


Optimal pattern-based near-field path planning for mobile robots, under consideration of obstacles in intralogistics environments.

Ilef Mhgirbi ^{a,*}, Tino Krueger-Basjmeleh^b

^aStudent of Robotics and Autonomous Systems, Universität zu Lübeck, Lübeck, Germany

^bSTILL GmbH, Hamburg, Germany

*Corresponding author, email: lukas.freiling@student.uni-luebeck.de; musterprofessor@wisdom.uni-luebeck.de

© 2025 Freiling *et al.*;

This is an Open Access article distributed under the terms of the Creative Commons Attribution License (<http://creativecommons.org/licenses/by/4.0>), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract

Brownfield environments in intralogistics are highly dynamic, complex and tight areas, requiring mobile robots to navigate aside humans while being flexible, efficient and safe. These demands are most challenging for path planning algorithms, addressing pickup and drop zones requiring higher accuracy, integrating the sensorially perceived environment, addressing the robot's kinematic constraints, and optimizing the path for minimum travel time. This paper presents a combinatorial optimization approach for path planning of autonomous mobile robots in these intralogistics environments. The objective is to generate an efficient but also explainable path that minimizes travel distance and curvature changes while ensuring obstacle avoidance and adherence to kinematic constraints. The proposed methodology formulates path planning as a real-time pattern-based optimization problem, designed for crowded and small intralogistics areas where bulky vehicles frequently change direction. The path planning problem is modeled through an objective function that optimizes both travel efficiency and maneuverability, integrating environmental recognition data to guide the optimizer. In order to address this challenge, bio-inspired metaheuristic methods such as Particle Swarm Optimization (PSO), Differential Evolution (DE), and Genetic Algorithms (GA) are employed. These algorithms explore diverse path solutions and progressively refine performance. A comparative study evaluates their computational efficiency and solution quality. The proposed approach is integrated into the Robotics Application Construction Kit (RACK) and tested in a controlled environment to assess real-time feasibility. By reducing reliance on extensive commissioning data, this method enhances adaptability across logistics scenarios. The study contributes to combinatorial optimization by improving path feasibility, computational efficiency, and adaptability, making mobile robots more effective in dynamic and complex intralogistics operations.

I. Introduction

Perception is the main task for autonomous robots to respond to their environment. For that reason, raw sensor data gets abstracted into refined features. Line segments can help to perceive the environment better. They can describe borders or other structures in the

surrounding. Therefore they are commonly used in mobile robots [1].

Line detection in machine and robot vision is a frequently studied problem. One popular technique is the Hough Transform. Darlitz *et al.* [2] showed an iterative Hough Transform where lines are subsequently found. Fernandes *et al.* stated that "the computational

cost ... prevented software implementations to achieve real-time performance" [3]. For that reason, the Hough Transform was not investigated in this paper.

Nguyen et al. [4] compared six algorithms in mobile robot scenarios. They showed that the Incremental and the Split and Merge algorithm had the best performance, while the Split and Merge was the fastest algorithm. These algorithms were only tested with 2D LiDAR data and not with unconstrained data from a point cloud.

Also presented by Nguyen et al. was the Random Sample Consensus (RANSAC). While they showed that the runtime performance was not as good as other algorithms on 2D LiDAR data, other researcher could show that the application in point cloud is in general good.

Reconstruction of line segments from large 3D point clouds was studied by Xin et al. [5]. They proposed a method using a nearest neighbor tree called Line Segment Extractor. It was shown that they could precisely estimate line segments. For their evaluation they used an high performance computer and were not restricted for real-time.

II. Methods

In this section, three different line segmentation techniques are presented. To estimate line parameters from a set of points the Principal Component Analysis (PCA) is used [6]. The PCA is a statistical technique used for dimensionality reduction by transforming data into a new coordinate system where the greatest variance by any projection of the data lies along the first component. This corresponds to a regression of data points as a line.

With the principal component (PC) found, only the line segment has to be determined. This can be achieved by projecting each point on the line and searching for the minimal and maximal value.

Split and Merge

Nguyen et al. showed a Split and Merge algorithm adapted to segment lines in 2D LiDAR scans. The input data has the constraint that consecutive data points have to be angular neighbours in a scan. The Split and Merge algorithm consists, as the name suggests, of two different parts. In the first part, the point set is getting split into smaller pieces that contributes to a line. This is achieved by fitting a line to the whole set of points and calculating the distance for each point to that line. If a point is further away as a certain threshold $t_{d_{max}}$ allows, the point cloud is split in two at that point. Then this procedure is repeated until there are only small pieces of the point cloud in which each point should sit on the corresponding line segment. To

each of these pieces, a line segment is fitted.

In the second part of this algorithm the calculated line segments get merged if they are colinear. To measure whether two line segments are colinear, the minimal distance between the endpoints of those lines and the angular difference is calculated. If the threshold t_{merge} for those conditions is not exceeded, the line segments can be merged by simply recomputing the parameters.

Algorithm 2 Split and Merge

```

1: function Split( $L, P$ ):
2:    $l = \text{FitLine}(P)$ 
3:    $p_{max}, d_{max} \leftarrow l.\text{findPointMaxDist}(P)$ 
4:   if  $d_{max} > t_{d_{max}}$  then
5:      $P_1, P_2 \leftarrow \text{SplitAt}(P, p_{max})$ 
6:     Split( $L, P_1$ )
7:     Split( $L, P_2$ )
8:   else
9:      $L.append(l)$ 
10:
11:  $L \leftarrow \emptyset$ 
12: Split( $L, P$ )
13:  $L \leftarrow \text{MergeIfColinear}(L, t_{merge})$ 
```

RANSAC

The Random Sample Consensus (RANSAC) [4] is a stochastic algorithm that tries to randomly sample a line from a point cloud. It works by selecting two points uniformly distributed from the set and fitting a line to it. Then the so-called inlier set is calculated. That means that the distance for each point to that line is calculated and the points that are below a threshold $t_{d_{max}}$ are counted as inliers. For this inlier set, the line gets recomputed and the inlier points are getting removed from the point cloud. This can be done until a maximum number of iterations is reached or there are not enough points in the point cloud.

Line Segment Extractor

The Line Segment Extractor is a algorithm which was first published by Xin et al. [5] in 2024. The main idea is to use local information to find line segments instead of searching in the whole data.

Therefor a k-nearest neighbor tree is first created. It is a data structure which can return the k (arbitrary number) nearest points for one specific point. For each point a PC is calculated in its neighborhood. Afterwards, a breadth first search (BFS) is started to find line segments. Therefore a random point is chosen as a start point for the BFS. To ensure that a parent node and it's children face the same direction, the absolute value of the scalar product of their principal components has to be bigger than a certain threshold

Algorithm 4 RANSAC

```

1: function RANSAC(L, P):
2:   if Max Iteration or #P < threshold then
3:     return
4:   inlier ← ∅
5:   p1, p2 ← X ~ U(0, #P - 1)
6:   l = FitLine(p1, p2)
7:   for p ∈ P do
8:     distp = l.GetDistance(p)
9:     if distp < threshold then
10:      inlier.append(p)
11:   if #inlier > threshold then
12:     L.append(l)
13:     P.remove(inlier)
14:     RANSAC(P)
15:   else
16:     RANSAC(P)

```

$t_{LSE} \in [0, 1]$. The paper suggests 0.85 as a reasonable value. The line parameters will be recomputed for the resulting subset. It is then removed from the tree.

Algorithm 6 Line Segment Extrator

```

1: function LSE(L, P):
2:   if Max Iteration or #P < threshold then
3:     return
4:   inlier ← ∅
5:   tree = buildKNearestNeighbourTree(P)
6:   for Node ∈ BFS(tree).start(rand) do
7:     if PCNode · PCNode.child > tLSE then
8:       BFS.queue(Node.child)
9:       inlier.append(Node.point)
10:  P.remove(inlier)
11:  L.append(FitLine(inlier))
12:  LSE(L, P)

```

III. Results and Discussion

To validate the performance of the line segmentation algorithms a series of test cases were generated to analyze specific properties of the algorithms. Therefore a selection of line segments were defined with start and end points for each test case. Afterwards, the line segments were abstracted into point sets, incorporating noise and different resolution. Of these test cases, four are presented here to illustrate the advantages and disadvantages of the methods. To measure the quality of the line segmentation algorithms two metrics were chosen which are explained in the following.

Structural distance

One way to measure the similarity between two lines (l_i, l_j) was presented by Ivanov et al. [7]. The approach shows two different ways of measure distance between two lines. One is called the structural distance. It calculate the minimal distance sum between the start and end points of both lines. A visualization is shown in figure ?? and the formula in (1).

$$d_s(l_i, l_j) = \min\{\|p_j^1 - p_i^1\|_2 + \|p_j^2 - p_i^2\|_2, \|p_j^1 - p_i^2\|_2 + \|p_j^2 - p_i^1\|_2\} \quad (1)$$

Using the minimum of either the start-to-start points or start-to-end points distances solves the problem of the unknown orientation of two lines.

Orthogonal distance

The second mentioned method is the orthogonal distance. It calculates the distance to the projected start and end points of a line to another. A visualization is shown in figure ?? and the formula in (2).

$$d_1(l_i, l_j) = \frac{d_a(l_i, l_j) + d_a(l_j, l_i)}{2} \quad (2)$$

$$d_a(l_i, l_j) = \|p_j^1 - \pi_{l_i}(p_j^1)\|_2 + \|p_j^2 - \pi_{l_i}(p_j^2)\|_2 \quad (3)$$

Here π_{l_i} is defined as the projection of a point p_j onto the line l_i . As the name suggests, this distance only takes into account the orthogonal displacement but not the tangential as the structural distance does. Although these two metrics can not give a single score for a line, the interpretation of the meaning of them can be very helpful.

Evaluation

The structural distance of each method for the test cases can be seen in figure ?. The Line Segment Extractor showed the lowest structural distance in all test cases. Further Split and Merge and RANSAC algorithm could not correctly segment the lines in the scenario "Interrupted Line". The Split and Merge algorithm was also not able to succeed the "Parallel Lines" test case, because the previous state requirement for the algorithm was not fulfilled.

The results using the orthogonal distance can be seen in figure ?. Here, the Line Segment Extractor has the best precession. While Split and Merge and RANSAC could not segment all line elements in the test case "Interrupted Line", they still managed to create a line that has a low orthogonal distance below 0.2 pixel.

Runtime was first measured with Python to show the relative behavior of the algorithms in each test case.

The results of the runtime test can be seen in figure ???. Except for the "Range-like" test case, RANSAC was the fastest algorithm. On the other hand the Line Segment Extractor was the slowest, except for the "Parallel Lines" test case, the Split and Merge algorithm took the most time. Both showed a runtime above 130 ms.

Using this tests, it was decided to implement the RANSAC and Line Segment Extractor in C++. The runtime of 1800 data points was between 12 to 30 ms for the RANSAC and around 40 ms for the Line Segment Extractor. This is fast enough to argue that it is feasible to run the algorithms in real-time.

IV. Conclusion

The Split and Merge algorithm showed very different results depending on the test case. It tended to long runtime for unconstrained input data. The precision was also not as good as the Line Segment Extractor, and compared to the RANSAC it had no big benefit since it succeeded less test cases.

Both the RANSAC and the Line Segment Extractor proved to be applicable in real time, but they have a trade off. The RANSAC was a little faster with 15 ms less in average, with the tendency to approximate lines and omit fine details. On the other side the Line Segment Extractor is a bit slower, but can detailed estimate fine line segments in the cloud. The best line segmentation algorithm is therefore depending on the specific use case.

Acknowledgments

The work has been carried out at the STILL GmbH, Hamburg and supervised by Prof. Dr. Georg Schild-

bach, Institute for Electrical Engineering in Medicine, Universität zu Lübeck.

Author's statement

Conflict of interest: Authors state no conflict of interest.

References

- [1] L. Zhang and B. Ghosh, Line segment based map building and localization using 2d laser rangefinder, 3, 2538–2543, IEEE, 2000. doi:[10.1109/robot.2000.846410](https://doi.org/10.1109/robot.2000.846410).
- [2] C. Dalitz, T. Schramke, and M. Jeltsch. Iterative hough transform for line detection in 3d point clouds. *Image Processing On Line*, 7:184–196, 2017, doi:[10.5201/ipol.2017.208](https://doi.org/10.5201/ipol.2017.208).
- [3] L. A. Fernandes and M. M. Oliveira. Real-time line detection through an improved hough transform voting scheme. *Pattern Recognition*, 41(1):299–314, 2008, doi:<https://doi.org/10.1016/j.patcog.2007.04.003>.
- [4] V. Nguyen, A. Martinelli, N. Tomatis, and R. Siegwart, A comparison of line extraction algorithms using 2d laser rangefinder for indoor mobile robotics, 1929–1934, 2005. doi:[10.1109/IROS.2005.1545234](https://doi.org/10.1109/IROS.2005.1545234).
- [5] X. Xin, W. Huang, S. Zhong, M. Zhang, Z. Liu, and Z. Xie. Accurate and complete line segment extraction for large-scale point clouds. *International Journal of Applied Earth Observation and Geoinformation*, 128, 2024, doi:<https://doi.org/10.1016/j.jag.2024.103728>.
- [6] J. Shlens, A tutorial on principal component analysis, 2014. arXiv: [1404.1100](https://arxiv.org/abs/1404.1100) [cs.LG]. url: <https://arxiv.org/abs/1404.1100>.
- [7] K. Ivanov, G. Ferrer, and A. Kornilova, Evolin benchmark: Evaluation of line detection and association, 2023. arXiv: [2303.05162](https://arxiv.org/abs/2303.05162) [cs.CV]. url: <https://arxiv.org/abs/2303.05162>.