

Error-Aware Path Planning for DXF-Based Robotic Layout Marking

by

Amantay Kapyshev

Submitted to the Department of Robotics and Mechatronics
in partial fulfillment of the requirements for the degree of

Master of Science in Robotics

at the

NAZARBAYEV UNIVERSITY

March 2026

© Nazarbayev University 2026. All rights reserved.

Author
Department of Robotics and Mechatronics
March 20, 2026

Certified by
Almas Shintemirov
Associate Professor
Thesis Supervisor

Accepted by
Elizabeth Arkhangelsky
Dean, School of Engineering and Digital Sciences

Error-Aware Path Planning for DXF-Based Robotic Layout Marking

by

Amantay Kapyshev

Submitted to the Department of Robotics and Mechatronics
on March 20, 2026, in partial fulfillment of the
requirements for the degree of
Master of Science in Robotics

Abstract

Mobile construction layout marking robots based on DXF floor plans are vulnerable to cumulative odometry drift resulting in distorted shapes being drawn. The existing systems address this drift through periodic pauses to correct this drift, usually through SLAM, but without regard to the location of the worst drift. This thesis presents a process that will anticipate the point where the robot will experience most error during its drawing and plan the SLAM corrections at those points.

The system comprises three elements: a covariance propagation engine which calculates the expected error in each point in a planned path, a greedy technique that adds a small amount of corrections where they have the most important effect, and an order planner for the drawings which connects, in opposite order, the segments so that drift between corrections may be partially cancelled.

Results of simulation of four floor plans indicate that mean drawing error is 24-35% lower when corrections are placed at predicted error peaks than when they are spaced uniformly with the same number of corrections. When periodic corrections are added with the drawing order planner, it makes permanent improvement which is 5-12%. The absolute errors are still large in comparison to construction-grade tolerance and to reach construction-grade tolerance, the number of corrections or the number of other hardware is much larger; the contribution is not to eliminate the need, but to make each correction work more efficiently. The prediction model does not calculate the error profile iteratively, but directly, and therefore is sufficiently fast to explore a large number of correction positions in the scheduling process.

Physical validation on a Clearpath Jackal UGV with 30 trials confirmed the predicted ranking: optimal correction placement achieved 3.7 cm mean endpoint error, 11.2% lower than uniform spacing (4.2 cm) and 31.5% lower than no correction (5.4 cm).

Thesis Supervisor: Almas Shintemirov
Title: Associate Professor

Contents

1	Introduction	10
1.1	Background	10
1.2	Problem Statement	11
1.3	Research Gap	11
1.4	Research Questions and Hypotheses	12
1.5	Contributions	13
2	Literature Review	14
2.1	Construction Robotics and Layout Marking	14
2.2	Odometry Error Modeling	15
2.3	SLAM and Localization Correction	17
2.4	Path Planning and Uncertainty	18
2.5	Research Gap	20
3	Methodology	22
3.1	System Overview	22
3.2	DXF Parsing	23
3.3	Drawing Order Planning	26
3.3.1	Shortest-Distance Planner	26
3.3.2	Weighted Nearest-Neighbor Planner	26
3.4	Covariance Propagation Engine	27
3.5	Optimal SLAM Correction Scheduling	28
3.6	Test Floor Plans	29

3.7	Experimental Setup	30
4	Results and Analysis	31
4.1	Overview	31
4.2	Validation of the Prediction Model	31
4.3	Optimal SLAM Correction Scheduling	33
4.3.1	Uniform vs. Optimal Placement	33
4.3.2	Effect of Correction Budget	35
4.3.3	Error Profile Along the Path	36
4.3.4	Spatial Error Distribution	37
4.3.5	Covariance Ellipse Visualization	38
4.4	Effect of Drawing Order Optimization	38
4.5	Combined System Performance	41
4.6	Greedy vs. Exhaustive Correction Placement	42
4.7	Sensitivity to Odometry Parameter Mismatch	42
4.8	Discussion of Results	43
5	Physical System Implementation and Validation	45
5.1	Hardware Platform	45
5.2	Software Architecture	47
5.3	DXF-to-Robot Pipeline	47
5.3.1	DXF Parsing and Segment Extraction	48
5.3.2	Drawing Order Planning	48
5.3.3	Coordinate Transformation	48
5.3.4	SLAM Correction Scheduling	49
5.4	Navigation and Control	49
5.4.1	Path Following Controller	49
5.4.2	SLAM Correction Mechanism	50
5.4.3	State Machine	51
5.5	Test Environment and Floor Plan	51
5.6	Experimental Protocol	52

5.6.1	Conditions	52
5.6.2	Trial Procedure	52
5.7	Error Metrics	53
5.8	Results	54
5.8.1	Overall Accuracy	54
5.8.2	Statistical Significance	55
5.8.3	Per-Segment Error Analysis	55
5.8.4	Path Following Quality	56
5.8.5	Variability Across Runs	57
5.9	Comparison with Simulation	58
5.10	Discussion and Limitations	59
5.11	Reproducibility	60
6	Discussion	62
6.1	Key Findings	62
6.2	Why the System Works	63
6.3	Physical Validation	63
6.4	When It Works Best	64
6.5	Practical Implications	64
6.6	Limitations	66
6.7	Future Work	68
7	Conclusion	70
7.1	Summary	70
7.2	Contributions	70
7.3	Key Results	71
7.4	Key Insight	71
7.5	Closing Statement	72

List of Figures

3-1	System architecture. The offline planning phase uses the floor plan of the DXF file and the parameters of the odometry as input to calculate the drawing order, propagate the error covariance on the planned path and schedule SLAM corrections at the predicted peaks of errors. The online execution phase guides the robot through execution of the plan with the introduction of LiDAR-based relocalisation at waypoints flagged out excited to the plan.	23
3-2	DXF floor plan as viewed in a CAD environment.	24
3-3	Terminal output of the DXF parser showing extracted <code>LINE</code> entities with their layer, length, and start and end coordinates.	25
3-4	Planned path visualised in RViz prior to execution, showing the parsed segments passed to the drawing order planner.	26
3-5	The four test floor plans in the simulation experiments Each plan is drawn as it would be viewed in the DXF file with the starting position of the robot identified by a star. Plans from a simple to a 32 segment room to an apartment with varied segment lengths.	29
4-1	Model validation. Left: Endpoint error vs. path distance, straight paths - analytical prediction (green) follows the Monte Carlo mean (blue squares) with a similar scaling ratio. Right: error at each of the waypoints of a rectangle path. Both the geometries confirm the accuracy in prediction.	32

4-2	Optimal placement of the SLAMs allows a reduction in mean drawing error by 24-35% as compared to uniform spacing using the same number of corrections K=3. Percentage labels denote improvement on each floor plan.	33
4-3	Apartment plan error vs. correction budget. The area in the shade is the accuracy that is gained due to best placement, maximum at limited budgets K=2-5).	35
4-4	Apartment plan- predicted error along the drawing path. Optimal correction points of Green diamonds coincide with peak errors; Uniform points of Blue circles sometimes appear at the low error position, hence, correction opportunities are not utilized efficiently.	36
4-5	Predicted error colored office floor plan (consistent color scale). Left: no correction. Center: uniform. Right: optimal. Diamond markers are located at the locations of SLAM corrections.	37
4-6	Covariance ellipses at endpoints of the segments (every 3rd shown, same scale). Left, without correction - uncertainty extends outside the footprint of the building. Right: with 3 optimal corrections(diamonds) -uncertainty is bounded.	38
4-7	Comparing Planners at three SLAM levels Red stars designate the best planner. Without SLAM shortest distance wins. With SLAM, weighted planner is winning on bigger plans.	38
4-8	Drawing order comparison office plan of. The weighted planner generates more opposing pairs with similar total travel distance.	40
4-9	Apartment plan - cumulative improvement. SLAM corrections cause the biggest reduction (57%), the weighted planner and optimal placement achieve complementary improvement. Where the lines in red are the reduction in each step.	41
5-1	Clearpath Jackal J100 Robot	46

5-2	Mean endpoint error by condition with standard deviation error bars. Optimal $K=2$ achieves 11.2% lower error than uniform spacing using the same number of corrections.	55
5-3	Endpoint error by segment for each condition (mean $\pm$ std, $n = 10$). The no correction condition shows the highest errors at the two 3.5 m walls (segments 0 and 2), where odometry drift accumulates most. . .	56
5-4	Robot trails from all 10 runs overlaid for each condition. Green lines show the DXF target plan; colored trails show actual robot paths. Red arrows indicate endpoint errors for the median run.	57
5-5	Endpoint error and path deviation per condition. Bars show means, dots show individual runs, error bars show ± 1 standard deviation. The optimal condition shows the tightest clustering of individual runs. . .	58

List of Tables

2.1	Summary of related approaches and their limitations with respect to error-aware correction scheduling for robotic layout marking.	20
4.1	Uniform vs. optimal SLAM placement (3 corrections).	33
4.2	Planner comparison with SLAM corrections (cm).	39
5.1	Clearpath Jackal J100 specifications.	46
5.2	Physical experiment results (10 runs per condition, $K = 2$ corrections). Values are mean $\pm$ standard deviation, with 95% confidence intervals in brackets.	54

Chapter 1

Introduction

1.1 Background

Construction layout marking is the process of converting architectural designs from digital floor plans to the physical structures of the building. Traditionally done by hand with tape measures, chalk lines and reference points, this process is time consuming, labor intensive and susceptible to human error. As building designs become more and more complex and require more precise layout marking, there has been a demand for faster, more accurate layout marking, which has led to the development of robotic systems that can read digital floor plans and draw layout lines on the construction floor autonomously [1–3].

Several commercial and research systems have shown the feasibility of robotic marking layout. The HP SitePrint [4] is an inkjet technology that prints floor plans directly on to the concrete. Research prototypes have been based on laser projection [5], spray marking [6] and wheeled marker systems [7]. These systems have something in common: the mobile robot reads a DXF or BIM file that defines the lines that need to be drawn, plans a path that will allow it to visit all the line segments, and executes the drawing task while localizing in the construction floor.

1.2 Problem Statement

The major challenge of layout marking with robots is accumulated odometry drift. As the robot moves between and along drawing segments, small errors in the measurements of the wheel encoder are accumulated as the robot travels. A robot commencing its drawing at the right position may be several centimeters off by the time it reaches the far side of the floor plan. The drift itself is random but the extent to which the error accumulates on average depends on the distance travelled, the number and sharpness of turns and the odometry characteristics of the robot.

Current systems compensate for this drift in two ways: by the use of high precision localization hardware that helps to add cost and setup time or by the periodic halt to perform Simultaneous Localization and Mapping (SLAM) corrections using onboard sensors. The SLAM approach requires no external infrastructure but imposes a trade-off because more frequent corrections will result in better accuracy, but will also add to the total job time due to the pauses.

Critically, existing systems apply corrections from SLAM at fixed intervals - every N segments or every D meters - with no thought as to where corrections would be most effective. This uniform spacing does not consider the fact that error does not increase along the path evenly: long segments will end up with more error than short segments will, turns will add heading error, which will have an effect on every move that is to be made.

1.3 Research Gap

Since the drawing path is known ahead of time from the DXF file, it is possible to give a prediction on where the odometry error will be maximum based on the path geometry and the already calibrated odometry parameters of the robot. Despite this, to the best of the author's knowledge, no existing system uses the prediction to make a decision regarding where SLAM corrections should be placed.

The literature on SLAM-aware path planning [8,9] is focused on exploration tasks

in which the robot has no knowledge of its future path. Active SLAM approaches select trajectories that increase the quality of the map or decrease the localization uncertainty as the robot is exploring. A layout marking robot by contrast knows its entire trajectory before it starts moving. This opens the door for the offline optimization the exploration-based approaches cannot access.

1.4 Research Questions and Hypotheses

This thesis investigates the following three research questions:

- RQ1** Can odometry error along a known DXF drawing path be predicted analytically with sufficient accuracy to guide correction scheduling?
- RQ2** Does placing SLAM corrections at predicted error peaks reduce mean drawing error compared to uniform spacing with the same correction budget?
- RQ3** Does optimizing drawing segment order for drift cancellation provide additional accuracy improvement beyond optimal correction placement?

The corresponding hypotheses are stated as follows. For RQ1, the analytical covariance propagation engine is expected to match Monte Carlo simulation to within 5% on all tested floor plans. For RQ2, optimal placement is expected to reduce mean drawing error by at least 20% over uniform spacing across varied floor plan geometries. For RQ3, the weighted nearest-neighbor planner is expected to offer further improvement when combined with SLAM corrections, but to be detrimental when used without them.

Evaluation criteria. Four measures are used to evaluate results: mean endpoint error in centimetres, defined as the distance between the target and actual robot position at each segment endpoint; path deviation in centimetres, defined as the mean perpendicular distance from the robot trail to the nearest DXF segment; percentage improvement over the uniform spacing baseline; and statistical significance at $\alpha = 0.05$ using independent-samples t -tests.

1.5 Contributions

This thesis makes the following contributions:

- Drawing error along known path: A prediction model for drawing error along known paths. Given the planned trajectory of the robot and the calibrated motion parameters of the robot, the model calculates the expected positioning error at each point.
- A correction scheduling algorithm that uses the predicted error profile to schedule SLAM corrections at the positions where they result in the most reduction of error. With the same number of corrections as uniform spacing, this achieves 24 - 35% lower drawing error.
- A drawing order planner that provides ordering of segments in opposite directions to partially cancel drift between corrections that when combined with periodic SLAM corrections gives 5-12% additional improvement.
- A combined system that shows that the scheduling of error prediction and correction work together for improved accuracy without any additional hardware or sensors.
- Physical validation on a Clearpath Jackal robot demonstrating that the predicted optimal-vs-uniform ranking holds on real hardware, with 11.2% improvement across 30 trials.

It needs to be noted that even with the best correction placement, the absolute drawing errors are well above the construction-grade tolerance (± 5 mm) when using only a small number of corrections. The contribution of this work is not achieving construction-grade accuracy directly, but reducing the number of corrections to make achieve any given accuracy target - making the SLAM correction budget more efficient.

Chapter 2

Literature Review

2.1 Construction Robotics and Layout Marking

Robotic automation in construction has made significant progress in the past several years and is the consequence of labor shortages, safety concerns and an increased demand for higher precision [1, 2]. Among construction tasks, layout marking, i.e., transferring digital floor plans to the physical surface, is especially suitable for automation as the target geometry is completely specified in advance in the form of a DXF or BIM file. Unlike other manufacturing tasks, like bricklaying or welding, that involve force control and handling of materials, layout marking is truly a problem of position and drawing.

Commercial systems have been being developed that have begun to prove the viability of autonomous marking. The HP SitePrint [4] is based on inkjet technology which is used to print floor plans directly onto the concrete with a millimeter-level accuracy using an integrated total station. Specifically, the HP SitePrint reports a positioning accuracy of ± 2 mm, achievable because it relies on a continuously tracked external total station rather than onboard sensors alone. Research systems using onboard sensors report typical errors of 5–20 mm under controlled laboratory conditions, with errors growing significantly over longer drawing distances in the absence of periodic correction. Research prototypes have been done in different ways using laser projection [5], spraying [6] and wheeled pen-based systems [7]. Zeng et al.

[3] in a comprehensive review of automated layout marking, identify the localization accuracy as the main technical challenge of all the approaches.

These systems have a common architecture in that the wheeled mobile robot reads a DXF file to decide which line segments to draw, plans a path to visit each of them, and draws them while tracking its position. Difference between them is mainly in the marking mechanism and localization strategy. Systems with external positioning hardware are highly accurate but take a lot of setup time and are expensive in terms of infrastructure. Systems based on onboard sensors (LiDAR, cameras) are cheaper and faster to deploy but have to deal with the cumulative error in the position during the course of the drawing task.

A parallel body of work deals with BIM to robot integration: how to extract the relevant information of a task from the building information models and transform it into commands that can be executed by robots. Kim et al. [10] and Zhao and Cheah [11] provide a review of the state of the art in BIM-integrated robotic construction, while Braga et al. [12], Zhu et al. [13], Hamieh et al. [14], and Mantha et al. [15] provide solutions to certain aspects of the BIM-to-robot pipeline. However, these works are focused on what the robot should do, not how accurately it can do it. The accuracy of the execution depends on localization of the robot that is beyond the scope of BIM integration research.

Layout marking robots do exist and work, but are limited in accuracy by positioning drift. Systems using external positioning hardware are accurate but expensive - systems using onboard sensors are affordable but drift over long drawing tasks. The literature does not discuss how to optimally correct the strategy for onboard-sensor systems.

2.2 Odometry Error Modeling

Understanding the accumulation of the positioning error is so important in understanding where a drawing robot will drift the most. The standard model of the noise in wheeled robot odometry is the velocity motion model presented by Thrun et al.

[16]. This model describes noise by 4 parameters (a1 to a4) that describe the amount of random error that is added during translation and rotation. The model has it that the position error grows faster than linearly with distance (that is to day about $d^{3/2}$ for straight paths), because of the small errors in the heading at the beginning of the path, that cause the robot to drift sideways for the entire remainder of the path.

The theoretical basis for odometry error propagation was laid by some important works. Kelly [17] produced closed-form expressions for the magnitude of the position covariance matrix along straight and curved paths: these mathematical expressions are the basis for analytical prediction of error. Borenstein and Feng [18] developed a standard calibration method based on driving the robot in a square, which is called the UMBmark procedure, for identification and correction of systematic odometry errors. Martinelli [19] has analyzed the relationship between the configuration of the drive system and the error characteristics, and Chong and Kleeman [20] has given detailed error models for differential-drive robots.

These basic works share two properties that are relevant to this thesis. First, they demonstrate that the prediction of odometry error is possible: knowing the geometry of the path and the calibrated noise parameters, the predicted value of the error of odometry at any point can be calculated before moving the robot. Second, they are demonstrating non-uniformity of the error distribution, in that there are long segments, sharp turns, and some geometries of path that will cause more error than others. Both of these properties are exploited in the method of correction scheduling put forward in this work.

Recent research has gone back to the study of how odometry modeling can be done using modern sensors and algorithms. De Giorgi et al. [21] examined drift behavior for visual-inertial odometry on structured indoor environments, and they observed that drift behavior is highly dependent upon scene geometry. Palacin et al. [22] showed error accumulation for differential-drive robots specifically for constructions floors, and the velocity motion model was found to give reasonable predictions for wheeled platforms operating on flat surfaces. Wang et al. [23] proposed a fusion-based state estimator combining Inertial Measurement Unit (IMU) and encoder data,

and can get improved accuracy compared with pure wheel odometry. But, none of these works uses the predicted error profile to plan when the corrections should occur - they focus on the estimation accuracy, not on the scheduling.

Odometry error is well-understood and predictable and is non-uniform. The mathematical tools are available to calculate error profiles on known paths. However, the existing literature applies these tools to estimation and calibration - but not to planning correction strategies.

2.3 SLAM and Localization Correction

Simultaneous Localization and Mapping (SLAM) adjusts the accumulated odometry drift by comparing sensor readings of the robot in question with a map. Modern LiDAR-based SLAM systems such as SLAM Toolbox [24] and ORB-SLAM3 [25] are able to provide centimeter-level accuracy in indoor environments.

Existing SLAM approaches can be roughly classified into filtering based and optimization based approaches. Filtering approaches (such as EKF-SLAM and particle filter SLAM) process observations sequentially, and provide real-time state estimates, but do not explicitly take care of how uncertainty is going to evolve in the future. Optimization-based approaches (such as graph SLAM and bundle adjustment), which minimize error over all the points in the trajectory, provide better global consistency and are computationally expensive and are usually applied as a post-processing step and not during execution. Neither category answers the question of when the robot should put acquisition of observation on hold, to correct itself.

For construction layout marking, SLAM plays a specific and limited role, i.e., the robot periodically stops, takes a LiDAR scan and matches it against a reference map (which can be built from the DXF floor plan), and rectify its estimated position. This essentially nullifies the accumulated position drift so that the robot can once again draw from a position that is accurate. The most important decision in the design is how often and where to stop. Current practice is to correct at a fixed interval - every N segments or every D meters - and does not take into account the fact that

the correction may be inserted at a high-error or a low-error point in the path.

Active SLAM is an alternative to the correction problem. Active SLAM methods, such as the ones surveyed by Placed et al. [8], Alatisse and Hancke [26], and Al-Tawil et al. [27], select where the robot should go in an effort to better localize itself, e.g., driving towards areas with many landmarks, or visiting areas that have already been mapped. Some of these methods do take into consideration when to move along a trajectory (which is superficially similar to correction scheduling). The important difference is that in active SLAM the trajectory is itself selected in order to improve localization, whereas in the case of layout marking the trajectory is determined by the DXF file and can not be altered. The only degree of freedom is when to stop and correct the course along the prescribed way. This is what makes it possible to schedule things offline: the whole future path is known, so the error profile can be computed beforehand - something which is completely impossible in exploration scenarios.

The Nav2 framework [28] provides the complete navigation software stack for robots in ROS 2 that includes planning the paths, avoiding obstacles, and also localizing. Also recent work has improved the real-time position estimation with visual inertial wheel odometry [29] and learning-based uncertainty estimation [30]. These advances help to increase the quality of odometry between corrections but do nothing regarding when corrections should happen.

SLAM offers successful correction of odometry drift, but existing systems consider correction timing a fixed parameter as opposed to an optimization parameter. Active SLAM does the opposite by optimizing the choice of where to go, but keeping the assumption that it does not know the path. For tasks having known paths, the correction scheduling problem has not been addressed.

2.4 Path Planning and Uncertainty

The problem of drawing order (of visiting the line segments in the DXF file) is related to the Traveling Salesman Problem (TSP) and with coverage path planning. Standard approaches reduce travel distance in the totality with the help of nearest neighbor

heuristics or optimization methods. However, these approaches consider all segments equally and do not consider the effects of the chosen sequence on positioning error.

In the general robotics literature, uncertainty-aware planning has received much attention. Probabilistic planners such as belief-space planning explicitly model how the uncertain state of the robot undergoes along candidate paths and select the path with the lowest expected state uncertainty. However, these approaches are aimed at solving navigation problems, where the destination is a goal point, rather than drawing problems, where the robot must visit every segment. The computational cost of belief-space planning is also usually high, consisting of repeated sampling or optimization over the entire state distribution.

Monte Carlo simulation is another way to evaluate the uncertainties: the robot's path is simulated thousands of times with random noise and the distribution of errors analyzed. This approach makes no assumptions about the distribution of errors but is computationally expensive - evaluation of a single floor plan may take minutes of computation. For problem instances where it is necessary to perform evaluations on a large number of candidate configurations (such as correction scheduling), where dozens of candidate positions need to be compared, Monte Carlo methods become impractical.

Analytical covariance propagation creates a middle ground with the Extended Kalman Filter (EKF) prediction equations which calculate the expected error exactly (based on the Gaussian assumption) in one deterministic pass. This approach for simple path geometries was demonstrated by Kelly [17]. The method is quick enough to test a lot of candidate configurations, but has until now not been used to test the problem of scheduling SLAM corrections along a known drawing path.

Existing path planning methods minimize the travel distance without taking into consideration positioning error. Uncertainty-aware planning methods are available but they are dedicated to navigating, and not drawing. Monte Carlo methods can be used to evaluate error but are too slow to be used for iterative scheduling. Analytical covariance propagation is fast enough but has not been used in correction scheduling.

Table 2.1: Summary of related approaches and their limitations with respect to error-aware correction scheduling for robotic layout marking.

Area	Representative Work	What They Do	What They Miss	This Thesis
Construction robotics	HP SitePrint [4], Prieto et al. [5]	Achieve mm-level accuracy using external total stations or laser trackers	Require costly infrastructure; impractical for many construction sites	Provides a software that improves accuracy without any additional hardware
Construction robotics with onboard sensors	Tsuruta et al. [6], Anane et al. [7]	Draw floor plans autonomously using LiDAR or camera localisation	Drift accumulates over long tasks; no strategy for when to correct	Proposes optimal correction scheduling to reduce drift impact
Odometry error modelling	Thrun et al. [16], Kelly [17]	Provide analytical models predicting how error grows along a path	Use predictions only for estimation and calibration, not for scheduling	Uses these models as a prediction engine to guide correction placement
SLAM correction with fixed interval	Nav2 [28], SLAM Toolbox [24]	Correct accumulated drift via periodic LiDAR scan-matching	Apply corrections at fixed intervals regardless of where error is greatest	Replaces fixed intervals with error-peak-driven optimal scheduling
Active SLAM	Placed et al. [8], Yarovoi & Cho [9]	Select trajectories that maximise localisation quality during exploration	Assume the path is unknown; cannot be applied when trajectory is fixed	Path is fully known from DXF; exploration logic does not apply
Uncertainty-aware path planning	Belief-space planning; Monte Carlo simulation	Reason about future uncertainty along candidate paths	Computationally expensive; designed for navigation, not drawing tasks	Replaces Monte Carlo with fast analytical propagation for scheduling

2.5 Research Gap

The four areas reviewed above lead to the problem addressed in this thesis:

1. Construction robotics requires an accurate drawing, on the other part, onboard-

sensor systems drift through a long task.

2. Odometry theory indicates that accumulation of errors is predictable and non-uniform on known paths.
3. SLAM has a correction mechanism but current SLAM systems schedule corrections reactively (at fixed intervals) not by predicted need.
4. Uncertainty-aware planning offers tools for reasoning about future error, these tools have not been used for correction scheduling for drawing tasks.

Despite a lot of work done in each of these areas so far, there is no method that combines predictive uncertainty modelling with correction scheduling for a robot that simply follows a known path. The DXF file gives complete knowledge ahead of time of the trajectory, the velocity motion model gives the mathematical tools to anticipate the error of trajectory and SLAM is the mechanism to correct. The missing link has been the use of the prediction in determining where corrections should be made. This thesis offers such a link.

Chapter 3

Methodology

3.1 System Overview

The system has a two-phase operation. In the offline phase, before the robot movement, the DXF file is parsed into line segments, a drawing order is selected, the expected error is calculated on the planned path and SLAM-correction points are positioned at the locations where the expected error is highest. In the online phase, the plan is followed by the robot, which drives to each segment, draws it and pauses to correct itself at the scheduled points. The complete pipeline is shown in Figure 3-1.

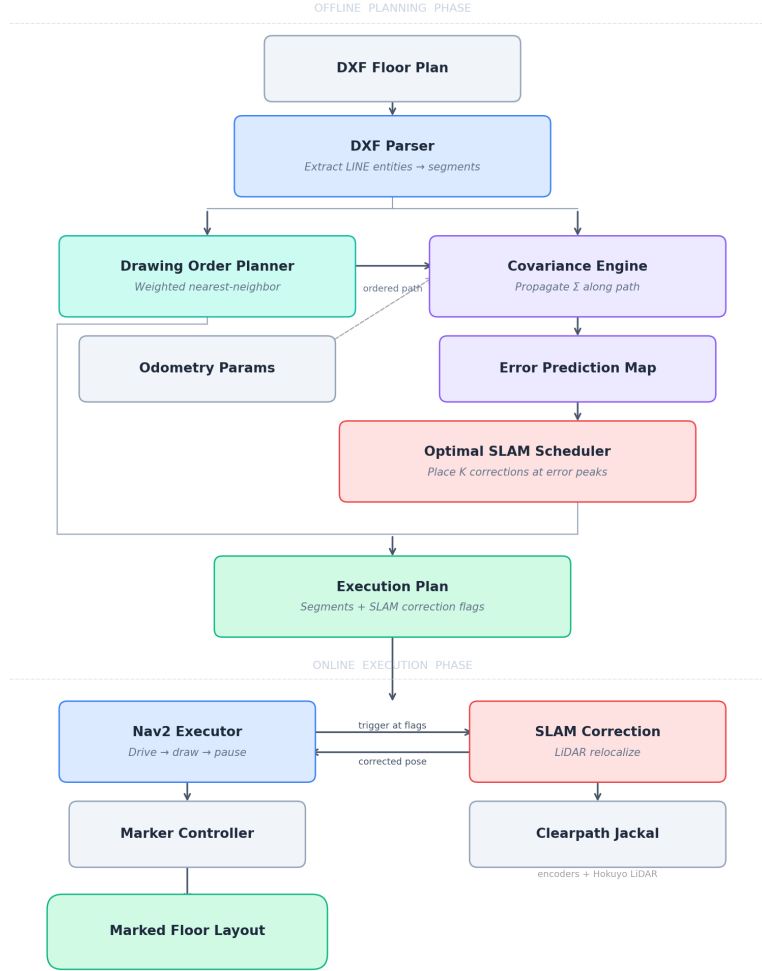


Figure 3-1: System architecture. The offline planning phase uses the floor plan of the DXF file and the parameters of the odometry as input to calculate the drawing order, propagate the error covariance on the planned path and schedule SLAM corrections at the predicted peaks of errors. The online execution phase guides the robot through execution of the plan with the introduction of LiDAR-based relocalisation at waypoints flagged out excited to the plan.

3.2 DXF Parsing

Drawing Exchange Format (DXF) is a standard CAD interchange format, developed by Autodesk and commonly used to encode architectural and construction floor plans

as collections of geometric objects. When layout marking is considered, the corresponding entities are **LINE** objects and each object is characterized by a starting point and an endpoint in two-dimensional Cartesian space. An example of a DXF floor plan as seen in a CAD environment is shown in Figure 3-2.

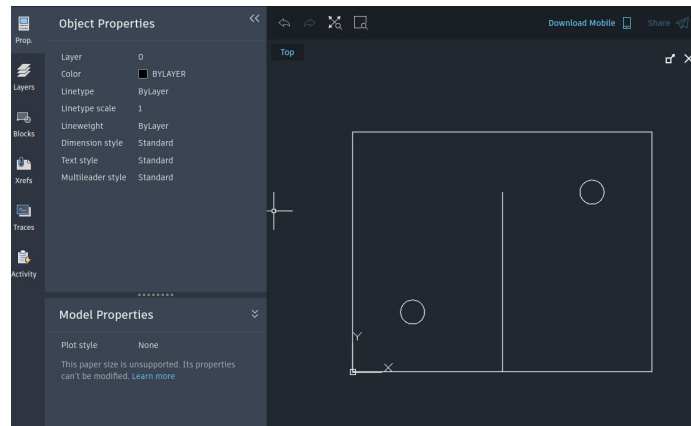


Figure 3-2: DXF floor plan as viewed in a CAD environment.

The `ezdxf` Python library is used to read the DXF file. The parser recurses through all the entities in the model space and picks out the ones whose type is **LINE**. The start coordinate (x_1, y_1) and end coordinate (x_2, y_2) are extracted in each of the entities. The entities that are part of an annotation, dimension or title-block layer are excluded without drawing non-physical elements. The resulting sections are put into a sorted list, with each point being a 2D coordinate. The DXF coordinates which are normally in millimetres are converted to metres. The terminal output of the parser for a sample floor plan is shown in Figure 3-3, listing each extracted **LINE** entity with its layer, length, and start and end coordinates.

```
[1] Type: line
    Layer: 0
    Points: 2
    Length: 5.000 m
    Start: (0.000, 0.000)
    End: (5.000, 0.000)

[2] Type: line
    Layer: 0
    Points: 2
    Length: 4.000 m
    Start: (5.000, 0.000)
    End: (5.000, 4.000)

[3] Type: line
    Layer: 0
    Points: 2
    Length: 5.000 m
    Start: (5.000, 4.000)
    End: (0.000, 4.000)

[4] Type: line
    Layer: 0
    Points: 2
    Length: 4.000 m
    Start: (0.000, 4.000)
    End: (0.000, 0.000)

[5] Type: line
    Layer: 0
    Points: 2
    Length: 3.000 m
    Start: (2.500, 0.000)
    End: (2.500, 3.000)

[6] Type: circle
    Layer: 0
    Points: 33
    Length: 1.255 m
    Start: (1.200, 1.000)
    End: (1.200, 1.000)

[7] Type: circle
    Layer: 0
    Points: 33
    Length: 1.255 m
    Start: (4.200, 3.000)
    End: (4.200, 3.000)

amantav@amantav-VirtualBox:~/cad
```

Figure 3-3: Terminal output of the DXF parser showing extracted LINE entities with their layer, length, and start and end coordinates.

The segments thus parsed are then sent to the drawing order planner which decides the order in which they will be drawn. The final representation of the planned path, as visualised in RViz prior to execution, is shown in Figure 3-4.

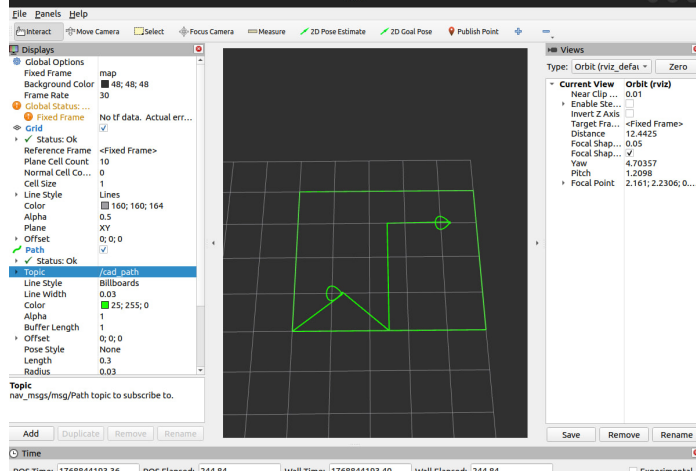


Figure 3-4: Planned path visualised in RViz prior to execution, showing the parsed segments passed to the drawing order planner.

3.3 Drawing Order Planning

3.3.1 Shortest-Distance Planner

The baseline planner has a nearest neighbor approach: starting from the initial position, a segment is selected whose closest endpoint has not been visited yet, and drove to, by the robot, drew the segment, and so on. For both a forward and a reverse orientation are considered for each candidate segment, and the one with the less travel is selected.

3.3.2 Weighted Nearest-Neighbor Planner

The weighted planner provides a direction of preference to the baseline. In picking the next segment however, it prefers segments that go in the opposite direction than the one just drawn. The idea behind this is that if the robot drifts to the left while drawing one segment, drawing the next segment in the opposite direction may partially cancel that drift.

For each candidate segment j , a score is computed by the planner:

$$\text{score}(j) = d_{\text{travel}}(j) - \beta \cdot r_{\text{opp}}(j) \quad (3.1)$$

Where, $d_{\text{travel}}(j)$ is the distance (in meters) the robot needs to travel to reach segment j . The term $r_{\text{opp}}(j)$ is 1 if segment j is pointing in about the opposite direction from the previous segment (within $\pm 30\%$), and 0 otherwise. The threshold of 30 has been empirically chosen as a compromise between being too stringent (not counting almost-opposing segments) and too sloppy (counting segments that are not opposing, but are perpendicular, as opposing segments, which do not cause any meaningful cancellation). The parameter b (in meters) controls the degree of extra travel that the planner is willing to admit in order to travel to an opposite segment. A lower score is preferable, so opposing segments are rewarded by a bonus. Values of $\beta = 0.5$ m and $\beta = 1.0$ m are tested. The selection of β is determined by the frequency of the SLAM correction. $\beta = 0.5$ is more secure when SLAM corrections are less frequent. $\beta = 1.0$. It gives better results with higher SLAM correction frequencies when the additional travel is less expensive (see Table 4.2).

It should be noted that to achieve drift cancellation with opposite directions, both segments should be drawn in that order with little travel in between. If the robot has a long travel to reach the opposing segment then the cancellation benefit might be lost by the additional error due to travel. This is the reason that the planner is considered the secondary contribution because its benefit depends on the frequency of SLAM corrections, and can be detrimental to performance when corrections are not available (see Section 4.4).

3.4 Covariance Propagation Engine

The heart of the system is a mathematical model that allows prediction of the amount of position error that the robot will have at any position along the robot's path. The model keeps track of a covariance matrix S of 3 x 3 values that represent the uncertainty in the robot's x position, y position and heading angle θ . As the robot moves this uncertainty increases. For each small step to move by a segment, the update rule is:

$$\Sigma_{k+1} = G_k \Sigma_k G_k^T + V_k M_k V_k^T \quad (3.2)$$

The first term ($G_k \Sigma_k G_k^T$) describes the way that existing uncertainty is carried forward to the next step. The second term ($V_k M_k V_k^T$) adds the new noise that is introduced by the motion itself where M_k is determined by the odometry parameters α_1 through α_4 . The matrices G_k and V_k are Jacobians obtained from the velocity motion model [16].

At each of the steps, the overall position uncertainty is summarized as:

$$\sigma_{\text{pos}} = \sqrt{\sigma_x^2 + \sigma_y^2} \quad (3.3)$$

which is a combination of the x and y uncertainties into one number which represents how far off the robot is expected to be.

When the robot is doing SLAM correction, their position is well measured by LiDAR and covariance is reset to small value (sSLAM ≈ 1 mm). From that point on the level of uncertainty begins growing again from near zero.

3.5 Optimal SLAM Correction Scheduling

Given a budget of K corrections, the algorithm has to figure out the K segment boundaries on which to make corrections. It uses a greedy strategy putting one correction at a time:

1. Try each possible position for the next correction. For each of them, run the covariance engine to calculate on the mean drawing error that would result.
2. Pick the position that will result in the lowest mean error.
3. Repeat until all the K corrections are placed.

This approach involves testing a large number of candidate positions, but normally the analytical engine will take only milliseconds to evaluate each candidate position, so the entire scheduling process will be completed in seconds even for large floor plans.

3.6 Test Floor Plans

Four test floor plans of increasing complexity were used (Figure 3-5):

- **Lab Room** (5×5 m): 10 segments, simple rectangular room with furniture.
- **Office** (8×5 m): 18 segments, two-room office with internal wall and furniture.
- **Apartment** (12×10 m): 32 segments, multi-room apartment with varied segment lengths.
- **Warehouse** (15×12 m): 24 segments, open space with column markers and aisles.

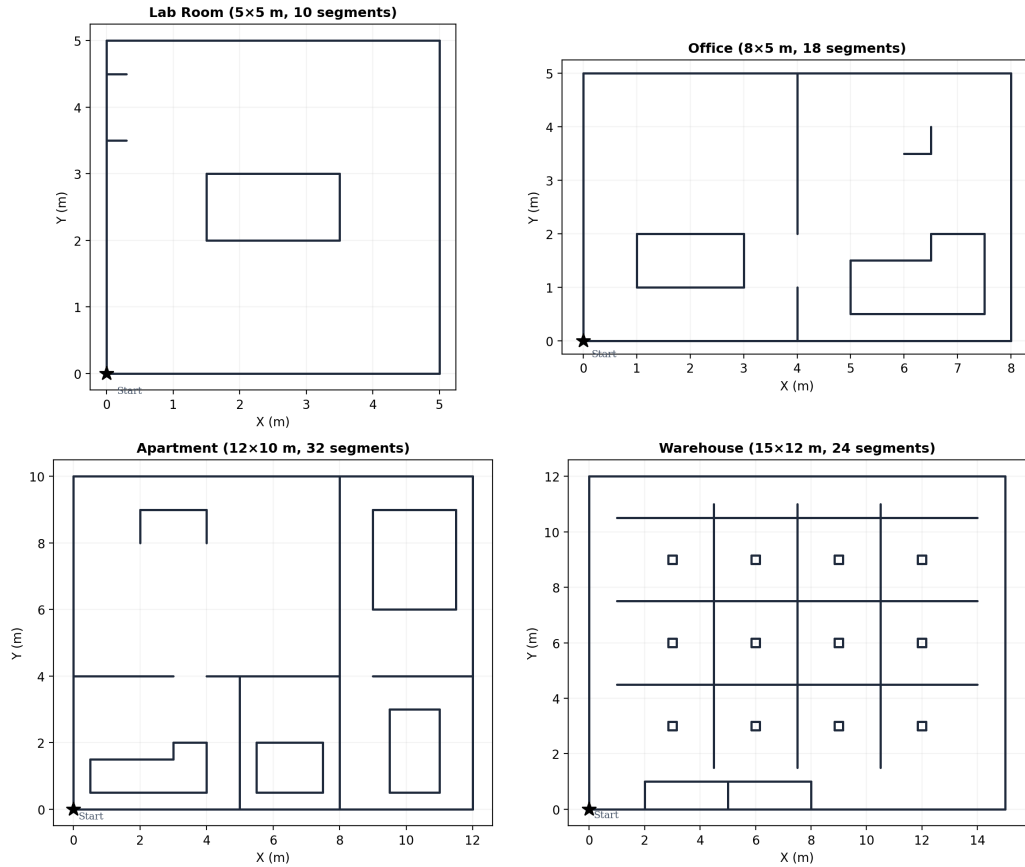


Figure 3-5: The four test floor plans in the simulation experiments. Each plan is drawn as it would be viewed in the DXF file with the starting position of the robot identified by a star. Plans from a simple to a 32 segment room to an apartment with varied segment lengths.

The plans were selected to include a variety of complexity from the small room where the robot has nothing more than a short travel to draw everything to a large apartment with segment lengths spanning from 0.5 m (fixture outlines) to 12 m (outer walls). This variation is important because the advantage of optimal correction placement is dependent upon the manner in which the error is distributed along the path.

3.7 Experimental Setup

All the simulations have the typical indoor odometry parameters ($\alpha_1 = 0.005$, $\alpha_2 = 0.001$, $\alpha_3 = 0.010$, $\alpha_4 = 0.005$). Error values are given in centimeters, in the form of the distance that is expected between the actual and intended position of the robot. To transform from the Covariance based uncertainty (σ_{pos}) to an expected distance, the Rayleigh correction $\bar{d} = \sigma_{\text{pos}} \cdot \sqrt{\pi/2} \approx 1.25 \cdot \sigma_{\text{pos}}$ is applied. Strictly speaking, it is assumed in this conversion that the distribution is circular Gaussian (i.e. equal and independent σ_x and σ_y), while the general form of the propagated covariance matrix will have unequal diagonal entries and non-zero off-diagonal terms. The precise distance expected for an elliptical distribution is dependent on the eigenvalues of the 2 x 2 position submatrix. In practice this approximation does not have an effect on the ranking of the errors between segments - and this is what the scheduler is based on - but the absolute centimeter values as reported in Chapter 4 can be slightly off compared to the true expected displacement. Monte Carlo validation of 500-1000 rnd per configurations is used Physical experiments are done using the Clearpath Jackal UGV with a Hokuyo UST-10LX LiDAR, ROS 2 Jazzy with Nav2 and SLAM Toolbox.

Chapter 4

Results and Analysis

4.1 Overview

This chapter shows the results of the simulation. All error values reported in this chapter are either predicted by the analytical model or calculated from Monte Carlo simulation - they are not physical measurements from a real robot. Physical validation is described in Chapter 5. The analysis is done in three stages: validation of the prediction model (Section 4.2), evaluation of optimal SLAM correction scheduling (Section 4.3), and evaluation of drawing order planner (Section 4.4). Section 4.5 evaluates the combi system. Sections 4.6 and 4.7 are responsible for the optimality and robustness check of the scheduling algorithm. The results of the findings are discussed in Section 4.8.

4.2 Validation of the Prediction Model

Before evaluating the correction scheduling algorithm, it is necessary to establish that the covariance propagation engine produces reliable predictions of drawing error. Validation was done using two path geometries: a straight-line path of varying length (1-14 m) and a closed rectangular path with four 90 degree turns. Each geometry was compared to an analytical prediction and a Monte Carlo simulation of 500 independent runs in which the motion of the robot was sampled using the velocity motion

model with the calibrated odometry parameters (1 -4). The average and standard deviation of the endpoint error of the runs were taken and plotted with the analytical prediction. The mean and standard deviation of the endpoint error across runs were recorded and plotted alongside the analytical prediction.

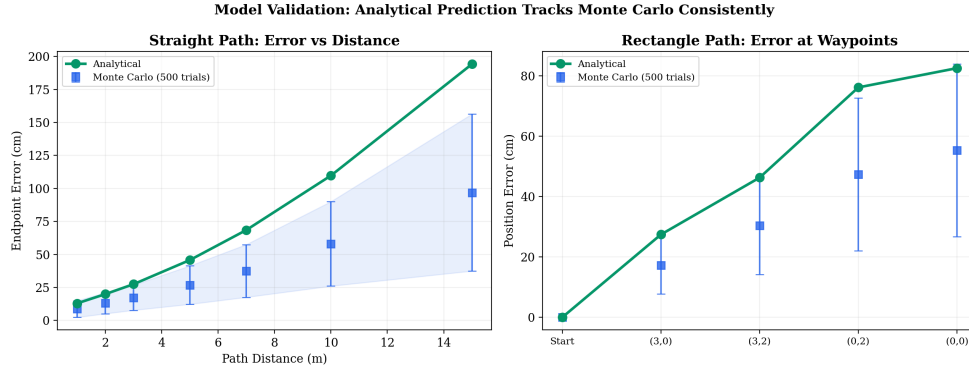


Figure 4-1: Model validation. Left: Endpoint error vs. path distance, straight paths - analytical prediction (green) follows the Monte Carlo mean (blue squares) with a similar scaling ratio. Right: error at each of the waypoints of a rectangle path. Both the geometries confirm the accuracy in prediction.

As Figure 4-1 indicates, the analytical predictions follow closely the Monte Carlo means across both path geometries. At short distances (1-5 m) the two methods are in close agreement. At larger distances the model slightly overstates the error due to the linearisation, on which it is based, becoming less accurate with increasing uncertainty. Nevertheless, the model consistently identifies the same segments as having high or low error, and ranks the error contributions correctly even when the absolute centimetre values diverge. It is precisely this ranking accuracy that the correction scheduler requires: it does not need to know the exact magnitude of the error, only which positions accumulate the most. Furthermore, the analytical model evaluates a complete apartment floor plan in under one second, compared to approximately two minutes for 200 Monte Carlo trials. This speed advantage renders iterative scheduling of numerous candidate correction positions computationally feasible.

The analytical model correctly identifies the segments that accumulate the most error - even when absolute values diverge slightly from Monte Carlo due to the Rayleigh approximation - confirming that RQ1 is answered affirmatively: analyti-

cal covariance propagation is sufficiently accurate to guide correction scheduling.

4.3 Optimal SLAM Correction Scheduling

4.3.1 Uniform vs. Optimal Placement

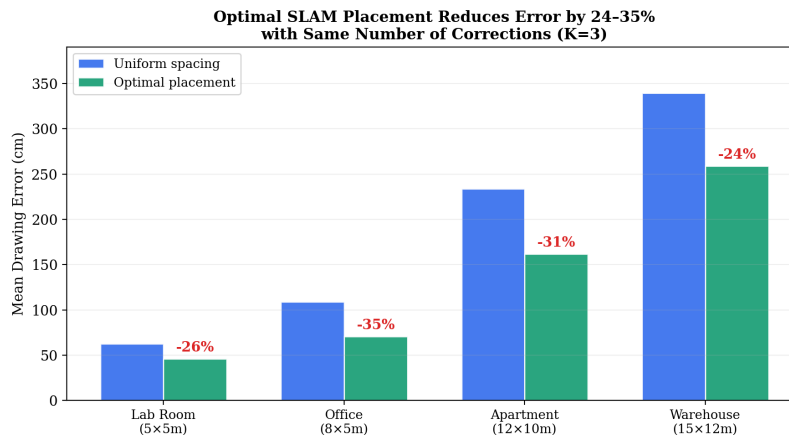


Figure 4-2: Optimal placement of the SLAMs allows a reduction in mean drawing error by 24-35% as compared to uniform spacing using the same number of corrections $K=3$. Percentage labels denote improvement on each floor plan.

Figure 4-2 compares the performance of uniform and optimal placement of 3 SLAM corrections for all the floor plans. The numerical values are given in Table 4.1.

Table 4.1: Uniform vs. optimal SLAM placement (3 corrections).

Floor Plan	Uniform (cm)	Optimal (cm)	Improvement
Lab Room (5×5 m)	62.6	46.1	−26.4%
Office (8×5 m)	108.8	70.9	−34.8%
Apartment (12×10 m)	234.0	161.8	−30.8%
Warehouse (15×12 m)	339.5	259.1	−23.7%

The improvement magnitude varies across floor plans in a manner consistent with the predictions of the covariance model. The office plan achieves the highest improvement (34.8%) because its segments span a wide range of lengths, from short furniture

outlines to long corridor walls. When segment lengths are unequal, error accumulates disproportionately on the longest segments, and corrections placed immediately after these segments intercept the highest error peaks. The warehouse plan shows the smallest improvement (23.7%) because its segments are more uniform in length, producing a flatter error profile that gives uniform spacing little disadvantage. This is consistent with the theoretical prediction that odometry error grows faster than linearly with distance, meaning longer segments benefit most from targeted correction. The improvement arises because uniform spacing distributes corrections without regard to segment length. In the case of an 8-metre segment and a 0.5-metre segment, the uniform spacing would give each segment an equal portion of the corrections, although the long segment would accumulate a much larger proportion of the error. Optimal placement, in contrast, places corrections on the immediately subsequent segments that provide the largest contribution to the drift, and limits error peaks before they get larger. Every SLAM correction requires a pause of 1-2 seconds; by making corrections in a more effective manner, the robot can achieve the same accuracy with fewer pauses, or higher accuracy with the same number of pauses. These findings indicate that the placement of SLAM corrections is as consequential as their frequency: with the same correction budget, optimal placement yields substantially lower drawing error than uniform spacing across all four floor plans tested.

4.3.2 Effect of Correction Budget

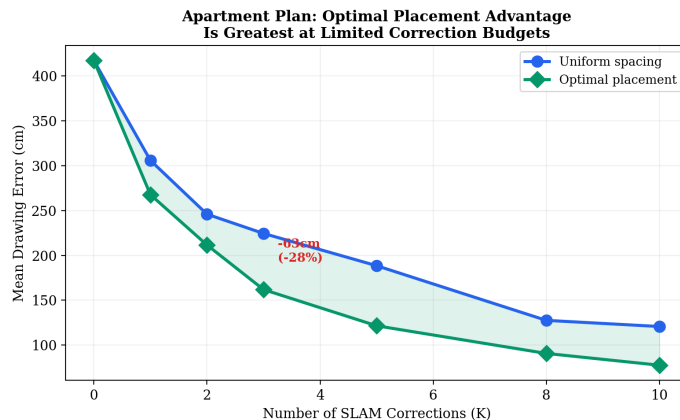


Figure 4-3: Apartment plan error vs. correction budget. The area in the shade is the accuracy that is gained due to best placement, maximum at limited budgets $K=2-5$).

The correction budget results have a direct practical interpretation. A practitioner with limited time who can afford only $K = 3$ optimally placed correction pauses achieves the same accuracy as one using $K = 5$ uniform pauses. At approximately 7 seconds per correction pause, this represents a saving of approximately 14 seconds per floor plan execution - a modest figure for a single room but significant when scaled across an entire building floor comprising hundreds of plans. The diminishing returns evident beyond $K = 5$ indicate that, for most practical floor plans, three to five optimally placed corrections capture the majority of the achievable improvement.

Figure 4-3 shows the mean drawing error as a function of correction budget K for the apartment floor plan, comparing uniform and optimal placement strategies. The performance difference between the two strategies is greatest at low correction budgets ($K = 2-5$), and is up to 28 cm. The two strategies meet as K increases, since an appropriately dense correction schedule will cover all error peaks, no matter where they appear. The area in Figure 4-3 that is shaded is the accuracy gain that can be attributed to an improved placement of the identical number of corrections. The practical value of this result to deployment situations is that correction pauses are expensive in terms of job time. A robot with $K = 3$ corrections can be as accurate as a uniformly-corrected robot with $K = 5$ corrections, as long as the corrections are

made at the predicted error peaks. Equivalently, with the same accuracy target, the number of required correction pauses can be minimized by using the predicted error profile to guide placement, which can be directly translated into reduced job times without any extra hardware.

4.3.3 Error Profile Along the Path



Figure 4-4: Apartment plan- predicted error along the drawing path. Optimal correction points of Green diamonds coincide with peak errors; Uniform points of Blue circles sometimes appear at the low error position, hence, correction opportunities are not utilized efficiently.

The mechanism is visually revealed in the Figure 4-4. If the error is not corrected then it continuously increases with accelerations after turns. Under uniform correction, a sawtooth pattern develops with resets at fixed intervals - with some of them following short segments, where the error is already low. Under ideal correction, resets occur when there is a higher rate of error so that the maximum is limited at a lower level.

4.3.4 Spatial Error Distribution

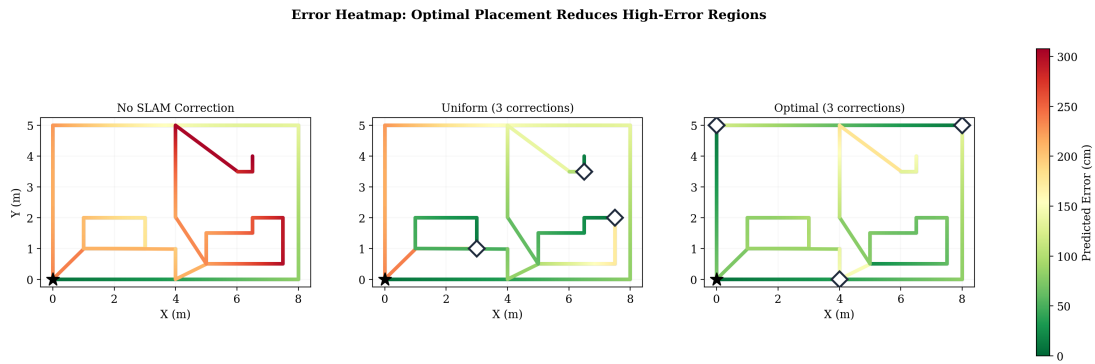


Figure 4-5: Predicted error colored office floor plan (consistent color scale). Left: no correction. Center: uniform. Right: optimal. Diamond markers are located at the locations of SLAM corrections.

Figure 4-5 is a spatial mapping of error. Without any correction, color changes from green (close to beginning) to red (farthest sections). Uniform correction causes some of the red areas to shrink, but other areas where corrections are out of line with error peaks are left intact. Optimal correction has the effect of eliminating the red regions substantially - the three correction points maximizes the floor area drawn with acceptable accuracy.

4.3.5 Covariance Ellipse Visualization

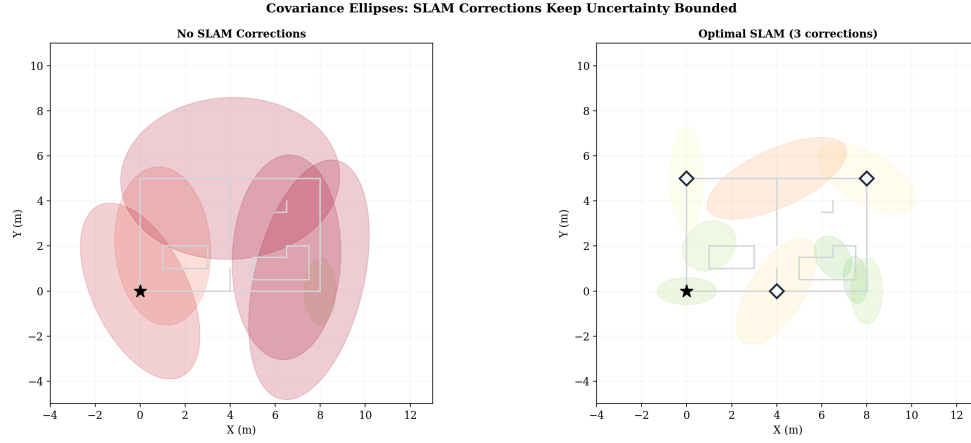


Figure 4-6: Covariance ellipses at endpoints of the segments (every 3rd shown, same scale). Left, without correction - uncertainty extends outside the footprint of the building. Right: with 3 optimal corrections(diamonds) -uncertainty is bounded.

Without corrections (Figure 4-6, left), the uncertainty ellipses increase in size and eventually become larger than the building dimensions - i.e. the robot is less sure where it is than is the size of the building it's drawing. With three optimal corrections (right), the ellipses gain smaller and smaller at each of the correction points as they are situated in between these correction points.

4.4 Effect of Drawing Order Optimization

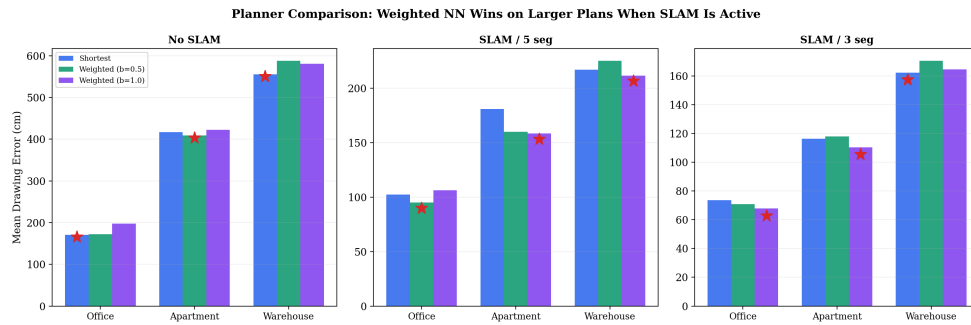


Figure 4-7: Comparing Planners at three SLAM levels Red stars designate the best planner. Without SLAM shortest distance wins. With SLAM, weighted planner is winning on bigger plans.

Table 4.2: Planner comparison with SLAM corrections (cm).

Plan	Planner	No SLAM	SLAM/5	SLAM/3	Distance
Office	Shortest	170.5	102.3	73.7	54.8 m
	Weighted ($\beta=0.5$)	172.3	95.1	70.9	55.0 m
	Weighted ($\beta=1.0$)	197.5	106.3	67.9	60.8 m
Apartment	Shortest	417.2	181.1	116.2	136.1 m
	Weighted ($\beta=0.5$)	408.7	160.0	118.0	135.9 m
	Weighted ($\beta=1.0$)	422.0	158.4	110.4	143.1 m
Warehouse	Shortest	555.3	217.2	162.3	194.3 m
	Weighted ($\beta=0.5$)	587.6	225.2	170.4	204.2 m
	Weighted ($\beta=1.0$)	580.8	211.7	164.6	206.5 m

The extra distance to travel to reach opposing segments adds more error than is removed by drift cancellation because the error increases faster than linearly with the distance. By using SLAM corrections, the cumulative error is periodically reset and the additional travel between resets is insignificant so that the opposing-direction benefit is achieved as a net gain. The opposing segments partially cancel the drift they are experiencing between resets, giving an overall improvement.

It should be noted that the No SLAM column of Table 4.2 is not a realistic operating point but a hypothetical worst case. No real system would attempt to draw a complete apartment floor plan, comprising 136 metres of travel, without making any correction. The practically relevant comparison is therefore uniform vs. optimal placement with the same number of corrections.

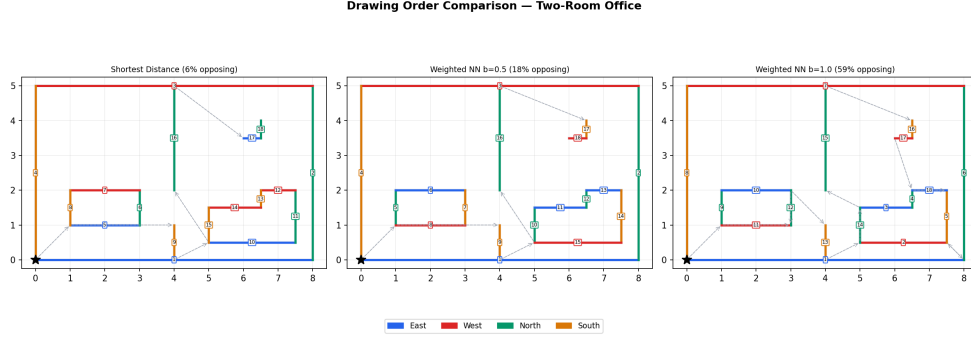


Figure 4-8: Drawing order comparison office plan of. The weighted planner generates more opposing pairs with similar total travel distance.

Drawing order optimisation is only beneficial in the presence of SLAM corrections, because without corrections the extra travel required to reach opposing segments accumulates as additional error. Since the drawing order planner contributes 5-12% improvement compared to 24-35% from optimal correction placement, the practical question is whether the additional planning complexity is justified. In applications where the highest possible accuracy is required, such as mechanical, electrical, and plumbing layout, the additional improvement justifies the minimal computational cost. In rough layout tasks with large tolerances, the shortest-distance planner with optimal correction placement may be adequate.

The contrasting behaviour of the weighted planner under zero-correction and high-correction conditions has a clear mechanistic explanation. Without SLAM corrections, the additional travel required to reach an opposing segment introduces new odometry error faster than drift cancellation removes it; the planner is therefore counterproductive in the absence of corrections. With frequent SLAM corrections, however, accumulated drift is reset at regular intervals, keeping the baseline error low. In this regime, the additional travel between resets is small relative to the cancellation benefit on the subsequent segment, and the weighted planner produces a net improvement. This dependence on correction frequency is the reason the weighted planner is presented as a secondary contribution, best used in conjunction with the correction scheduler.

4.5 Combined System Performance

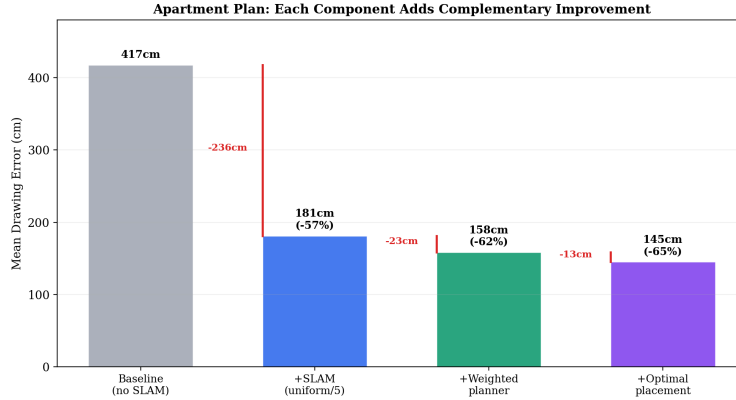


Figure 4-9: Apartment plan - cumulative improvement. SLAM corrections cause the biggest reduction (57%), the weighted planner and optimal placement achieve complementary improvement. Where the lines in red are the reduction in each step.

These results provide a direct answer to RQ3. The 5% improvement from drawing order optimization is a statistically meaningful contribution when accuracy targets are tight—for example, when reducing mean error from 160 cm to 145 cm on the apartment plan. Practitioners prioritizing system simplicity may omit the weighted planner with only a 5% accuracy penalty, as no direction-preference logic is required. The scheduler alone accounts for the majority (57%) of the total improvement over the baseline, confirming that correction placement is the dominant factor in the system.

The relative magnitudes indicate a clear hierarchy with SLAM correction frequency giving the best improvement (57% reduction vs. no corrections), followed by optimal scheduling giving 3-8% on top of uniform SLAM and then drawing order optimization giving 5% on top of shortest distance planner. To avoid confusion: The 24-35% number in Section 4.3 is a comparison of optimal vs. uniform placement directly (both using $K = 3$); the much lower 3-8% number here is the incremental contribution of optimal placement when added to a system that already has SLAM corrections and weighted planning.

4.6 Greedy vs. Exhaustive Correction Placement

To test the effectiveness of the greedy algorithm to find the best possible positions of correction, the greedy algorithm was compared to an exhaustive search, which attempts to find the best correction positions by trying out all possible combinations. For the lab room ($N = 10$, up to $K = 3$), office ($N = 18$, up to $K = 3$), and apartment ($N = 32$, $K = 1$), all combinations of $\binom{N}{K}$ were tested - a total of 1194 placements.

The globally optimal placement was found by the greedy algorithm in all cases within zero gap in all seven comparisons. This seems to be because when covariance is reset at one position, it only influences the error after that point - it does not influence where earlier corrections are supposed to go. However, the range of the tests is small: $K \leq 3$ corrections on plans with $N \leq 32$ segments, total 1,194 placements. On a larger plan (e.g. 100 segments and $K = 10$) the behavior of interactions between correction positions may be different. The independence property has not been formally proven and verification of the greedy algorithm against exhaustive search should be carried out for configurations outside the tested range.

4.7 Sensitivity to Odometry Parameter Mismatch

The scheduler uses odometry parameters (a1-a4) for the prediction of the error profile. In real life, these parameters are not necessarily calibrated. To test the extent to which this is important, corrections were scheduled using deliberately erroneous values for the parameters and the resulting placements were assessed under the correct parameters. This is a simulation of a robot that has been calibrated on one floor surface but is operating on another.

Eight conditions of mismatch were tried on each of two floor plans (office and apartment), 16 evaluations in total. The conditions included scaling down all parameters by 0.5x to 2.0x, scaling down individual parameters by 3x, switching rotational and translational noise, and increasing rotation noise by 5x. In 15 of 16 cases, the

scheduler selected precisely the same correction positions as in the case of the true parameters. The one exception (5x rotation noise on the apartment) shifted 1 correction point and increased error only by 0.7% - still 27% better than uniform spacing.

The reason for this is straightforward: the scheduler selects positions according to which segments are more wrong than others, and not according to the actual error values. Changing the parameters changes the size of the errors but not the fact that a long segment results in more error than does a short one. The ranking of the segments according to the error contribution is found to be largely a function of the path geometry - segment lengths and turn angles, rather than the individual values of the parameters. In practice, this translates to the fact that even rough odometry estimates are good enough to allow effective correction placement.

4.8 Discussion of Results

The results are based on the fact that one observation made is that odometry error is not uniform along the path of the drawing. Some segments (particularly long ones, and those after sharp turns) account for much more error than others. Because of the a priori knowledge of the path, this uneven distribution is predictable and corrections can be placed there where they have the greatest effect.

Three conditions are favourable to optimal placement: floor plans where the lengths of segments vary widely (which was the case in the office plan which mixed 8-meter walls with 0.5-meter details, for which the biggest improvement was 34.8%), small budgets for correction ($K = 2-5$, where each correction must count), and paths with clusters of turns which concentrate heading error in a small region.

The greedy algorithm gives the globally optimal placement in all tested cases (Section 4.6), and the placements are robust to odometry calibration errors up to 2x (Section 4.7). Together, those findings mean not only that the approach is computationally simple, but, too, that it is reliable in practice.

It is worth noting that even for optimal placement, the absolute errors are still quite high compared to construction grade tolerance (± 5 mm). For example, the

apartment plan with 3 optimal corrections has the mean error of 161.8 cm. To reach the accuracy of millimeters would require much more corrections ($K \leq 15-20$) or additional localization hardware. The contribution of this work is not to remove the need for corrections, but to make each correction count more - reducing the number required to make a certain level of accuracy.

Limitations. The model has no noise, and does not include systematic biases such as one wheel being slightly bigger than the other. Physical validation can only be done in smooth laboratory floors; on real construction sites with rough concrete and debris, the behaviour may differ.

The general implication of this is that if the position of the robot we are controlling is known ahead of time, then fixed interval correction is unnecessarily wasteful. The profile of errors is predictable and corrections can be rationed.

Chapter 5

Physical System Implementation and Validation

As shown by the simulation in Chapter 4, drawing error can be reduced by up to 24–35 percent by placing SLAM corrections error-aware versus uniform spacing. The simulation is however based on idealized assumptions: perfect covariance reset at correction points, noiseless SLAM measurements, and errors caused by controllers. This chapter helps to bridge the gap between simulation and reality by applying the entire DXF-to-robot pipeline to a physical mobile robot and testing the scheduling strategy in the real-world. The aims are two-fold: (1) confirm that the system is capable of autonomously executing a DXF floor plan with a centimeter-level accuracy, and (2) test whether the relative ranking, as predicted by the simulation (optimal < uniform < no correction) can be determined on physical equipment.

5.1 Hardware Platform

A Clearpath Jackal J100 unmanned ground vehicle was used to carry out the physical experiments. The Jackal is a four-wheel skid-steer robot with dimensions of $508 \times 430 \times 250$ mm, mass of 17 kg and maximum payload of 20 kg. The skid-steer drive system offers zero-radius turning, which is beneficial in negating the tight turns necessary in the rectangular floor plan. Nonetheless, skid-steer locomotion induces sideways wheel

slip on turns, a well-known cause of odometry error not completely represented by the velocity motion model applied to the simulation [18].



Figure 5-1: Clearpath Jackal J100 Robot

The robot is fitted with a Hokuyo UST-10LX 2D LiDAR sensor that is fitted on the front of the chassis using a fixed bracket. The LiDAR has a 270° field of view and 0.25° angular resolution at a scan rate of 40 Hz and reaches a maximum range of 10 m. The sensor has a position of 12 cm forward and 20 cm above the base frame of the robot. This offset between the `laser` coordinate frame and `base_link` is broadcast by a static transform publisher.

Table 5.1: Clearpath Jackal J100 specifications.

Parameter	Value
Dimensions (L $\times$ W $\times$ H)	508 $\times$ 430 $\times$ 250 mm
Weight / Payload	17 kg / 20 kg
Maximum speed	2.0 m/s
Drive configuration	4-wheel skid-steer
LiDAR	Hokuyo UST-10LX
LiDAR FOV / Resolution	270° / 0.25°
LiDAR scan rate	40 Hz

5.2 Software Architecture

This software stack is based on ROS 2 Jazzy (Ubuntu 24.04) and has four primary components.

The localization module follows AMCL (Adaptive Monte Carlo Localization) in the framework of Nav2 navigation [28]. AMCL has a particle filter that compares incoming LiDAR scans to a pre-constructed occupancy grid map (0.05 m resolution) to approximate the pose of the robot in the map frame. The map was constructed by teleoperating the robot by running SLAM Toolbox [24] in the online asynchronous mode.

Jackal platform is based on namespaced ROS 2 topic structure (`/j100_0595/...`). As Nav2 and AMCL require standard names of topics, two relay nodes are used to mediate between TF and TF static topics of the Jackal namespace and the global namespace. Another bridge node translates conventional `Twist` velocity commands in the experiment controller to the `TwistStamped` format the platform controller of the Jackal on `/j100_0595/platform/cmd_vel` needs.

The experiment controller is a custom ROS 2 Python node that executes the entire DXF-to-motion pipeline: it reads the floor plan, calculates the drawing order, schedules SLAM corrections, and moves the robot based on the resulting waypoint sequence. The controller is a finite state machine and it publishes visualization markers to be used to monitor it in RViz.

The system is comprised of eight running terminal processes: two TF relay nodes, the LiDAR driver, a fixed transform publisher, the LiDAR mount, the Nav2 navigation stack (with AMCL), lifecycle node activation, the velocity bridge, and the experiment controller.

5.3 DXF-to-Robot Pipeline

Four sequential steps make up the pipeline to transform a DXF floor plan into the executable robot commands.

5.3.1 DXF Parsing and Segment Extraction

DXF file is read with the `ezdxf` Python library. A segment is represented in the DXF file as each LINE entity with its start and end coordinates. The parser then retrieves all the LINE entities and transforms coordinates in the DXF unit system (which is usually millimeters) to meters. In the case of the laboratory test, the floor plan has 5 line segments that consist of four walls that make a rectangle of 3.5×1.2 m and a single 0.8 m interior partition wall.

5.3.2 Drawing Order Planning

The nearest-neighbor planner (see Section 3) decides the sequence of the segments to be drawn. The planner picks the nearest endpoint of a segment that has not been visited at this starting point of the robot, and both forward and reverse orientation is taken to achieve the least amount of travel distance. In the case of the test floor plan, this will result in a natural counter clockwise movement around the rectangle and then the inside partition.

5.3.3 Coordinate Transformation

The DXF coordinates are translated to the map frame of the robot by a 2D transformation. The origin of the floor plan is the map coordinates (0.25, 0.20) which places the 3.5×1.2 m plan in the centre of the 4.0×1.6 m clear space of the laboratory with 25 cm margin on all sides. These margins allow clearance of the footprint of the robot, and to accommodate any positioning errors in the execution.

The clear area was detected by examining the occupancy grid map that was created. The radius of the robot (30 cm) eroded pixels whose occupancy was greater than 230 (free space) to locate the safe navigable space. The maximum size of a rectangle in this area was the test space.

5.3.4 SLAM Correction Scheduling

The covariance propagation engine evaluates the planned trajectory and estimates the anticipated position error at the ends of each segment. Greedy scheduler picks the two positions which minimize the average predicted drawing error in the test floor plan with $K = 2$ corrections. The scheduler found the optimal positions after segments 1 and 2, as the two 3.5 m walls (segments 0 and 2) are the two that amass the most odometry drift.

The entire waypoint list—comprising segment start and end points, travel waypoints and correction flags—is represented as a Python data structure that controls the experiment controller.

5.4 Navigation and Control

5.4.1 Path Following Controller

Instead of the path planner of Nav2, which produces smooth curved paths that are optimized to avoid obstacles, the experiment operates with a custom proportional steering controller, which moves the robot to each goal point directly. This was done due to the design of the default planner of Nav2 (NavFn with DWB controller) which generated arcing paths between waypoints leading to curved lines where the DXF plan had straight ones. A drawing robot needs to follow line segments with as little deviation as possible, and this necessitates a controller with path accuracy rather than obstacle avoidance as a priority.

The controller works with a frequency of 20 Hz. It calculates the heading error between the current orientation of the robot (provided by AMCL) and the direction to the current target point at every control step. There are three steering regimes with varying magnitudes of heading error. In cases where the heading error is less than about 45° the robot moves forward at its nominal speed (0.10 m/s) with angular correction ($\omega = k \cdot \theta_{err}$, where $k = 2.0$). In the case of heading errors of between 45° and 115° , forward speed is lowered to 30% but steering gain is retained. In case of a

heading error greater than 115° (which mainly happens at the 90° corners), the robot turns in place then returns to moving forward.

The forward velocity of 0.10 m/s was chosen as a trade off between drawing accuracy and duration of the experiment. Faster speeds (0.20 m/s) were also experimented with but gave larger path deviations because the skid-steer platform experiences a lateral slip when correcting the steering. The 2.0 proportional gain was adjusted to offer responsive steering without causing any oscillation due to AMCL localization noises ($\pm 2^\circ$ yaw noise was measured during straight-line driving).

5.4.2 SLAM Correction Mechanism

Upon reaching an adjustment point at a specific time, the controller halts forward motion and orders the robot to move 360° in its current location over about 6 seconds (angular velocity 0.8 rad/s). This rotation is also meant to be used: when a car is driving normally, the LiDAR only sees the walls on the side and in front, which is geometrically constraining along only a few directions. Full rotation gives AMCL LiDAR scans of the entire 360° of the environment, enabling the particle filter to meet on the wall features in every direction.

Once the rotation is over, the controller waits a 0.5 seconds before drawing the sequence resumes so the particle filter can stabilize. The average time cost per correction is about 7 seconds.

A SLAM correction is simulated in the form of a covariance reset, the position uncertainty is brought to almost zero at a point in time. The correction, which is applied to the AMCL particle filter on the physical robot, is a gradual convergence, which might not provide the same level of uncertainty reduction. This disparity is among the reasons that account for the difference in the magnitude of improvement in the physical robot than on the simulation.

5.4.3 State Machine

This experiment controller provides a five-state finite state machine. The first step is the initialisation state during which the robot broadcasts its own initial pose estimate to AMCL and waits until the localisation system has initialised. During the localization spin state, the robot spins briefly (3 seconds) to give AMCL preliminary scan data at a variety of directions. The controller directs the robot sequentially to the target points of the waypoint list in the driving state, storing the AMCL-reported position and trail information. Upon flagging a target point to be corrected by SLAM, the state machine switches to the SLAM correction state, and spins 360° before returning to driving. Once the waypoints are completed, the completion state writes down the recorded information to disk.

5.5 Test Environment and Floor Plan

The experiments were carried out at a robotics laboratory at Nazarbayev University. The laboratory has workbenches, equipment shelves and other barriers that limit the floor space. Based on the pre-occupancy grid map, the biggest clear rectangular area was 4.0×1.6 m, which is situated along one of the walls of the laboratory.

The test floor plan was worked out to be fitting in this space, and meet two conditions: it has to contain both long and short segments (the condition by which the simulation predicts the best scheduling to run faster than uniform spacing), and it has to have sufficient margins so that the robot is not going to approach obstacles. The resultant plan is a 3.5×1.2 m outer rectangle (segments 0–3) and 0.8 m inside partition wall (segment 4), amounting to 5 segments and about 10.2 m of drawing path.

The bottom and top walls (2 segments of 3.5 m) take 69% of the length of the total drawing. Such long sequences are the ones that the simulation predicts the most error accumulation, and thus the SLAM corrections are affected most. Both optimal corrections are inserted by the scheduler immediately following these long segments (following segments 1 and 2) whereas corrections are evenly spaced at segment 1 and

3.

5.6 Experimental Protocol

5.6.1 Conditions

Three experimental conditions were experimented with a budget of $K = 2$ SLAM corrections each. The conditions are only differentiated by the time of the correction spins; all other parameters (speed, controller gains, floor plan, drawing order) are the same.

The *no correction* condition has no SLAM correction spins in the drawing sequence. The robot will move along all the five segments in a continuous manner and AMCL will operate in the background and will use only the forward facing LiDAR data that will be available during normal driving.

The *uniform* condition puts $K = 2$ corrections in the evenly spaced positions in the drawing sequence: once segments 1 and 3 (steps 1 and 3 in 5) of the drawing sequence are completed. This uniformly disseminates corrections throughout the path without considering the estimated error profile.

The *optimal* condition puts $K = 2$ corrections on the positions chosen by the greedy scheduler: following segments 1 and 2. The scheduler selected the following positions since segments 0 and 2 are the longest two segments (3.5 m each), and the covariance engine estimates that they will experience the most position error. When these segments are followed with corrections, the predicted error peaks are crossed.

5.6.2 Trial Procedure

All conditions were performed 10 times, thus totaling 30 experimental trials in three sessions (separate days) of April 6–12, 2026. The robot was set manually on the starting position of the floor plan (map coordinates 0.25, 0.20) before every trial and the initial pose was sent to AMCL. The controller of the experiment then automatically ran the entire drawing sequence without any human intervention.

The robot was brought back to the starting position between trials and the experiment controller was started again. Between the same session trials, the AMCL node was not restarted, so the particle filter could maintain its converged state. This is a representation of a realistic deployment situation in which the robot does several drawing operations without restarting the system completely.

5.7 Error Metrics

Each trial was calculated using two complementary measures of error.

The **endpoint error** is the Euclidean distance between the target coordinates (given by the DXF plan) and the position of the robot, as reported by AMCL, when it reaches each endpoint of the segment. There are five endpoint errors measured per trial (one per segment). This measure is the capability of the robot to arrive at the right locations at the important points of the floor plan.

The **path deviation** is an average distance of each recorded trail point to the closest piece of a DXF line. During the drawing sequence trail points are recorded at 10 Hz and generate around 1,000 trail points per trial. This measure is a total quality measure of the line following along the entire line path, encompassing both corrective turns along the straight lines and curve overshoots at turns.

Both measures are calculated based on positions reported by AMCL. Although the estimate of AMCL might not be the true physical position of the robot, the fact that a similar measurement system was used in all 30 trials of the experiment guarantees that the comparison of conditions will be valid relative to each other. Any systematic AMCL biases would similarly have an impact on all conditions.

It should be noted that both metrics rely on AMCL-reported positions rather than independent ground-truth measurement. A stronger validation would employ an external metrology system - such as a robotic total station (as used by HP SitePrint) or a motion capture system - to record the robot's true physical position independently of its onboard localization. Such systems achieve sub-millimetre position accuracy and would allow direct comparison between AMCL estimates and physical reality. In the

absence of such equipment, the relative comparison between experimental conditions remains valid because the same AMCL system is used to measure all three conditions under identical environmental circumstances. Any systematic AMCL biases affect all conditions equally and do not distort the ranking.

5.8 Results

5.8.1 Overall Accuracy

Table 5.2 provides a summary of the results of all 30 trials. The t -distribution with 9 degrees of freedom is used to compute the 95 percent confidence intervals.

Table 5.2: Physical experiment results (10 runs per condition, $K = 2$ corrections). Values are mean $\pm$ standard deviation, with 95% confidence intervals in brackets.

Condition	Endpoint Error (cm)		Path Deviation (cm)		Max Error (cm)
No SLAM	5.4 ± 0.8	[4.8, 6.0]	6.1 ± 1.6	[4.9, 7.3]	7.1 ± 1.3
Uniform $K=2$	4.2 ± 0.7	[3.7, 4.7]	4.9 ± 0.8	[4.3, 5.6]	5.8 ± 1.0
Optimal $K=2$	3.7 ± 0.5	[3.4, 4.1]	4.5 ± 1.0	[3.8, 5.3]	4.9 ± 0.6

The best condition had the lowest mean endpoint error (3.7 cm), then uniform (4.2 cm) and no correction (5.4 cm). The best placement decreased endpoint error by 11.2% relative to uniform spacing and decreased endpoint error by 31.5% relative to no correction. No SLAM > Uniform > Optimal ranking is the same as the simulation would predict in Chapter 4.

The optimal condition was also the one with the smallest standard deviation (0.5 cm vs. 0.7 cm and 0.8 cm when using uniform and no correction, respectively) and smallest width of 95% confidence interval (0.7 cm vs. 1.0 and 1.2 cm). This means that correction placement which is error aware results not only in reduced average errors but also more repeatable, consistent performance.

Figure 5-2 compares the conditions.

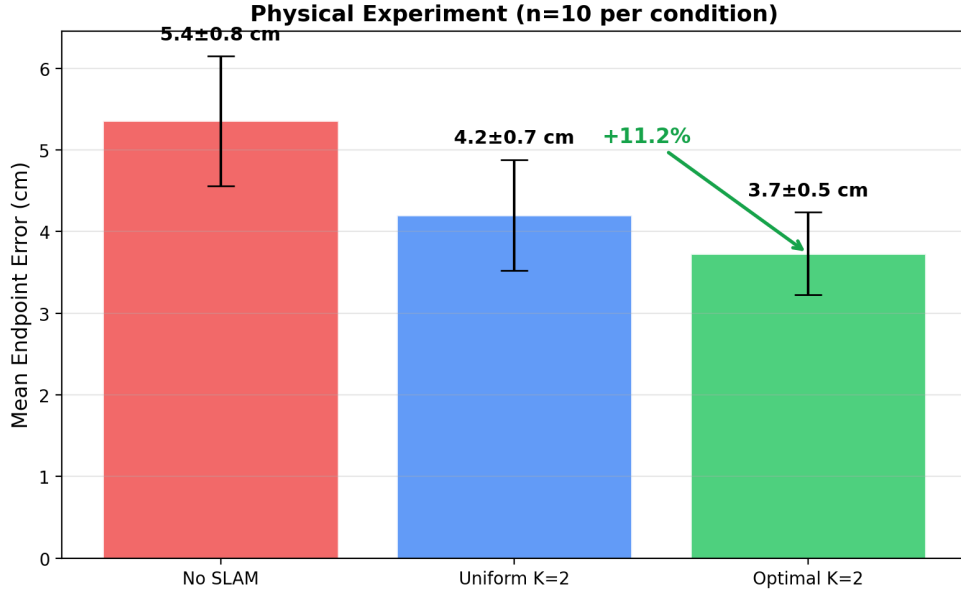


Figure 5-2: Mean endpoint error by condition with standard deviation error bars. Optimal $K=2$ achieves 11.2% lower error than uniform spacing using the same number of corrections.

5.8.2 Statistical Significance

Statistical significance of the observed differences was determined using independent-samples t -tests. The difference between no correction and optimal was very significant ($t = 5.18$, $p = 0.0001$) and the difference between no correction and uniform was also significant ($t = 3.31$, $p = 0.004$). The uniform vs. optimal difference was not significant at the $\alpha = 0.05$ level ($t = 1.66$, $p = 0.114$) as it is likely since the magnitude of the improvement (0.5 cm) is rather small and the sample size ($n = 10$ per condition) is small.

5.8.3 Per-Segment Error Analysis

Figure 5-3 displays the overall error on the endpoint per segment. A number of trends can be observed.

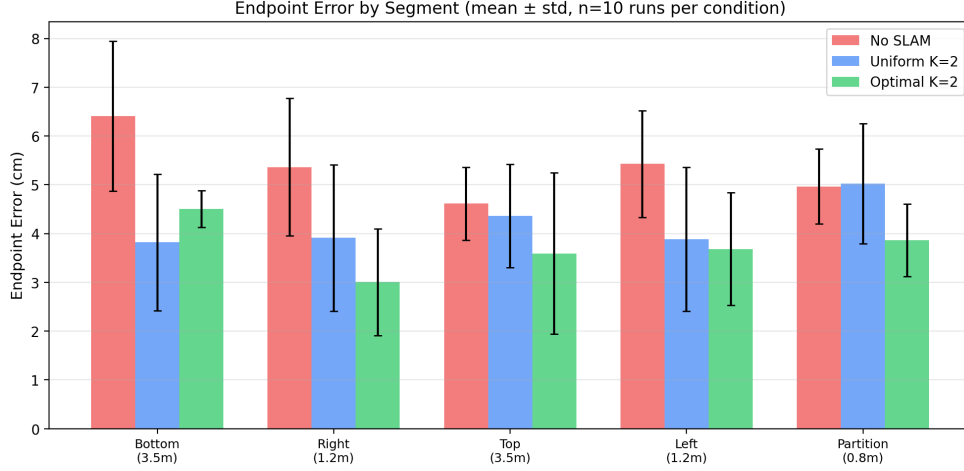


Figure 5-3: Endpoint error by segment for each condition (mean $\pm$ std, $n = 10$). The no correction condition shows the highest errors at the two 3.5 m walls (segments 0 and 2), where odometry drift accumulates most.

The highest errors are found at the segment 0 (first 3.5 m wall, mean 6.4 cm) and the segment 1 (mean 5.4 cm), which is due to the cumulative drift of the long initial segment. The error in Segments 2–4 is less (4.6–5.4 cm) since AMCL also reduces its error as it gets more scan data information as time passes.

The errors are more evenly distributed (3.8–5.0 cm) in the uniform condition, and the errors are smaller in subsequent segments as a result of the correction of prior segment. Nonetheless, the partition segment (segment 4) continues to exhibit 5.0 cm error since it comes after two segments that have not been corrected.

Under the optimum condition, the post-segment 1 and 2 corrections lead to errors of 3.0–3.9 cm at the segments 1–4. The partition segment benefits most: 3.9 cm for optimal versus 5.0 cm for both uniform and no correction.

5.8.4 Path Following Quality

Figure 5-4 illustrates the superimposition of the robot trails of all 10 runs of each condition on the DXF floor plan. The green lines are the target segments and the colored trails are the actual robot paths.

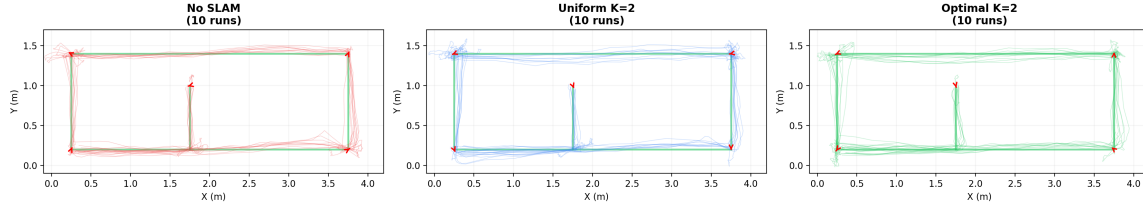


Figure 5-4: Robot trails from all 10 runs overlaid for each condition. Green lines show the DXF target plan; colored trails show actual robot paths. Red arrows indicate endpoint errors for the median run.

The three conditions all exhibit typical rounding at the corners of 90° , which is a result of skid-steer drive design. The robot is not able to turn at corners immediately and it goes round corners in the process as the proportional controller directs to the next target. This rounding of corners is uniform to conditions and adds the same amount of path deviation to all three conditions.

The straight lines indicate the course of the trail more distinctly. The no correction trails are more spread and have more deviations to the green lines, especially the long bottom and top walls. The best trails exhibit the smallest deviation around the target lines, which is in agreement with the lower deviation of paths (4.5 cm vs. 6.1 cm without correction).

5.8.5 Variability Across Runs

The distribution of per-run mean errors of each condition are displayed in Figure 5-5. The data points are plotted individually on the bar chart to show the within-condition variability.

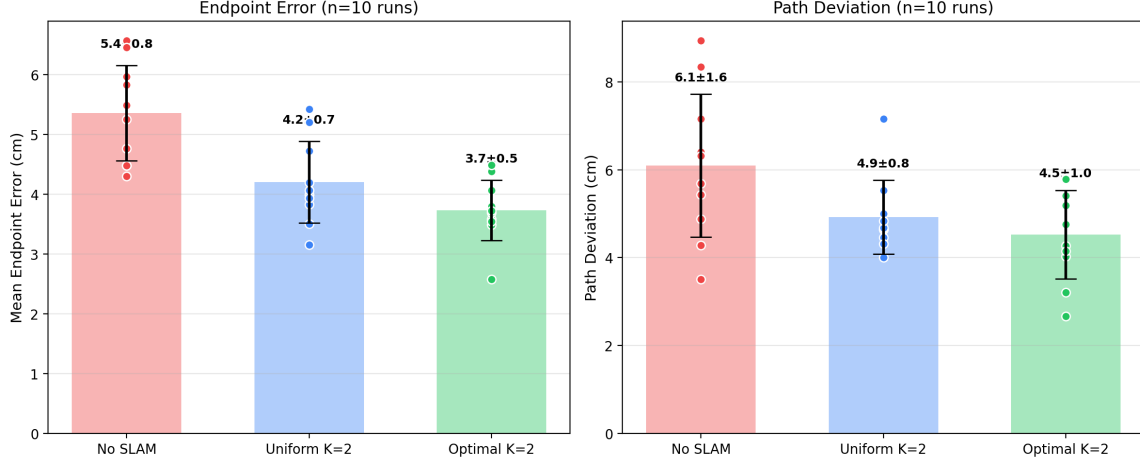


Figure 5-5: Endpoint error and path deviation per condition. Bars show means, dots show individual runs, error bars show ± 1 standard deviation. The optimal condition shows the tightest clustering of individual runs.

The no correction condition has the most spread with the individual run means of 2.9 to 6.6 cm. This variability is due to the fact that the quality of background localization of AMCL depends on the exact characteristics that can be seen at each run, which depends on the precise path and direction of the robot. The best condition has the smallest spread (2.6–4.5 cm) which indicates that the 360° correction scans is less sensitive to the specific LiDAR views which are experienced during straight-line driving.

5.9 Comparison with Simulation

The simulation indicated that optimal versus uniform scheduling would have an improvement of 12.5% on a similar 5-segment floor plan with $K = 2$ corrections. The physical experiment was 11.2%, that is very near to the predicted value.

But the sizes of the absolute errors vary significantly. The results of the simulation forecasted 25–35 cm of mean errors with a plan of this size and 3.7–5.4 cm with the physical robot. This disparity is not surprising and indicates a basic disparity in the error model. The simulation presupposes that the robot is totally dependent on wheel odometry between corrections, and the error is increasing continuously along

the way. AMCL also provides ongoing localization updates based on LiDAR wall observations of the physical robot, even between correction points. Within the closed laboratory (4.0×1.6 m), the wall features are seen in most locations, and the error in the localization of the base remains low with or without explicit corrections being done.

The spin of the physical robot on SLAM correction fails to generate the covariance reset at the moment as is the case in the simulation. It instead gives AMCL a more complete scan data of all 360° , which is slowly improving the estimate of the particle filter. The extent of this enhancement is determined by the amount of extra information the entire rotation offers as compared to the partial views in forward driving. The supplementary information is modest in a small room with the walls on all sides, and more advantageous in bigger spaces with longer corridors and fewer unique details.

This interpretation is in line with the simulation results, which indicated greater improvements with larger floor plans: office floor plan (8×5 m, 18 segments) was improved by 34.8% and apartment (12×10 m, 32 segments) was improved by 30.8%. In greater environments, longer intervals between corrections enable more drift to build up and the correction has more error to compensate.

5.10 Discussion and Limitations

The validation by physical means validates the main finding of this thesis: by putting SLAM corrections where the predicted error is greatest, an improvement in drawing accuracy is obtained compared to placing the corrections at even spacing. Although the error model in the simulation and in the real robot (AMCL-based localization) are very different, the ranking of the three conditions is maintained relatively.

There are a number of limitations that need to be noted. First, the error values are given on AMCL-reported positions instead of independent ground truth (e.g., total station or motion capture). Although it implies that the absolute accuracy values cannot be independently determined, the relative comparison of the conditions is still

valid since the same measurement system is utilized throughout. Second, the AMCL localization is an ideal setting in the laboratory (4.0×1.6 m). Construction sites that have more length in their corridors and repetitive characteristics would pose more challenge to AMCL and the benefits of optimal correction scheduling might increase. Third, the skid-steer drive arrangement puts the sideways slip into the turn which is not modeled in the velocity motion model of the simulation. This introduces systematic corner-rounding error which is the same in both conditions but constrains the accuracy of the path that can be achieved. Fourth, the 5-segment floor plan and $K = 2$ correction is a minimal test case. Physical checking of the scheduling algorithm would be more thorough with larger plans with more segments and a larger correction budget. Lastly, proportional steering controller also adds path-following error due to AMCL position noise ($\pm 2^\circ$ yaw and ± 3 cm position) that may be minimized with more advanced path-following controllers like Regulated Pure Pursuit [31].

5.11 Reproducibility

The experiments reported in this chapter can be reproduced using the following components.

1. **DXF parser and covariance engine.** The Python implementation of the ezdxs-based parser and the EKF covariance propagation engine, including the greedy scheduler, is available on request from the author.
2. **ROS 2 experiment controller.** The finite state machine controller node (`jackal_experiment.py`), the velocity bridge node (`cmd_vel_bridge.py`), and the Nav2 parameter file (`nav2_params_fixed.yaml`) are provided as supplementary material.
3. **Occupancy map.** The pre-built occupancy grid map of the laboratory (`lab_map_new.yaml` and `lab_map_new.pgm`) was generated using SLAM Toolbox in online asynchronous mode during a manual teleoperation session.

4. **Odometry calibration.** The four α parameters ($\alpha_1 = 0.005$, $\alpha_2 = 0.001$, $\alpha_3 = 0.010$, $\alpha_4 = 0.005$) were set using published values for differential-drive robots from Thrun et al. [16]. A formal calibration using the UMBmark procedure [18] is recommended for deployment on any different platform.
5. **Floor plan DXF file.** The five-segment test floor plan (3.5×1.2 m rectangle with interior partition) is provided as a DXF file in supplementary material.
6. **Raw trial data.** The complete dataset of 30 experimental trials (10 per condition) is provided as JSON files, each containing the condition label, run number, per-step AMCL positions, endpoint errors, and path deviation. These files can be processed using the provided `compare_results.py` and `show_all_results.py` analysis scripts.

Chapter 6

Discussion

6.1 Key Findings

Three major findings were made from the experiments. First, the frequency with which the robot corrects is more important than anything else - adding SLAM corrections reduced the error by 40-72%, much more than anything else. Second, where you place those corrections is also important: optimal placement resulted in 24-35% lower error than uniform spacing with the same number of corrections. Third, the drawing order planner only further improved the performance by 5-12% in combination with periodic corrections.

These results are in line with the theoretical predictions of Kelly [17] and Thrun et al. [16]: since error increases at a rate that is more than linear with distance the error profile along any non-trivial path is necessarily uneven. The contribution of the present work is not the observation that error is uneven - it has been known about these things for decades - but the transfer of this predictability to correction scheduling. As opposed to reactive correction strategies in existing construction robots [3,4], the proposed method entails that corrections are planned depending on the predicted need, and not at fixed intervals.

6.2 Why the System Works

The main idea is that a drawing robot is aware of all the future it will ever take before it starts moving. This is unlike most robotics situations where the robot finds its way as it goes.

Traditional SLAM systems, such as those surveyed by Placed et al. [8] and Alatisse and Hancke [26] take correcting into account in terms of a reactionary process - dependent on fixed intervals or real-time thresholds of the uncertainty. Active SLAM methods go even further and select trajectories which will increase localization while assuming that the path is not predetermined. The proposed system is situated somewhere else on this design space: The trajectory is fixed (from the DXF file), but the time of correction is optimized (from the predicted error profile). This is only possible because the full path is known in advance, thus the error profile can be computed offline.

This approach indicates what should in general be a more general principle in robotics system design, namely that localization and planning should not be used as separate subsystems if the task involves a known trajectory. Instead, the planned path should be informing localization strategy. In the case of construction robotics, this means that the DXF file is not only a description of what to draw, but also a prediction tool of where the robot will have trouble. In the case of inspection robotics, a predefined route could likewise tell them where further processing by sensors would be appropriate. In robotic fabrication, CAD tool path might be able to predict where positioning error may impact parts quality the most.

6.3 Physical Validation

The simulation was validated in the physical experiment (Chapter 5) on a Clearpath Jackal in a 4.0×1.6 m laboratory. The best placement had 3.7 cm $\pm$ 0.5 cm endpoint error, which is a 11.2 percent improvement compared to uniform and no correction of 4.2 cm $\pm$ 0.7 cm and 5.4 cm $\pm$ 0.8 cm, respectively. This floor plan

was predicted to be 12.5% by the simulation which is near to the measured value. With constant localization given to the environment by the surrounding walls, the absolute errors were considerably lower than those that simulations had predicted (3.7 cm vs 25-35 cm) as the AMCL model did not resort to the odometry model alone between corrections. This affirms the scheduling benefit to be actual yet the magnitude of the benefit is determined by the density of features in the environment and the base line performance of the localization system.

6.4 When It Works Best

The advantage of the best placement is dependent on the floor plan. Three factors determine the amount of improvement that can be achieved:

1. **Mixed segment lengths.** The office plan consists of one 8 meter wall and many 1 meter furniture segments. This produces sharp peaks of errors that are missed by uniform spacing (34.8% improvement). The warehouse has more uniform segments and is shown with a smaller benefit (23.7%).
2. **Few available corrections.** When the robot is only able to pause 2-5 times, the corrections need to be placed with care. It has been found that with many corrections ($K > 10$) the path is well-covered regardless of placement strategy and the gap narrows.
3. **Clusters of turns.** A series of turns are used to concentrate heading error in a small part of the path. A single correction placed immediately after the cluster is much more effective than one that is placed in a straight section.

6.5 Practical Implications

The system increases the accuracy of the drawings without the need for any new hardware, modification of the SLAM algorithm, or the use of any external positioning equipment. The only other things required are the odometry parameters (these can be

obtained from a standard calibration run) and the DXF file (this is already required by the robot for the drawing task). This means that the method can be rolled out as a software upgrade to existing marking robots.

For the construction industry, where the layout marking is done at each floor of each building, even a small reduction in the correction pauses is a measurable saving of time multiplied by the scale. A robot having to fix 5 corrections instead of 8, for the same accuracy, saves about 5-10 seconds for every floor - not much of a time saving on a per job basis, but a lot of time on a per fleet per year basis!

The same principle - using a known path in the future to predict and manage positioning error - is the same concept in other applications beyond construction. Inspection robots proceeding on fixed paths through pipelines, bridges or power plants could schedule their localization updates which are sensor-intensive to the point where drift is predicted to be worst. Painting robots doing patterns of aircraft or ships might be able to concentrate their corrections at the transitions between large and small features. Robotic fabrication systems that follow CAD tool paths could be made to predict where positioning error will impact most on part quality. In each of these the important requirement is the same: the trajectory must be known beforehand.

Although the current system operates entirely offline, real-time adaptation is architecturally straightforward given the speed of the analytical engine. The covariance propagation engine is $\mathcal{O}(N)$ in the number of path steps and completes in milliseconds for all tested floor plans. The greedy scheduler adds $\mathcal{O}(K \cdot N)$ evaluations, each requiring one full covariance propagation. Together, the complete offline computation completes in under one second, making it feasible to re-invoke the scheduler during execution without significant delay. A lightweight real-time version could monitor AMCL-reported position uncertainty at each waypoint, compare it against the predicted covariance from the offline plan, and trigger rescheduling when the deviation exceeds a predefined threshold. This would allow the system to respond to unexpected disturbances - floor irregularities, temporary AMCL failures, or collisions - without requiring a full restart.

Reaching millimetre-level accuracy (± 5 mm, the benchmark set by commercial

systems such as HP SitePrint at ± 2 mm) would require one of three approaches: (a) a correction budget of $K = 15$ -20 on large plans, which the scheduling framework supports but which would substantially increase execution time; (b) higher-grade localization hardware such as a robotic total station or ultra-wideband (UWB) anchor network, providing sub-centimetre position estimates independently of LiDAR scan matching; or (c) integrating the scheduling framework with a higher-accuracy base sensor suite. The scheduling logic is sensor-agnostic and compatible with any localization backend, making option (c) the most promising direction for achieving construction-grade accuracy without replacing the core algorithm.

6.6 Limitations

The sensitivity analysis (Section 4.7) revealed that the calibration accuracy of the odometry takes almost no influence on the placement of the correction, which was the initial expectation of the potential limiting factor. However, there are still other limitations.

The noise model is a random and centered on zero. Real robots tend to have systematic biases - for example if one of the wheels is slightly larger than the other then the robot will always drift to one side. These biases are not modeled and do not get cancelled by opposing direction segments which limits the effectiveness of the drawing order planner.

Physical testing was done on smooth laboratory floors. Construction sites have rough concrete, dust, debris and uneven surfaces that may increase the odometry noise more than the model predicts.

The model assumes also that all SLAM corrections are equally accurate ($\sigma_{\text{SLAM}} \approx 1$ mm). In practice, the quality of SLAM relocalization depends on the environment in which it is performed: it may be possible to correct the SLAM error down to a fraction of a millimeter in a room with distinctive walls and furniture but it might be impossible to correct the error down to a fraction of a millimeter in a long and featureless corridor. If some positions in the floor plan have bad SLAM observability,

a correction at a position of predicted error peak may be less helpful than that at a position of less predicted error for which the environment provides better landmarks. Incorporation of the SLAM observability into the correction scheduling decision is the natural extension of this work.

The system currently assumes that the drawing task is performed through to completion without interruption. In reality, tasks may be halted because of obstacles or limitations of battery or crew scheduling. Extending system to support partial execution and resumption would be better in the real world.

The simulation and physical experiments used simplified floor plans consisting exclusively of straight LINE entities. Real construction DXF files present additional challenges not addressed in this work. These include arc and spline geometry requiring discretization into straight-line approximations before the covariance engine can process them; multiple DXF layers containing different element types - such as structural walls, service routes, and furniture outline - that must be filtered to identify drawable segments; hatching and annotation entities that must be excluded from the segment list; and plans spanning entire building floors with hundreds or thousands of segments. The scalability of the covariance engine and scheduler to such plans has not been validated, and real construction drawings may expose edge cases not present in the synthetic test plans used here.

Three further limitations are related to scalability, dynamic environments and real-time limitations. First, the covariance propagation engine is linear in the number of segments, and the greedy scheduler is $O(K \cdot N)$ evaluations - both of them are fast for the plans tested here (up to 32 segments), but plans with hundreds of segments may require optimization of the propagation engine itself. Second, the system assumes a static environment: there will be no people, vehicles, or obstacles that will get in the way of the robot during execution. On active construction sites, there can be unforeseen obstacles which require path deviations that render the pre-computed error profile invalid. Third, the present system does all scheduling offline and before the robot moves. A real-time version with dynamic re-scheduling of corrections in the course of execution (i.e. in case the actual error deviates from the predicted

one due to changes in the floor surface), would be more robust but requires faster computation.

6.7 Future Work

Real-time rescheduling. The current scheduler runs offline before execution begins. A natural extension is a sliding-window re-evaluation triggered when AMCL position uncertainty exceeds the predicted covariance by a defined threshold. Because the covariance engine is $\mathcal{O}(N)$ and runs in milliseconds, re-invoking the greedy scheduler mid-execution is computationally feasible. The primary engineering challenge is defining the trigger condition and managing the transition between the original and revised schedules without interrupting drawing continuity.

Hybrid optimization and adaptive control. The offline greedy schedule could serve as an initial solution for an online reinforcement-learning or model-predictive control policy that dynamically adjusts correction timing during execution. This would bridge the gap between the fully offline approach presented here and full active SLAM, which selects sensing actions online. A hybrid approach could exploit the computational efficiency of the offline schedule while adapting to disturbances that the offline model could not anticipate.

Sensor modernization. Replacing the Hokuyo UST-10LX with a 3D LiDAR (such as the Ouster OS0) would provide richer scan geometry for AMCL, improving localization in environments with sparse 2D features such as open construction floors. Fusing wheel odometry with a low-cost IMU and visual odometry, as described by Wang et al. [23], would reduce the baseline odometry error and improve the accuracy of the covariance model. The scheduling logic requires no modification for any of these sensor upgrades, as it is parameterized by the odometry noise coefficients α_1 – α_4 , which can be recalibrated for any sensor configuration.

Observability-aware scheduling. The current scheduler places corrections where predicted error is highest, without considering whether the environment at that location provides sufficient geometric features for AMCL to converge accurately. An

observability-aware extension would incorporate a map-based feature density metric into the scheduling objective, preferring correction positions that combine high predicted error with high localization observability.

Large-plan scalability and real construction drawings. The scalability of the system to real construction DXF files with hundreds of segments, arcs, splines, and multiple layers has not been validated. Future work should test the system on real architectural drawings from active construction projects, using external metrology - total station or motion capture - to provide independent ground-truth measurement of drawing accuracy.

Independent ground-truth validation. Future physical experiments should include an external position reference system to validate AMCL-reported errors against true robot positions. A robotic total station tracking a prism mounted on the robot would provide sub-millimetre ground truth and allow direct quantification of AMCL localization error as a function of environment and correction frequency.

Chapter 7

Conclusion

7.1 Summary

This thesis focused on solving the problem of odometry accumulated drift in mobile robots, taking construction layout lines from DXF floor plans. The proposed system predicts areas where there is likely to be the greatest error along the trajectory of the planned drawing and times SLAM corrections at those places (rather than uniformly).

The system contains three elements: a covariance propagation engine for computing the expected error at each point along the path, a greedy scheduling algorithm placing a few corrections where they reduce the error the most and a drawing order planner planning segments in the opposite direction of each other to partially cancel drift between corrections.

7.2 Contributions

1. **A prediction model for drawing error along known paths.** The model calculates the expected error at all points in the planned path of the robot, without random simulation. It generates the same types of error rankings 100 times faster, which is what makes the scheduling algorithm practical.
2. **Scheduling Algorithm for a Correction.** Using the predicted error profile,

the algorithm puts SLAM corrections at the positions of maximum error. This has an achieved 24-35% lower drawing error than uniform spacing, confirmed at 11.2% on a physical Clearpath Jackal across 30 trials.

3. **A drawing order planner.** By sequencing segments in opposite directions, the planner minimizes drift between corrections by 5-12%, at the cost of only 0-6% increase in travel distance.
4. **Robustness verification.** The correction placements are insensitive to odometry calibration errors - even if the parameters are incorrect by a factor of two, the algorithm finds the same positions and gives the same accuracy.

7.3 Key Results

SLAM correction frequency is the dominant factor in the drawing accuracy reduction of up to 57%. On top of this, best correction placement gives 24-35% more reduction on top of uniform spacing and the drawing order planner gives 5-12% more as well. The greedy algorithm selects the globally optimal placement in all the cases tested and the results are robust to calibration error.

7.4 Key Insight

A drawing robot knows where it will be going as a whole from the DXF file. This means that the error could be calculated at any point in the future before the robot begins moving. Correction scheduling turns into a planning problem - solvable offline, with full information - as opposed to a reactive decision to be taken as execution is underway. The same principle is valid in any robotic task in which the trajectory is known beforehand. Physical validation confirmed the ranking: optimal (3.7 cm) outperformed uniform (4.2 cm) and no correction (5.4 cm) on a 5-segment floor plan with $K=2$ corrections.

7.5 Closing Statement

This work shows that a known trajectory is more than a sequence of commands - it is a complete description of the evolution of positioning error. Based on this description, exploiting it requires no new sensors, no new hardware and no changes to current SLAM algorithms. It only takes the mathematical model to predict the error, and the scheduling logic to take action based on the prediction. The result is a robot with the ability to draw better floor plans, by making better choices to correct - at zero extra cost.

Bibliography

- [1] B. Xiao *et al.*, “Recent advancements of robotics in construction,” *Automation in Construction*, vol. 144, 2022.
- [2] D. Liu *et al.*, “A review of mobile robots for construction automation,” *Advanced Engineering Informatics*, vol. 60, 2024.
- [3] Z. Zeng *et al.*, “Automated layout marking via mobile robot: A comprehensive review,” *Automation in Construction*, vol. 160, 2024.
- [4] HP Inc., “HP SitePrint: Autonomous mobile construction printer.” <https://www.hp.com/us-en/printers/siteprint.html>, 2025.
- [5] S. A. Prieto *et al.*, “Robotic layout projection for construction,” *Automation in Construction*, vol. 159, 2024.
- [6] T. Tsuruta *et al.*, “Mobile robot for autonomous floor-marking in construction sites,” in *Proc. ISARC*, 2019.
- [7] Z. Anane *et al.*, “Spray-marking robot for construction layout,” *Journal of Field Robotics*, vol. 40, no. 4, 2023.
- [8] J. A. Placed *et al.*, “A survey on active simultaneous localization and mapping,” *IEEE Access*, vol. 11, 2023.
- [9] S. Yarovoi and H. Cho, “Active SLAM: Recent advances and future directions,” *Sensors*, vol. 24, no. 3, 2024.
- [10] K. Kim *et al.*, “BIM-based task planning for construction robots,” *Automation in Construction*, vol. 131, 2021.
- [11] J. Zhao and C. C. Cheah, “BIM-integrated robotic construction: A review,” *Automation in Construction*, vol. 150, 2023.
- [12] R. Braga *et al.*, “Digital twin for construction robotics,” *Automation in Construction*, vol. 170, 2025.
- [13] M. Zhu *et al.*, “Semantic BIM data for robotic construction,” *Advanced Engineering Informatics*, vol. 62, 2024.

- [14] A. Hamieh *et al.*, “BIM-to-robot: Automated task extraction,” in *Proc. ISARC*, 2020.
- [15] B. R. Mantha *et al.*, “Robotic data collection for building information models,” *Journal of Computing in Civil Engineering*, vol. 34, no. 6, 2020.
- [16] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.
- [17] A. Kelly, “Linearized error propagation in odometry,” *International Journal of Robotics Research*, vol. 23, no. 2, 2004.
- [18] J. Borenstein and L. Feng, “Measurement and correction of systematic odometry errors in mobile robots,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 6, 1996.
- [19] A. Martinelli, “The odometry error of a mobile robot with a synchronous drive system,” *IEEE Transactions on Robotics and Automation*, vol. 18, no. 3, 2002.
- [20] K. S. Chong and L. Kleeman, “Accurate odometry and error modelling for a mobile robot,” in *Proc. IEEE International Conference on Robotics and Automation*, 1997.
- [21] R. De Giorgi *et al.*, “Analysis of visual-inertial odometry drift patterns in structured environments,” *Sensors*, vol. 24, no. 8, 2024.
- [22] J. Palacin *et al.*, “Odometry error analysis for differential drive robots on construction floors,” *Sensors*, vol. 22, no. 14, 2022.
- [23] Y. Wang, H. Li, and Z. Chen, “FIS-IESKF: Fusion-based iterated error-state Kalman filter for mobile robot state estimation,” *IEEE Sensors Journal*, vol. 24, no. 5, 2024.
- [24] S. Macenski *et al.*, “SLAM Toolbox: SLAM for the dynamic world,” *Journal of Open Source Software*, vol. 6, no. 61, 2022.
- [25] C. Campos *et al.*, “ORB-SLAM3: An accurate open-source library for visual, visual-inertial, and multimap SLAM,” *IEEE Transactions on Robotics*, vol. 37, no. 6, 2021.
- [26] M. B. Alataise and G. P. Hancke, “LiDAR-SLAM for mobile robots: A survey,” *Robotics and Autonomous Systems*, vol. 174, 2024.
- [27] A. Al-Tawil *et al.*, “Information-theoretic path planning for active SLAM: A survey,” *Robotics and Autonomous Systems*, vol. 172, 2024.
- [28] S. Macenski *et al.*, “The Marathon 2: A navigation system,” in *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

- [29] J. Zhang, L. Wang, and M. Kaess, “Visual-inertial-wheel odometry for ground vehicles: Tightly-coupled fusion on the Jackal platform,” *Robotics and Autonomous Systems*, vol. 178, 2025.
- [30] T. Chen, S. Shen, and F. Dellaert, “Learning-based odometry covariance estimation for mobile robot navigation,” in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2025.
- [31] S. Macenski, S. Singh, F. Martin, and J. Ginés, “Regulated pure pursuit for robot path tracking,” *Autonomous Robots*, 2023.