TagSLAM: Robust SLAM with Fiducial Markers

Bernd Pfrommer[1,2] Kostas Daniilidis[1]

***Abstract*— TagSLAM provides a convenient, flexible, and robust way of performing Simultaneous Localization and Mapping (SLAM) with AprilTag fiducial markers. By leveraging a few simple abstractions (bodies, tags, cameras), TagSLAM provides a front end to the GTSAM factor graph optimizer that makes it possible to rapidly design a range of experiments that are based on tags: full SLAM, extrinsic camera calibration with non-overlapping views, visual localization for ground truth, loop closure for odometry, pose estimation etc. We discuss in detail how TagSLAM initializes the factor graph in a robust way, and present loop closure as an application example. TagSLAM is a ROS based open source package and can be found at `https://berndpfrommer.github.io/tagslam_web`.**

I. Introduction

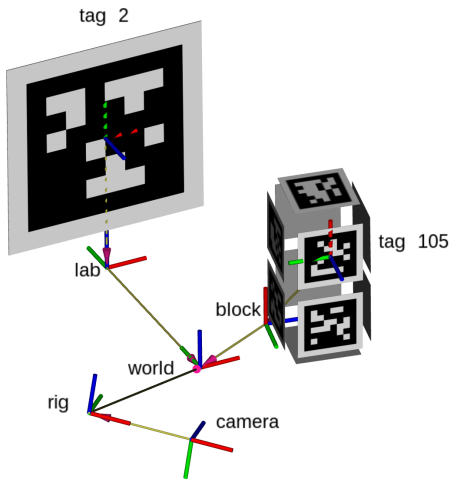

Fig. 1. Single-camera TagSLAM scene with dynamic "rig" and "block" bodies, and a static "lab" body.

Accurate and robust Simultaneous Localization And Mapping (SLAM) algorithms are arguably one of the corner pieces for building autonomous systems. For this reason, SLAM has been studied extensively since the 1980s [1]. In a nutshell, SLAM methods take sensor data such as camera images as input and extract easily recognizable features ("landmarks"). The landmarks are entered into a map such that when the same landmark is detected again later, the observer's pose can be determined by triangulation. In graph based SLAM, a nonlinear optimizer such as GTSAM [2] is used to simultaneously optimize the pose of the observer and the location of the landmarks, given the sensor measurements.

Much research effort [1] has gone into solving the hard aspects of SLAM. One of them is recognizing previously seen landmarks ("loop closure"). This is often difficult if landmarks are observed from a different viewpoint, or under different lighting conditions. The other is maintaining a map of landmarks. To limit memory consumption and achieve fast retrieval, landmarks must be at some point discarded, efficient retrieval databases must be updated and queried, adding considerable complexity [3]. If loop closure fails, the estimated camera pose starts to drift, which is a well-known problem with local-map-only algorithms such as visual-inertial odometry (VIO).

In many situations, and in particular for laboratory experiments these problems can be sidestepped with a mild intervention to the environment. By placing visual fiducial markers such as the popular AprilTags [4] in a scene, one can create a small set of artifical landmarks, typically numbering less than 100. The TagSLAM framework presented in this paper is designed for such a situation. With only a few landmarks to consider, memory and retrieval speed are no longer critical, and loop closure is guaranteed once a tag is successfully decoded by the AprilTag library. If one or more tags are visible at all times, then TagSLAM can perform both tracking and loop closure.

But more importantly, tags permit the introduction of geometric constraints that have been observed by other means. For instance, markers can be placed inside a building at coordinates that are known because a highly accurate building plan is available [5]. Another alternative is measuring the locations of select tags with a laser distance measuring tool. When a measured marker is later recognized by the moving camera, it will pull the trajectory to the reference location. As we demonstrate in the experimental section, this can augment existing SLAM or VIO methods. By utilizing well-known tags in a few places, and ubiquitous but anonymous feature points elsewhere, one can leverage the advantages of both approaches.

Even though tag based SLAM is much simpler than general SLAM, some remaining difficulties are addressed in this paper. Since TagSLAM utilizes GTSAM [2], a graph based non-convex optimizer, care must be taken to properly initialize all starting values. A bad initial value can cause the optimizer to fail or to converge to a local minimum. In the present work we give a detailed description of how

[1]University of Pennsylvania School of Engineering and Applied Science. We gratefully acknowledge support through the following grants: NSF-IIP-1439681 (I/UCRC RoSeHuB), NSF-IIS-1703319, NSF MRI 1626008, and the DARPA FLA program.

[2]Thanks to Chao Qu and Ke Sun for many useful discussions.

we achieve robust initialization to make TagSLAM a tool that can be applied to a wide range of real-world situations without parameter tuning.

It is worthwhile noting that SLAM is a rather general algorithm, and therefore can solve several related, simpler problems as well. If for example the camera pose is known, one can estimate the pose of the landmarks. Likewise, if a map of the landmarks is known, the camera pose can be inferred. TagSLAM inherits all of this flexibility. In contrast to traditional SLAM packages however, where moving landmarks are filtered out, TagSLAM can explicitly model and track multiple objects that have tags attached to them. This means TagSLAM can also be used for pose estimation of objects other than cameras, as shown in Fig. 1. In fact, TagSLAM has been deployed for the University of Pennsylvania's Smart Aviary project in a scenario where *all* cameras are static. This is not directly possible with any SLAM package that we are aware of.

TagSLAM's robustness and availability as an open-source ROS [6] package make it particularly accessible for researchers new to robotics. It is now being used at the University of Pennsylvania's GRASP lab for extrinsic camera calibration with non-overlapping views, loop closure on visual-inertial odometry benchmarks, tag mapping, object pose estimation, and more. TagSLAM is also employed in a forthcoming project by the University of Michigan's DASC laboratory.

II. Related Work

A review of the vast SLAM literature is beyond the scope of this paper. For a thorough survey we refer to [1]. The focus of this section will be on SLAM or localization techniques that operate off of fiducial markers.

Several works on this topic however do not employ graph based optimizers. For instance, Ref. [7] presents a Kalman-filter based approach for localizing off of multiple AprilTags. The authors use localization via tags for aerial construction surveillance because often GPS signals are compromised by surrounding buildings. In further contrast to TagSLAM, there is no mapping: all tag poses must be entered into a map beforehand.

Ref. [8] presents a visual-inertial approach for obtaining ground truth positions from a combination of inertial measurement unit (IMU) and camera. They also utilize AprilTags, but in contrast to TagSLAM use an Extended Kalman Filter (EKF), a method that is well suited for their goal to effectively provide a real-time, outdoor, low-cost motion capture system. Since TagSLAM uses a graph optimizer instead of a filter, it can be operated in non-real time to leverage observations from both the past and the future, resulting in smoother trajectories. TagSLAM also does not require the presence of an IMU and can work with cameras alone. This simplifies the experimental setup, but also reduces robustness to motion blur during quick rotations. In case IMU data is available, TagSLAM can be set up to operate in a similar fashion to the system described in [8]. One could for example run a standard monocular VIO algorithm without the use of any tags, and feed the computed odometry updates into TagSLAM to be combined with AprilTag observations for loop closure and drift removal.

A combination of AprilTags and GTSAM were used in Ref. [5] to provide ground truth for VIO benchmarking. The software developed there served as a precursor to TagSLAM, but it was lacking a framework for generalization to other scenarios, and required that approximate tag world poses be available beforehand.

Closely related to the present paper is the underwater SLAM performed in [9]. The authors also leverage AprilTags and the GTSAM factor graph optimizer to obtain vision based ground truth poses and extrinsic calibration. The focus of their work however is more on providing a solution for a particular problem, whereas TagSLAM aims to be a general purpose framework that can be easily applied to many different settings. In fact, all the factors in [9] that are related to AprilTags are already implemented in TagSLAM. The code structure of TagSLAM is designed such that adding the problem-specific XHY and ZPR factors from [9] should be straight forward.

Some of the difficulties with optimizer initialization discussed in section V have been nicely described in Ref. [10]. The authors overcome these issues by using additional depth sensor information. In contrast, TagSLAM leverages a combination of multiple tags, multiple cameras, or pose history to arrive at a robust starting guess. This works also outdoors or over longer distances, where depth sensing is not an option.

III. Model Setup

TagSLAM uses a few simple abstractions with which complex scenarios can be built without the need to write any code. This section will introduce the concepts and establish the necessary notation.

We denote as ${}^{B}\mathbf{T}_{A}$ an SE(3) transform that takes vector coordinates ${}^{A}\mathbf{X}$ in reference system A and expresses them in B:

$$ {}^{B}\mathbf{X} = {}^{B}\mathbf{T}_{A}\,{}^{A}\mathbf{X}. \tag{1} $$

Such a transform defines a pose. We distinguish two kinds:

- Static pose: the transform ${}^{B}\mathbf{T}_{A}$ is independent of time. A static poses is represented by a single variable for the optimization process. Note that this does not mean the pose must be known from the beginning (i.e have a prior), but could be discovered as image data becomes available.
- Dynamic pose: the transform ${}^{B}\mathbf{T}_{A}(t)$ is time dependent. Such a pose is assumed to change, i.e. the optimizer will allocate a new variable for every time step t. Sufficient input data must be provided at time t to solve for ${}^{B}\mathbf{T}_{A}(t)$.

A *body* is an object that can have tags and cameras attached to it. Its pose is always given with respect to world coordinates, and can be classified as static or dynamic depending on the nature of the body. For static bodies, an optional prior pose may be specified.

Tags must have a unique id, and each tag must be associated with a body to which it is attached. Tags without association are ignored, unless a default body is specified to which any unknown tag will be attached upon discovery. In contrast to body poses, tag poses are *always static*, and are expressed with respect to the pertaining body. A tag pose prior is optional, so long as that is not required to determine the body pose.

Camera poses, like tag poses, are *static*, and given with respect to the body to which the camera is attached. This body is referred to as the camera's "rig", although from a modeling point of view, it is a body like any other, and can have for instance tags attached to it. A prior camera pose (extrinsic calibration) is optional provided the optimization problem can be solved without it.

With bodies, tags, and cameras, a rich set of SLAM problems can be modeled, as shown for a simple single-camera scenario in Figure 1. It is sufficient to provide the static priors for the lab-to-world (${}^{\mathrm{w}}\mathbf{T}_l$), tag-2-to-lab (${}^{l}\mathbf{T}_2$), tag-105-to-block (${}^{\mathrm{b}}\mathbf{T}_{105}$), and camera-to-rig transform (${}^{\mathrm{r}}\mathbf{T}_{\mathrm{c}}$). The remaining poses ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{b}}(t)$ and ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{r}}(t)$, as well as the missing poses of the tags on the block can be determined from the images arriving at the camera.

IV. Factor Graph

Our SLAM is formulated as a bipartite factor graph with two types of nodes: the variables (poses) which are elements of the set Θ, and the factors, which constrain the variables via the set of measurements $\mathcal{Z}$. The factor graph defines a probability $P(\Theta|\mathcal{Z})$ that assumes its maximum a posteriori (MAP) value for the optimal variable set Θ^*, given the measurements:

$$\Theta^* = \arg\max_{\Theta} P(\Theta|\mathcal{Z}) \ . \tag{2}$$

The set of variables $\mathcal{Z}$ contains:

- The camera poses ${}^{\mathrm{body(cam}\ j)}\mathbf{T}_{\mathrm{cam}\ j}$, with respect to the bodies they are attached to.
- The tag poses ${}^{\mathrm{body(tag}\ k)}\mathbf{T}_{\mathrm{tag}\ k}$, relative to their respective bodies.
- The world poses ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}\ l}$, of static bodies.
- The time-dependent world poses ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}\ m}(t)$, $t = 1 \dots N_{\mathrm{t}}$ of dynamic bodies.

The likelihood is expressed [2] as a product of factors $p^{(i)}$ that connects the variables with each other via measurements to form the desired graph structure:

$$P(\Theta|\mathcal{Z}) = \prod_i p^{(i)}(\Theta|\mathcal{Z}) \ . \tag{3}$$

To make the factors $p^{(i)}$ computationally tractable, we follow the standard approach [2] and model them as Gaussians:

$$g(x;\mu,\Sigma) = \exp(-\frac{1}{2}||x \ominus \mu||^2_{\Sigma}) \ . \tag{4}$$

Here, x is the variable, μ the center of the Gaussian, and Σ defines the Mahalanobis distance. Note the use of the $\ominus$ operator, which reduces to straight subtraction for elements of a vector space, but produces 6-dimensional Lie algebra coordinates when applied to elements on the SE(3) manifold:

$$\mathbf{T}_A \ominus \mathbf{T}_B = \left[[\log(\mathrm{Rot}(\mathbf{T}_B^{-1}\mathbf{T}_A))]^{\vee}_{\vee}, \mathrm{Trans}(\mathbf{T}_B^{-1}\mathbf{T}_A)^{\top} \right]^{\top} \ . \tag{5}$$

In (5) Rot() and Trans() refer to the rotational and translational part of the SE(3) transform, respectively, log() is the matrix logarithm, and $\vee$ denotes the *vee* map operator. Equipped with the definition of a Gaussian on SE(3), we can now introduce the basic factors $p^{(i)}$ from Eq. (3).

Absolute Pose Prior. This unary factor can be used to specify a prior pose $\mathbf{T}_0$ with noise Σ for e.g. a tag or a camera:

$$p_A(\mathbf{T}|\mathbf{T}_0,\Sigma) = g(\mathbf{T};\mathbf{T}_0,\Sigma) \ . \tag{6}$$

Relative Pose Prior. With this binary factor, a known transform $\Delta\mathbf{T}$ between two pose variables can be specified, with noise Σ:

$$p_R(\mathbf{T}_A,\mathbf{T}_B|\Delta\mathbf{T},\Sigma) = g(\mathbf{T}_B^{-1}\mathbf{T}_A;\Delta\mathbf{T},\Sigma) \ . \tag{7}$$

If odometry body pose differences $\Delta\mathbf{T}_{\mathrm{odom}}(t)$ with noise σ are available from e.g. a VIO algorithm running alongside TagSLAM, a relative pose prior of $p_R({}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}}(t), {}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}}(t-1)|\Delta\mathbf{T}_{\mathrm{odom}}(t),\sigma)$ can be used to insert the odometry updates into the pose graph.

Tag Projection Factor. The output of the tag detection library is a list of tag IDs and the corresponding image coordinates $\mathbf{u}_c$ (in units of pixels) of the corners $c = 1 \dots 4$ of every tag. This gives rise to one quaternary tag projection factor per tag:

$$p_T({}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}}, {}^{\mathrm{rig}}\mathbf{T}_{\mathrm{cam}}, {}^{\mathrm{body}}\mathbf{T}_{\mathrm{tag}}, {}^{\mathrm{w}}\mathbf{T}_{\mathrm{rig}}|\{\mathbf{u}_c\},\sigma_p) = \prod_{c=1\dots4} g(\Pi({}^{\mathrm{cam}}\mathbf{T}_{\mathrm{rig}}\,{}^{\mathrm{rig}}\mathbf{T}_{\mathrm{w}}\,{}^{\mathrm{w}}\mathbf{T}_{\mathrm{body}}\,{}^{\mathrm{body}}\mathbf{T}_{\mathrm{tag}}\mathbf{s}_c);\mathbf{u}_c,\sigma_p) \ . \tag{8}$$

Here, $\mathbf{s}$ refers to the corner coordinates in the tag reference frame, i.e. $\mathbf{s}_1 = [-l/2,-l/2,0]^{\top}$, $\mathbf{s}_2 = [l/2,-l/2,0]^{\top}$, $\mathbf{s}_3 = [l/2,l/2,0]^{\top}$, $\mathbf{s}_4 = [-l/2,l/2,0]^{\top}$ for a tag of side length l. A sequence of transforms expresses $\mathbf{s}$ in camera coordinates, after which the function Π projects [11] the point onto the sensor plane and converts it to pixel coordinates. The noise parameter σ_p is a diagonal matrix that reflects the accuracy of the tag library's corner detector, which is usually assumed to be about one pixel.

We can visualize the factor structure of Eq. (3) by means of a graph as shown in Fig. 2 for the scene from Fig. 1. In Fig. 2, black squares represent factors, whereas circles denote pose variables to be optimized. The prior factors p_A constrain the static poses ${}^{l}\mathbf{T}_2$, ${}^{\mathrm{w}}\mathbf{T}_l$, ${}^{\mathrm{r}}\mathbf{T}_{\mathrm{c}}$, and ${}^{\mathrm{b}}\mathbf{T}_{105}$. A tag projection factors p_T arising from an observation of Tag 2 determines the dynamic rig pose ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{r}}(t)$, whereas an observation of Tag 105 likewise yields the block pose ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{b}}(t)$. Assuming that odometry for the rig is provided by some external algorithm, there is a relative pose factor p_R connecting the rig poses for t and $t+1$. Two more tag observations at $t+1$ generate additional factors that further constrain rig and block poses at $t+1$.

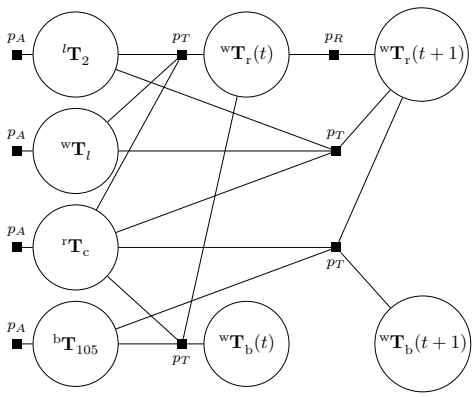

Fig. 2. Factor graph of Eq. 3 for the scene shown in Fig. 1

V. Robust Initialization

Nonlinear non-convex optimizers such as GTSAM [2] are iterative solvers and hence rely on a starting guess that is reasonably close to the optimum. If for example a camera pose is initialized such that one of the observed tag corner points lies behind the camera, the optimizer will likely fail.

Several sources of error can contribute to poor pose initialization. TagSLAM has been used for several projects already, and in our experience, human errors are frequently the root cause, such as inaccuracies or outright typos when entering measured tag poses, intrinsic or extrinsic calibration errors, duplicate tag ids, errors in tag size due to printing or misspecfication, using tags that are not planar, or supplying unsychronized stereo images. One of the design goals of TagSLAM was to produce, as much as possible, a reasonable result with a bounded error even with compromised input data, such that at least the scene can be visualized for further analysis.

Even when all input data is correct, initializing a camera or tag pose can be challenging, for example because the corner of the tag is not detected accurately, which may happen under motion blur, or when a tag is partially occluded. As described in Ref. [10], there are two camera poses from where a single tag looks quite similar, with only perspective distortion distinguishing between them. For a tag that is barely large enough to be detected and is viewed at a shallow angle, a tag corner error of just a single pixel can lead to a dramatically different pose initialization. Localizing off of a single tag is therefore intrinsically difficult, and should be avoided. Satisfactory results from a single tag can only be expected in combination with odometry input or when the tag image is sufficiently large.

When multiple cameras and several tags are involved with known or unknown poses it is anything but obvious which measurements to use, and in which order. For example, should the pose of camera 1 be established first from the tag corners, then the pose of camera 2 via a known extrinsic calibration, or the other way round? Which tags should be used for this purpose if several are visible?

In case the tag poses are known, one might be tempted to answer the last question with: use the corner points of all tags simultaneously with a perspective n-point method [12] (PnP). In practice we find that PnP is not robust to misspecified tag poses. Moreover when it fails it is not clear which of the tags is in error, necessitating an expensive elimination process. For this reason we base all pose initialization on homographies [11] from a minimum set of tags only, carefully picking which tags to use, and in which order. The remainder of this section will describe how exactly this is done.

To fully exploit the history of observations, two separate graphs are maintained: the *full graph* contains factors and variables pertaining to all measurements up to the current time, whereas the *optimized graph* only has those variables and factors that are sufficiently constrained to form a well conditioned optimization problem. All incoming measurements are thus entered immediately into the *full graph*, but only find their way into the *optimized graph* when the variables can actually be initialized.

Usually several static poses can be initialized right away because a prior is available. In Fig. 2 for example, the poses ${}^{l}\mathbf{T}_2$, ${}^{\mathrm{w}}\mathbf{T}_l$, ${}^{\mathrm{r}}\mathbf{T}_{\mathrm{c}}$, and ${}^{\mathrm{b}}\mathbf{T}_{105}$ are determinable due to the prior factors shown to their left, and can therefore be directly inserted into the *optimized graph*.

A. Subgraph discovery

As measurements arrive, they give rise to new factors that create edges in the graph between existing and new variables. Variables connected to these factors may be rendered determinable, which in turn, through existing factors, may cause other variables to become determinable as well. Thus every new factor can give rise to a subgraph of newly determinable variables. We refer as *subgraph discovery* to the exhaustive traversing of the graph until no more new variables can be determined. During this process, all deteriminable variables and factors are entered into the subgraph. Further, an *initialization list* is created that contains the factors in the order they are discovered. The initialization list later governs the order in which the subgraph will be initialized. The discovered subgraph depends on the new factor from which the discovery is started, so potentially, each new factor gives rise to a different subgraph. In practice the subgraphs are connected, and often a set of new factors generates only a single subgraph.

Note that also factors arising from past measurements may enter the subgraph. For example, if only tags A and B with unknown pose were seen at time step t, no camera rig pose could be determined. But if at $t+1$ tags B and C are observed, and tag C has known pose, then the subgraph at $t+1$ will contain poses for tags A, B, and C, although tag A was observed in the previous step.

All variables of the so generated subgraph are determinable, but if any dynamic poses from previous time steps are present, they must be constrained or the problem is ill determined. This is done by inserting absolute priors for those poses. Fig. 3 shows a subgraph, derived from the graph in Fig. 2 during time step $t+1$. Note the removal of the tag

projection factors and of the block pose for time t, and the addition of a prior (colored red) on pose ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{r}}(t)$.

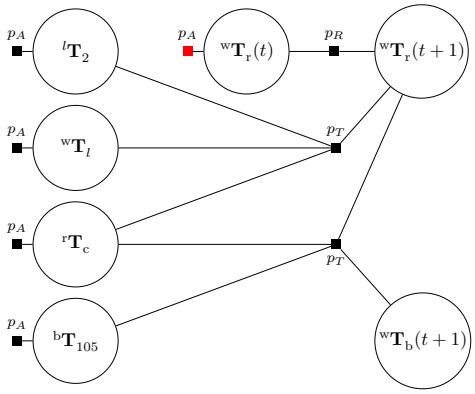

Fig. 3. Subgraph of graph in Fig. 2

To arrive at a complete set of subgraphs, all new factors are entered into the *new factor list*. The order of the factors is important, and will be discussed further below. An iteration over this list is performed, and, unless the new factor is already part of an already discovered subgraph, a new subgraph is generated by discovery. This procedure results in a set of disjoint subgraphs with all determinable variables and connected factors, as well as the corresponding *initialization lists*.

B. Order of subgraph discovery

What still remains to be settled is the order in which factors are entered into the *new factor list*. This strongly affects the *initialization lists* generated during subgraph discovery, and hence the order in which measurements are used for initialization. For instance in Fig. 3, there are two ways to initialize ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{r}}(t+1)$. One is through the relative pose factor with respect to ${}^{\mathrm{w}}\mathbf{T}_{\mathrm{r}}(t)$, the other through the tag projection factor on tag 2.

By examining the typical failure cases of several alternative approaches, the following order for entering factors into the *new factor list* was established:

1) Any relative pose priors. Note that this prefers odometry updates, which are typically more reliable in establishing a body pose than initializing it from tag observations.

2) Any tag projection factors. These factors are sorted in descending order by pixel area of the observed tag, irrespective which camera they were observed from. This means that larger tags will be used first to establish a pose.

3) Any factors that do not establish a pose. Examples are problem specific additional factors such as distance measurements between tag corners.

C. Optimization

Once the subgraphs have been obtained, their variables are initialized in the order specified by the corresponding *initialization list*. Then all subgraphs are optimized using GTSAM, in non-incremental mode, i.e. without using iSAM2 [2], and their error is evaluated. This step is analogous to model validation in RANSAC [12]. If a subgraph's error falls below a configurable threshold, the subgraph is accepted, and its factors and optimized values are transferred to the *optimized graph*. Subsequently, the *optimized graph* is optimized with an iSAM2 update step.

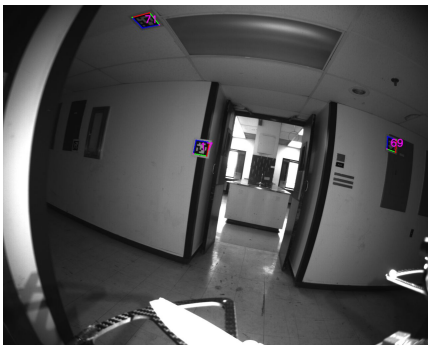

Fig. 4. View from the left camera of an Open Vision Computer. Detected AprilTags are shown in color.

In the rare case where a subgraph is rejected due to excessive error, an initialization with different ordering is attempted. Since exhaustively trying all possible orderings is computationally too expensive, an ad hoc procedure is adopted that was found to work satisfactorily in practice: the *initialization list* of the subgraph is rotated such that the first factor goes to the end of the list, and all other elements of the list advance by one. This implies that successively smaller tags are used to seed the initialization process. The subgraph is initialized with the new ordering, followed by optimization. In case the subgraph error is still too high, this process is repeated until the original ordering is reached again. If no initialization ordering leads to an acceptable error, the subgraph is rejected, thus preventing an outlier measurement from contaminating the *optimized graph*.

D. Diagnostics

Rejection of a subgraph is usually a strong indicator of faulty input parameters or misdetected tag corners. In most cases however, subgraph rejection is not fatal, and TagSLAM will successfully handle subsequent incoming data. For diagnosis, TagSLAM produces per-factor and time resolved error statistics. Such output is essential for tracking down e.g. incorrectly specified tag poses.

VI. Application Example

We illustrate the versatility of TagSLAM by showing how it can be used to achieve loop closure for VIO. The synchronized images and IMU data that serve as input for the VIO are collected with an Open Vision Computer [13]. During 13 minutes, a total of 15595 stereo frames are recorded at 20Hz along the 630m long trajectory through the rooms of an abandoned chemical laboratory. Fig. 4 shows an example image of some of the 57 tags that are strategically placed along the corridor. Their poses are deterimined from the wall orientations and from laser distance measurements with a Leica Disto D3a. The odometry is computed with the

stereo VIO algorithm as described in Ref. [14] but, running offline with abundant CPU resources available, we use a larger number of visual feature points to improve drift.

Fig. 5 shows the trajectories for VIO (cyan), loop-closed TagSLAM (magenta), and stereo ORB-SLAM2 [3] (yellow). The tag locations are visible in the map as well. All trajectories start at the same point at the bottom of the map, but only the TagSLAM trajectory returns correctly to the starting point. Both VIO and ORB-SLAM2 exhibit drift, and evidently ORB-SLAM2 does not achieve loop closure. This is not surprising since the hallway images look very different while returning. By combining tag projection factors from the camera images with relative pose factors from the odometry, TagSLAM by design closes the loop.

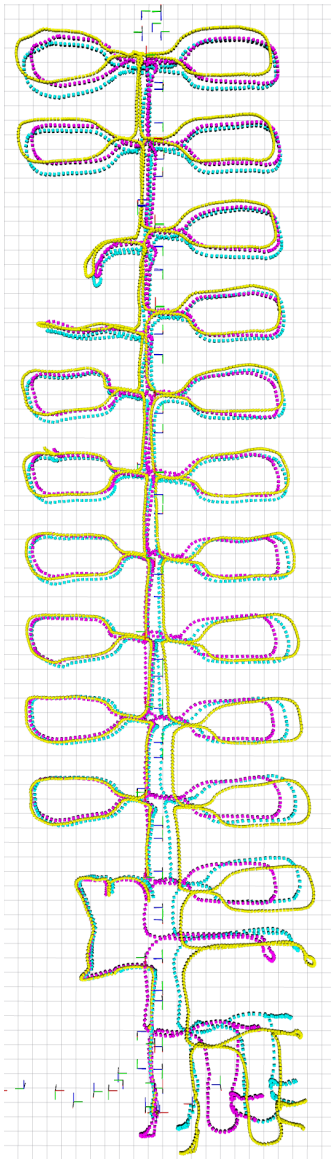

Fig. 5. Trajectory using VIO, TagSLAM, and ORB-SLAM2. The grid cell size is 1m.

Creating and incrementally optimizing the graph upon factor insertion takes 188s on an Intel i9-9900K 3.6GHz CPU, which is an average performance of about 12ms per frame. A final full (non-incremental) graph optimization adds another 4.3s to the total processing time. While 12ms time per frame seems to indicate the possibility of running in real time, individual frames can take longer to process, depending on iSAM2 relinearization. As the graph grows over time, so does the CPU load, and individual frames can take as much as 260ms. However, for situations where there already is a trusted map of tag poses available, TagSLAM can be configured to retain only the last two time steps in the graph, making it suitable for real-time operation.

VII. Conclusion

In this paper we present TagSLAM, a highly flexible front-end to the factor graph optimizer GTSAM that makes fiducial based visual SLAM and related tasks accessible to the robotics community by means of an open source ROS package located at `https://berndpfrommer.github.io/tagslam_web`.

References

[1] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Trans. Rob.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[2] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *IEEE Intl. Conf. Robot. Automat. (ICRA)*, May 2011, pp. 3281–3288.

[3] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras1," *IEEE Trans. Rob.*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *IEEE Intl. Conf. Intel. Rob. Sys. (IROS)*, Oct. 2016, pp. 4193–4198.

[5] B. Pfrommer, N. Sanket, K. Daniilidis, and J. Cleveland, "PennCOSY-VIO: A challenging Visual Inertial Odometry benchmark," in *IEEE Intl. Conf. Robot. Automat. (ICRA)*, 2017, pp. 3847–3854.

[6] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an Open-Source Robot Operating System," in *IEEE ICRA: Workshop on Open Source Robotics*, May 2009.

[7] N. Kayhani, A. Heins, W. Zhao, M. Nahangi, B. Mccabe, and A. P Schoellig, "Improved tag-based indoor localization of uavs using extended kalman filter," in *36th Intl. Symp. Automat. Robot. Constr. (ISARC)*, May 2019.

[8] M. Neunert, M. Bloesch, and J. Buchli, "An open source, fiducial based, visual-inertial motion capture system," in *19th Intl. Conf. Inform. Fus. (FUSION)*, July 2016, pp. 1523–1530.

[9] E. Westman and M. Kaess, "Underwater AprilTag SLAM and calibration for high precision robot localization," Carnegie Mellon University, Pittsburgh, PA, USA, Tech. Rep. CMU-RI-TR-18-43, Oct. 2018.

[10] P. Jin, P. Matikainen, and S. S. Srinivasa, "Sensor fusion for fiducial tags: Highly robust pose estimation from single frame RGBD," in *IEEE Intl. Conf. Intel. Rob. Sys. (IROS)*, Sept. 2017, pp. 5770–5776.

[11] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An Invitation to 3-D Vision*. SpringerVerlag, 2003, pp. 134–139.

[12] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, June 1981.

[13] M. Quigley, K. Mohta, S. S. Shivakumar, M. Watterson, Y. Mulgaonkar, M. Arguedas, K. Sun, S. Liu, B. Pfrommer, R. V. Kumar, and C. J. Taylor, "The Open Vision Computer," in *IEEE Intl. Conf. Robot. Automat. (ICRA)*, 2019, pp. 1834–1840.

[14] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.