A FAST AND FLEXIBLE METHOD FOR META-MAP BUILDING FOR ICP BASED SLAM

A. Kurian[a], K. W. Morin [a,*]

[a] Leica Geosystems Ltd., 245 Aero Way NE, Calgary, Alberta, Canada, T2E 6K2 - (ajeesh.kurian, kristian.morin)@nwgeo.com

Commission III, WG III/2

KEY WORDS: LiDAR, 3D point cloud, ICP, SLAM, Mobile Mapping

ABSTRACT:

Recent developments in LiDAR sensors make mobile mapping fast and cost effective. These sensors generate a large amount of data which in turn improves the coverage and details of the map. Due to the limited range of the sensor, one has to collect a series of scans to build the entire map of the environment. If we have good GNSS coverage, building a map is a well addressed problem. But in an indoor environment, we have limited GNSS reception and an inertial solution, if available, can quickly diverge. In such situations, simultaneous localization and mapping (SLAM) is used to generate a navigation solution and map concurrently. SLAM using point clouds possesses a number of computational challenges even with modern hardware due to the shear amount of data. In this paper, we propose two strategies for minimizing the cost of computation and storage when a 3D point cloud is used for navigation and real-time map building. We have used the 3D point cloud generated by Leica Geosystems's Pegasus Backpack which is equipped with Velodyne VLP-16 LiDARs scanners. To improve the speed of the conventional iterative closest point (ICP) algorithm, we propose a point cloud sub-sampling strategy which does not throw away any key features and yet significantly reduces the number of points that needs to be processed and stored. In order to speed up the correspondence finding step, a dual kd-tree and circular buffer architecture is proposed. We have shown that the proposed method can run in real time and has excellent navigation accuracy characteristics.

1. INTRODUCTION

The mobile mapping community has benefited from recent developments in rotating multi-beam LiDAR sensors which can give fairly accurate range measurements of the sensing environment in real-time and with 360^o field of view (Muhammad and Lacroix, 2010) (Puente et al., 2013). With the limited range of the sensors and the presence of obstacles in the environment, the user needs to move in order to capture the full 3D model. To build the full map from these individual scans, we need precise position (location and attitude) with which one can transform and stack up individual scans. With a good GNSS-INS solution available, this is a trivial problem. In GNSS denied environments, however, we need to seek other ways to generate navigation solutions. Simultaneous localization and mapping (SLAM) (Durrant-Whyte and Bailey, 2006)(Bailey and Durrant-Whyte, 2006) using cameras (Strasdat et al., 2010) or LiDAR (Zhang and Singh, 2014) is an enabling technology to curb the error growth of IMU based solutions in GNSS denied situations.

Iterative closest point (ICP) algorithm is used to estimate the relative orientation and translation between two arbitrary point clouds (Besl and McKay, 1992). There are many variants of ICP: point to point and point to plane are the two most widely used methods (Segal et al., 2009). In point to point ICP we compute the optimum translation and rotation that minimize the distance between corresponding points from two scans; while in the point to plane method we minimize the distance between two planes. Irrespective of the ICP flavour used, one can continuously integrate the navigation solution by estimating relative position of the current scan at time t_n with the previous scan at t_{n-1}; or with a meta-map created from all the previous scans (i.e., using scans from t_0 to t_{n-1} by appropriately stacking each scan). The meta-map based solution is shown to have superior performance although it requires higher storage and computation (Nüchter et al., 2007).

The complexity of the meta-map based algorithm is twofold. As we keep a larger number of points in the memory, the storage footprint linearly increases with the number of frames used to form the meta-map. At the same time, computational complexity for finding correspondence increases quadratically with the number of points. A kd-tree (Moore, 1991) forms the heart of ICP correspondence algorithms and it is the most computationally demanding step (Greenspan and Yurick, 2003). The cost of computation depends on the amount of data stored in the kd-tree and the number of queries performed to find the correspondence. In this paper, we propose two simple but effective strategies to reduce the burden from using a kd-tree. Our primary objective is to have speed of operation so that SLAM can be performed in real-time without performance degradation. The second objective is to have the flexibility of keeping only a relevant portion of the previous map in the volatile memory. The rest can be loaded/unloaded from/to hard disk when the user needs it.

Our first strategy is to sub-sample the 3D point cloud without loosing any key information. In point to plane ICP, our focus is to capture all the planes available so that the full 6DoF can be estimated. We use the sudden changes in the range measurements obtained by individual lasers of the LiDAR, which is an indicator of laser scanning a new surface, to achieve this task. In addition, since the point cloud obtained from scanning LiDAR is not uniform, we performed a non-uniform sampling based on the depth measurements from individual surfaces. The set of points thus obtained are used for building the 3D meta-map. Our second strategy was to reduce the number of points used in the kd-tree by implementing a ring buffer. In that way we only kept a part of the map in the volatile memory at any given point of time. To further increase the speed, we developed a two layer approach for building the meta-map. In this strategy, two maps are kept in the memory which are maintained at a different rate and queried differently. The first map is built and queried in every frame. However, this map contains only a few previous frames. The second map, which contains 10 times more points than the first one is created at a lower frequency (such as 1 in 50 frames). If we

*Corresponding author

have a sufficient number of points from the first map using a radius search, the second map is not queried at all. In this way we can increase the speed so as to perform real time operations.

Leica Geosystems has recently introduced the Pegasus Backpack system, which makes 3D map building fast and effortless[1]. We use the LiDAR point cloud generated by the Pegasus backpack to assess the performance of our algorithms. The backpack contains Velodyne VLP-16 LiDAR scanners, multiple high resolution cameras and a NovAtel SPAN GNSS-INS navigation system. We continuously integrate the SPAN data with navigation information from the SLAM module to obtain an optimal estimate of the backpack position irrespective of GNSS signal availability. The total processing time taken as well as the relative accuracy are used to assess the performance of our algorithm. From the results one can see that even at a high data rate, the proposed solution has excellent navigation accuracy and real time performance characteristics.

The paper is organized as follows. In Section 2, we give a quick introduction to SLAM algorithm and details of the proposed map building strategies. We evaluate the performance of the new methods in terms of processing speed accuracy in section 3. Finally in Section 4, we provide some concluding remarks.

2. PROBLEM FORMULATION

In point to plane ICP we want to minimize

$$\mathbf{T}_{opt} = \arg\min_{\mathbf{T}} \sum_{i=1}^{N} \left((\mathbf{T}\mathbf{p}_i - \mathbf{q}_i) . \mathbf{n}_i \right), \quad (1)$$

where the point $\mathbf{p}_i = \{p_x, p_y, p_z, 1\}_i$ on the body frame is transformed using $\mathbf{T}$ such that it will minimize the error between the plane defined by the point $\mathbf{q}_i = \{q_x, q_y, q_z, 1\}_i$ and surface normal $\mathbf{n}_i$. $\mathbf{T}$ is defined as

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \quad (2)$$

where, $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation. If we know $\mathbf{n}_i$ and $\mathbf{q}_i$ in advance, the cost of computation depends only on the number of points used to solve the minimization of the objective function. However, in practice, we have to estimate $\mathbf{n}_i$ for each point $\mathbf{p}_i$ which are sampled from the same surface at different times and sensor positions. A kd-tree or octree can be used to find the corresponding points of current scans in the meta-map.

There are different costs involved in building, maintaining and querying a kd-tree. For example, the cost associated with building a kd-tree is $O(n \log(n))$ where n is the number of points in the data set. However, in the worst case scenario this can reach up to $O(n \log^2(n))$ (Wald and Havran, 2006). Similarly for search we have an average cost of $O(\log(n))$ with a worst case cost of $O(n)$. Thus, the kd-tree becomes the most computationally expensive module in a SLAM pipeline. Although one could build and incrementally update the kd-tree from frame to frame, due to the sensor noise it is advantageous to preprocess the points in the meta-map to improve the quality of the solution. Thus, we end up in building the map every frame. Keeping this cost in mind, we aim to minimize the number of points that are used to build the kd-tree (in other words, the meta-map). We propose two approaches for this solution.

[1]Pegasus Website http://leica-geosystems.com

2.1 Point Cloud Sub-Sampling

Figure 1 shows the change of depth observed by an individual laser in one single scan of 360 degree rotation. Due to the sensor noise, we have applied a low pass filter. As discussed in (Grant et al., 2013) one can use the first, second and third derivatives of the measurements to detect the locations where the laser beams switch surfaces. Since we are getting 10 frames per second, we do not need to keep all the points from these individual surfaces. Also, as the density of the points close to the sensor is high, we get an over representation of such surfaces. Hence, our second objective is to sub-sample these individual surfaces such that we will retain a minimum number of points without compromising the integrity of the scans. We use an inverse depth relation to sub-sample the cloud. i.e., points are sampled more frequently from the farthest distance and less frequently from surfaces close to the sensor. The points thus obtained are used for building the meta-map. To reduce the number of queries and to reduce the nonlinear least square problem, we further subdivide these points so that a handful of points are selected from each surface.

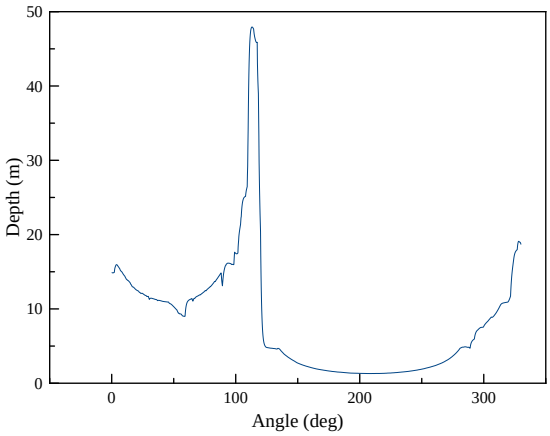

Figure 1. Filtered depth measurement from an individual laser.

2.2 Meta-Map Building

Meta-scan based SLAM solutions tend to be superior, but they come with computational costs. To reduce the costs, the first question we wish to ask is how many frames should be used for the meta-map building. Ideally the larger number of frames we keep (due to the sparse scans), the better we could build a model of the environment. This results in better normal estimation. On the other hand, the map which we build depends on the quality of the navigation solution. The SLAM solution, like an inertial navigation solution, is a dead-reckoning result and it is expected to have an error component which grows over time. This can lead to poor map building over a longer time period, which can further degrade the SLAM solution (due to poor normal estimation).

2.2.1 Ring Buffer To mitigate this problem, we propose to use a simple ring buffer. The architecture of the ring buffer is shown in Fig. 2. Each time a new frame is obtained, we will increment the index pointer m (which points to a buffer inside) and we flush the buffer or move the content to hard disk with appropriate tags (such as current translation and rotation information) for future use. Once the pointer reaches a maximum value M we reset $m = 0$ and repeat the process. When the kd-tree needs to be built, the content of the individual buffers are combined; they undergo some preprocessing operations (such as voxel grid filtering); and result is used to build the kd-tree. We have found that

selecting an appropriate value for M (such that we could store one or two minutes of data) is sufficient to give good navigation solution. The advantage of this strategy is that we are dealing with significantly fewer number of points at any given time, and the cost of storage and computation is greatly reduced.

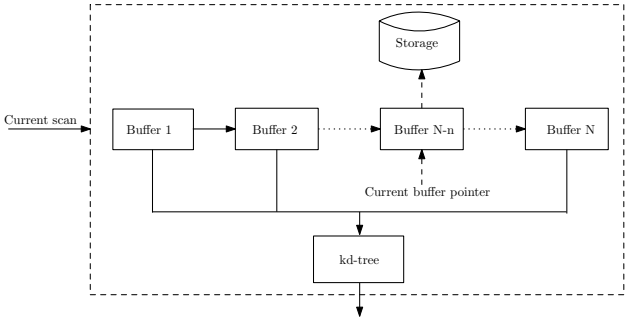

Figure 2. Architecture of ring buffer based storage.

2.2.2 Layered kd-tree Approach When sensor noise is higher, it is advantageous to increase the number of points that are used for surface normal estimation. Instead of keeping fixed nearest neighbors, one can perform a radius search to obtain all of those points satisfying the minimum distance criteria. We also note that when the user moves away from an area which has been scanned thoroughly, we do not have to update the meta-map every frame. Instead, we could update it once in every N frames. To reduce the computation cost, we split the operation into two steps as shown in Fig. 3. The idea is to have two sets of kd-trees which are maintained and queried at different rates. The first kd-tree (which we call as the primary) holds fewer number of points; say from the last M frames of scans. This kd-tree is rebuild for every new frame and all the feature points query the primary kd-tree first. The secondary kd-tree is built and maintained at a lower rate and holds much more points than the primary one. The length of the secondary buffer N depends on the amount of data we want to hold in memory. If the query from primary tree results in a sufficient number of points, the secondary tree is not queried with the same point. Although this approach increases the cost of queries, in practice, most of the closer points can be searched in the primary kd-tree itself and a significant performance improvement can be achieved.

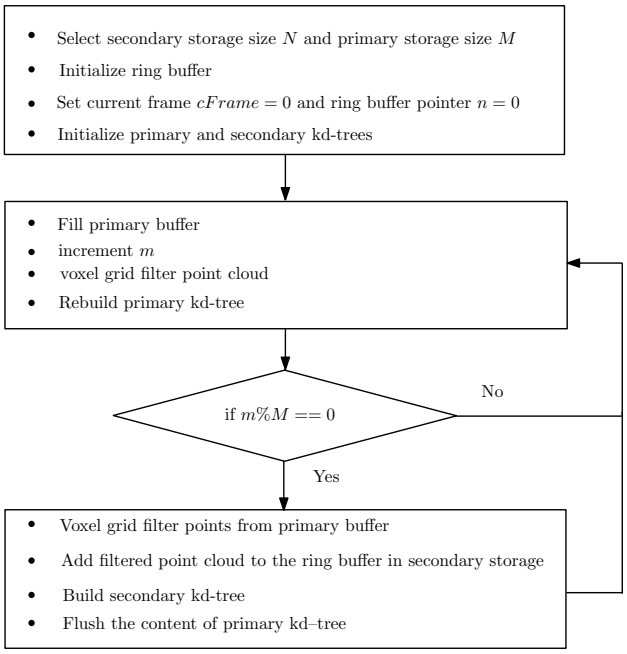

Figure 3. Flow chart of the proposed algorithm.

3. RESULTS

The Pegasus Backpack (Fig. 4) generates approximately 600000 point observations every second from scanners rotating at 10Hz. In principal this means one needs to process 60000 points in 100ms in order to make the system work in real time. In practice however, the number of points can significantly vary depending on the reflective surfaces present in the environment. Figure 5 shows the point cloud obtained from the Pegasus Backpack in an indoor settings. The data from multiple scanners are transformed to a common reference frame. The gray points represent the full point cloud. We apply the sub-sampling technique outlined in this paper to get the green points by first finding individual surfaces scanned by each beam and uniformly sampling it. Such point are are typically 10 to 15% of the full cloud. From this, we further decimate the point selection (up to 50%) for least square computation (shown in red). This reduces the computation burden on the nearest neighbor search. We can see that we are able to capture all the relevant surfaces from the scan.

Figure 4. Pegasus Backpack.

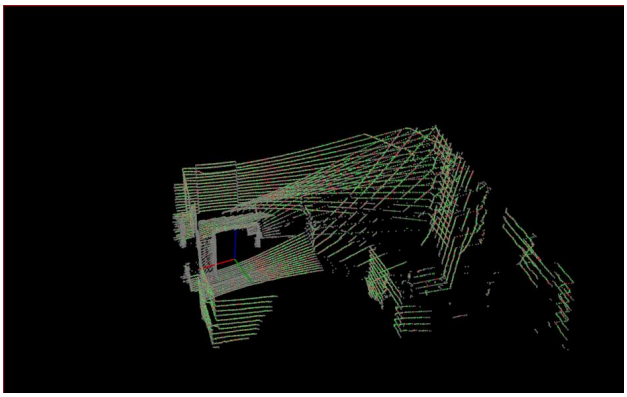

Figure 5. Point cloud obtained from the Pegasus backpack after mapping it to the common reference frame (grey), selected surface points (green) and features (red) used for ICP.

Although our goal of SLAM augmentation is to navigate in the absence of GNSS, in order to validate our algorithms we need to compare our results with a GNSS controlled solution. To do that we selected a partial data set collected outdoors in an environment which contains buildings, vehicles, trees and bushes in close proximity to the user (Data Set 1). The data under test cov-

method	time (s)	average error (m)
dual kd-tree	218.6	0.1562
single kd-tree with ring buffer	608.0	0.1515
all data	1020.5	0.1319

Table 1. Execution time and average mean square error.

ers a time period of 260s. With this data we were able to track GNSS signals and we processed the control GNSS/INS solution using NovAtel/Waypoint Inertial Explorer (IE) software. After processing the control solution has as estimated position accuracy of 2cm in XY and 3cm in Z.

We ran SLAM using all of the data, a part of the data using the ring buffer and layered kd-tree and compared its performance against the control solution. Fig. 6 the XY plot of the trajectory is given. We start at (0, 0) and walk towards (12.964, 29.379). One can see that the trajectory of all the SLAM methods slowly drifts away from the control as a function of time. This is expected since SLAM is a dead reckoning system which accumulates error with time. However, one can see that the difference between using the full data and the layered kd-tree are minimal. To get a better understanding, we plot the mean square error in Fig. 7. One could verify that difference in RMSE between the full data and layered methods has a maximum value of 6cm. On the other hand, one can observe from the table 1 the processing time for the layered method is 1/4 the time compared to full data. After obtaining the navigation solution we built the point cloud with the full resolution which is shown in Fig. 8. From these figures, we can see that the integrity of the map is not compromised with the proposed solution.

To further assess the performance of the proposed methods, we use a longer data set (Data Set 2), which contains sections of tall trees, bushes, skyscrapers and corridors. This data set includes a GNSS outage which results in a larger inertial drift from the conventional GNSS-INS processing. The overall length of the data is 1140s. As in the previous data set, we start our walk from (0, 0) and walk towards (18.343, 4.315). In Figs. 9 and 10, the navigation solution from GNSS/INS (shown in red) can be seen to differ from the final SLAM solution. It can be inferred by plotting the inconsistencies in the point cloud and the changes in erroneous height measurements while travelling over a flat surfaces that the initial GNSS/INS solution drifts in accuracy over time as a result of the missing GNSS updates. We use the double layered kd-tree approach to run ICP SLAM on this data set and the results are shown in blue in these figures. The overall time taken to process this data set with our method is 1030 seconds. The point cloud generated is shown in Fig. 11. From all these analysis, we can conclude that as the SLAM processing time is less than the collection time, our proposed method is suitable for real-time operation without compromising the quality of the navigation solution or the 3D map built.

4. CONCLUSION

In this paper, we propose methods to increase the speed of ICP based SLAM. We proposed two strategies: (1) to sample the point cloud from all the possible plane segments and (2) to use a novel method to store and retrieve spatial data using a layered kd-tree approach. By using the point cloud collected by Pegasus backpack, we showed that the proposed method has real-time processing capabilities with excellent performance characteristics.

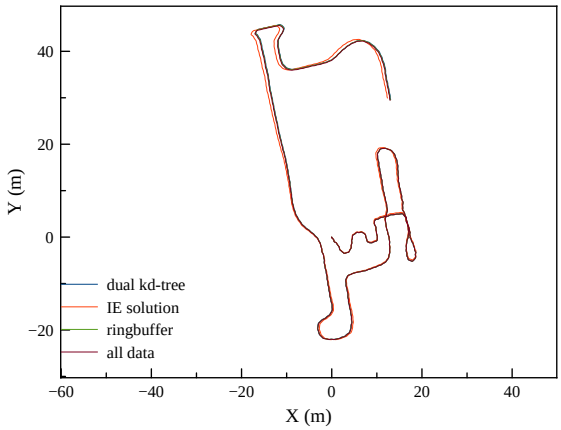

Figure 6. XY plot of the trajectory along with the ground truth which is obtained from GNSS/INS for Data Set 1. The new SLAM method has a maximum difference of 6cm to the control.

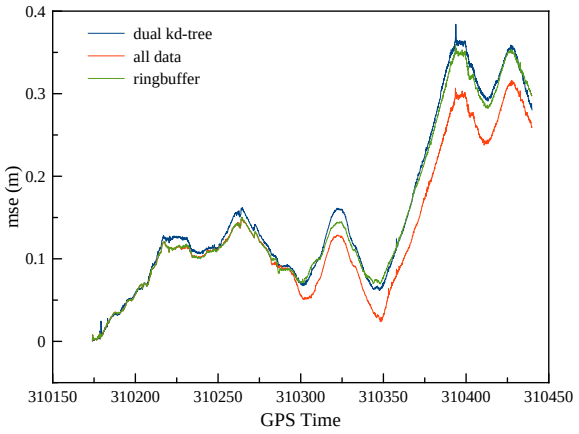

Figure 7. Mean square difference to control for Data Set 1.

REFERENCES

Bailey, T. and Durrant-Whyte, H., 2006. Simultaneous localization and mapping (slam): Part ii. IEEE Robotics & Automation Magazine 13(3), pp. 108–117.

Besl, P. J. and McKay, N. D., 1992. Method for registration of 3-d shapes. In: Robotics-DL tentative, International Society for Optics and Photonics, pp. 586–606.

Durrant-Whyte, H. and Bailey, T., 2006. Simultaneous localization and mapping: part i. Robotics & Automation Magazine, IEEE 13(2), pp. 99–110.

Grant, W. S., Voorhies, R. C. and Itti, L., 2013. Finding planes in lidar point clouds for real-time registration. In: Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on, IEEE, pp. 4347–4354.

Greenspan, M. and Yurick, M., 2003. Approximate kd tree search for efficient icp. In: 3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on, IEEE, pp. 442–448.

Moore, A. W., 1991. An intoductory tutorial on kd-trees.

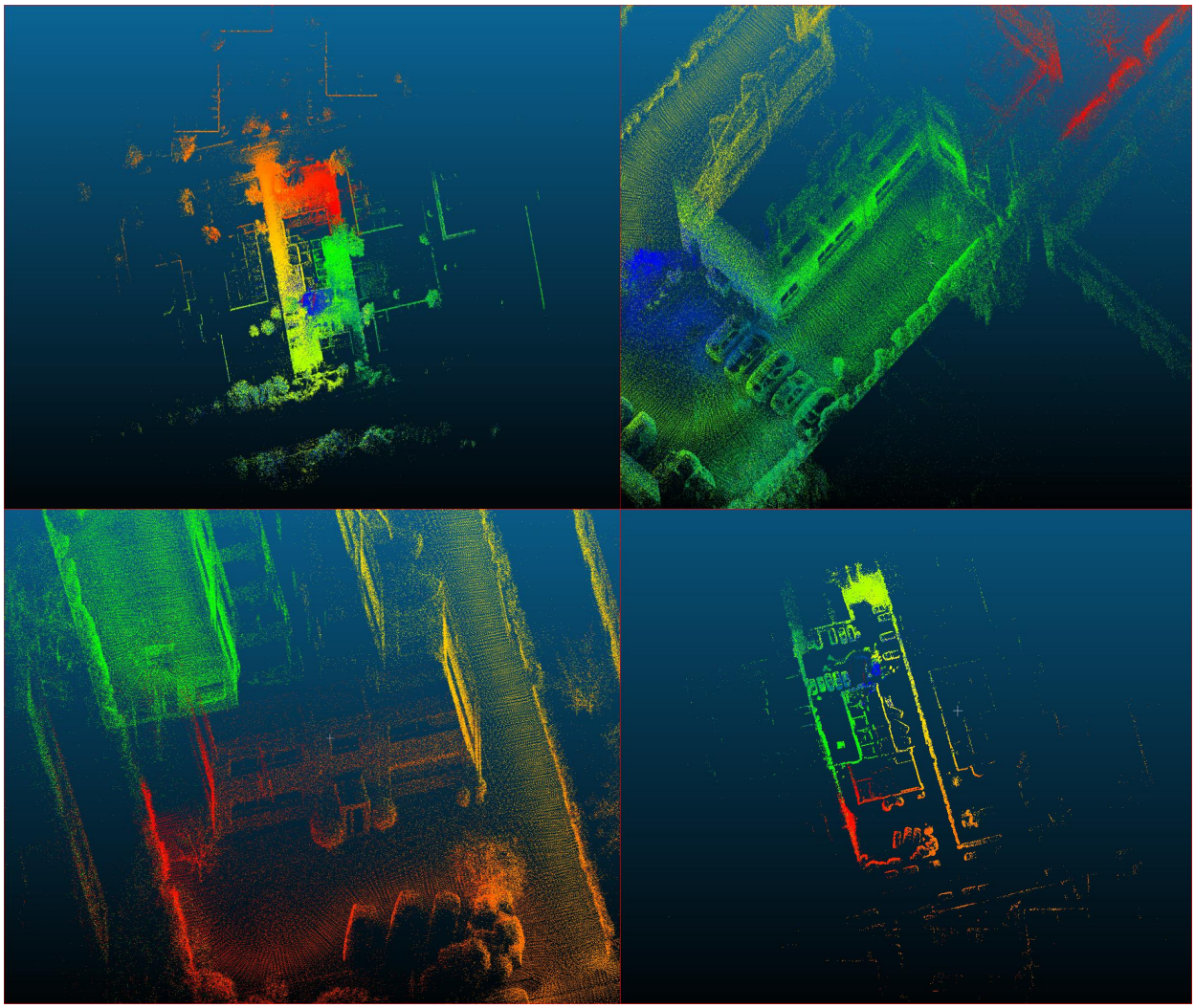

Figure 8. Full point cloud obtained using the layered kd-tree and sub-sampling approach with data set 1.

UL: Top View of Point Cloud UR: Isometric View of Point Cloud
LL: Isometric View of Point Cloud LR: Cross-Section Showing Building Outlines

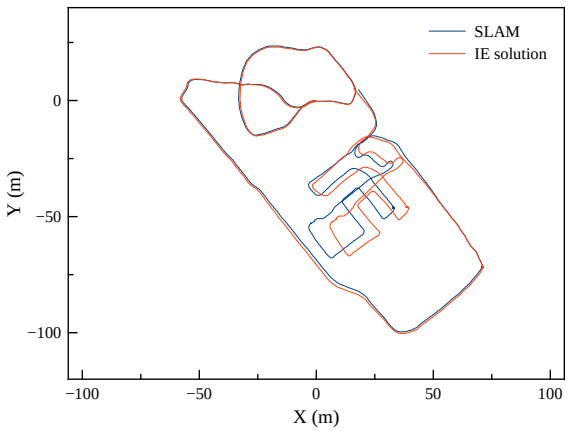

Figure 9. XY profile of Data Set 2. IE Solution drifts while user moves indoors.

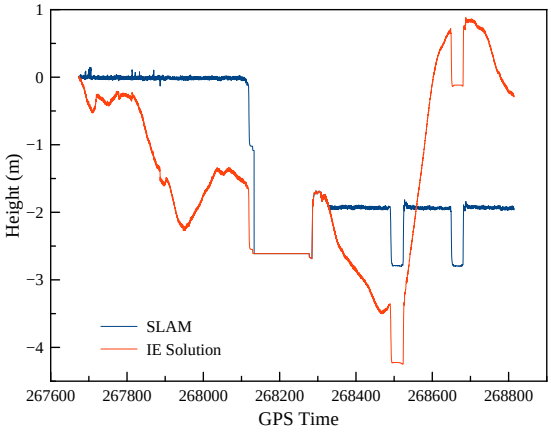

Figure 10. Height profile of Data Set 2. IE Solution shows height errors without GNSS updates. SLAM solution shows a consistent flat trajectory indoors.

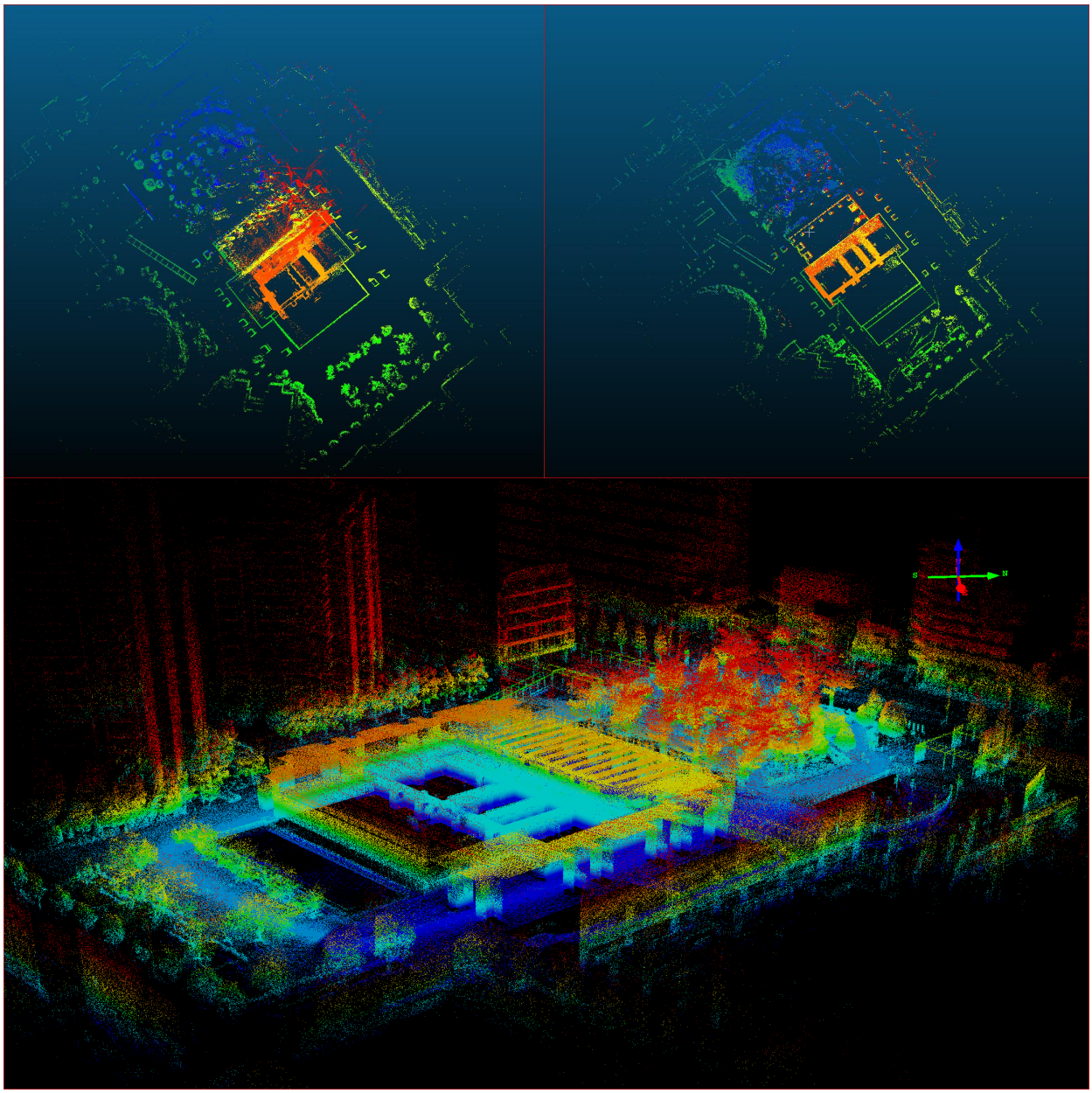

Figure 11. Point cloud generated using data set 2.

UL: Initial Point Cloud without SLAM Aiding UR: Final Point Cloud with SLAM/INS Trajectory
Bottom: Isometric View of Final Point Cloud

Muhammad, N. and Lacroix, S., 2010. Calibration of a rotating multi-beam lidar. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE, pp. 5648–5653.

Nüchter, A., Lingemann, K., Hertzberg, J. and Surmann, H., 2007. 6d slam3d mapping outdoor environments. Journal of Field Robotics 24(8-9), pp. 699–722.

Puente, I., González-Jorge, H., Martínez-Sánchez, J. and Arias, P., 2013. Review of mobile mapping and surveying technologies. Measurement 46(7), pp. 2127–2145.

Segal, A., Haehnel, D. and Thrun, S., 2009. Generalized-icp. In: Robotics: Science and Systems, Vol. 2number 4.

Strasdat, H., Montiel, J. and Davison, A. J., 2010. Real-time monocular slam: Why filter? In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE, pp. 2657–2664.

Wald, I. and Havran, V., 2006. On building fast kd-trees for ray tracing, and on doing that in o (n log n). In: Interactive Ray Tracing 2006, IEEE Symposium on, IEEE, pp. 61–69.

Zhang, J. and Singh, S., 2014. Loam: Lidar odometry and mapping in real-time. In: Robotics: Science and Systems Conference (RSS).