level parking structure. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, Kobe, pp 3395–3400
22. Leonard J, Durrant-Whyte H (1992) Dynamic map building for an autonomous mobile robot. The International Journal on Robotics Research 11(4):286 – 298
23. Li H, Hartley R (2006) Five-point motion estimation made easy. In: International Conference on Pattern Recognition, pp 630–633
24. Lourakis M, Argyros A (2009) Sba: A software package for generic sparse bundle adjustment. ACM Trans on Mathematical Software 36(1), software: www.ics.forth.gr/~lourakis/sba
25. Lowe D (2004) Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision 60(2):91 – 110
26. Neira J, Tardós J (2001) Data association in stochastic mapping using the joint compatibility test. IEEE Trans on Robotics and Automation 6(17):890 – 897
27. Nüchter A (2009) 3D Robotic Mapping. Springer Tracts in Advanced Robotics (STAR), Springer Verlag, software: openslam.org/slam6d.html
28. Press W, Teukolsky S, Vetterling W, Flannery B (1992) Numerical Recipes, Second Edition. Cambridge University Press, Cambridge
29. Quigley M, et al (2009) ROS: an open-source robot operating system. In: Proceedings of the ICRA-Workshop on Open-Source Robotics, software: ros.org
30. Rodriguez-Losada D, Matia F, Galan R (2006) Building geometric feature based maps for indoor service robots. Robotics and Autonomous Systems 54(7):546–558
31. Snavely N, Seitz S, Szeliski R (2008) Modeling the world from internet photo collections. International Journal of Computer Vision 80(2):189–210, software: phototour.cs.washington.edu/bundler/
32. Stachniss C, Mozos OM, Burgard W (2009) Efficient exploration of unknown indoor environments using a team of mobile robots. Annals of Mathematics and Artificial Intelligence 52(2):205–231
33. Thrun S, Leonard J (2009) Springer Handbook of Robotics, Springer Verlag, chap 34. B. Siciliano and O. Khatib (eds.)
34. Thrun S, Burgard W, Fox D (2005) Probabilistic Robotics. MIT Press
35. Thrun S, et al (2000) Probabilistic algorithms and the interactive museum tour-guide robot minerva. The International Journal of Robotics Research 19(11):972–999
36. Thrun S, et al (2004) Autonomous exploration and mapping of abandoned mines. Robotics and Automation Magazine 11(4):79–91
37. Triggs W, McLauchlan P, Hartley R, Fitzgibbon A (2000) Bundle adjustment – A modern synthesis. In: Triggs W, Zisserman A, Szeliski R (eds) Vision Algorithms: Theory and Practice, LNCS, Springer Verlag, pp 298–375
38. Wurm KM, Hornung A, Bennewitz M, Stachniss C, Burgard W (2010) Octomap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In: Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation, software: octomap.sf.net