Digital Analytics and Robotics for Sustainable Forestry

CL4-2021-DIGITAL-EMERGING-01
Grant agreement no: 101070405

DELIVERABLE 3.1

Document Defining Map Server Interfaces

Due date: month 6 (February 2023)
Deliverable type: R
Lead beneficiary: UOXF

Dissemination Level: PUBLIC

Main author: Nived Chebrolu, Maurice Fallon

1 Abstract

Overview This document describes a map representation format which will allow individual robots to create maps and write them to file — specifically the robots which will carry out mapping duties such as ANYmal, the Leica BLK2Fly, NTNU RMF drone and the handheld mapping systems from Leica and Oxford. We will prefer to use common/standard tools such as open source file formats and assume that data from robots is primarily created using Robot Operating System (ROS). It is intended that these maps (created online in the forest) will be later read back by the map fusion engine which University of Oxford is developing. The will also support the inclusion of GPS data and we will be able to merge maps between different robots and different mapping sessions. The map fusion engine will provide an abstraction between the robots and down stream processing and human interface (especially by foresters and decisions support systems). Significant thought has been given to defining a system which enforces little in the way of dependency between partners for their own robot-specific systems.

A key point is that this map representation will assume a deformable and flexible graph representation to ensure that intra-map consistency can be achieved and so that no permanent mapping errors are *burned in* which would be the case with a black-box commercial solution.

Relation to overall DigiForest project This mapping API is inteded to be used by all the partners developing robots (and handheld systems). It's output will feed to the human operators and foresters — including for growth analysis, mission planning and operations scheduling.

2 Overview of Pose Graph SLAM

As an initial pre-amble, we distinguish between *Pose Odometry* and *Simultaneous Localization and Mapping (SLAM)* in the following section.

2.0.1 Odometry

Odometry is an iterative algorithm which processes incoming sensor measurements (by default a multi-beam lidar and an IMU). The pose of the sensor/robot at time k is defined to be $T_k \in \mathrm{SE}(3)$. The odometry system computes an incremental relative odometry estimate between two consecutive timestamps, $\Delta T_{k-1:k}$. This estimate is the relative transfromation between the sensor pose estimate at the current pose and a previous iteration:

$$\Delta T_{k-1:k} = T_{k-1}^{-1} * T_k$$

We assume that this odometry is at lidar or camera frame rates — so close to 10Hz. It may perhaps be computed at IMU frame rates (e.g. 200Hz) or even be the output of a continuous time estimator. The odometry estimator concerns itself only with the sensor data around the robot at that particular point in time. When the robot moves away from that location, the old sensor measurement will have no influence on the algorithm. For lidar odometry, typically when the robot moved by about 50m, old sensor measurements will no longer have an influence on the robot's odometry.

This odometry estimate is assumed to be gravity aligned (such that the downwards direction is consistent) which is achieved by fusing the IMU's accelerometery into the motion estimate (known as lidar odometry and perhaps additionally lidar-inertial odometry). Typically we assume that we can estimate the robot's attitute accurately because the pitch and roll are observable using the accelerometery.

However, unavoidably the pose of the sensor relative to its starting point will become less and less certain over time. The estimator will suffer unavoidable drift in the X, Y and Z dimensions as well as Yaw (rotation in the horizontal plane). Ultimately, the odometry system is imprecise and our sensors are not perfect. An example rate of drift is 1m per 100m travelled - small but gradual.

2.0.2 Pose Odometry and Simultaneous Localization and Mapping (SLAM)

Alternatively SLAM, in more specifically *pose graph SLAM*, is a technique which attempts to correct for odometry drift. It does this by keeping track of the old data (in our case point clouds and robot poses) collected by the robot and builds up a running history of the path taken by the robot and its map. In pose graph SLAM we do this by keeping a sequence of *pose nodes* — robot poses which are placed along the path taken by the robot — perhaps 1m apart. This corresponds to the orange edges in Fig 1. Alternatively poses can be placed according to the degree of measurement overlap - with new nodes created more frequently in confined space and less frequently in open environments.

The system then attempts to carry out *place recognition* to uncover when the robot has returned to a location that it has previously visited and it uses that knowledge to correct the trajectory and the map.

When such a recognition occurs, we can establish where the robot is within the historical map. By precisely positioning the robot relative to a previously part of the map we will have achieved a *loop closure place registration*. However this will have

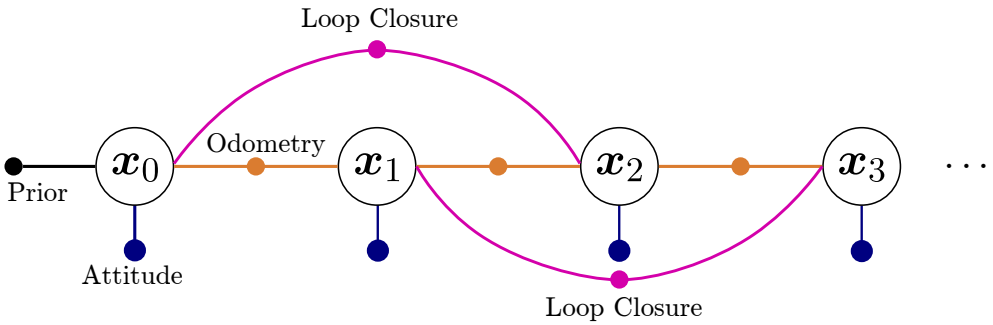

Figure 1: Pose-graph representation of the SLAM system. Each node represents the pose of the sensor whereas the edges represent the constraints coming from multiple sources.

indentified the unavoidable error which has occured in the trajectory — which is due to the drift in the odometry system — as described above.

What remains is to consider the entire trajectory and the map and to determine how to best distribute the error along the entire robot path so as to maintain *local consistency*. Effectively the error is spread along the map.

We do this using *pose graph optimization (PGO)*. Having built up the path the robot took and the constraints due to *place registration*, we can create a more complex version of the pose graph. Shown in magenta in Fig 1 are loop closures. Note that they represent specifically non-consecutive constraints.

This constraint set can be relaxed using least squares optimization to distribute the error along the path — by moving the poses gently so as to minimize the squared error on the constraints. The map is in turn corrected by moving the map data according to the poses. (In all cases the map is simply the result which occurs by deforming the pose graph — it is entirely a function of the location of the poses. This is important as it avoids dealing with the raw underlying map data everytime a new loop closure happens.)

To solve this optimization problem we (Oxford) use an open source library called GTSAM (Georgia Tech Smoothing and Mapping) [5] but this approach is not proscriptive and theis map representation can be solved by other optimization algorithms such as g2o [10] and CERES [1].

Note that two other constraints are shown in the figure, a Prior factor and an Attitude factor. See below for more details. Lastly, we have not gone into any technical detail about the optimisation such as claims of optimality, computational efficiency or robustness here by intention. See [5] for further technical data.

3 File Format for Pose Graph SLAM Maps

As described above we can solve (and update) large pose graph problems of multiple kilometres travel in real-time. Point cloud maps made from such mapping sessions could be in the order of gigabytes of point clouds data. However the challenge comes in storing this map, reusing the map and combining it with maps from other runs. In the following we describe an file input/output format to do this.

Currently we support writing 3 datatypes to file, which do we using a variant of the 'g2o format', these can be used to write a SLAM problem to file and to recreate it afterwards in computer memory - as a GTSAM/iSAM pose graph optimisation

problem.

The *g2o*[1] file format is widely used in the SLAM community (but it's not really a 'standard' format).

The datatypes described below can be easily parsed in C++ code using Eigen datatypes. In the following, where possible, we prefer to use the approach of Furgale [8] when representating transformation conventions and what is used commonly in ROS.

Note that in all experiments we will also record the raw sensor signals from the platforms. Typically this will be using ROS but where our collaborators prefer we have considered folder structures and file formats to store the raw data as png files, CSV imu streams and individual lidar point cloud files. For this approach we will likely follow or extend the well known EuROC dataset format [4].

3.1 Odometry Edges

The primary constraints of the poses graph are odometry and loop closure edges which are relative contraints between two individual poses. In the g2o text format they can be written as a string in the following way:

```
EDGE_SE3:QUAT tail_id head_id [3 value pos] [4 value quat] [21 value info matrix]
```

The position and quaternion elements together represent an estimated transform in 3D (SE(3)). Specifically this is the *relative transformation from the tail to the head node, as represented in the coordinate frame of the tail node*. The tail and head node correspond to the pose of the robot/sensor corresponding to the respective ids.

The last block of terms is the information matrix — giving the uncertainty about this relative constraint. It is a 21 element vector which corresponds to the upper triangular elements of a 6x6 matrix (with the matrix being symmetric about the diagonal).

Typically the information matrix terms are zero for off diagonal terms and the diagonal terms have values in the range of 100-1000. An important point is that the terms are ordered with three translation elements first and three rotation elements after that. This is the g2o variable ordering convention. Note that in GTSAM these terms need to be reordered when parsing as it uses a rotation-then-translation ordering.

For odometry edges, an edge corresponds to the relative odometry from one node to the next one consecutively — typically triggered about 1m apart. The head_id is precisely an integer one greater than the tail_id. If there are N nodes in the graph, we will have $N-1$ odometry edges. Again these are the orange edges in Figure 1.

For loop closure edges, an edge corresponds to the relative transformation between two non-consecutive poses. These are identified using a loop closure detector (as described above and detail below). Thus the head_id and tail_id will **not** be consecutive. There will typically be many fewer loop closure nodes than odometry ones. These are coloured magenta in the figure.

The edges are now explained. But by themselves this set of edges does not define the placement of the pose graph — the origin of the graph could be anywhere and we could translate the entire graph to be at any arbitrary location.

3.2 Pose Priors

To fix the graph in a specific location in the world coordinate frame — such as being aligned to building floor plan or in the GNSS/GPS frame we need to fix at least one

[1] g2o is an alternative SLAM backend which popularised this file format.

of the nodes. We can do that by defining a *pose prior* for the first nodes/vertex. The following is the format we use — which adds a unique timestamp to each node:

```
VERTEX_SE3_SE3:QUAT_TIME id [3 value pose] [4 value quat] [sec & nsec timestamp]
```

This corresponds to a special transformation: the estimated transformation between this vertex and the origin of the world frame. Without GPS, we typically place the first pose of the pose graph into the world using this vertex constraint.

Typically when running SLAM the first pose prior is placed at the origin (X,Y,Z elements) with the orientation defined by the IMU's attitude (as described above).

Side note: When writing to disk the SLAM problem of a particular mapping session, we also save the estimated pose of each pose node — as estimated by the SLAM problem. This can be useful because (1) its the best estimate from GTSAM up to that point and (2) its our best estimate of the attitude (pitch and roll) of each pose — regardless of the position and yaw of the graph.

3.3 Gravity Alignment

To *establish* gravity alignment of the SLAM map, we use the orientation element of the first node/vertex i.e. so that the z-axis is upwards. To complete the SLAM problem we actually need only a single VERTEX constraint — we dont need to use all of the attitude estimates mentioned above.

However, if we use only the initial attitude in our SLAM problme we might suffer from a loss of gravity alignment over time e.g. due to inaccurate loop-closures — despite the initial alignment.

So, to *maintain* gravity alignment we go a step further. We store the orientation of each robot pose - as computed by the odometry system (and stored in the VERTEX element). To each node we can add an additional attitude factor which ensures that the pitch and roll — as estimated in the original odometry system is used to constrain the attitute in the pose graph problem.

Note that an alternative approach would be to retain the biases of the IMU so as to be able to refine the attitude vector. We have not done this.

What is described above represents a single SLAM graph — for a single robot moving in an environment once. Reasonably a single session could cover about 1000m traveled (or roughly 30 mins of a person walking) with poses spaced 1 m apart. The pose graph optimizer (iSAM2 from GTSAM) can run in real-time with this number of nodes — triggered about once per second. Again see [5] for more details on the algorithms underlying iSAM and GTSAM.

3.4 Platform ID

To support tuning specific to individual platforms and to better link different missions to a specific robot we also add a platform specific ID. This is a unique integer specific to each robot, for example.

```
PLATFORM_ID 0
```

4 Multi-Session Pose Graph Optimization

To scale this approach to cover multiple hectares, multiple robots and/or multiple mapping sessions we need to merge mapping sessions one-by-one. A simple approach would be to just merge the two point clouds from a pair of mapping sessions together

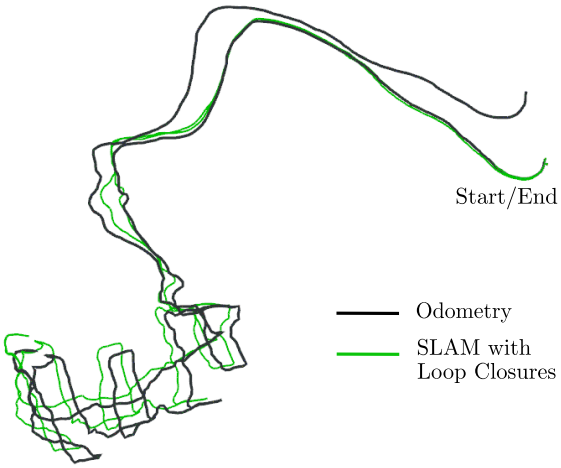

Figure 2: Odometry trajectory compared to a SLAM trajectory for a 1.3km session. The odometry contins 25m of total drift — largely as a result of minor orientation error. This error has been distributed in the SLAM solution such that the map is still locally consistent.

into a single cloud using ICP registration.

However the devil is in the detail: ultimately the SLAM maps and trajectories will have local residual error within them - so there will be duplicated trees that do not line up and are inconsistent. Afterall, the SLAM algorithms job is to retain local consistency by spreading error around the graph. The error still exists and for large maps (1000m sized), there is going to be several metres of residual error within the poses which form the SLAM point cloud map.

This issue is illustrated in Fig 2. The odometry trajectory is over 1km and 25m of drift has built up (about 2%). Yet this would have resulted in a smeared and incorrect map. The SLAM system recognises the drift, distributes it across the graph so that the local transforms are slightly different. However, still remaining drift exists and needs to be accounted for — in particular some dimensions (the off horizontal dimensions) are inherently poorly constrained.

We account for this issue by never freezing the layout of the trajectory and the map — until it is fully complete, well constrained and georeferenced. Instead, we keep the constraints from each session and merge them into a single multi-session map which optimises all the constraints.

This concept is illustrated in Fig. 3. You can see two pose graphs shown in the figure with *inter-session loop closures* linking between them. We do this by loading the first pose graph (which we call the *primary pose graph*) and then for each subsequent pose graph (a *secondary pose graph*) we read the nodes and their corresponding point cloud scans and try to find pair wise loop closures/registrations. When pairwise registrations are found, inter-session loop closures are added.

As a sidenote, when we add new sessions to the SLAM instance, we have to ensure each node ID is actually unique. We do this by offsetting each session. Another issue is having an initial guess for each new node that is inserting into the factor graph (which GTSAM requires), to achieve that we take the original SLAM solution for the secondary pose graph and transform the graph into alignment with the primary graph using the first inter-session loop closure.

The key challenge was glossed over here - how to identify inter-session loop closures — i.e. how to do place recognition between the different mapping sessions. This is a challenging research problem.

4.1 Place Recognition in Forests

Place Recognition is an open research problem in robotics and autonomous vehicles which isn't specific to forest environments — with many papers published on this topic. In the following we will merely present a short overview of a technique we are currently using (in Oxford). The method we are using is called ScanContext and was proposed by Giseop Kim and Ayoung Kim [9].

ScanContext creates a single global descriptor to represent a typical lidar scan. The descriptor creates a type of polar grid around the robot where the range measurements and their heights are used to fill the polar rings. This is shown in Fig. 4. The descriptor is a matrix of about 60 x 10 (angle x height) and is built for each scan in the pose graph. As the SLAM session continues the algorithm builds a database of descriptors and uses a nearest neighbour matching back-end to quickly find a list of possible matches (with a confidence score). Importantly, ScanContext is also CPU-based (i.e. no deep learning, no GPU) which makes it tremendeously easy to use on any robot or device.

When a proposal match with high confidence is found we form a pair with the

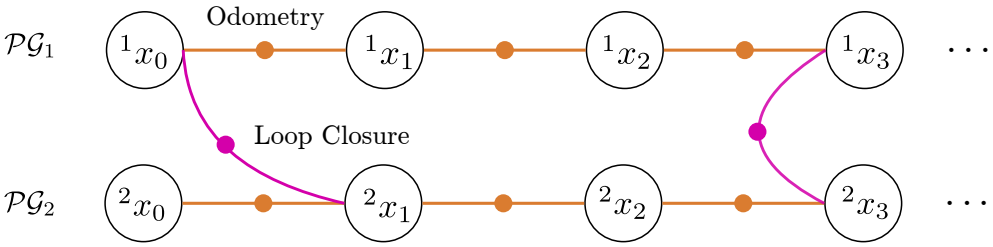

Figure 3: Multi-Session Pose Graph.

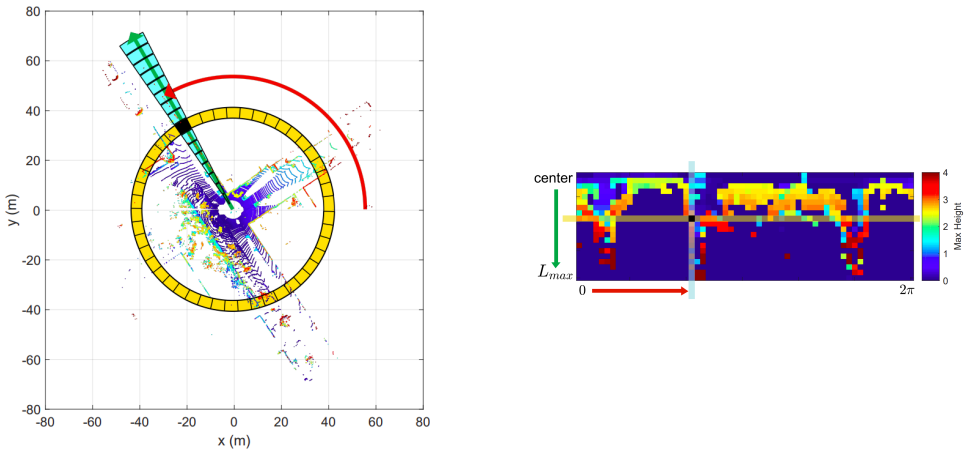

Figure 4: Scan Context Descriptor Summary. Left shows a top down view with the points in the yellow ring used to determine the maximum height for those cells which then transfers to the descriptor seen on the right hand side. Image taken from [9].

test/current scan and the proposal scan the proposal match. We then use this pair of scans (and an estimate of the relative rotation between them) to attempt ICP registration and to determine a loop closure transformation — this is without any initial relative offset guess. Computation time (finding the descriptor and testing it against the existing database) is small — less than 100 milliseconds.

4.1.1 Performance of ScanContext

We have found ScanContext to be able to do reliable place recognition in certain outdoor locations, despite its simplicity. Where the environment contains structure (e.g. buildings or individual free standing trees) and is not dominated by rough bushes and wild growth it has good precision and recall. It would seem that a scan taken in a well-managed forest can produce a relatively unique and distinct descriptor.

However performance is much worse in poorly maintained and unmanaged young forest growth (e.g. forests where all the oldest trees are merely a few centimetres in diameter. However, in such an environment, better pre-processing of the scan clouds could help improve performance as would the application of existing research on learning methods for lidar place recognition such as the following works from consortium partners [14, 6, 7].

Similar principles for detecting a re-visited are also applied to visual place recognition. In built environment and autonomous vehicle settings, the performance of visual place recognition is well understood; however there are a multitude of challenges in the natural environments. We anticipate data collections being months and possibly years apart such so that visually recognising a location will be challenging. Example works such as [13] and [11] either use lighting invariant features or learn databases of different 'visual experiences' to capture different lighting conditions. Visual recognition in these enviroments as relatively more challenging than lidar and is not the focus of our initial efforts.

Returning to lidar place recognition, we typically find that to achieve a good loop proposal, the scanner needs to have passed very close to a suitable loop closure pose — e.g. within 2m. this is good in situations where the person or robot is likely to have done this anyway e.g. when there is only a narrow path or corridor open for the robot to follow. However if the operator doesnt do this, e.g. walking 3m apart along a roadway or crossing perpendiclarly over a previously walked path, no loop closures can be found. This is a problem in a forest where directly finding an overlapping path isnt always easy. Humans and robots need to plan their routes with this in mind.

Improving on this issue would be possible by improving the method's view point invariance (e.g. ScanContext++). We also envisage developing filtering techniques which allow us to produce descriptors which are more meaningful i.e. by deleting loose branches and leaves and retaining only points with stable local normals (typically the trunks).

Other more complex research includes extracting individual trees specifically, attaching a descriptor to them and finding explicit tree-to-tree pairwise matches as attemped in other works. This is an open research problem in this environment.

4.1.2 Fusing Multi-Session Maps with ScanContext

Using this approach we can find inter-session loop-closures so as to align two or more map sessions into a single super map — as described above. We iterate through the pose nodes and lidar scans and find pairwise matches spanning the missions. We typically tune the ScanContext/Registration pipeline to be conservative — to only

accept matches we are quite confident in. More work could be done to reduce this tuning. Also we could like to add local clique checking (e.g. checking that sets of loop closures are consistent before accepting them entirely).

Finally, our approach then carries out a wave of brute force search to find pairwise matches using a geometric prior. This is important to ensure that any local drift is corrected for — and that the map is well constrained and consistent.

Further detail can be found here [12].

4.2 Incorporating GPS/GNSS into the SLAM graph

The map created using the SLAM system (Sec. 2.0.2) is initialized in an arbitrary (gravity-aligned) local coordinate frame. Typically, the starting pose of the sensor at

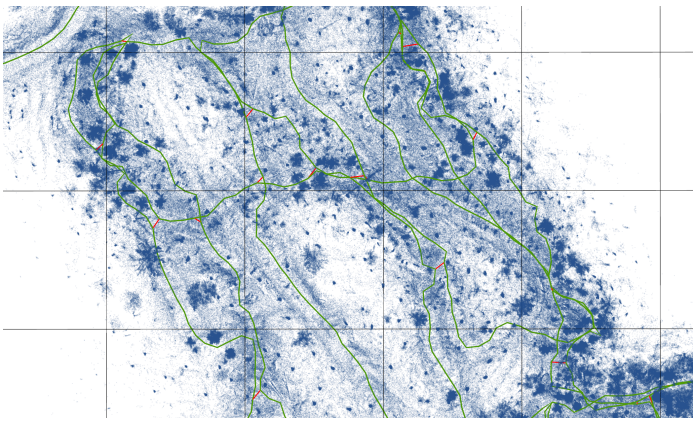

Figure 5: A top-down details view showing how loop closure edges constrain the SLAM graph and ensure map crispness. Red lines correspond to loop closures.

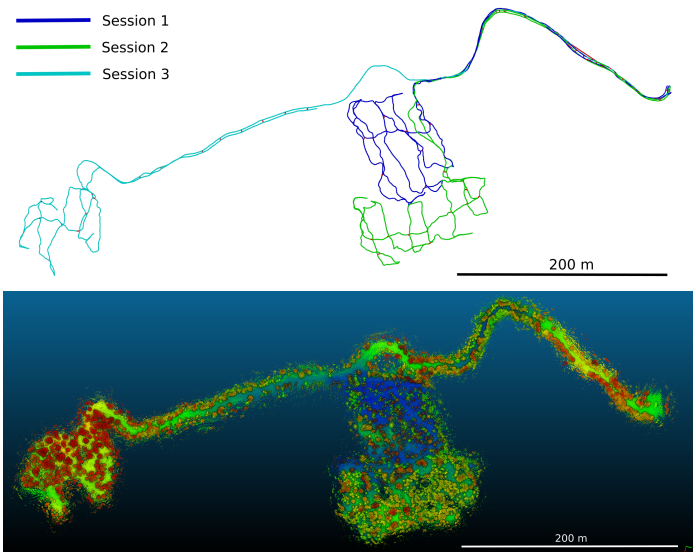

Figure 6: A 3 session SLAM map where the red edges (zoom in to see them!) indicate the inter-session loop closures.

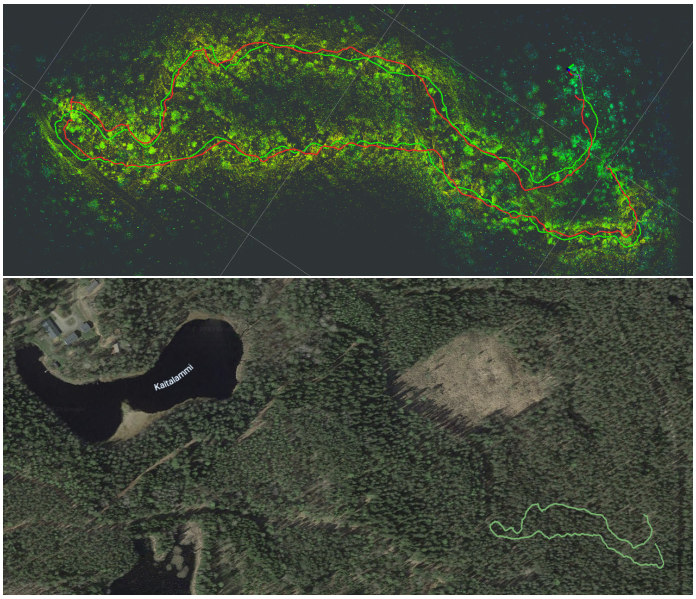

Figure 7: Trajectory from a handheld mapping mission at Evo, Finland (upper image). The trajectory in red is the GNSS sensor measurements transformed into the local mapping frame. It has been aligned with the SLAM trajectory (green). The squares are 100m. This then allows the SLAM trajectory to be represented in a GNSS frame and overlaid on commercial satellite imagery (lower image, Google Earth).

the beginning of the survey mission is chosen as the origin of the map. However, it is often necessary and useful to register these maps into a global GNSS coordinate system. By registering the maps in GNSS coordinates, we are able to bring maps created by different robots over multiple sessions into an common/unique global coordinate frame. This means that multiple survey missions conducted in the same areas of the forest can be queried by using GNSS coordinates and compared over time for monitoring as well as detecting changes in a given region. Geo-referencing the SLAM maps also allows us to visualize the results by overlaying them on satellite imagery. An example of Geo-referenced SLAM trajectory is illustrated in Fig. 7.

To incorporate GPS/GNSS measurements into our pose graph, we first need to compute a rigid transformation between the local SLAM coordinate frame and the global GPS/GNSS coordinate frame. The GPS/GNSS coordinate frame uses an ellipsoid model for earth as described in WGS-84 standard. However, our pose-graph optimization procedure assumes the positions to be specified in a Cartesian coordinate system. Therefore, we convert the GNSS measurements, i.e. `[Latitude, Longitude, Altitude]` into a ENU (East-North-Up) Cartesian coordinates that is computed as a linearization around a reference GNSS coordinate.

After converting the GPS/GNSS measurements to Cartesian coordinates, we estimate the local-to-global rigid transformation by registering a portion of the trajectory in the two coordinate frames. We use pose estimates of k nodes ($k = 10$) in the graph and the corresponding GNSS measurements (in ENU frame) to compute a rigid transformation using the SVD based alignment algorithm described in [15]. Note that as our pose-graph nodes are gravity aligned, we constrain this transformation to 4 degrees of freedom only, consisting of a 3D translation vector and an yaw angle. After computing the transformation, we add it as a single prior factor/constraint to the pose

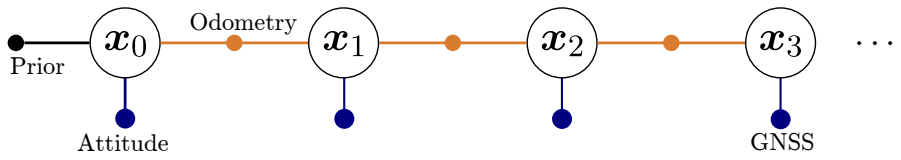

Figure 8: Pose-graph representation of the SLAM system with GNSS prior factor. A global transformation between the trajectory in the local coordinate frame and the global GNSS (ENU) coordinate frame is estimated. This transformation is added as a factor/constraint to the pose graph.

graph which effectively transforms the poses in the graph to the GNSS coordinate frame. A diagram illustrating the integration of GNSS prior factor is shown in Fig. 8).

The format of the g2o file entries for this process are listed below. The first line is latitude, longitude, altitude for a local origin. `GNSS_LLA_TO_MAP` corresponds to an alignment between that origin and the SLAM map frame (with only x, y and yaw populated).

```
GNSS_LLA_REF 61.1958 25.1407 163.423
GNSS_LLA_TO_MAP 0.853813 0.341663 0 0 0 0.950391 -0.311057
```

As future work, we aim to integrate multiple GPS/GNSS factors to the pose graph over time. This would allow us to integrate GNSS measurements continuously based on the uncertainty of the measurements which typically depend on the number of visible satellites. Further, adding GPS/GNSS measurements across the trajectory would also act as additional loop closures in the pose-graph optimization. Initial work in this direction has recently published in this work from TUM partners [3] (with a visual SLAM focus) and the upcoming work from Oxford [2].

One particular challenge is that the quality of GNSS measurement varies significantly as a result of foliage cover and even the height of the sensor from the ground. In Fig. 9 we show this behaviour — a Ublox GNSS sensor's height estimate moved in a manner highly uncorrelated with the SLAM height estimate. This image is taken from an experiment with an Oxford Frontier attached to a backpack. At 450seconds the backpack was taken off and put on the ground for 150 seconds — where the GNSS height estimates jumps by many metres.

4.3 Basic Interface for Visualizing Pose-Graph based SLAM map

The g2o file format (see Sec. 3) can be used to store the pose graph offline. We read the pose for each node in the graph from the g2o file and load the corresponding data (eg. pointclouds) using the timestamp. We then transform the data stored in sensor-centric frame to the 'map/world' frame according to the pose estimate in the graph. This means that if some new loop closures are found in a post-processing step or two pose-graph missions are merged offline, new maps can be created simply by using the optimized poses allowing for flexibility in offline usage.

Currently, we assume the data to be time-synchronized to the pose graph nodes to make the association. This, however, may not be convenient/possible in a setup with multiple sensors operating at different frequencies. In such cases, the poses from

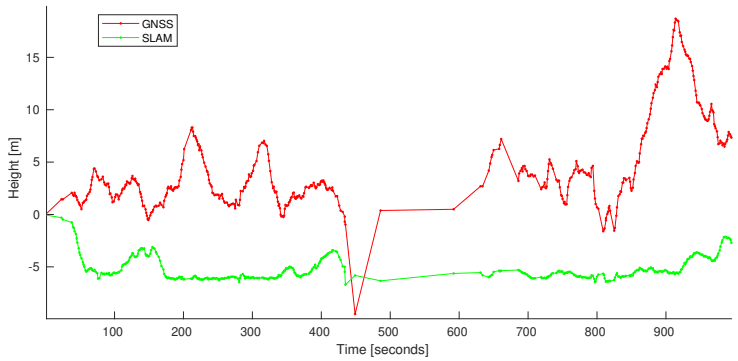

Figure 9: Comparison between the height/altitude of the SLAM trajectory and the GNSS trajectory for the Evo, Finland forest run (each offset to start at zero height). This shows that the height of the GNSS trajectory within the forest is erratic and difficult to use.

graph need to be interpolated to the timestamps of the corresponding sensors.

A sample program reading a g2o file can be downloaded from this location

```
https://github.com/ori-drs/multi_session_slam_viewer
```

It run using the command:

```
$ rosrun multi_session_slam_viewer single_pose_graph_publisher
  g2o_path:= path_to_g2o_file
```

Rviz is used for visualizing the data published by the node. An illustrative example from a SLAM session is shown in Fig. 10.

Presently, the sample program is implemented for visualizing raw pointcloud data. We plan to support visualization of tree semantics such as trunks, branches, leaves etc. We aim extend the functionality to visualize other sensor data such as images, GNSS measurements (for overlaying SLAM results on satellite images/maps). The final goal of the module is also to support visualization of data from multiple sessions in a common view.

4.4 Terrestrial and Above-Canopy Map Merging

Above-canopy mapping (e.g. from with Silvere's aerial drone) typically has access to GNSS and will not suffer from the problems described here and will typically achieve sub metre accuracy when fusing IMU and GNSS sensing.

5 List of Constraints Supported

Odometry or Loop closure edges:

```
EDGE_SE3:QUAT tail_id head_id [3 value pos] [4 value quat] [21 value info matrix]
```

Vertex/node including time. Initial node used for initial orientation. Subsequent nodes constrain attitude:

```
VERTEX_SE3_SE3:QUAT_TIME id [3 value pose] [4 value quat] [sec & nsec timestamp]
```

GNSS origin and a single world-to-map frame alignment:

```
GNSS_LLA_REF 61.1958 25.1407 163.423
GNSS_LLA_TO_MAP 0.853813 0.341663 0 0 0 0.950391 -0.311057
```

Platform ID:

```
PLATFORM_ID 0
```

6 Summary

In conclusion, we have presented an overview of the 3D mapping and localisation approach developed by Oxford as part of the Digiforest project. An open data format has been developed which allows a SLAM problem to be created by the other partners robots and shared between other users. We also discussed how maps can be used across mapping sessions and (in future) robots. GNSS fusion was discussed and an initial integration presented.

Future work will focus on introducing this representation to other robots of the project — specifically the ANYmal quadruped and the aerial platforms of NTNU, Leica and TUM.

References

[1] Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. *Ceres Solver*. Version 2.1. Mar. 2022. URL: https://github.com/ceres-solver/ceres-solver.

[2] Jonas Beuchert, Marco Camurri, and Maurice Fallon. "Factor Graph Fusion of Raw GNSS Sensing with IMU and Lidar for Precise Robot Localization without a Base Station". In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2022.

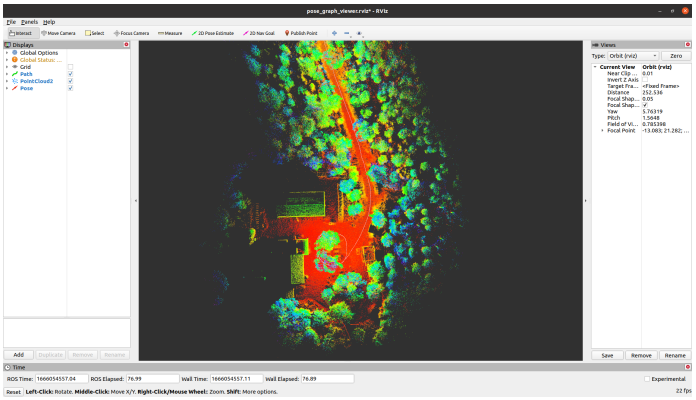

Figure 10: Rviz visualization of pose-graph data being read from a g2o file. The white line shows the trajectory of the sensor. The pointclouds are projected to the map frame according to the corresponding pose save in the g2o file.

[3] S Boche et al. “Visual-Inertial SLAM with Tightly-Coupled Dropout-Tolerant GPS Fusion”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2022, pp. 7020–7027.

[4] Michael Burri et al. “The EuRoC micro aerial vehicle datasets”. In: *The International Journal of Robotics Research* (2016). DOI: 10.1177/0278364915620033.

[5] Frank Dellaert, Michael Kaess, et al. “Factor graphs for robot perception”. In: *Foundations and Trends in Robotics* 6.1-2 (2017), pp. 1–139.

[6] Renaud Dubé et al. “SegMap: 3D Segment Mapping using Data-Driven Descriptors”. In: *Robotics: Science and Systems (RSS)*. 2018.

[7] Renaud Dubé et al. “SegMatch: Segment based place recognition in 3D point clouds”. In: *International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 5266–5272.

[8] P. Furgale. *Representing Robot Pose: The good, the bad, and the ugly*. 2014. URL: http://paulfurgale.info/news/2014/6/9/representing-robot-pose-the-good-the-bad-and-the-ugly.

[9] Giseop Kim and Ayoung Kim. “Scan Context: Egocentric Spatial Descriptor for Place Recognition Within 3D Point Cloud Map”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 4802–4809. DOI: 10.1109/IROS.2018.8593953.

[10] Rainer Kuemmerle et al. “g2o: A General Framework for Graph Optimization”. In: *2011 IEEE International Conference on Robotics and Automation (ICRA)*. 2011.

[11] Michael Paton et al. “Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016.

[12] Alexander Proudman, Milad Ramezani, and Maurice Fallon. “Online Estimation of Diameter at Breast Height (DBH) of Forest Trees Using a Handheld LiDAR”. In: *European Conference on Mobile Robotics (ECMR)*. Bonn, Germany (Virtual), 2021.

[13] Li Sun et al. “Robust and Long-term Monocular Teach and Repeat Navigation using a Single-experience Map”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021.

[14] Georgi Tinchev, Adrian Penate-Sanchez, and Maurice Fallon. “Learning to See the Wood for the Trees: Deep Laser Localization in Urban and Natural Environments on a CPU”. In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1327–1334.

[15] Zichao Zhang and Davide Scaramuzza. “A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018.