2023 IEEE International Conference on Robotics and Automation (ICRA 2023)
May 29 - June 2, 2023. London, UK

kollagen: A Collaborative SLAM Pose Graph Generator

Roberto C. Sundin and David Umsonst

Abstract— **In this paper, we address the lack of datasets for – and the issue of reproducibility in – collaborative SLAM pose graph optimizers by providing a novel pose graph generator. Our pose graph generator, *kollagen*, is based on a random walk in a planar grid world, similar to the popular `M3500` dataset for single agent SLAM. It is simple to use and the user can set several parameters, *e.g.*, the number of agents, the number of nodes, loop closure generation probabilities, and standard deviations of the measurement noise. Furthermore, a qualitative execution time analysis of our pose graph generator showcases the speed of the generator in the tunable parameters.**

In addition to the pose graph generator, our paper provides two example datasets that researchers can use out-of-the-box to evaluate their algorithms. One of the datasets has 8 agents, each with 3500 nodes, and 67645 constraints in the pose graphs, while the other has 5 agents, each with 10000 nodes, and 76134 constraints. In addition, we show that current state-of-the-art pose graph optimizers are able to process our generated datasets and perform pose graph optimization.

The data generator can be found at `https://github.com/EricssonResearch/kollagen`.

I. Introduction

Both robots and extended reality devices, such as AR/VR glasses, permeate the public and private sector more and more in the form of, *e.g.*, industrial applications and consumer devices. For an autonomous operation, the robots and devices often need to build a map of the environment by themselves and orient themselves in said map. For this, Simultaneous Localization and Mapping (SLAM) is used [1]. Due to the sheer amount of deployed devices, the large scale environments, or both, it is deemed beneficial to perform collaborative SLAM (C-SLAM) among the devices [2]. Therefore, C-SLAM has become an active research area in recent years.

In C-SLAM, each agent has a front-end and a back-end. The front-end processes measurements and generates loop closures, which indicate correspondences between places that the agent has visited. Loop closures are not only made between poses of a single agent as in SLAM (called *intra-agent loop closures*), but also between poses of different agents (called *inter-agent loop closures*). Since the front-end of an agent is highly dependent on the types of sensors used by the agent, we put our focus on the back-end in this paper. In the back-end, the map is abstracted into a graph and graph optimization is performed to optimize over the agent's pose and the position of landmarks. The graphs of different agents will be connected via inter-agent loop closures. Often, the landmarks are marginalized out, such that the graph representing the map is a *pose graph*. The nodes of the pose graph represent the poses of an agent, while the edges represent constraints coming, *e.g.*, from odometry and loop closures. For SLAM, pose graph optimizers have been developed that are able to solve the pose graph optimization problem of the back-end in an incremental manner, such as iSAM2 [3], or provide a certifiably correct solution in an offline manner such as SE-Sync [4]. In addition to this, there are many datasets for pose graph optimization in the single agent case available, such as `M3500` [5], and `torus` and `rim` [6]. There are also several methods available for pose graph optimization in the C-SLAM case. For example, DPGO [7] presents a distributed way to obtain a certifiably correct solution to the collaborative (distributed) pose graph optimization problem, while a majorization minimization approach is proposed in [8]. Since the increased interest in SLAM led to an increase in SLAM datasets, one would expect that the increased interest in C-SLAM would lead to an increase in datasets for C-SLAM. However, to the best of the authors' knowledge, this has not been the case. Currently, SLAM datasets are often split into several parts to substitute as C-SLAM datasets, as it is, for example, done in [7].

To address this issue, we present *kollagen*, a collaborative SLAM pose graph generator. Inspired by the `M3500` dataset, *kollagen* generates pose graphs of agents moving randomly on a planar grid. For the sake of simplicity, *kollagen* only generates planar pose graphs but it could be easily extended to three-dimensional pose graphs. The generator provides both ground truth for the agents, and noisy odometry and loop closure measurements, where the generation of a loop closure is random and the generation probability is based on the proximity of the nodes. The generated pose graphs are saved in an intuitive file structure for the C-SLAM pose graphs, which lets the user easily extract the pose graph for each agent alone such that it also can be used for the single agent SLAM case. Moreover, we give the option to generate the dataset from specifications in a `.json` file. A qualitative analysis of the execution time indicates that the pose graph generator scales linearly with the number of agents and the number of nodes of each agent, but scales worse than linear in the radius used for determining loop closures due to the quadratic dependency on the radius. Furthermore, we demonstrate how to use the pose graph generator and provide two datasets generated with our pose graph generator, which can be readily used as example datasets by the research community to evaluate their algorithms and compare them to other algorithms.

The remainder of the paper is organized as follows. In Section II, we review related work concerning pose graph

Roberto C. Sundin and David Umsonst are with Ericsson Research, Stockholm, Sweden. E-Mail: {roberto.castro.sundin, david.umsonst}@ericsson.com

optimization datasets. In Section III, we present *kollagen* and describe its inner workings on a high level, while two example datasets are presented in Section IV, generated with our pose graph generator and optimized with state-of-the-art pose graph optimizers. Finally, Section V concludes the paper stating possible extensions of the presented pose graph generator.

Notation: The sets of natural and real numbers, and integers are $\mathbb{N}$, $\mathbb{R}$, and $\mathbb{Z}$, respectively. A zero-mean normal distribution with standard deviation σ is denoted by $\mathcal{N}(0, \sigma^2)$, while $\mathcal{U}\{a, b\}$ for integers $a < b$ denotes the discrete uniform distribution over the integers c such that $a \leq c \leq b$. Here, $\text{wrap} : \mathbb{R} \to [-\pi, \pi)$ is a function that maps an angle into its equivalent angle in the range $[-\pi, \pi)$[1], $\%$ is the modulo operator, and $\delta_i = \delta_{i0}$, where δ_{ij} is the Kronecker delta. A planar pose is defined as $\mathbf{x} = (x, y, \theta)$, where x and y are the position in the plane, and θ is the heading. We define $\|\mathbf{x}\|_{2,p}$ as the Euclidean norm of the position in the pose, *i.e.*, $\|\mathbf{x}\|_{2,p} = \sqrt{x^2 + y^2}$.

II. Related Work

The increased interest in SLAM in the last decades also raised the need for datasets for the proposed SLAM algorithms. A classical dataset for planar SLAM is `M3500`, introduced in [5]. In this dataset, a mobile agent navigates a grid world, similar to moving around city blocks in Manhattan. It consists of 3500 poses and 5600 constraints, which include both odometery and loop closure constraints. To investigate the robustness to noise of their SLAM algorithm, Carlone *et al.* [9] introduce three variations of `M3500`, which are based on the same ground truth as `M3500` but with increased levels of angular noise. A `M3500`-like dataset with 10000 nodes (called `W-10000` in [10] and `M10000` in [11]) is introduced in [12] as well as the `sphere` dataset. There also exist many datasets for 3D SLAM algorithms, such as `torus`, `cube`, `cubicle`, and `rim` introduced in [6]. Here, `rim` and `cubicle` are pose graphs obtained from real data, while `torus` and `cube` are generated via simulation. Often, datasets are also published when a new pose graph optimization algorithm for the SLAM back-end is published, as was the case for the datasets `City10000`, `CityTrees10000`, and `sphere2500`, which were released with iSAM [13]. A pose graph generator is provided with the g^{2}o framework [14], which is a general graph optimization framework. Their simulator allows for landmark nodes in the graph and different measurement models for the agents. However, it requires the g^{2}o framework such that it is not possible to easily integrate it with other pose graph optimizers.

While there exists an abundance of datasets for SLAM, there are not many publicly available datasets for C-SLAM [2]. Nine different datasets with real world data from multi-robot cooperative localization and mapping are presented in [15]. These nine datasets contain the movements of five ground robots and landmark positions. Five additional real world datasets with three ground, and one aerial, vehicle are provided by [16]. These five datasets include IMU measurements and camera images as well as AprilTags for each robot. Golodetz *et al.* [17] investigate collaborative 3D reconstruction in four different environments and provide the datasets for these environments. These C-SLAM datasets concentrate on real world examples, which can be used to evaluate the whole C-SLAM pipeline, *i.e.*, both the front-end and the back-end.

To obtain datasets for the back-end, it is common to split datasets for single agent SLAM algorithms, such as `M3500`, `torus` and `cubicle`, into several distinct parts for testing C-SLAM algorithms or to use simple simulations of agents trajectories and loop closures, such as a lawnmower pattern, see, for example, [7]. However, splitting single SLAM datasets will not necessarily lead to realistic multi-agent datasets, because the obtained trajectories of the agents are often confined to certain areas of the map and do not move as independent agents would potentially move. One could also use the data provided in [15]–[17] to obtain a pose graph, similar to how `cubicle` and `rim` are created. These pose graphs can be useful for benchmarking due to their realistic pose graphs, but for each dataset we would only obtain one pose graph for a certain amount of agents.

Due to the lack of pose graph datasets for C-SLAM, we present, *kollagen*, a data generator for C-SLAM pose graphs, in the following section, which lets the user generate an arbitrary amount of pose graphs efficiently, with an arbitrary amount of agents.

III. The kollagen Pose Graph Generator

In this section, we introduce *kollagen*, a collaborative SLAM pose graph generator. The *kollagen* data generator is a C++20 header-only library without third-party dependencies, making it easy to incorporate into existing projects. It produces a pose graph for each of the N_{agents} agents as well as inter-agent loop closures connecting the pose graphs of the single agents, where the number of agents, $N_{\text{agents}} \in \mathbb{N}$, is user-specified. Furthermore, it offers the ability to output pose graphs in the `.g2o` file format for use with popular graph optimizing frameworks such as, g^{2}o [14] and GTSAM [18], and ground truth in the `.tum` file format[2] for use with SLAM evaluation tools such as evo [19]. However, since the `.g2o` format does not extend naturally to the multi-agent case, we also present the *multi-g2o* structure (Section III-B) along with a parser for reading and writing to said structure.

Next, we describe how the collaborative pose graphs are generated, followed by an introduction to our new structure, *multi-g2o*, for saving the collaborative pose graphs, a section on the interoperability of *kollagen*, and finally a qualitative analysis of execution time of *kollagen*.

Remark 1. *Please note that in this section, we describe the inner workings of* kollagen *on a high level. If the reader desires more insights on how to use* kollagen*, we invite them to look at the documentation provided on our GitHub*

[1] A possible representation of wrap could be $\theta \mapsto \arctan 2(\sin\theta, \cos\theta)$.

[2] See *Ground-truth trajectories* section at `https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats`

page: `https://github.com/EricssonResearch/kollagen`

A. Data Generation

The data generator is inspired by the `M3500` dataset, and we, therefore, initiate this section with a quick review of `M3500`. The pose graph is generated by performing a random walk on a planar grid. The agent randomly chooses a direction and then takes four steps in that direction before choosing a new direction. This is done in an iterative manner to produce the ground truth. By taking the ground truth of `M3500` and comparing the relative translation between each node with the provided odometry measurements, we find that the measured position and heading are seemingly influenced by a zero-mean normal distribution with a standard deviation of 0.023 as depicted in Fig. 1.

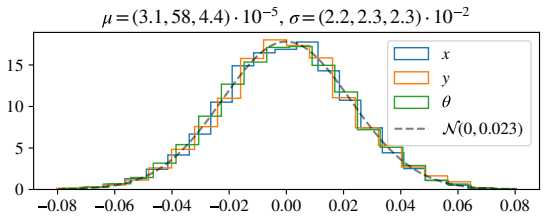


Fig. 1: Distribution of the noise influencing the position, (x, y), and the heading, θ, in the `M3500` dataset

1) Agent pose propagation: Inspired by the `M3500` [5] dataset, the agents are constricted to moving within a planar Manhattan-type world, that is, a grid of horizontal and vertical lines with equal spacing. Each agent's k:th pose, $\mathbf{x}_k = (x_k, y_k, \theta_k)$, is defined by its position $(x_k, y_k) \in \mathbb{Z}^2$ in the xy-plane, and its heading angle $\theta_k \in \frac{\pi}{2} \cdot \{-2, -1, 0, 1\}$. To generate the ground truth, the agents' poses are propagated as a two-dimensional random walk for N_{steps} steps, where $N_{\text{steps}} \in \mathbb{N}$ is defined by the user.

Each agent turns in a randomly selected direction (including not turning at all) before taking s steps forward. The parameter s is, by default, the same for all agents and was added taking inspiration from `M3500` in which four steps are taken (corresponding to $s = 4$) after possibly turning. From this description follows that each pose $\mathbf{x}[k]$ can be described as a function of the previous pose as follows

$$\theta_k = \text{wrap}\left(\theta_{k-1} + \delta_{k\%s} \cdot \frac{\pi}{2}\Theta_k\right), \tag{1a}$$

$$x_k = x_{k-1} + \cos\theta_k, \tag{1b}$$

$$y_k = y_{k-1} + \sin\theta_k, \tag{1c}$$

where $\Theta \sim \mathcal{U}\{-2 + n_d, 1\}$ for a user-defined parameter $n_d \in \{0, 1\}$, which enables the possibility of an agent to perform a 180° turn when randomly choosing a direction, and $\delta_{k\%s}$ guarantees that turning can only happen every s:th step. The initial pose, $\mathbf{x}_0$, of each agent is by default set to zero, but can also be adjusted by the user.

Remark 2. *In* kollagen*, the ground truth is generated on an integer grid according to Eq.* (1) *for numerical reasons. However, the results get re-scaled by a factor* $\frac{d}{s}$ *during saving, where* $d \in \mathbb{R}_{>0}$ *is specified by the user.*

2) Odometry measurement: In the previous section, we presented the ground truth pose propagation of an agent given by Eq. (1). In this section, we describe how an agent takes relative measurements between consecutive poses.

What we expect to obtain from a real system would be noisy measurements of the change in angle

$$\Delta\hat{\theta}_k = \theta_k - \theta_{k-1} + \tilde{\theta}_k \tag{2}$$

and the distance travelled $\Delta\hat{\ell}_k = \|\mathbf{x}_k - \mathbf{x}_{k-1}\|_{2,p} + \tilde{\ell}_k$, where $\tilde{\theta} \sim \mathcal{N}(0, \sigma^2_{\text{ang}})$ and $\tilde{\ell} \sim \mathcal{N}(0, \sigma^2_{\text{pos}})$. Here, σ_{pos} and σ_{ang} are the (user-defined) standard deviations of the position and angle, respectively. This then leads to an estimate of the relative change in xy-coordinates given by

$$\Delta\hat{x}_k = \Delta\hat{\ell}_k \cos\Delta\hat{\theta}_k, \tag{3a}$$

$$\Delta\hat{y}_k = \Delta\hat{\ell}_k \sin\Delta\hat{\theta}_k, \tag{3b}$$

Note that the measurement model described in Eqs. (2) and (3) is biased and has a correlated covariance matrix.

3) Pose graph alignment: Since each agent performs a random walk according to Eq. (1), it is likely that the agents' trajectories diverge due to the randomness in each agent's movement. Depending on the collaborative SLAM application, this can be considered unrealistic behavior, for example, in a constrained environment. Therefore, we provide an (optional) alignment step which increases the proximity of the agent poses. In this step, an anchorpoint corresponding to the arithmetic mean of the positions of each agent is calculated. Setting the first agent as a reference point, the pose graphs of all the other agents are translated such that their anchorpoints are aligned with the anchorpoint of the first agent.

4) Loop closures: Once the ground truth trajectory and odometry measurments are generated, and the pose graphs have potentially been aligned, *kollagen* generates loop closures as follows.

For each pose $\mathbf{x} \in \mathcal{P}_1$, we check whether there exists a pose $\mathbf{z} \in \mathcal{P}_2$ such that $\|\mathbf{x} - \mathbf{z}\|_{2,p} \leq R_{\text{lc}}$, where $R_{\text{lc}} \geq 0$. If $\|\mathbf{x} - \mathbf{z}\|_{2,p} = 0$, then a loop closure between $\mathbf{x}$ and $\mathbf{z}$ is added with probability $p_{\text{lc}} \in [0, 1]$. If $0 < \|\mathbf{x} - \mathbf{z}\|_{2,p} \leq R_{\text{lc}}$, then a loop closure between $\mathbf{x}$ and $\mathbf{z}$ is added with a probability proportional to a function that has its maximum at $\|\mathbf{x} - \mathbf{z}\|_{2,p} = 0$ and decreases with an increasing $\|\mathbf{x} - \mathbf{z}\|_{2,p}$. By default the function is set to $\exp\left(-5\left(\|\mathbf{x} - \mathbf{z}\|_{2,p}/R_{\text{lc}}\right)^2\right)$.

To generate *intra-agent* loop closures we choose $\mathcal{P}_1 = \{\mathbf{x}_k\}$ and $\mathcal{P}_2 = \{\mathbf{x}_i\}_{i=0}^{k-1}$, where $\mathbf{x}_i$ is the i:th pose of one agent. Then, for all agents, we iterate through the procedure described above for all $k \in \{1, \ldots, N_{\text{steps}}\}$ to obtain the intra-loop closures for each agent.

Inter-agent loop closures are performed similarly to the intra-agent loop closures, but they are performed pairwise between the agents for the $\binom{N_{\text{agents}}}{2}$ possible agent pairs, where $\mathcal{P}_1$ is then the set of all poses of one and $\mathcal{P}_2$ of the other agent, respectively. Note that both p_{lc} and R_{lc} do not have to be the same for intra- and inter-agent loop closures.

The measurement model for the loop closures is based on the true relative change in the coordinates and heading

with additive noise. For example, for an intra-agent LC let $\mathbf{x}_i \in \mathcal{P}_1$ and $\mathbf{x}_j \in \mathcal{P}_2$. Then the measurement is given by

$$\Delta\hat{\theta}_{ij} = \theta_i - \theta_j + \tilde{\theta}_{ij} \tag{4a}$$

$$\begin{bmatrix}\Delta\hat{x}_{ij}\\ \Delta\hat{y}_{ij}\end{bmatrix} = \begin{bmatrix}\cos\theta_j & \sin\theta_j\\ -\sin\theta_j & \cos\theta_j\end{bmatrix}\begin{bmatrix}x_i - x_j\\ y_i - y_j\end{bmatrix} + \begin{bmatrix}\tilde{x}_{ij}\\ \tilde{y}_{ij}\end{bmatrix}, \tag{4b}$$

where $\tilde{\theta} \sim \mathcal{N}(0, \sigma^2_{\text{ang}})$, $\tilde{x} \sim \mathcal{N}(0, \sigma^2_{\text{pos}})$, $\tilde{y} \sim \mathcal{N}(0, \sigma^2_{\text{pos}})$, and σ_{pos} and σ_{ang} are again the (user-defined) standard deviations of the position and angle for the loop closure, respectively. The measurement for a inter-agent loop-closure is defined in a similar manner.

Remark 3. *We choose a different measurement model for the loop closures than the one in Eqs.* (2) *and* (3)*, because the loop closures can be generated with different sensing modalities, which could results in more information than the distance travelled and the angle changed. Furthermore, note that in contrast to Eqs.* (2) *and* (3) *the measurement model in Eq.* (4) *is unbiased and has a diagonal covariance matrix.*

5) Generation of the information matrices: In a pose graph, each measurement has an (Fisher) information matrix I attached to it, which specifies how much information is contained in the measurement. The information matrix can also be interpreted as the inverse of the covariance matrix. For the odometry, the information matrix is correlated due to the measurement model given by Eqs. (2) and (3). If an agent moves in x-direction, the information matrix has correlated θ and y entries and similarly for a move in y-direction. For loop closures, we have $I = \text{diag}(\sigma^{-2}_{\text{pos}}, \sigma^{-2}_{\text{pos}}, \sigma^{-2}_{\text{ang}})$ due to the measurement model given by Eq. (4). Furthermore, we also give the user the ability to choose incorrect diagonal information matrix for both the odometry and loop closures, which can be used to evaluate how their pose graph optimization algorithms handle errors in the information matrix.

B. The multi-g2o structure

Here, we will introduce the new structure, *multi-g2o*, to save collaborative pose graphs, but first we review the `.g2o` format and point out its drawbacks for collaborative pose graphs.

The `.g2o` format is a plaintext format which consists of vertices (initial guesses), and edges (odometry and loop closures). To take the 2D planar case as an example, a `.g2o` file would consist of lines of

i) vertices of the form:
`VERTEX_SE2 K x_K y_K yaw_K`
for an identifier `K`$\in \mathbb{N}$, and (absolute) position/heading estimates given as floating point numbers `x_K`, `y_K`, and `yaw_K`.

ii) edges of the form:
`EDGE_SE2 K_A K_B Dx_K Dy_K Dyaw_K I`
which describes the relative transformation `Dx_K`, `Dy_K`, and `Dyaw_K` from the vertex with identifier `K_A` to the vertex with identifier `K_B`. The entry `I` denotes the estimated information matrix for the transformation and is given as the upper triangular entries in row-major order. In the 2D case this would hence be given by the six entries:
`I_11 I_12 I_13 I_22 I_23 I_33`

It could be argued that the `.g2o` format could easily be extended to multi-agent data by simply allocating a range of identifiers to each agent. The downside to this approach is that the identifiers given to each agent are not apparent within the `.g2o` file itself, and it is therefore up to the parser of the file to make this connection. To this end, we propose the *multi-g2o* structure as illustrated in Listing 1, where `MultiG2oFolder` is a directory containing

Listing 1: Multi-g2o structure

```
MultiG2oFolder
|-- inter_agent_lc.dat
|-- agent1
|   |-- posegraph.g2o
|   |-- agent1_GT.tum
|-- agent2
|   |-- posegraph.g2o
|   |-- ...
|-- agent...
|   ...
```

- `inter_agent_lc.dat`: a plaintext file with rows of similar structure as the edges in a `.g2o` file, that is,
  `A1 K1 A2 K2 Dx_K Dy_K Dyaw_K I`,
  where `A1` and `A2` indicate the agents connected by the inter-agent loop closures, while `K1` and `K2` are the identifiers of the connected nodes of the respective agents.
- one folder for each agent (here `agent1`, `agent2` etc.) containing the agent's pose graph with intra-agent loop closures as a `.g2o` file, and the ground truth of the agent in `.tum` file format (necessary for some SLAM evaluation frameworks).

This structure has the following benefits:

1. It clearly distinguishes data for each agent: it is possible to use the `.g2o` file of each agent separately in, *e.g.*, GTSAM, g^{2}o, etc.;
2. It is human-readable and easily navigable through the OS file browser;
3. The provided ground truth `.tum` file makes it easy to use with the popular evaluation tool evo [19].

C. Interoperability

The *kollagen* library can be used in a number of ways depending on where and how the data is meant to be used. The highest number of possibilities is given to users targeting C++: these users can include the library header `kollagen.h`, from which they can generate data for immediate use, without any need of saving to disk. An example of this is provided in `iSAM2example.cpp` where data is generated and converted to GTSAM types, allowing direct usage with, *e.g.*, iSAM2. If the end target is not C++, a more convenient option is to generate data by building and

running `generate.cpp` or to install the python package using pip. The generated data can then be saved as a single `.g2o` file which then needs to be parsed (no parser provided, but see, *e.g.*, [7] for an example), or saved as *multi-g2o*, for which a parser for C++ is provided. In addition to that, the datasets can be generated via specifications given in a `.json` file, which means that whole datasets can be shared and regenerated via just one single file.

D. Execution time

While the execution time may not be critical in most use-cases, some users of *kollagen* might – for practical reasons – want the generation to be both fast and scalable. One such case could be the need for massive data-generation in learning-based applications. Another might be to generate a large variation of datasets to get a statistically sound perception of performance or accuracy. For this reason we provide Figs. 2(a) to 2(c) which showcase the execution times when increasing the number of steps taken, the number of agents, and the size of the distance used in the intra- and inter-agent loop closure generation, respectively. In Fig. 2(c), we set $R_{\text{lc}} = R$ for both intra- and inter-agent loop closures and vary R. The tests were performed on a laptop with an Intel i7-1185G7 CPU @ 3.0 GHz and 32 GB RAM.

The data suggests that the execution time scales linearly both in terms of the number of steps and agents. However, the data indicates a worse than linear time increase when we increase R. While this might be expected due to the number of look-ups being proportional to R^2, we are pointing this out to users that want to work with high values of R, since they can expect increased times for generating datasets. For details on the exact parameters and procedures used for generating these figures, we refer the reader to the source file `timing.cpp` in the *kollagen* library.

In general, *kollagen* has been optimized for readability, maintainability, and ease of use rather than for speed. This means that there are several close-at-hand code optimizations that users with stricter speed requirements can perform. One such optimization would be to replace the $\mathcal{O}(\log N)$ `std::map` containers with $\mathcal{O}(1)$ `std::unordered_map` containers. However, since not all `std::map` containers in *kollagen* are holding types that are hashable by default (such as `std::pair`), such a change would require either implementing hashing functions for such types, or to include the Boost C++ Libraries (`https://www.boost.org/`) as a dependency and use the hashing functions they provide.

IV. Example Datasets

In this section, we use the data generator to provide two example datasets generated with *kollagen* that can be used by the community out-of-the-box to evaluate their pose graph optimization algorithms for collaborative SLAM. We first describe the datasets and then apply three state-of-the-art algorithms on these example datasets to showcase that our generated datasets can be processed by state-of-the-art algorithms.

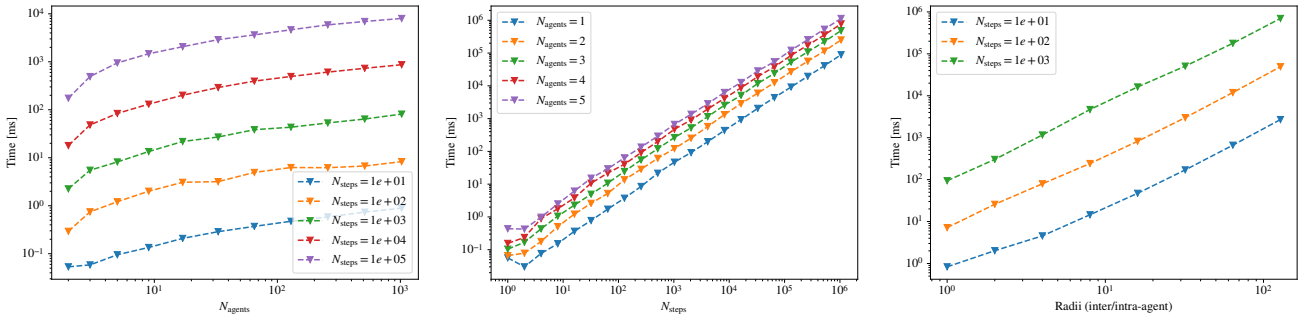
Let us now describe the two example datasets that are provided with this paper and are generated with *kollagen*. The first dataset, `Multi3500x8`, is inspired by `M3500` and consists of eight agents, which have 3500 nodes in their respective factor graphs. Since the agent in `M3500` is able to perform 180° turns, we also enable that option for this dataset (as described below Eq. (1)). In total, the dataset has 67645 constraints, where 39645 out of these constraints are loop closures. The ground truth trajectories of the eight agents are shown in Fig. 3(a). The parameter settings used to generate `Multi3500x8` can be found in `Multi3500x8.json`

The second dataset, `Multi10000x5`, is inspired by the `M10000` dataset and consists of five agents, which have 10000 nodes in their respective factor graphs. In this dataset, we disabled the 180° turns of the agents such that they can only turn left and right, or not keep the same direction. This dataset has a total of 76134 constraints out of which 26134 are loop closure constraints. The ground truth trajectories of the five agents are shown in Fig. 3(b). We foresee that the use of this dataset is to evaluate the performance of collaborative pose graph optimizers for large datasets, *e.g.*, one can analyze the speed of the pose graph optimizer. The parameter settings used to generate `Multi10000x5` can be found in `Multi10000x5.json`

Next, we will run state-of-the-art algorithms for pose graph optimization on our two example datasets. The state-of-the-art algorithms we run our example datasets on are SE-Sync [4], DPGO [7], and g^{2}o [14]. For SE-Sync we use the default parameters with chordal initialization and we let DPGO run for 50 iterations with default parameters. For g^{2}o we initialize with *spanning tree* and use the `gn_var_cholmod` solver with default parameters for 10 iterations. SE-Sync, DPGO, and g^{2}o expect a single `.g2o` file as an input. Therefore, we need to concatenate all agent graphs for both investigated datasets in one `.g2o` file, which is handled by *kollagen*. These `.g2o` files can then be directly processed in SE-Sync or g^{2}o. For DPGO, we use the accompanying `MultiRobotExample.cpp`.

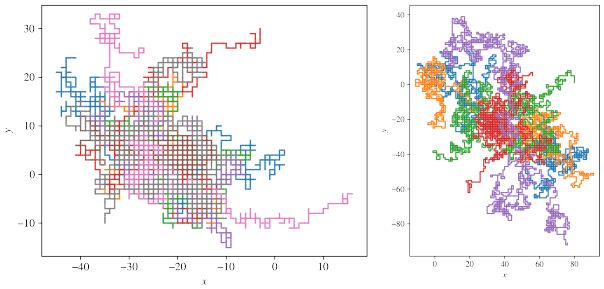
Since we give the user the option to save the `.g2o` files with the correct information matrices and incorrect diagonal matrices (see Section III-A.5), we use both correct and incorrect information matrices for the odometry, where the incorrect matrices are given by $I = \text{diag}(\sigma_{\text{pos}}^{-2}, \sigma_{\text{pos}}^{-2}, \sigma_{\text{ang}}^{-2})$. The loop closures have the correct information matrices though. We call the case with incorrect diagonal information matrices `Diagonal` and the case with the correct information matrices `Exact`. Since the objective functions of the optimizers typically depend on the information matrices, we cannot simply compare the optimal values of the objective values in the two considered cases. To obtain a fair comparison of the pose estimates, we will use the unaligned mean average pose error (APE) with respect to the translations, as determined by evo [19], between the ground truth and the pose estimates.

The results for `Multi3500x8` and `Multi10000x5` are shown in Table I and Table II, respectively. First, we note each of the pose graph optimizers has reduced the mean



(a) (b) (c)

Fig. 2: (a) Execution time VS the number of agents N_{agents}, for a selection of values of N_{steps}; (b) Execution time VS the number of steps N_{steps}, for a selected number of agents N_{agents}; (c) Execution time VS the size of the radii, for a selected number of steps N_{steps}.



(a) (b)

Fig. 3: Ground truth trajectories of (a) the eight agents in `Multi3500x8`; (b) the five agents in `Multi10000x5`.

TABLE I: Results for the mean APE on `Multi3500x8`

Case	g^{2}o	DPGO	SE-Sync	Odometry
`Diagonal`	0.385161	0.966504	0.733618	14.883697
`Exact`	0.390065	0.252501	0.212746	14.883697

TABLE II: Results for the mean APE on `Multi10000x5`

Case	g^{2}o	DPGO	SE-Sync	Odometry
`Diagonal`	1.019478	3.098239	0.791197	17.427713
`Exact`	1.426272	1.178893	0.446346	17.427713

APE significantly compared to the mean APE obtained from the odometry estimates. This shows that the state-of-the-art optimizers have sucessfully used our example datasets to generate a better estimate of the poses than the odometry. For SE-Sync and DPGO, we observe that if the diagonal information matrices are used for the odometry the mean APE increases compared to using the exact information matrix, which is an expected outcome. Furthermore, we see that SE-Sync outperforms DPGO, which is again expected since SE-Sync is a global solver and we have limited DPGO to 50 iterations. However, g^{2}o behaves in an interesting way in the diagonal case. For `Multi3500x8`, the obtained mean APE for g^{2}o is even smaller than the one obtained for SE-Sync and DPGO. While SE-Sync has returned the globally optimal pose estimates for its objective function, we would like to point out that g^{2}o solves a different objective value, which could be one explanation why the estimates of g^{2}o are closer to the ground truth than the estimates of SE-Sync. Furthermore, returning the globally optimal solution to an objective function does not imply that the obtained solution will be close to the ground truth, since the objective function is influenced by the noisy measurements, as, *e.g.*, pointed out in [20]. In the case with the true information matrices, SE-Sync performs better than g^{2}o, which can be explained with the fact that SE-Sync now has the correct information matrices in its objective function such that it will get closer to the ground truth.

V. Conclusions

In this paper, we presented *kollagen*, a collaborative SLAM pose graph generator. *kollagen* provides a simple and intuitive way to generate pose graphs in a collaborative SLAM setting, which has been lacking so far. Therefore, it provides the means to generate datasets for collaborative SLAM pose graph optimizers, which will enhance both the reproducibility of results as well as the comparison of different optimizers with high quality datasets.

The current version of *kollagen* produces a random walk on a planar grid for a user-specified amount of agents and also allows the user to specify the number of nodes for the agents' pose graphs. The odometry and loop closure measurements are influenced by normally distributed noise processes with user-defined covariances, while the probability of creating a loop closure can be adjusted by the user. Finally, we also showcased how state-of-the-art pose graph optimization algorithms can run on two provided datasets.

There are several possible extensions that could be made to *kollagen*. First, the current version produces only planar pose graphs, where the robot at each time step either moves straight ahead, left, right, or backwards. A simple extension for the three dimensional case is to let the robot also move up and down randomly. This would lead to a similar dataset as `cube` but with several agents. Second, the odometry and loop closure measurements generated are based on normally distributed noise. One extension could be to use a different noise distribution to have more realistic measurements, while another extension could be to modify the code to randomly introduce outliers.

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 Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, Dec 2016. [Online]. Available: https://doi.org/10.1109%2Ftro.2016.2624754

[2] P.-Y. Lajoie, B. Ramtoula, F. Wu, and G. Beltrame, "Towards collaborative simultaneous localization and mapping: a survey of the current research landscape," *Field Robotics*, vol. 2, no. 1, pp. 971–1000, Mar 2022. [Online]. Available: https://doi.org/10.55417%2Ffr.2022032

[3] 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 *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011. [Online]. Available: https://doi.org/10.1109%2Ficra.2011.5979641

[4] D. M. Rosen, L. Carlone, A. S. Bandeira, and J. J. Leonard, "SE-Sync: A certifiably correct algorithm for synchronization over the special euclidean group," *The International Journal of Robotics Research*, vol. 38, no. 2-3, pp. 95–125, Aug 2018. [Online]. Available: https://doi.org/10.1177%2F0278364918784361

[5] E. Olson, J. Leonard, and S. Teller, "Fast iterative alignment of pose graphs with poor initial estimates," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. [Online]. Available: https://doi.org/10.1109%2Frobot.2006.1642040

[6] L. Carlone, R. Tron, K. Daniilidis, and F. Dellaert, "Initialization techniques for 3d SLAM: A survey on rotation estimation and its use in pose graph optimization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2015. [Online]. Available: https://doi.org/10.1109%2Ficra.2015.7139836

[7] Y. Tian, K. Khosoussi, D. M. Rosen, and J. P. How, "Distributed certifiably correct pose-graph optimization," *IEEE Transactions on Robotics*, vol. 37, no. 6, pp. 2137–2156, Dec 2021. [Online]. Available: https://doi.org/10.1109%2Ftro.2021.3072346

[8] T. Fan and T. Murphey, "Majorization minimization methods for distributed pose graph optimization with convergence guarantees," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5058–5065.

[9] L. Carlone and A. Censi, "From angular manifolds to the integer lattice: Guaranteed orientation estimation with application to pose graph optimization," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 475–492, Apr 2014. [Online]. Available: https://doi.org/10.1109%2Ftro.2013.2291626

[10] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, May 2010. [Online]. Available: https://doi.org/10.1109%2Frobot.2010.5509407

[11] L. Carlone, R. Aragues, J. A. Castellanos, and B. Bona, "A fast and accurate approximation for planar pose graph optimization," *The International Journal of Robotics Research*, vol. 33, no. 7, pp. 965–987, May 2014. [Online]. Available: https://doi.org/10.1177%2F0278364914523689

[12] G. Grisetti, C. Stachniss, and W. Burgard, "Nonlinear constraint network optimization for efficient map learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 10, no. 3, pp. 428–439, Sep 2009. [Online]. Available: https://doi.org/10.1109%2Ftits.2009.2026444

[13] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, apr 2007. [Online]. Available: https://doi.org/10.1109%2Frobot.2007.363563

[14] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, May 2011. [Online]. Available: https://doi.org/10.1109%2Ficra.2011.5979949

[15] K. Y. Leung, Y. Halpern, T. D. Barfoot, and H. H. Liu, "The UTIAS multi-robot cooperative localization and mapping dataset," *The International Journal of Robotics Research*, vol. 30, no. 8, pp. 969–974, Mar 2011. [Online]. Available: https://doi.org/10.1177%2F0278364911398404

[16] R. Dubois, A. Eudes, and V. Fremont, "AirMuseum: a heterogeneous multi-robot dataset for stereo-visual and inertial simultaneous localization and mapping," in *2020 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*. IEEE, sep 2020. [Online]. Available: https://doi.org/10.1109%2Fmfi49285.2020.9235257

[17] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray, and P. H. S. Torr, "Collaborative large-scale dense 3d reconstruction with online inter-agent pose optimisation," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 11, pp. 2895–2905, nov 2018. [Online]. Available: https://doi.org/10.1109%2Ftvcg.2018.2868533

[18] F. Dellaert, R. Roberts, A. Cunningham, Varun Agrawal, C. Beall, Duy-Nguyen Ta, Lucacarlone, F. Jiang, Nikai, J. L. Blanco-Claraco, S. Williams, Ydjian, A. Melim, Zhaoyang Lv, J. Dong, J. Lambert, Krunal Chande, Akshay Krishnan, G. Chen, Balderdash-Devil, DiffDecisionTrees, Sungtae An, Mpaluri, Ellon Paiva Mendes, M. Bosse, A. Patel, Ayush Baid, P. Furgale, Matthewbroadwaynavenio, and Roderick-Koehle, "borglab/gtsam: 4.1.1," 2021. [Online]. Available: https://zenodo.org/record/5794542

[19] M. Grupp, "evo: Python package for the evaluation of odometry and slam." https://github.com/MichaelGrupp/evo, 2017.

[20] K. J. Doherty, D. M. Rosen, and J. J. Leonard, "Performance guarantees for spectral initialization in rotation averaging and pose-graph slam," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5608–5614.