Paper: Rb*--**-****:**

Robust Posegraph Optimization Using Proximity Points

Yuichi Tazaki, Kotaro Wada, Hikaru Nagano, and Yasuyoshi Yokokohji

Graduate School of Engineering, Kobe University, Rokkodai-cho, Nada-ku, Kobe, Hyogo, Japan
E-mail: [tazaki] at mech.kobe-u.ac.jp

[Received 00/00/00; accepted 00/00/00]

This paper proposes a robust posegraph optimization (PGO) method for posegraphs with keypoints. In the conventional PGO formulation, a loop constraint is defined between a pair of nodes, whereas in the proposed method, it is define between a pair of keypoints. In this manner, robust PGO based on switch variables can be realized in a more fine-grained manner. Loop constraint is defined based on the unique geometric property of proximity point, and implemented as a new edge type of the g^{2}o solver. The proposed method is compared with other robust PGO methods using real world data recorded in Nakanoshima Robot Challenge 2021.

Keywords: Mobile Robot, Proximity Points, Robust Posegraph Optimization

1. Introduction

In recent years, rapid aging of the society and rising cost of human labor have been strongly pushing many fields of industry and service towards automation. The SLAM technology, which enables a robot to build a map of the working environment, localize itself in the map, and navigate itself in the environment, is one of the key building blocks of autonomous robots [1]. One fundamental technical issue that arises when a robot operates in a large-scale environment for a long duration of time is how to build and maintain a map in a concise and usable form.

Posegraph is a graphical map representation in which places are represented by nodes and traversability between places are expressed by edges. It is considered more suitable to large-scale mapping compared to grid-based maps, whose data size increases proportionally with the covered area. Two important techniques related to the posegraph SLAM is loop closure detection and posegraph optimization. The former technique is used for detecting loops; matches between different portions of recorded data with high similarity. The latter is used for correcting erroneous travel data by using loops as constraints to obtain a posegraph with higher accuracy [2, 3, 4, 5, 6, 7, 8, 9].

The g^{2}o (General Graph Optimization) solver was proposed in [3] as a powerful computational tool for posegraph optimization. It implements several iterative methods for nonlinear least square problems such as the Gauss-Newton method and the Levenberg-Marquardt method. In addition to its computational efficiency, a major advantage of g^{2}o is its extensibility; new types of vertices and edges can be added by minimum amount of additional programming. One widely studied extension is robustification of the posegraph optimizer. Because of errors of loop detection, any posegraph optimization problem based on real data may include spurious loop constraints, which may cause catastrophic error in optimization results. Robust posegraph optimization is a technique to handle posegraphs including such outliers. In [4, 5] the idea of switch variables was proposed, and this idea was implemented as an extension of g^{2}o. This method defines a switch variable for each loop constraint. By simultaneously optimizing loop constraints and switch variables, loop constraint do not agree with other constraints are automatically supressed. In [6], another method base on M-estimators was proposed. This method realizes robust posegraph optimization without introducing extra variables.

One shortcoming of existing robust PGO framework is that suppression of erroneous loop constraints can only be performed on a node-pair basis; that is, even if an erroneous constraint is caused by only partial mismatch of observation data, the loop constraint must be suppressed entirely. The contribution of this paper is to propose a robust posegraph optimization method for keypoint-based posegraphs and evaluate its effectiveness using real-world data. Proximity point was proposed in the authors' previous work as a type of keypoints with several useful properties. A loop-closure detection algorithm that measures the similarity of two locations based on proximity-point matching was proposed in [10]. Moreover, a real-time detection algorithm of proximity points from 3D pointcloud and a particle-filter based self-localization method that utilizes proximity points were proposed in [11]. However, posegraph optimization based on proximity points, and its robustification, in particular, has not been discussed in the previous studies.

The organization of the rest of this paper is as follows. In Section 2, the definition and geometric characteristics of proximity points are reviewed. In Section 3, robust posegraph optimization based on proximity points is described. In Section 4, experimental results using real-world data are presented. Concluding remarks are given

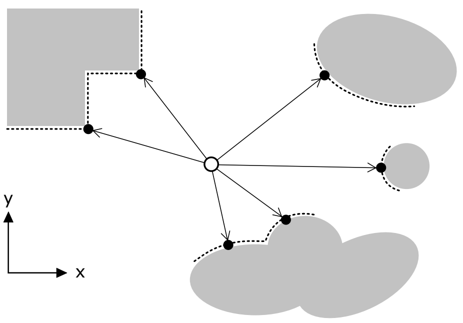


Fig. 1. : A top-view of an environment. Multiple proximity-points (black dots) are detected on different surrounding objects (gray shapes) from a single observation point (hollow circle)

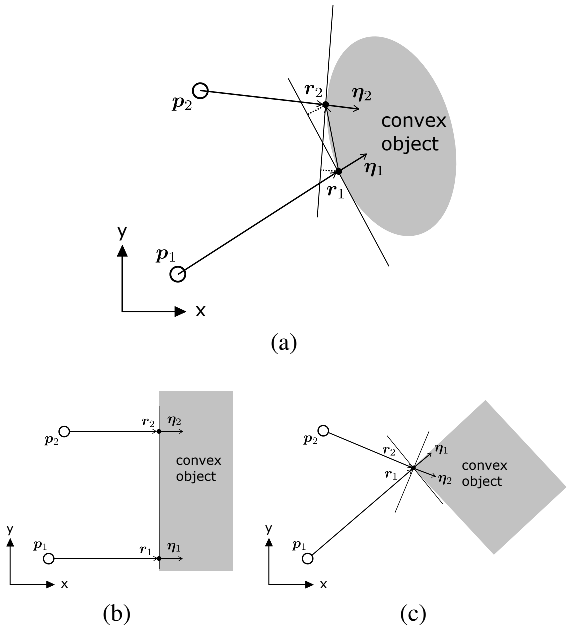


Fig. 2. : Geometric relationship of a pair of proximity-points detected on a single convex object from different observation points

in Section 5.

2. Definition and Geometric Properties of Proximity Points

In the following, the basic definition and geometric characteristics of proximity points are briefly reviewed. Let $\mathscr{P} = \{\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N\}$ be a 3D pointcloud where each $\boldsymbol{r}_i \in \mathbb{R}^3$ denotes the position of a point expressed in a 3D coordinate frame whose origin is located at the observation point. Moreover, let us define the following functions that transform a given $\boldsymbol{r} = [r_\mathrm{x}\, r_\mathrm{y}\, r_\mathrm{z}]^\mathsf{T}$ into spherical coordinates.

$$d(\boldsymbol{r}) = \|r\|,$$
$$\theta(\boldsymbol{r}) = \mathrm{atan2}(r_\mathrm{y}, r_\mathrm{x}),$$
$$\phi(\boldsymbol{r}) = \mathrm{atan2}(r_\mathrm{z}, \sqrt{r_\mathrm{x}^2 + r_\mathrm{y}^2})$$

As illustrated in Fig. 1, a proximity point is a point that is *locally nearest* to the observation point. More precisely, a point $\boldsymbol{r}$ is a proximity point if the following condition is satisfied.

$$\boldsymbol{r} = \arg\min_{\boldsymbol{r}' \in N(\boldsymbol{r})} d(\boldsymbol{r}') \tag{1}$$

Here, $\mathscr{N}(\boldsymbol{r})$ is the set of neighboring points of $\boldsymbol{r}$, defined as follows.

$$\begin{aligned}\mathscr{N}(\boldsymbol{r}) = \{&\boldsymbol{r}' \in \mathscr{P} \mid \\ &|\theta(\boldsymbol{r}') - \theta(\boldsymbol{r})| < \sigma_\theta, |\phi(\boldsymbol{r}') - \phi(\boldsymbol{r})| < \sigma_\phi\}\end{aligned} \tag{2}$$

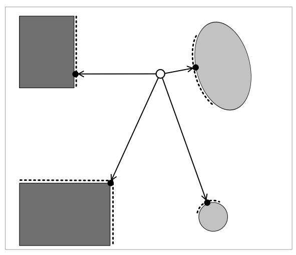
Here, σ_θ and σ_ϕ are parameters that determine the size of the neighborhood. Theoretically, if one could assume that the pointcloud is infinitely dense, then σ_θ and σ_ϕ could be infinitesimal; in practice, however, these parameters should be chosen carefully considering the actual spatial resolution of the pointcloud.

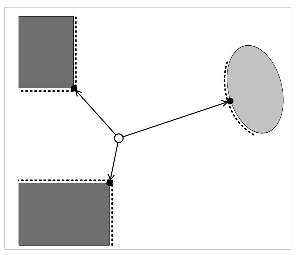
Proximity points have some useful geometric properties that can be utilized for loop-closure detection and posegraph optimization. Consider a convex object and two different observation points whose poses in the global coordinate frame are given by $\boldsymbol{x}_1 = (\boldsymbol{p}_1, \theta_1)$ and $\boldsymbol{x}_2 = (\boldsymbol{p}_2, \theta_2)$, respectively, as illustrated in Fig. 2(a)-(c). Moreover, let $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ be two proximity points detected on the surface of the convex object from these observation points, each expressed in the local coordinate frame of the corresponding observation point. The convexity of the object implies that the following inequality generally holds.

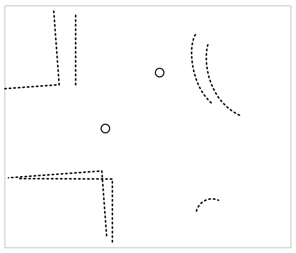
$$\begin{aligned}&\boldsymbol{e}(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{r}_1, \boldsymbol{r}_2) \\ &:= \begin{bmatrix}(R(\theta_1)\boldsymbol{\eta}_1)^\mathsf{T}((\boldsymbol{p}_2 + R(\theta_2)\boldsymbol{r}_2) - (\boldsymbol{p}_1 + R(\theta_1)\boldsymbol{r}_1)) \\ (R(\theta_2)\boldsymbol{\eta}_2)^\mathsf{T}((\boldsymbol{p}_1 + R(\theta_1)\boldsymbol{r}_1) - (\boldsymbol{p}_2 + R(\theta_2)\boldsymbol{r}_2))\end{bmatrix} \\ &\geq \mathbf{0}\end{aligned} \tag{3}$$

Here, $\boldsymbol{\eta}_i = \boldsymbol{r}_i / \|\boldsymbol{r}_i\|$, and the vector inequality is evaluated componentwise.

The implication of the inequality above is the following. Consider a tangent plane of the object at the proximity point $\boldsymbol{r}_1$. Then, because of the convexity of the object, the other proximity point $\boldsymbol{r}_2$ must be on this tangent plane or in the opposite side of the plane from the observation point $\boldsymbol{p}_1$. The same relationship holds with the indices 1 and 2 swapped. Examples are shown in Figs. 2(a)-(c). In special cases where the curvature of the object is extremely small (e.g., a wall) or extremely large (e.g., a rectangular corner of an object), these inequalities approximately hold with equality. In fact, in cases illustrated in Fig. 2(b)(c), $\boldsymbol{e} = \mathbf{0}$ holds.


(a) First visit


(b) Second visit


(c) Relative pose obtained by scan matching

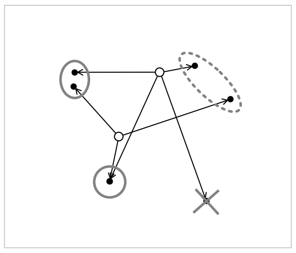

(d) Proximity point matching constraints

Fig. 3. : Comparison of scan matching and proximity-point matching

3. Robust Posegraph Optimization Based on Proximity Points

In general, posegraph optimization is formulated as a large-scale nonlinear optimization problem in which the poses of the vertices are optimized to minimize the sum of costs defined on the edges. In the conventional formulation, each edge cost is defined as a cost function that penalizes the deviation of the relative pose of the connected two vertices from the desired relative pose. A major difference of our formulation is that each edge cost penalizes the geometric error of a matched pair of proximity points. The loop-detection algorithm proposed in our previous study [10] outputs a series of matched proximity points $\mathscr{M} = \{m_i\}$. Here, the i-th match is a 4-tuple (j_i, j'_i, k_i, k'_i), which indicates that the k_i-th proximity point of the j_i-th vertex and the k'_i-th proximity point of the j'_i-th vertex is a matching pair. It defines a constraint between the two poses $\boldsymbol{x}_{j_i}$ and $\boldsymbol{x}_{j'_i}$ based on (3).

Another major difference is in how robust posegraph optimization is realized. In the conventional formulation, a switch variable is defined for each edge cost defined between a pair of vertices. In this manner, the weight of edges could only be adjusted in a per-vertex-pair basis. This means, even if only a small portion of pointcloud is causing the mismatch, the entire edge cost has to be weakened. On the other hand, we introduce one switch variable for each proximity-point pair. Considering that generally several matched proximity-point pairs are defined between every pair of vertices, even if a few wrongly matched proximity-point pairs get suppressed, the remaining (correct) pairs could be used to correct the relative pose of the vertices.

Figures 3(a)-(d) illustrate the advantage of proximity-point-based loop constraints over conventional relative-pose-based constraints. Consider that a robot visited the same place twice (Figs. 3(a),(b)), and pointcloud and proximity points were detected. Here, in the second visit, the objects shown in light gray either changed its position or disappeared completely. Clearly, the relative pose constraint computed by scan matching between (a) and (b) will be erroneous (Fig 3(c)). This constraint could be suppressed by setting the switch variable as zero, but in any case, this constraint will be useless at best. On the other hand, matched proximity points are shown in Fig. 3(d). Here, valid matches (the ones surrounded by solid circles) will be used for constraining the vertex poses, while the erroneous match (the one surrounded by a dashed circle) will be suppressed, and the unmatched proximity point (the crossed one) will be simply ignored. In this manner, the proximity-point-based method enables selection of valid and invalid (suppressed) matches with higher flexibility.

In this study, the general graph optimizer (g^{2}o) [3] is used as a basic framework of posegraph optimization. One of the useful features of g^{2}o is its extensibility; that is, one can add a new definition of edge type while utilizing pre-defined vertex and edge types and powerful optimization functionality of g^{2}o. To implement proximity-point-based PGO in the framework of g^{2}o, a new vertex named VERTEX_PROX is defined. Similar to VERTEX_SE2, which is one of the predefined vertex types, VERTEX_PROX parametrizes a pose in SE(2) as

$$\boldsymbol{x}_j = \begin{bmatrix} \boldsymbol{p}_j \\ \theta_j \end{bmatrix} \tag{4}$$

where j is the vertex index. $\boldsymbol{x}_j$ expresses the pose of the robot at the j-th observation point in the global coordinate frame. In addition, each VERTEX_PROX vertex is assigned a set of proximity points $\mathscr{R}_j = \{\boldsymbol{r}_{j,1}, \dots, \boldsymbol{r}_{j,n_j}\}$. Note that while $\boldsymbol{x}_j$ is a decision variable, proximity points in $\mathscr{R}_j$ are used as parameters that define proximity-point-based edge constraints. For expressing relative pose constraints based on odometry, a predefined edge type EDGE_SE2 defined below is used.

$$J_{\mathrm{se2},i} = (\boldsymbol{x}_{j'_i} - \boldsymbol{x}_{j_i} - \Delta\boldsymbol{x}_i)^{\top} \Omega_{\mathrm{se2},i} (\boldsymbol{x}_{j'_i} - \boldsymbol{x}_{j_i} - \Delta\boldsymbol{x}_i) \tag{5}$$

Here, j_i and j'_i are the index of the connected vertices, $\Delta\boldsymbol{x}_i$ is the desired relative pose, and $\Omega_{\mathrm{se2},i}$ is the covariance matrix.

Moreover, a new edge type named EDGE_PROX is defined to implement loop constraints between proximity points. The i-th edge has a 4-tuple of indices $m_i = \{j_i, j'_i, k_i, k'_i\}$ whose definition has been given previously. Moreover, it has a switch variable s_i. The cost function of EDGE_PROX is defined as follows.

$$\begin{aligned} J_{\mathrm{prox},i} &= s_i^2 \boldsymbol{e}_i^{\top} \Omega_{\mathrm{prox},i} \boldsymbol{e}_i + \xi (1 - s_i)^2, \\ \boldsymbol{e}_i &= \boldsymbol{e}(\boldsymbol{x}_{j_i}, \boldsymbol{x}_{j'_i}, \boldsymbol{r}_{k_i}, \boldsymbol{r}_{k'_i}) \end{aligned} \tag{6}$$

Here, $\boldsymbol{e}$ is the proximity point matching error defined in (3), and $\Omega_{\mathrm{prox},i}$ is the covariance matrix. The switch variable s_i, by definition, is an unconstrained real variable,

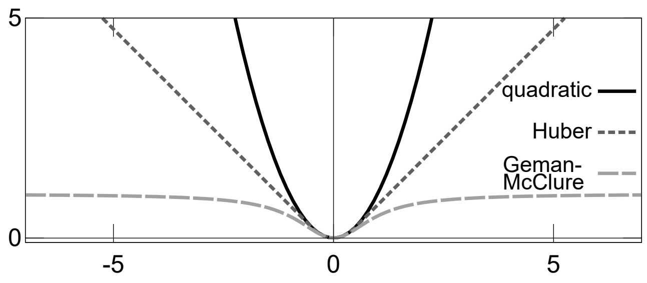


Fig. 4. : Profile of cost functions: quadratic, Huber, and Geman-McClure

but the optimal value of s_i is always between 0 and 1 (it is clear from (6) that s_i outside the range $[0,1]$ can never be optimal). When the switch variable s_i is close to zero, the weight of the error term $\boldsymbol{e}_i^{\mathsf{T}} \Omega_i \boldsymbol{e}_i$ becomes small, and this loop constraint is effectively suppressed. The second term, $\xi(1-s_i)^2$, is a penalty that must be paid for making s_i close to zero. This penalty term is needed to avoid a trivial solution with all switch variables set as zero. The value of the parameter ξ, which control the strength of this penalty term, must be determined manually.

The overall posegraph optimization problem is formulated as follows.

$$\begin{array}{ll} \text{find} & \boldsymbol{x}_j \ (j \in \mathscr{V}), \ s_i \ (i \in \mathscr{E}_{\text{prox}}) \\ \text{minimize} & \displaystyle\sum_{i \in \mathscr{E}_{\text{prox}}} J_{\text{prox},i} + \sum_{i \in \mathscr{E}_{\text{se2}}} J_{\text{se2},i} \end{array} \tag{7}$$

Here, $\mathscr{V}$ is the set of VERTEX_PROX vertices, $\mathscr{E}_{\text{se2}}$ is the set of EDGE_SE2 edges, and $\mathscr{E}_{\text{prox}}$ is the set of EDGE_PROX edges. Note that the node poses $\boldsymbol{x}_j$ and the switch variables s_i are optimized simultaneously in the single optimization problem (7). As a result of optimization, outlier edges that do not agree with other edges are suppressed, and node poses that minimize the total error of unsuppressed edges are computed.

As described in Section 2, the matched proximity points should satisfy the inequality (3). In other words, it is no problem if the components of $\boldsymbol{e}$ take positive values. In this sense, penalizing the magnitude of $\boldsymbol{e}$ as defined in (6), which essentially require $\boldsymbol{e}$ to be zero, may seem inappropriate. In practice, however, many objects appear in the environment have either very small curvature or very large curvature, in which case $\boldsymbol{e}$ become close to zero. Moreover, it is technically difficult to handle inequality constraints in numerical optimization, especially when we do no want to modify the core optimization routine of g^{2}o. For this reason, we consider that cost definition (6) is practically acceptable.

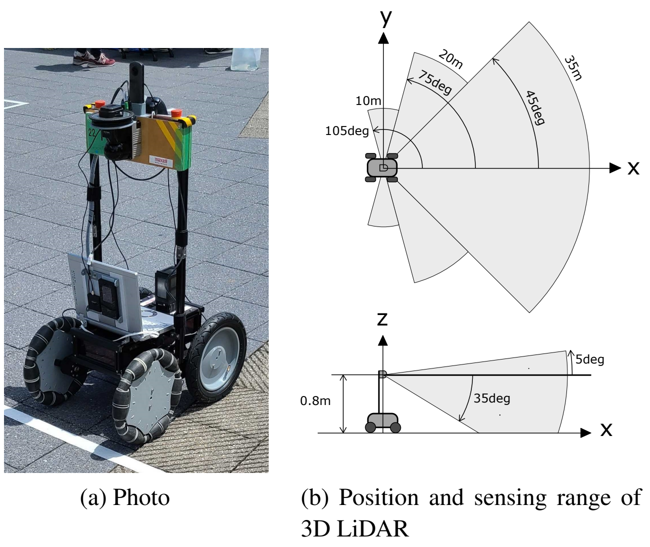
For comparison, we also consider robust optimization based on the M-estimators: the Huber method and the Geman-McClure method. The essential idea of the M-estimator-based methods is to filter the original cost function so that the gradient of the filtered cost function saturates to a certain limit as the error becomes greater than the specified threshold. Consider a quadratic edge cost defined as $J = \boldsymbol{e}^{\mathsf{T}} \Omega \boldsymbol{e}$. The Huber method defines the filtered cost as

$$\rho_{\text{H}}(J) = \begin{cases} J & \text{if } J \leq \sigma^2 \\ 2\sigma\sqrt{J} - \sigma^2 & \text{otherwise} \end{cases} \tag{8}$$

while in the Geman-McClure method, it is defined as

$$\rho_{\text{G}}(J) = \frac{J_i}{\sigma^2 + J}. \tag{9}$$

The profile of these functions are shown in Fig. 4 for a scalar quadratic cost $J(x) = x^2$. An advantage of M-estimator-based robustification is that there is no need to introduce extra decision variables. Its shortcoming is poor convergence caused by small gradient of ρ function. Another shortcoming is that it can weaken but cannot completely nullify the influence of spurious edges.



(a) Photo

(b) Position and sensing range of 3D LiDAR

Fig. 5. : Mobile robot used for data recording

4. Experimental Results

4.1. Experimental Setup

Experiments were conducted to evaluate the proposed robust posegraph optimization method. Figure 5(a) shows the mobile robot Omnia3 used for data recording. It is equipped with a 3D-LiDAR (Hokuyo YVT-35LX-F0), whose installation position and sensing range are illustrated in Fig. 5(b). At a period of 500ms, the 3D-LiDAR measures a 3D pointcloud consisting of 20,000 points, and from this pointcloud, proximity points are detected by the algorithm described in detail in [11]. An example of 3D pointcloud and proximity points detected from it is shown in Fig. 6. Statistically, the number of proximity points detected from a single set of pointcloud is 8 in average and 20 at most.

Data recording was conducted in two environments shown in Fig. 7. The first one is the course of Nakanoshima Robot Challenge 2021 Extra Challenge.

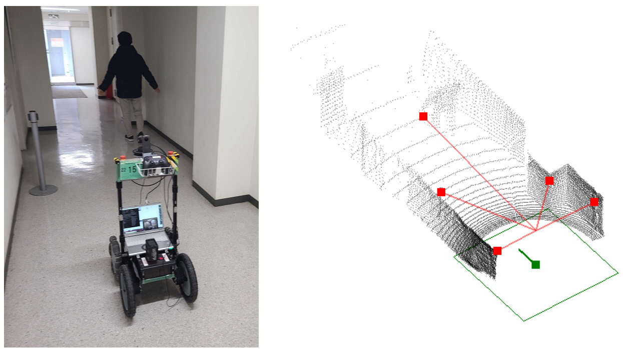
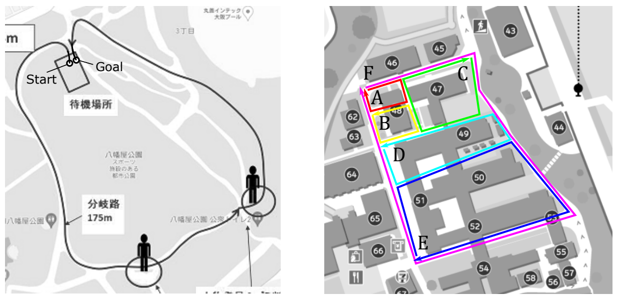


Fig. 6. : An example scene (left), 3D pointcloud (right, black dots), and detected proximity-points (right, red square markers)



(a) Nakanoshima Robot Challenge 2021 Extra Challenge Course (excerpt from the challenge proposal)

(b) Kobe University engineering department building

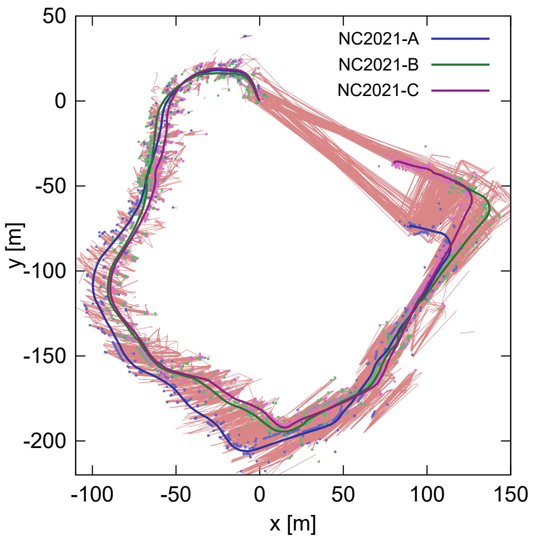
Fig. 7. : Test environment

Three data-recording runs were conducted, and recorded time-series were labled NC2021-A to NC2021-C. In each run, the mobile robot was manually operated from the starting point along the course to the goal point, and returned to the starting point. Since many pedestrians and other participating robots are in the scene, there are many sources of observation noise that cause erroneous loop-closure detection. The second environment is the engineering department building of Kobe University. In this environment, the mobile robot was manually operated to drive along several difference cyclic routes as shown in Fig. 7(b), and the recorded data were labeled KU-A to KU-F. In the next section, we evaluate posegraph optimization results obtained by different methods in terms of the magnitude of error. Here, error refers to the value of the cost function used for optimization, and it does not refer to error from some kind of ground truth.

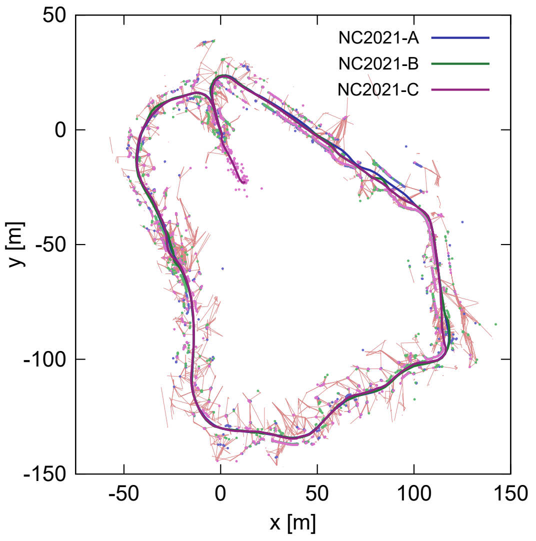
4.2. Posegraph optimization results

4.2.1. NC2021

Figure 8(a)(b) show the posegraph of NC2021 before and after posegraph optimization. Trajectory points and proximity points that belong to different time-series (NC2021-A to NC2021-C) are depicted in different colors. We can see in Figure 8(a) that large odometry error is present before optimization. Loop constraints that are detected by the loop detection method presented in [10] are depicted by pale red lines connecting matched proximity points. Loops are mainly detected between regions near the start and the goal of the same time-series, and between close regions of different time-series. Figure 8(b) shows the posegraph after the application of the proposed robust PGO. We can see that error is greatly reduced. There are some remaining loop constraints shown as red lines, but they are mostly spurious loops that were suppressed thanks to robust optimization technique.



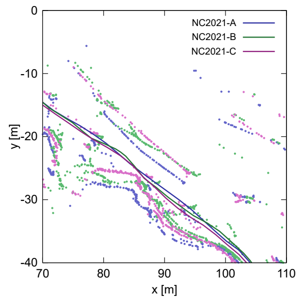
(a) before



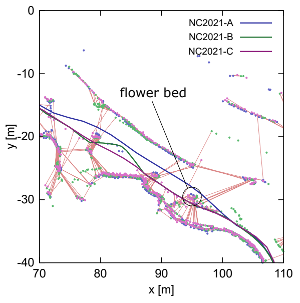
(b) after

Fig. 8. : Before and after PGO (NC2021Ex)

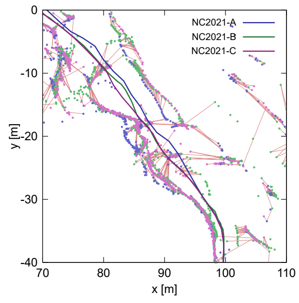
Different robust PGO methods were applied to NC2021. All compared methods could successfully reduce large odometric error that were observed before optimization. Nevertheless, small but notable differences are



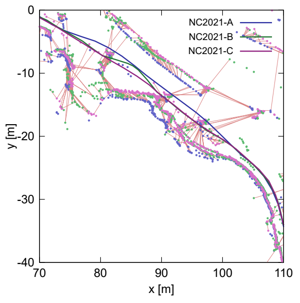
(a) loop constraints by relative pose



(b) loop constraints by proximity-point pair



(c) Huber M-Estimator



(d) Geman-McClure M-Estimator

Fig. 9. : Comparison of robust PGO methods

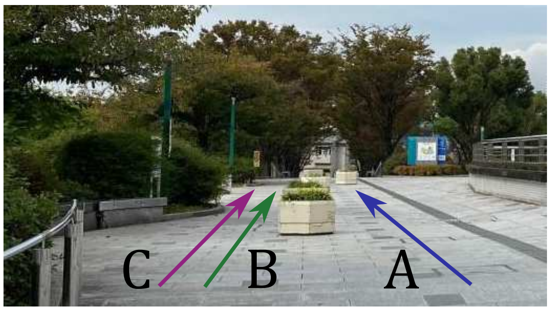
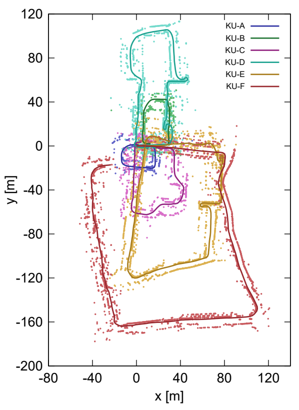
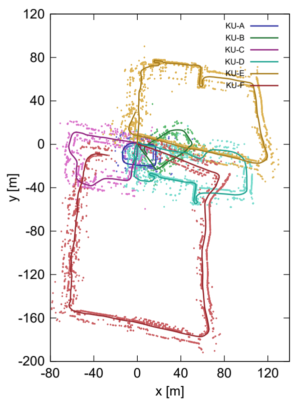


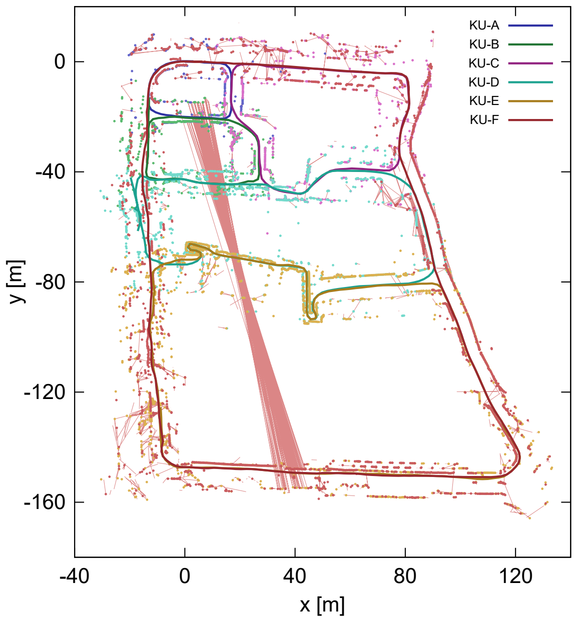
Fig. 10. : Difference of routes taken around the flower bed



(a) Before



(b) After rotation-only optimization



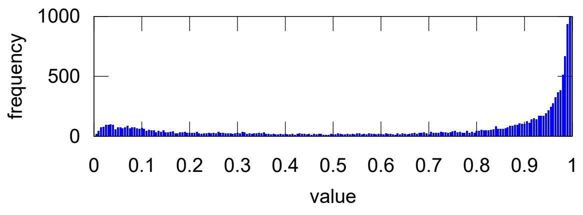
(c) After full optimization

Fig. 11. : Posegraph optimization results of KU

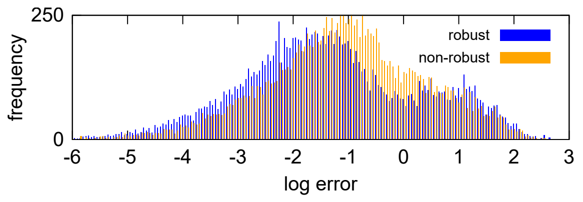
observed in some regions. One example of such regions is shown in the close-up images Figs. 9(a)-(d). A photo of the same region is shown in Fig. 10. A flower bed was located in the middle of the walkway, and in NC2021-A, the robot avoided to the right, whereas in NC2021-B and NC2021-C, it avoided to the left. This small difference of travel route caused relatively a large number of erroneous loop detections around this region. In the relative-pose-based formulation, the switch variable corresponding to loop constraints defined between nodes in this region was weakened uniformly, and there remained large error as shown in Fig. 9(a). In the result of proximity-point-based formulation is shown in Fig. 9(b). Wrong proximity-point matches were weakened, while correct ones remained in effect, and as a reult, a better optimization result was achieved. The results of M-estimator methods, Huber and Geman-McClure, are shown in Figs. 9(c) and (d), respectively. Here, the value of the parameter σ was hand-tuned to obtain the best result. Compared to the proposed method, average error remained large. From this result, it was found that the M-estimator method was not as good as the switch-variable method in rejecting spurious loops.

4.2.2. KU

Six series of data, KU-A to KU-F were recorded and used for map construction. Loop detection were performed on every pair with overlapping routes. Figs. 11(a)-(c) show the posegraphs before and after optimization. Before optimization, the first node of each data was set to the origin and its orientation angle was zero. When PGO was directly applied to this initial configuration, it was hard to avoid converging to an erroneous local minimum. To avoid this problem, PGO was performed in two

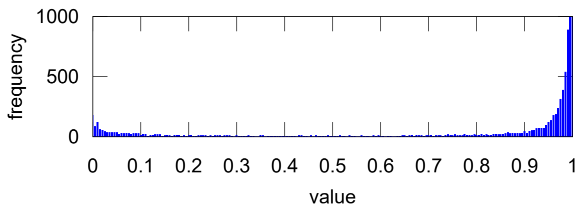


(a) Histogram of s_i

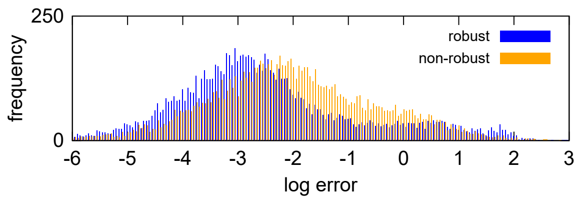


(b) Histogram of $\log(\|\boldsymbol{e}_i\|)$

Fig. 12. : Comparison of error distribution



(a) Histogram of s_i



(b) Histogram of $\log(\|\boldsymbol{e}_i\|)$

Fig. 13. : Comparison of error distribution of KU

steps; in the first step, the orientation of nodes were optimized while the position was fixed, and in the second step, full optimization was computed. A set of erroneous loops were detected between KU-B and KU-F. Thanks to robust optimization, these loops were suppressed and made little influence to the result.

4.3. Effect of Switch Variables

For NC2021, histograms of the switch variables and the edge costs after optimization are shown in Figs. 12(a) and (b), respectively. In Fig. 12(b), the error distribution of two cases are shown: one is when the switch variables were optimized (labeled robust) and the other is when all switch variables were fixed to 1 (labeled non-robust). Fig. 12(a) shows that most switch variables are concentrated around the two extremes, 0 and 1, where the cluster around 1 is much larger. As shown in Fig. 12(b), the error distribution without switch variable optimization (i.e., all s_i fixed to 1) has a single great peak, while the distribution with switch variable optimization shows two peaks. Here, the smaller peak on the right consists of suppressed edges, while the greater peak on the left consists of unsupressed ones. We can see that the latter peak is shifted to the left, indicating that the average error of unsupressed edges where reduced greatly thanks to the supression of a smaller number of outlier edges. Fig. 13(a),(b) show the same statistics for KU, in which similar observations can be made.

Table 1. : Number of iterations and total computation time of each method

Method	NC2021		KU2022	
	# iter	time[s]	# iter	time[s]
switch vars	11	2.50	36	6.06
Huber	11	1.55	-	-
Geman-McCLure	10	1.44	50	5.79

4.4. Computation Cost

Table 1 summarizes the number of iterations and total computation time of three methods: switch variables, Huber, and Geman-McClure, all based on proximity-point matching constraints. Computation was performed on a Windows computer with AMD Ryzen 9 5950X CPU with 16 cores and 32 parallel threads.

For NC2021, all three methods converged after almost the same number of iterations. Note that close-up images of optimization results are shown in Figs. 9(b)-(d). The switch-variable-based method required slightly greater computation time because of the increased dimensionality of the problem. Nevertheless, the overall computation is completed in a matter of seconds, therefore it is considered to be little problem in practice. For KU2022, the switch variable-based method converged successfully although a greater number of iterations was required compared to NC2021. On the other hand, the M-estimator methods performed poorly. Huber, in particular, failed to converge within 500 iterations. Geman-McClure converged, but large error remained in the output posegraph. The above results indicates the switch-variable-based method can achieve better convergence than the M-estimator methods. Although the computation time of each iteration increases, the total computation cost is reduced thanks to faster convergence.

5. Conclusion

This paper proposed a robust keypoint-based posegraph optimization method that can be used for large-scale mapping of outdoor environments. By assigning a switch variable to each keypoint pair, the proposed method was able to perform robust PGO in a more fine-grained manner compared to conventional methods that assign a switch variable to each node pair. One limitation of our method in the present form is that the generated posegraph retains the serial structure of recorded data it is based on. Our next goal is to extend the current framework to incremental topological map building that is capable of capturing the topological structure of the environment in the map.

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*, 32(6):1309–1332, 2016.

[2] D. M. Cole and P. M. Newman. Using laser range data for 3d slam in outdoor environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1556–1563, 2006.

[3] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g^2o: a general framework for graph optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, 2011.

[4] N. Sünderhauf and P. Protzel. Switchable constraints for robust pose graph slam. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 1879–1884, 2012.

[5] N. Sünderhauf and P. Protzel. Towards a robust back-end for pose graph slam. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1254–1261, 2012.

[6] P. Agarwal, G. D. Tipaldi, L. Spinello, C. Stachniss, and W. Burgard. Robust map optimization using dynamic covariance scaling. In *2013 IEEE International Conference on Robotics and Automation*, pages 62–69, 2013.

[7] G. Hu, K. Khosoussi, and S. Huang. Towards a reliable slam back-end. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 37–43, 2013.

[8] G. Agamennoni, P. Furgale, and R. Siegwart. Self-tuning m-estimators. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4628–4635, 2015.

[9] F. Wu and G. Beltrame. Cluster-based penalty scaling for robust pose graph optimization. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, pages 6193–6200, 2020.

[10] Y. Tazaki, Y. Miyauchi, and Y. Yokokohji. Loop detection of outdoor environment using proximity points of 3d pointcloud. In *IEEE/SICE International Symposium on System Integration (SII)*, pages 411–416, 2017.

[11] Y. Tazaki and Y. Yokokohji. Outdoor autonomous navigation utilizing proximity points of 3d pointcloud. *Journal of Robotics and Mechatronics*, 32(6):1183–1192, 2020.