2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

The Masked Mapper: Masked Metric Mapping

Acshi Haggenmiller[1] Cameron Kabacinski[2] Maximilian Krogius[1] Edwin Olson[1,2]

***Abstract*— In this paper, we propose a flexible mapping scheme that uses a *masking function* (*mask*) to focus the attention of a pose graph SLAM (Simultaneous Localization and Mapping) system. The masking function takes the robot's observations and returns true if the robot is in an important location. State-of-the-art methods in SLAM generate dense metric lidar maps, creating precise maps at a high computational cost by storing lidar scans for each pose node and continually attempting to close loops. In many cases, trying to always make loop closures is unnecessary for localization and even risky because of perceptual aliasing and false positives. By masking out these less useful positions, our method can create more accurate maps despite performing far fewer scan matches. We evaluate our system with three simple mask functions on a 2.5 km trajectory with significant angular drift. We compare the number of scan matches performed under each mask as well as the accuracy of the loop closures.**

I. Introduction

Typical robot navigation systems use dense metric map representations that are based around information-dense lidar maps for localization and route planning. In pose SLAM, a factor graph is constructed with nodes representing the pose of the robot at different points in time and edges representing odometry measurements and lidar-scan loop closures. There are efficient techniques to optimize these factor graphs as non-linear least squares problems [1]. By continually integrating odometry and lidar scans, the precise location of the robot and a detailed map of its environment can be maintained at all times. As the size of the map increases it becomes more and more difficult to maintain global consistency, as odometry drift will continue to accumulate until the robot returns to an area it has seen before. This accumulated error can be eliminated by matching the robot's current location to places it has seen before to form loop closures, but this process is computationally expensive and has the risk of incorrectly matching similar-looking regions (perceptual aliasing). The continuous loop-closing work of dense metric SLAM produces highly precise maps at a high computational cost and potential risk of false-positive loop closures.

Topological mapping methods address these problems by focusing on interpreting the semantics of the environment and using these to build models that more closely resemble the human cognitive map [2] with intersections, paths, places, regions, and so forth. These representations can be very simple and the robot will only need to close loops when it makes sense based on the topology and semantic labels. As a result, topological maps can generally define an environment with fewer nodes and loop closures, although the process of determining the correct semantic labels for the robot's location and of segmenting the environment can be complex.

[1]Robotics Institute, University of Michigan
[2]Computer Science and Engineering Department, University of Michigan
{acshikh,camkab,mkrogius,ebolson}@umich.edu

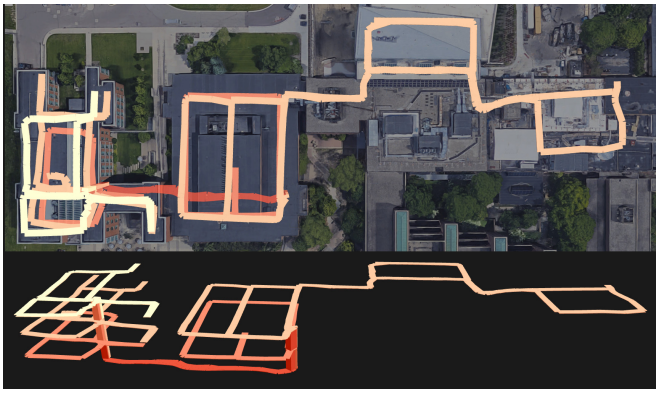

Fig. 1: Top-down (top) and perspective (bottom) views of a masked metric map, spanning several buildings on the University of Michigan North Campus. The floors are color-coded, using a lighter color for each floor from the first to the fourth. The robot drove 2.5 km and the total optimized path length over all floors is 1.9 km. Imagery from Google and Maxar Technologies.

The key advantage in topological mapping is that high-precision localization in hallways and similar environments is often unnecessary. A simple control law such as go-straight or wall-follow will reliably navigate these scenarios without centimeter-level localization. In addition, these hallways are the places where scan matching systems are most likely to fail from perceptual aliasing.

Our system takes advantage of the fact that some places need precise localization more than others by abstracting this distinction into a configurable masking function. A well-designed masking system can achieve both lower probability of failure and lower computational complexity, sacrificing precision in hallways and similar locations that is often unneeded.

The main contributions of this paper include:

1) A flexible approach to masking a high performance SLAM system to lower computational cost and minimize the chance of errors due to perceptual aliasing.
2) Two effective masking functions based on detecting openings and intersections that improve map quality while performing five times fewer scan matches than an "always true" mask function.
3) Demonstration of our system on a 2.5 km dataset.

II. Related Work

In building a consistent map, mapping methods must choose when and how to represent locations, how to detect

loop closures and validate them, and how to optimize the resulting map/graph. We generally refer to any specifically remembered location as a *node*.

In metric pose SLAM, nodes are generally placed at a regular spacing, with each node maintaining its measurements and object frame. Matches between nodes are used to close loops, and in an early paper, matches are searched for between all pairs of nodes with a matching score threshold for acceptance [3]. Thrun et. al. filter the node pairs to check with a series of geometric constraints before matching them, and repeat this in an iterative approach [4]. Google's Cartographer builds locally consistent submaps, and then later connects them. Whenever the system gets a new lidar scan, it attempts to match it with nearby completed submaps. Loop closures that have a high enough match score are added to the optimization with a Hilbert loss for robustness against false positives [5]. Our method follows from dense pose SLAM, focusing on how sparse node placement may make the matching and loop closure problems easier.

Just as our method is based around a selective masking function, topological methods often focus on node selection in their maps. The nodes can represent entire partitioned areas, as with how Thrun partitions an occupancy-grid map by finding the critical points and lines of the Voronoi skeleton where clearance is locally minimized [6]. The nodes can also be points in the robot's trajectory, with the trajectory segmented into straight lines and nodes placed at the "corners", fitting the segments to the original trajectory [7]. Although simple, this representation can be used for absolute self-localization and building consistent maps [8].

The Spatial Semantic Hierarchy (SSH) topological mapping framework also starts with abstract definitions and lends itself to extension. Nodes are abstractly defined as the end-state of a hill-climbing control law that should be unique for localization and consistent mapping, while the edges between nodes are also abstractly specified by a transition control law and a termination criterion [2]. The nodes are often defined to be decision points at gateways or openings in the map. They can be found in ways of varying complexity: by computing the medial axis of the free space in a local occupancy grid constructed around each node [9]; by computing the pruned Voronoi skeleton of an occupancy grid and selecting the portions of the skeleton that leave the "core" [10]; and by calculating the Voronoi skeleton, finding the isovist at each point, computing the isovist eccentricity, and finally selecting the locations of openings with a learned classifier [11]. These methods generally take advantage of semantic place-type labels and gateway locations to filter loop closure candidates and get initial guesses for the match transformations. Another hybrid approach grounds the nodes as areas of high sensory overlap and then optimizes the robot's path, the local metric maps, and the transformations between areas as a unified Bayesian problem [12].

Feature-based SLAM methods demonstrate more ways to close loops. An expectation-maximization approach based on Markov localization simultaneously optimizes both robot poses and the map but does not achieve real-time performance [13]. A covariance-entropy ratio test can be used to both place nodes ("keyframes") and to validate loop closures, using the entropy of a reference loop closure to set dynamic thresholds [14]. Loop closure hypotheses can be made with a RANSAC-based algorithm using geometric and locality constraints and verified with a joint compatibility branch-and-bound algorithm [15]. To make this loop closure validation more robust, it can be postponed as long as there is considerable overlap between different local maps [16]. Alternatively, a learned classifier can be used to generate candidate loop closures with validation provided by RANSAC and geometric constraints [17].

III. Approach

A. Problem statement

We consider the process of observing odometry and lidar scans in order to map an environment for localization.

As a pose SLAM problem seeking to build a consistent map, we have to choose which robot positions to represent in our map (the nodes), how to detect loop closures and validate them, and how to optimize the resulting map/graph.

We consider the use of an arbitrary masking function to determine when a robot position should be explicitly represented in the map as a node with its corresponding lidar scan. To maintain global consistency, the system will also need to detect and validate loop closures in a way that does not depend on the choice of mask. In order to find all valid loop closures, the mask must consistently fire at least once in the area around each ground-truth junction in the map.

We define the masking function as a function that maps to a Boolean from all of the robot's observations Z until the current time T:

$$MaskF : \{Z_t\}_{t=0}^{T} \rightarrow \{0, 1\} \tag{1}$$

Any arbitrary masking function can be used with our system. We aim to show the advantage of choosing a sparse mask.

B. Method overview

The first step in creating the metric map is deciding when to add nodes. We defer this choice to the specified masking function with a few exceptions: we choose a minimum edge length ($1.5\,\mathrm{m}$) to prevent duplicate scan-matching nodes and a maximum edge length ($9\,\mathrm{m}$) after which we add odometry-only nodes. We also add one of these odometry-only nodes when a floor change is detected. These odometry-only nodes are necessary because a rigid-body transformation can only approximate the actual motion of the robot on an edge, and does not fully couple the rotational and translational components. Limiting the maximum edge length gives the map optimizer a more accurate representation to work with.

After adding a scan-matching node to the graph, we determine if it might close any loops in the map. We start by computing the *Dijkstra Projection* [18] from this new node to all other nodes in the trajectory graph. This provides us with the expected rigid-body transformation and associated covariance of the minimum error path from the new node to each existing node. We then compute the Mahalanobis

distance of the displacement between each of those pairs. If this falls below some threshold, then the two nodes have a high likelihood of being in the same area.

Depending on the masking function, there may be either a high or low number of potential nodes to check, so we sort them by the Mahalanobis distance and only proceed with a limited number (MaxMatches; 5 in our evaluation) of the most likely nodes for further testing. We use the method described in Many-to-Many Multi-Resolution Scan Matching [19], and only keep the matches that pass a minimum scan-match similarity score.

We then use our loop validator to confirm these potential loop closures before adding them to the map for optimization. Whenever we validate a loop closure, we optimize our map factor graph by performing a batch update of the nonlinear least-squares problem.

We describe these steps more precisely in Algorithm 1.

C. Dijkstra projection

The Dijkstra Projection [18] predicts the rigid transformation and covariance of the minimum error path between a source node and all other nodes in the graph, using Dijkstra's algorithm. We could use any scalar uncertainty metric over covariance, but choose the trace for simplicity:

$$\text{path}_{ij\text{opt}} = \underset{\text{path}ij}{\arg\min}\, \text{tr}(\Sigma_{\text{path}ij}) \tag{2}$$

As the shortest path is computed, we keep track of the cumulative predicted transformation and covariance, composing each successive transformation z_b onto the current total path transformation z_a.

Rigid transformations z_a and z_b can be composed like so:

$$z_{ab} = z_a \circ z_b \tag{3}$$

$$\equiv \begin{bmatrix} z_{a_x} + \cos(z_{a_\theta}) z_{b_x} - \sin(z_{a_\theta}) z_{b_y} \\ z_{a_y} + \sin(z_{a_\theta}) z_{b_x} + \cos(z_{a_\theta}) z_{b_y} \\ z_{a_\theta} + z_{b_\theta} \end{bmatrix} \tag{4}$$

And their combined covariance can be estimated with the Jacobians J_1 and J_2 of Eq. 4 (see Appendix B of Bosse et. al. for details [18]):

$$\Sigma_{z_{ab}} = J_1(z_a, z_b) \Sigma_a J_1(z_a, z_b)^T + \tag{5}$$

$$J_2(z_a, z_b) \Sigma_b J_2(z_a, z_b)^T \tag{6}$$

We compute the Mahalanobis distance of a translation with an additional parameter R (2 m) of translation that we want to allow between x_i and x_j. Although the Dijkstra projection also provides a transformation, we use the transformation from our current optimized map, $z_{ij} = {x_i}^{-1} \circ x_j$:

$$\text{Maha}_{ij} = \sqrt{z_{ij} (\Sigma_{ij\text{opt}} + \text{diag}(R^2, R^2))^{-1} z_{ij}} \tag{7}$$

We allow this extra translation R because two nodes do not need to coincide exactly to form a good closure, they only need to be close enough for their scan match transformation to complete the loop. The R we use then represents the distance apart at which we can reliably match two lidar scans.

Algorithm 1:

$N \leftarrow \{\}$, $E \leftarrow \{\}$
$N_0 \leftarrow$ CreateNode(Pose(), LidarScan())
$t_{\text{last full}} \leftarrow 0$, $t \leftarrow 1$
Loop
  repeat
    $d \leftarrow ||{x_t}^{-1}_1 \circ \text{Pose}()||$
    if $d >$ MaxEdgeLen $\vee$ FloorChanged() **then**
      $n \leftarrow$ CreateNode(Pose(), $\emptyset$)
      $e \leftarrow$ CreateEdge($N_{t\ \ 1}$, n, ${x_t}^{-1}_1 \circ$ Pose())
      $N_t \leftarrow n$, $E \leftarrow E \cup e$, $t \leftarrow t + 1$
    end
    $d_{\text{full}} \leftarrow ||{x_{t_{\text{last full}}}}^{-1}_1 \circ \text{Pose}()||$
  until $d_{\text{full}} >$ MinEdgeLen $\wedge$ MaskF(Observations);
  $n \leftarrow$ CreateNode(Pose(), LidarScan())
  $e \leftarrow$ CreateEdge($N_{t\ \ 1}$, n, ${x_t}^{-1}_1 \circ$ Pose())
  $N_t \leftarrow n$, $E \leftarrow E \cup e$
  $t_{\text{last full}} \leftarrow t$, $t \leftarrow t + 1$
  dijk $\leftarrow$ DijkstraProjection(N, E, n)
  $\text{mahas}_m \leftarrow \text{Maha}_{nm} \forall (m, z_{nm}, \Sigma_{nm}) \in$ dijk
  mahas $\leftarrow$ lowest MaxMatches entries in mahas
  for $m \in$ mahas $\mid \text{mahas}_m <$ MaxMaha **do**
    if LidarSimilarity(n, m) $>$ MinScore **then**
      $\text{matches}_m \leftarrow$ LidarSimilarity(n, m)
    end
  end
  for match $\in$ matches **do**
    AddToLoopValidator(match)
  end
  for $m \in$ NewlyValidatedLoopClosures() **do**
    $e \leftarrow$ CreateEdge($n, m,$ MatchXform($n,\ m$)))
    $E \leftarrow E \cup e$
    OptimizeFactorGraph(N, E)
  end
EndLoop

D. Masking functions (masks)

In this section, we describe several masks conforming to Eq. 1 that we use to evaluate our system.

1) Never: By always returning false, this masking function results in the worst-case baseline of dead-reckoning.

2) Always: Similar to a fully metric SLAM system, this mask creates scan-matching nodes at the highest allowable density, and attempts to continually scan match between these nodes. Because we are not combining scans into a global occupancy grid, this setup will be more similar to metric SLAM systems that also perform scan matching across individual lidar scans. This function serves mainly as a worse-case baseline for computational complexity in the number of scan matches.

3) Isovist Eccentricity: Inspired by a previous use of isovist eccentricity in a decision point detector [11], this mask calculates the eccentricity of the robot's 2D lidar scan, and applies a threshold to it. The eccentricity essentially works as a measure of the narrowness or "hallwayness" of the lidar scan. The lidar scan naturally forms an isovist by considering each scan point as a vertex connected to

its two neighbors. We compute the image moments, M_{ij}, of this isovist by triangulating the polygon and calculating the moments analytically. Then we calculate the covariance matrix and eccentricity as:

$$C = \begin{bmatrix} \frac{M_{20}}{M_{00}} - \left(\frac{M_{10}}{M_{00}}\right)^2 & \frac{M_{11}}{M_{00}} - \frac{M_{10}M_{01}}{M_{00}^2} \\ \frac{M_{11}}{M_{00}} - \frac{M_{10}M_{01}}{M_{00}^2} & \frac{M_{02}}{M_{00}} - \left(\frac{M_{01}}{M_{00}}\right)^2 \end{bmatrix} \quad (8)$$

$$e = \sqrt{1 - \frac{\lambda_2}{\lambda_1}} \quad (9)$$

where λ_1 and λ_2 are the eigenvalues of the covariance matrix. We apply a threshold with hysteresis to this eccentricity value to decide when a node should be added to the graph. Unlike the previous work, we can be much simpler because we do not need to avoid false positives.

In order to also record nodes at dead-ends in the graph, we compute a measure of "drift" to determine if the robot is at an extreme edge of the isovist. This is computed as the Mahalanobis distance from the robot to the centroid of the isovist using the covariance matrix above. We apply a simple threshold to this drift measure. The mask returns true if either the drift threshold or the eccentricity threshold passes.

4) Openings: This mask is also based on the idea of placing nodes at likely intersections or decision points. We first determine which directions are currently viable for the robot to take and filter these data over a small time horizon to help reject noise. Finally, the mask returns true if one of these directions has an angular difference from the robot's current trajectory between 40 and 90 degrees, indicating that the robot has approached a newly possible direction.

We determine these directions by first splitting a 2D lidar scan such that each continuous segment only has ranges greater than 5 meters from the robot, indicating a potential opening for the robot to travel through. We then refine the end-points because we want to find the narrowest part of each segment and determine if the robot will be able to fit. Finally we apply a minimum width threshold of $0.75\,\mathrm{m}$, the approximate width of the robot, and a minimum angle threshold of 9 degrees, to help with false positives.

To refine the end-point choices, we start by defining a center line from the robot that passes through the mean of the original end-points. Then we choose the point that minimizes the sum of the opening width and the square root of the perpendicular distance to the center line, considering only points whose projection onto that line is between 0 and 5 meters and are on the same side of that line as the original end-point. We do this separately for each end-point.

To filter and select from these openings, we sort them by opening direction and only choose openings that have enough measurements over time with a low enough standard deviation in their direction measurement.

In addition, because this function will not natively mark dead-ends as useful nodes, we add the condition that if the robot turns in excess of 150 degrees away from the vector of travel since the last node, it should also return true.

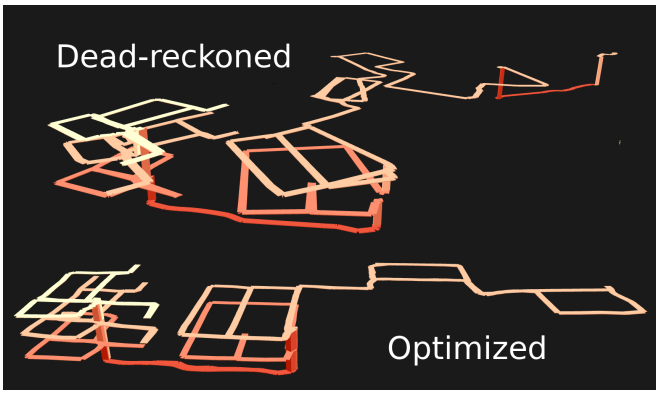

Fig. 2: Our map both before (top) and after loop closures and optimization (bottom). Our gyroscope had an uncorrected bias that increases the angular error over time unless corrected. Given the large size of some of these loops, the loop closures may have to correct as much as 30 degrees of rotation in our map at a time. The optimized map used a Mahalanobis threshold of 5.0, minimum scan-matching score of 42.5%, and the openings mask.

E. Dijkstra Projection Loop Validation

Because the masking function is a replaceable component of our system, we need a method for validating loop closures that does not make strong assumptions about the structure of nodes available for scan matching.

Our loop validation system maintains a set of loop closure hypotheses. When it finds a valid loop involving a hypothesis, it rewards that hypothesis a vote. After a threshold of votes (we use 6), that loop closure is validated.

The process of finding loops is based on the Dijkstra Projection [18], applied to finding the lowest error paths between the two nodes proposed for loop closure. In finding a path, we allow the search to use both validated and unvalidated loop closures in addition to odometry, always choosing validated over unvalidated loop closures when possible. From this minimum-error path, we form a complete loop by adding our loop closure edge. The resultant transformation should be unity. We judge the quality/error of the loop closure by the Mahalanobis distance between that final transformation and unity. If the final error is low enough (below 2), we give our new provisional loop closure and any other unvalidated loop closures on this path a single vote towards validation. Regardless of whether we accepted that prior loop, we repeat the process, first removing whichever edge in that last path was most redundant, or reachable from the greatest set of nodes in the search, indicating the potential presence of an alternative path. We continue this process until we either have enough votes to validate our new loop closure, or until a path can no longer be found to make another loop. In that case, the loop closure still has a chance to be validated later the next time we attempt to validate a new loop closure.

IV. Evaluation

To evaluate our system, we drove our robot through three buildings and four floors on the University of Michigan North

TABLE I: The number of scan matches that each masking function ends up making over the course of map creation, the number that pass the minimum score threshold, the number that are validated as loop closures, and the total time used for map creation. The always mask uses five times as many scan matches. The openings mask is more economical than eccentricity, getting more matches validated even with fewer attempts. These numbers used a minimum scan-match score of 42.5% and maximum Mahalanobis threshold of 5.0.

Masking func.	Total	Matched	Validated	Time (s)
Always	5353	4437	3833	195
Eccentricity	1172	860	627	50
Openings	974	760	637	46

Campus. The robot navigated a total of about $2.5\,\mathrm{km}$ to collect the data over 2.5 hours. The resultant map has a total path length of about $1.9\,\mathrm{km}$ and a total bounding area across four floors of about $34\,000\,\mathrm{m}^2$. The third floor has the largest individual bounding area of about 264 by 82 meters.

Our robot's fiber-optic gyro has an angular bias which we did not correct, so that the performance would be more similar to a less expensive sensor. This angular bias can be seen in the unoptimized dead-reckoned map in Fig. 2.

From these data, we construct and evaluate our masked metric maps to examine the trade-offs between the number of scan matches used, the accuracy of the final maps, and the sensitivity of our method to key parameters.

We evaluate the performance of our system by counting the number of scan-matching attempts, the number of successful matches, the number of matches that are validated as loop closures, and the computation time, shown in Table I. The ideal mask function would have low computational requirements, using only the minimum number of scan-matching attempts in order to form its loop closures. It would also only choose scan matching locations that are not prone to perceptual aliasing.

We evaluate the accuracy of the final maps by checking for important loop closures. With all the correct loops closed, every path we find in the map's graph should be as short as physically possible. If loop closures are missing, some paths will be longer. If incorrect loop closures are present, some paths will be shorter. To measure this, we calculate the lengths of paths in a randomly selected set of 999 starting and ending position pairs. We use the same set for all cases, and define the positions by their timestamp in the log file. This lets our comparison be fair because while each map may choose to place nodes in different places, the timestamps can still uniquely define locations. For each generated map, we interpolate the positions of the timestamps based on the two nodes in the map that precede and follow chronologically, and we calculate the shortest paths between these interpolated positions. We manually verify several maps to have no false-positive loop closures, although they may be missing loop closures, and take the minimum length of each path across them to use as ground truth. From these ground-truth lengths, we can calculate the mean path error of any map.

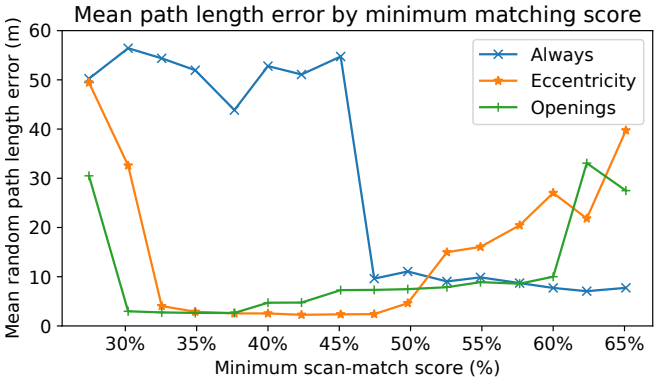

Fig. 3: Dependence of each masking function on the minimum matching score. A score of 100% would mean that the scans perfectly overlap. An error of close to $0\,\mathrm{m}$ indicates that all the loop closures are valid. Starting with more attempts, the always mask benefits from stricter thresholds to help manage false-positives, whereas the openings and eccentricity masks benefit from low match scores because the nodes are already less prone to perceptual aliasing. The Mahalanobis threshold is 5.0 for these runs.

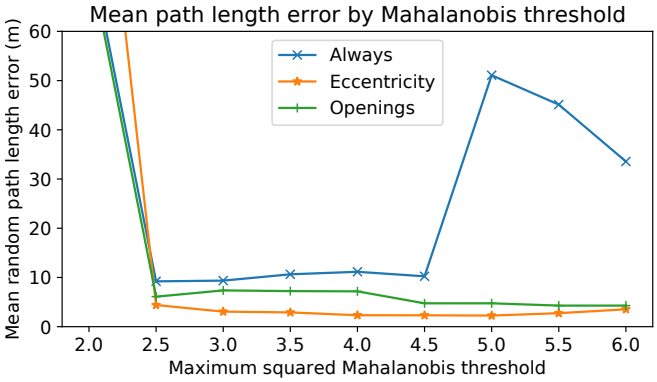

Fig. 4: Effects of the threshold that determines which nodes are close enough to scan match against for potential loop closures. An error of close to $0\,\mathrm{m}$ indicates that all the loop closures are valid. While the openings mask benefits from lenient thresholds allowing more matching with farther-away nodes, the always mask needs a smaller threshold to help limit perceptual aliasing. The minimum scan-matching score is 42.5% for these runs.

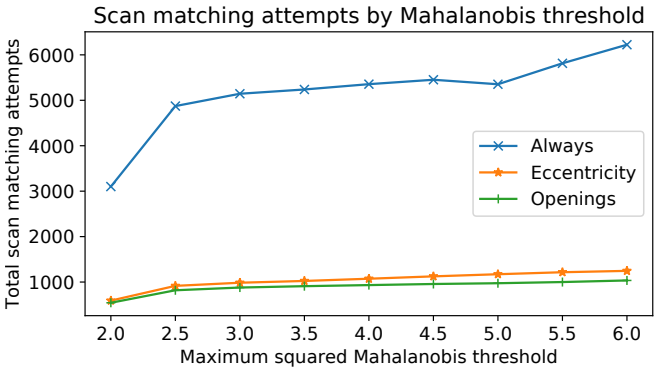

Fig. 5: Scan matching attempts increase with a larger Mahalanobis threshold. In our system, scan matching uses more computational processing than anything else, making it a proxy for the total computational resources needed. Furthermore, every attempt poses a risk for aliased nodes to incorrectly form a loop closure. Regardless of loop closing system characteristics, additional attempts can only increase the probability of false-positives. Our masks reduce the number of scan matches by five times.

V. Discussion

By manual inspection we were able to find that both the openings and eccentricity masks were able to generate maps without apparently missing or incorrect loop closures. An example of this is the map shown on the first page in figure 1. On a standard laptop with an i7-6820HQ processor, this map took 46 seconds to generate from a 2.5 hour log file using the openings mask function, far exceeding real-time.

Based on the path errors shown in figures 3 and 4, we see that the always mask often performs worse than the others. It was generally the case that the always mask would create false-positive loop closures due to perceptual aliasing between parts of the same hallway, and if the parameters were tuned to be more restrictive, it would fail to close a final loop at the end of the trajectory when the odometry's angular error is high. These conflicting situations made the always mask hard to tune.

Increasing the scan-matching score threshold reduces the availability of loop closure candidates to all the masks. In figure 3, we see that because masking reduces this availability even more, only the always mask benefits from this reduction, whereas the other masks benefit from using these lower-score matches. In figure 4, we see that good performance can be reached as long as the Mahalanobis threshold permits attempting the right loop closures. Because the system always prefers to match against nodes with lower Mahalanobis distances, it is relatively unaffected by larger thresholds, although they lead to an increase in the number of total scan matches, as we see in figure 5. For reference against these figures, the never/dead-reckoned map has an average path length error of $625\,\mathrm{m}$, because with no loops closed the map is treated as a one-dimensional chain.

Overall, we find that for our difficult map the always mask has the worst loop-closing performance while also being five times as computationally expensive. The openings and eccentricity masks both generate maps with low loop closure error, and that we can not visually distinguish from the the map presented in figure 1.

VI. Conclusion

Continually scan matching comes at a cost, requiring more computational resources and increasing the likelihood of failure through perceptual aliasing. In this paper we presented a flexible mapping scheme for creating sparse metric maps with a variable masking function that selects locations of interest for closing loops. Precise localization is rarely needed when driving down a hallway. This system is computationally efficient because it minimizes scan matching and optimizes a sparser map. We showed that our system can create high quality maps even in large environments with long loops and poor odometry. We compared two simple masking functions based on openings and isovist eccentricity with an "always" function that emulates a traditional dense SLAM system, and found that our masking functions produced maps with more true-positive and fewer false-positive loop closures, using only one-fifth as many scan matches.

A. Acknowledgements

This work was supported by grants from the NSF (1830615) and ARC (W56HZV-19-2-0001). Distribution A. Approved for public release; distribution unlimited. (OPSEC 3923).

Disclosure: Edwin Olson has a financial interest in a company that may have rights to foreground or background technology described in this paper.

References

[1] X. Wang, R. Marcotte, G. Ferrer, and E. Olson, "AprilSAM: real-time smoothing and mapping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2486–2493.

[2] B. Kuipers, R. Browning, B. Gribble, M. Hewett, and E. Remolina, "The spatial semantic hierarchy," *Artificial intelligence*, 2000.

[3] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[4] S. Thrun, D. Hahnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 3. IEEE, 2003, pp. 4270–4275.

[5] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2d lidar slam," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 1271–1278.

[6] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[7] T. Röfer, "Route navigation using motion analysis," in *International Conference on Spatial Information Theory*. Springer, 1999, pp. 21–36.

[8] A. Lankenau, T. Röfer, and B. Krieg-Brückner, "Self-localization in large-scale environments for the bremen autonomous wheelchair," in *International Conference on Spatial Cognition*. Springer, 2002, pp. 34–61.

[9] B. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli, "Local metrical and global topological maps in the hybrid spatial semantic hierarchy," in *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5. IEEE, 2004, pp. 4845–4851.

[10] P. Beeson, J. Modayil, and B. Kuipers, "Factoring the mapping problem: Mobile robot map-building in the hybrid spatial semantic hierarchy," *The International Journal of Robotics Research*, vol. 29, no. 4, pp. 428–459, 2010.

[11] C. Johnson, "Topological mapping and navigation in real-world environments," Ph.D. dissertation, University of Michigan, 2018.

[12] J.-L. Blanco, J.-A. Fernández-Madrigal, and J. Gonzalez, "Toward a unified bayesian approach to hybrid metric–topological SLAM," *IEEE Transactions on Robotics*, vol. 24, no. 2, pp. 259–270, 2008.

[13] S. Thrun, W. Burgard, and D. Fox, "A probabilistic approach to concurrent mapping and localization for mobile robots," *Autonomous Robots*, vol. 5, no. 3-4, pp. 253–271, 1998.

[14] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 2100–2106.

[15] J. Neira, J. D. Tardós, and J. A. Castellanos, "Linear time vehicle relocation in slam," in *ICRA*. Citeseer, 2003, pp. 427–433.

[16] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[17] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "Segmatch: Segment based place recognition in 3d point clouds," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5266–5272.

[18] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[19] E. Olson, "M3RSM: Many-to-many multi-resolution scan matching," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.