A Coverage-Driven Systematic Test Approach for Simultaneous Localization and Mapping

Philip Tasche
University of Twente
Enschede, The Netherlands
p.b.h.tasche@utwente.nl

Paula Herber
University of Münster
Münster, Germany
paula.herber@uni-muenster.de

***Abstract*—Simultaneous localization and mapping (SLAM) is a prerequisite for accurate navigation of autonomous vehicles. Although this is often safety-critical, systematic approaches for testing the correctness and accuracy of SLAM algorithms are missing. In this paper, we present an approach for automated and systematic testing of SLAM algorithms. We identify challenging environmental features for SLAM, define coverage criteria that characterize the SLAM problem's input space, and develop a method for automatically generating high-coverage tests. We demonstrate the effectiveness of our approach with a case study on an existing FastSLAM implementation.**

***Index Terms*—Autonomous Vehicles, Simultaneous Localization and Mapping, Test Automation, Coverage-driven Testing, Input Space Partitioning**

I. Introduction

With the advent of self-driving cars and autonomous vehicles, accurate navigation has become a safety-critical issue. Prerequisites for accurate navigation are a precise map and precise localization. This makes it crucial to ensure the correctness of algorithms for simultaneous localization and mapping (SLAM), for example by executing them with sample input data in simulation or in a test environment. This process is called SLAM testing.

To detect as many faults as possible, established software testing approaches typically aim at a high coverage of either the implementation (*structural testing*) or the input space (*functional testing*). Due to the lack of structure in their implementations [1], structural testing is not very effective for probabilistic SLAM algorithms, as even simple test cases can achieve complete structural coverage without triggering interesting behavior. In contrast to this, the idea of functional testing is to partition a given input or output domain into disjoint sub-domains such that all elements in one equivalence class are expected to provoke the same system behavior [2]. Then, by choosing one representative of each partition, a systematic coverage of the input space can be achieved with a reasonably small number of test cases. The selection of test cases from equivalence classes can be made, for example, by using border values, testing special values or randomly selecting test cases [3] [4]. The main challenge in applying functional testing to SLAM is the complexity of the input space, i.e. complex maps and sensor data, and SLAM algorithms' hard-to-predict behavior.

In this paper, we present an approach to partition the input space of the SLAM problem into equivalence classes such that we can automatically derive a set of interesting test cases with a high coverage with regard to these equivalence classes. Our key idea is threefold: First, we identify parameters that influence how challenging a test case is for SLAM algorithms, such as map size and sensor accuracy, but also parameters that can expose common flaws in SLAM implementations, such as phases of inactivity. Second, we define domain-specific coverage criteria by dividing these parameters' domains into equivalence classes in a declarative style. Third, we propose a method to automatically construct challenging test cases with a high coverage of the input space based on these coverage criteria. We demonstrate the applicability of our approach by generating test cases for an existing FastSLAM implementation [5]. Our coverage-driven approach improves the error detection capability for this case study and a number of example faults.

The rest of this paper is structured as follows: Sec. II introduces preliminaries, Sec. III discusses related work. Sec. IV presents our systematic test approach, Sec. V our coverage criteria and Sec. VI our coverage-driven test case generation. Sec. VII presents experimental results. Sec. VIII concludes.

II. Simultaneous Localization and Mapping (SLAM)

The aim of SLAM is to estimate both the state of a vehicle and the state of its environment without a priori knowledge of either. Since a SLAM algorithm must rely on noisy information about vehicle movement and sensor observations, this estimate takes the form of a probability distribution over all possible states. In the following, we briefly introduce the fundamentals of SLAM following [1], a selection of SLAM algorithms, including FastSLAM [6], and the particle deprivation problem.

A. Probabilistic Formulation of SLAM

The SLAM problem is characterized by the following random variables: The *vehicle position* is denoted by x_t at time t. The *map* is denoted by m and describes the state of the environment. It is independent of time as it is assumed to be static. It can be modeled as a *grid map* that independently estimates the occupancy probability of each grid cell, or as a *feature map* that estimates a number of landmark locations. *Motion information* is denoted by u_t and describes information about the vehicle's movement at time t, such as velocity or wheel rotation counts. *Sensor information* is denoted by z_t and describes the information gathered by the sensors about the environment at time t, such as range and bearing information to a nearby landmark.

The goal of an online SLAM algorithm, i.e. one that runs at exploration time, is to estimate the

Algorithm 1 Bayes Filter Algorithm

```
1: procedure BAYES FILTER($bel(s_{t-1})$, $z_t$, $u_t$)
2:     $\overline{bel(s_t)} = \int p(s_t|s_{t-1}, u_t)\ bel(s_{t-1})\ ds_{t-1}$
3:     $bel(s_t) = \eta\ p(z_t|s_t)\ \overline{bel(s_t)}$
4: end procedure
```

vehicle position x_t and map m based on sensor data $z_{1:t}$ and motion data $u_{1:t}$. Its belief bel is defined by a probability density function (PDF) p [1]:

$$bel(x_t, m) = p(x_t, m|z_{1:t}, u_{1:t})$$

Recursive Bayes Filter (RBF): An RBF is an abstract algorithm for online estimation of the PDF p under the assumption that the state behaves as a Markov process, i.e. the future is conditionally independent from the past [1]. Algorithm 1 displays the general algorithm for a state vector s [1] (in the context of SLAM, $s = (x_t, m)$). The central idea is to incorporate the input data in two steps. The *prediction step* in Line 2 predicts the current state based on the previous state and the measured motion. For all possible previous states s_{t-1}, it combines the probability $bel(s_{t-1})$ that the previous state of the system was s_{t-1} with the probability $p(s_t|s_{t-1}, u_t)$ that it evolved to s_t by executing the motion measured by u_t. This gives an initial approximation $\overline{bel(s_t)}$ of the PDF. The *correction step* in Line 3 uses the sensor observations to refine this approximation by taking into account the likelihood that the observation z_t would be made in state s_t.

B. Particle Filter

A *particle filter* is an implementation of the recursive Bayes filter. It uses a nonparametric method such as *Monte Carlo Localization* (MCL) [7] to estimate the state PDF by approximating it with weighted samples, called *particles* [1].

Like the Bayes filter, the particle filter algorithm (Algorithm 2) can be divided into two main steps. First, each particle in the previous particle set $\mathscr{X}_{t-1}$ is propagated according to the vehicle motion by sampling a possible new position from the *motion model* $p(x_t|x_{t-1}, u_t)$ in Line 4. To correct the prediction according to observation information, the predicted particle set is filtered in a process

Algorithm 2 Particle Filter Algorithm

```
1:  procedure PARTICLE FILTER($\mathscr{X}_{t-1}$, $z_t$, $u_t$)
2:    $\overline{\mathscr{X}}_t = \mathscr{X}_t = \emptyset$
3:    for $i = 1$ to $|\mathscr{X}_{t-1}|$ do
4:      sample $x_t^i \sim p(x_t|\mathscr{X}_{t-1}[i], u_t)$
5:      $w_t^i = p(z_t|x_t^i)$
6:      add $\langle x_t^i, w_t^i \rangle$ to $\overline{\mathscr{X}}_t$
7:    end for
8:    for $i = 1$ to $|\overline{\mathscr{X}}_t|$ do
9:      draw $k \in \{1, ..., |\overline{\mathscr{X}}_t|\}$ with probability $\propto w_t^k$
10:     add $x_t^k$ to $\mathscr{X}_t$
11:   end for
12: end procedure
```

called *resampling*. In Line 5, each particle is given a weight proportional to the *observation model* $p(z_t|x_t)$, meaning the probability that the given observation was made from the predicted position. In Lines 9 and 10, the algorithm then draws with replacement from the predicted particle set $\overline{\mathscr{X}}_t$ with probability proportional to the particle weight until the desired number of particles have been selected. The propagation and resampling together model the actual probability distribution over the problem domain. As the number of particles increases, their distribution approaches the true distribution [1].

FastSLAM: A prominent example of a particle filter is the *FastSLAM* algorithm [6]. Particles represent vehicle location histories to approximate the vehicle's state, using an algorithm similar to MCL [6]. FastSLAM separately estimates landmark locations with extended Kalman filters [1].

C. Particle Deprivation Problem

A particle filter's resampling step is vital to incorporate observations into the belief. However, filtering out particles can lead to loss of particle diversity. This can reduce the accuracy of the approximation [1]. To avoid this problem, the particle filter should resample only when necessary, use low-variance resampling [1] or a more accurate prediction step [8].

III. Related Work

The current standard for validating SLAM algorithms is execution on a public data set (e.g. [9] [10] [11]) or in a real or simulated sample environment (e.g. [12]). For example, MCL [7] and FastSLAM [6] have been validated in physical test environments. FastSLAM 2.0 [8] and GMapping [13] [14] used public data sets. These techniques remain standard in newer work, e.g. [15] [16] [17] [18], but they do not systematically cover the input space and may fail to detect errors in corner cases.

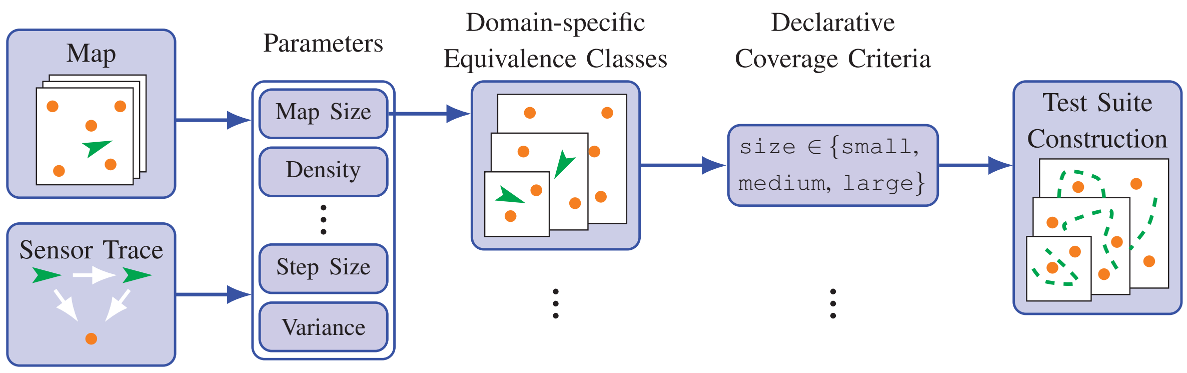
There have been some efforts to improve both the validity and the repeatability of SLAM tests. [12] and [19] provide metrics for SLAM evaluations. [20] defines principles for improving the quality of experimental robotics research by making testing efforts more repeatable and [21] seeks to implement these principles by using a set of floor plan maps and simulation software to generate a variety of test data sets. [22] [23] [24] provide a benchmarking framework for the performance and accuracy of SLAM algorithms. However, these works do not provide a systematic test approach.

The testing literature contains some approaches for testing difficult domains, for example, with evolutionary testing [25]. However, a literature survey on evolutionary testing by Rodrigues et al. [26] mentions no such approach for SLAM testing. [27] [28] use parameter-based equivalence class partitioning to test satellite image processing. However, their results are not applicable to SLAM.

Combinatorial testing [29] [30] [31] approaches are also related to our work. They are concerned with combinations and interactions between input parameters. To the best of our knowledge, combinatorial testing has not yet been applied to SLAM. We consider it to be complementary to our approach.

There has also been some work to identify and use environmental features that influence the accuracy of SLAM algorithms. [32] [33] analyze the relation between geometric features and the observed performance of SLAM algorithms to predict their performance in an unseen environment. [34] provides a challenging visual data set. These works identify challenging features but do not use these features for systematic test generation.

To the best of our knowledge, there exists no approach that enables systematic SLAM testing using coverage-driven test case generation.



Map
Sensor Trace
Parameters
Map Size
Density
Step Size
Variance
Domain-specific Equivalence Classes
Declarative Coverage Criteria
size ∈ {small, medium, large}
Test Suite Construction

Figure 1: Our coverage-driven systematic test approach for SLAM

IV. An Approach for SLAM Testing

In this paper, we present an approach to automatically and systematically test SLAM algorithms. The overall approach is shown in Fig. 1. Our input consists of landmark-based maps and sensor traces on these maps. To avoid the infeasible task of creating an equivalence class partitioning of the entire SLAM input space at once, we instead identify parameters that inform the composition of that space. For each parameter, we define domain-specific equivalence classes that we use to guide our test suite construction. To increase the effectiveness of our test approach, we are especially interested in coverage criteria that are likely to push faulty code into observable erroneous behavior. We achieve this by tailoring both the parameters we use and the equivalence class partitioning for the parameters according to difficulties for SLAM algorithms described in the literature. As these difficulties cannot be directly translated into input data, but stem from particular characteristics such as phases of inactivity and loops in the vehicle's path, we define the coverage criteria in a declarative style (e.g. map size small/medium/large) and systematically construct test cases based on these coverage criteria.

Our approach is based on feature maps [1] with point features. We assume a range-and-bearing sensor model. As the input space for our test approach, we use long sensor traces through landmark-based test maps. To consider the entire execution of the algorithm on this sensor trace is necessary in order to test the integration of new information into the existing belief, rather than just testing a single call of the update function. Another advantage is that a ground truth map is available for comparison against the SLAM algorithm's result.

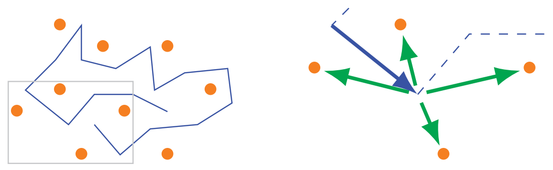


Figure 2: Test case design example

Fig. 2 shows an example test case. It consists of a map with a number of landmarks (orange circles) and the vehicle's path through the map (blue trace), separated into discrete time steps. Each step contains both motion (blue arrow) and sensor information (green arrows) obtained from simulation.

V. Domain-Specific Coverage Criteria

To systematically cover the input space and increase the test difficulty, we define parameters that influence the difficulty and expose typical errors. We have identified such parameters by a thorough analysis of the literature and searching for typical errors and difficulties. The parameters we have identified, our proposed partitioning for each parameter, and default ranges for each equivalence class are shown in Table I. The default ranges are predefined ranges that have been found to be useful in our experiments. Our coverage criterion for each parameter is defined as the number of equivalence classes that are covered by a given

	Parameter	Equivalence classes	Unit	Default ranges
1	*Directionality*	random, loop	–	–
2	*Rotation Error*	none, positive, negative	rad	$\{0\},\{2\pi n \mid n\in\mathbb{N}^{+}\},\{-2\pi n \mid n\in\mathbb{N}^{+}\}$
3	*Inactivity*	no inactivity, inactivity	steps	$\{0\},\{2000\}$
4	*Map Size*	small, medium, large	m^2	$[4,16],[16,100],[100,400]$
5	*Landmark Density*	low, medium, high	$\frac{\text{lm}}{m^2}$	$[1,3],[3,9],[9,15]$
6	*Step Size*	small, medium, large	cm	$[25,50],[50,100],[100,200]$
7	*Symmetry*	low, medium, high	%	$[0,10],[10,60],[60,90]$
8	*Outlier Probability*	none, low, high	%	$\{0\},[0,5],[5,15]$
9	*Field of View*	small, large	rad	$[\frac{\pi}{2},\pi],[\pi,2\pi]$
10	*Variance*	none, low, high	mm	$\{0\},[0,20],[20,100]$
			rad	$\{0\},[0,0.04],[0.04,0.2]$
11	*Bias*	high neg, low neg, none, low pos, high pos	mm	$[-25,-10],[-10,0],\{0\},[0,10],[10,25]$
			rad	$[-0.05,-0.02],[-0.02,0],\{0\},[0,0.02],[0.02,0.05]$

Table I: Equivalence classes by parameter (values outside the defined ranges are not considered)

test suite per total number of equivalence classes. Note that the number of and boundaries between equivalence classes can easily be adjusted by setting the corresponding parameters in our implementation to achieve a more fine-granular coverage if needed.

In the following, we summarize the main justification for the parameters and equivalence classes.

1) Directionality: Loops in the vehicle trajectory increase the time between re-observations of the same landmark and thus opportunities for correction of the state estimate. This makes the test case more difficult for SLAM algorithms [1] [12]. Thus, we include the equivalence classes loop, indicating a unidirectional map traversal, and random, indicating a random map traversal (Table I, Row 1).

2) Rotation Error: A common problem for motion and observation models is to account for rotations of less than 0 or more than 2π [1]. If the algorithm assumes that all rotations are between 0 and 2π, this may cause errors when comparing rotation values that differ by multiples of 2π. Intentionally inducing a positive and a negative rotation error will find this fault. Thus, we include equivalence classes with no such error as well as negative and positive deviations from the expected rotation interval (Table I, Row 2).

3) Inactivity: If a resampling procedure is implemented incorrectly, then a run of the algorithm without any input can cause erroneous behavior [1]. If a resampling step is executed at every empty update, as it would be in a naive implementation, then over time, this resampling filters out ever more samples until the belief converges on one single hypothesis that does not resemble the true distribution. A period of inactivity, followed by a normal test run, can expose such a design flaw. We include test cases with no inactivity (0 steps idle time) and with a longer phase of inactivity, e.g. 2000 steps idle time (Table I, Row 3).

4) Map Size: The size of the map is a challenge for SLAM algorithms [35], especially if it exceeds the effective range of the vehicle's sensors [1]. We include test cases with varying map sizes, e.g. small, medium and large maps (Table I, Row 4).

5) Landmark Density: The density of landmarks is a significant factor in SLAM performance. The fewer landmarks are available, the harder it becomes to localize the vehicle [1], which can also lead to inaccuracies in the estimated map [12] [34]. On the other hand, if the density of landmarks becomes too high, the chances of an error in data association increase [1]. Since both low and high landmark densities can cause difficulties, a variety of landmark densities should be included, e.g. low, medium and high density maps (Table I, Row 5, number of landmarks per square meter).

6) Measurement Step Size: The step size between measurements affects the difficulty of the test in multiple ways. A longer step size reduces the number of observations on landmarks, since it takes the vehicle out of the maximum observation range to a landmark faster. It also allows a larger accumulation of motion error, thus increasing the uncertainty of the vehicle's position [1]. Since the step size influences the difficulty for the SLAM algorithm [19], we consider various step sizes, e.g. small, medium and large steps (Table I, Row 6).

7) Symmetry: Symmetrical environments make it more difficult to localize the vehicle in the environment, since multiple hypotheses about the location may have to be maintained at any point in time [1]. Symmetry may also lead to faulty loop closures and consequent filter divergence [35]. Thus, we include maps with varying degrees of point symmetry around the center. This causes similar landmark constellations at different points on the map, increasing the difficulty for the SLAM algorithm [33] [35]. We achieve this by forcing the placement of a given percentage of landmarks to be symmetrical. The percentage is given by the equivalence classes, describing, e.g. a low, medium or high percentage of landmarks (Table I, Row 7). We exclude 100 % symmetry, as this would make it impossible to distinguish different map locations.

8) Outlier Probability: Outlier detection is an essential part of dealing with sensors prone to return false results. To detect flaws in the outlier detection, we include various outlier frequencies, e.g. equivalence classes with no outliers, low, and high outlier probability (Table I, Row 8).

9) Sensor Field of View (FoV): Many vehicles using SLAM do not have an ability to sense landmarks at arbitrary angles [32] [33]. Instead, they can only sense landmarks in a given range in front of them. This influences the accuracy of the result [33]. Thus, we include varying values for the FoV, e.g. small FoVs that allow only detection in front of the vehicle and large FoVs that allow detection also in an area behind the vehicle (Table I, Row 9). We exclude FoVs that are too small, as this would make the test too difficult.

10) Measurement Variance: Increasing the uncertainty of sensor and odometry measurements increases the difficulty of the test [1] [33] [19]. We include test cases with varying levels of uncertainty, e.g. none, low and high (Table I, Row 10).

11) Measurement Bias: Introducing a bias into the measurements reduces their accuracy and thereby increases the difficulty for the SLAM algorithm similarly to the variance. We include varying biases, e.g. none, low and high positive and low and high negative (Table I, Row 11).

There are some other candidate parameters that we currently ignore. [32] [33] suggest sensor range as a relevant feature, but this is subsumed by *map size* and *landmark density* in our abstract map representation. The frequency of SLAM updates can also be relevant, but is subsumed by *measurement step size* and *inactivity*. Further, we ignore some situational challenges that are incompatible with our abstract map and observation representation, e.g. lighting conditions for visual SLAM [34] [35] or an uneven road surface [34].

VI. Coverage-Driven Test Case Construction

The parameters we consider for the definition of domain-specific coverage criteria determine the structure of both map and vehicle path as well as the level of noise in the simulated observations. To systematically construct a test suite with a high coverage, we choose an equivalence class from each parameter, preferably one that is not yet included in the suite. From each equivalence class, we uniformly choose a value to derive constraints for the test case. We then generate a map and a sensor trace in compliance with these constraints to obtain a test case. Test cases are added to the suite in this way until every equivalence class of every parameter is covered at least once. Each such test case consists of a map and a simulated sensor trace of the vehicle driving around and observing landmarks in this map. However, the parameter values only provide constraints for the test case, not an executable procedure to generate the concrete map and sensor trace. In this section, we describe our method for constructing a test map and sensor trace such that it complies with its parameter values. We go into detail on three components, namely the map, the sensor trace and the observation simulation.

A. Map Generation

The map generation needs to take into account map size, landmark density and symmetry. For the sake of simplicity, our map design is limited to square feature maps with a side length determined by the size and a number of point features determined by the landmark density. Each landmark has a unique identifier, accommodating SLAM algorithms that require external object association because they assume known correspondences. The locations of landmarks on the map are randomly generated according to a uniform distribution. To control the symmetry in the map, the given percentage of landmarks are placed in pairs that are mirrored through the center of the map. After all symmetrical pairs have been placed, the remaining landmarks are placed individually. To preserve the distinctness of landmarks, a draw is rejected and repeated if a landmark is chosen at a position that is too close to another landmark.

B. Sensor Trace Generation

The sensor trace generation needs to take into account directionality, inactivity, measurement step size, and sensor FoV. A sensor trace consists of a predetermined number of update steps with information about the previous vehicle motion and simulated measurements of all obstacles within the vehicle's FoV. The distance between update steps is the measurement step size. To incorporate the inactivity parameter, a number of steps (given as parameter value) without any motion or observation information are inserted into the sensor trace. These steps are inserted in the middle of the sensor trace to maximize their effectiveness, as the inactivity is intended to cause particle deprivation in a poorly implemented resampling filter. Inserting it in the middle of the trace allows time for the filter to build up uncertainty, which is artificially removed through particle deprivation, and then also allows time for the resulting error to manifest itself in the estimate.

The most challenging aspect to take into account for the sensor trace generation is to control the direction of traversal. To overcome this challenge, we use the landmarks as guidance. With a fixed vehicle starting position in the center of the map, the generator picks a target landmark that has not been seen before and moves the vehicle towards that landmark until it is in view. If all landmarks have been seen, they are marked as unseen and the process repeats until the desired number of steps is reached. Fig. 3 illustrates the trace generation process. Starting near Landmark 1, the generator picks Landmark 3 as a target. It then simulates the vehicle driving towards this landmark until it is within sensor range. Once it is in view, the generator selects a new target, disregarding those that were already observed (marked as black), here Landmark 5. This process repeats until the predetermined number of time steps has elapsed.

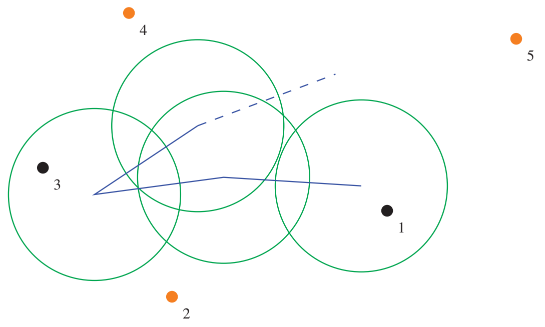


Figure 3: Sensor trace generation

The target selection ensures that all landmarks are visited at least once per complete traversal, giving the SLAM algorithm a chance to accurately map landmarks on the rim. Most importantly, using the landmarks as guidance allows us to control the direction of traversal and to incorporate the directionality parameter. If the parameter value specifies a random traversal, landmarks are randomly picked from the list of unobserved landmarks. If it specifies a unidirectional traversal, the unobserved landmarks are sorted by polar angle relative to the center of the map and the vehicle moves towards the next landmark according to that order. To avoid a deterministic and unchanging traversal, we add noise to the simulated vehicle path.

C. Observation Simulation

The observation simulation needs to account for rotation error, outlier probability, variance and bias. Measured values for range and bearing to landmarks as well as the vehicle motion are distorted according to a Gaussian distribution with the given bias and variance. Additionally, the outlier probability defines the probability of declaring an observation as an outlier. Outliers can be either maximum range readings or chosen from a uniform distribution between 0 and the actual observation distance. This roughly represents the expected distribution of outliers due to missed measurements and sensor obstruction [1]. Lastly, if a rotation error is specified, 2π is added to or subtracted from landmark bearing and vehicle turn information.

VII. Evaluation

To validate our test approach, we have implemented a compliant test generator and a simulation environment for executing the test cases. The source code, raw data and all material needed to replicate our evaluation are available at [36]. In the following, we present our evaluation methodology, evaluation metric, and our experimental setup and results.

A. Methodology

To demonstrate the error detection capability of our approach, we have injected three faults into a slightly modified version of the FastSLAM 1.0 implementation of the PythonRobotics software project [5]. Two of the three are based on common mistakes according to [1], namely a naive resampling filter and a missing treatment of angle values exceeding the full rotation. Both of these are covered by our parameters. The third fault is a sign error in the motion model, intended to evaluate the test suite's capability of detecting an error for which it was not explicitly designed, as well as the general ability of SLAM tests to find such errors. Please note that classical fault injection approaches like, e.g. mutation testing [37] [38] can not easily be applied to SLAM testing due to the data-intensity but little control flow of SLAM algorithms. However, our manually injected faults are representatives of typical faults in SLAM that are hard to detect and thus are well suited for the evaluation.

Our evaluation criterion is the increase in map error between the correct implementation and the faulty implementation. If this error increases by a value exceeding the expected noise, we consider the test case capable of finding the fault in the implementation. We set this threshold to 250mm. Since the noise in map error increases with larger map error, we additionally require an increase of at least 25 % for larger map errors to consider a test case effective at finding a fault.

An alternative approach would be to define a test oracle to judge whether a test passed or failed, and to judge error detection capability by counting the number of failed tests on faulty implementations. However, giving an appropriate test verdict is a major challenge for SLAM algorithms. As the sensor data is inherently imprecise and SLAM algorithms only give estimates of the vehicle positions, it is not trivial to distinguish whether a result is faulty or of acceptable precision with the given inputs. Especially varying map sizes and sensor uncertainties make it hard for our approach to compare different test cases. Since our aim is to evaluate the effectiveness of our testing approach and not to test the correctness of the algorithm under test, we leave the definition of a test oracle capable of dealing with these problems for future work.

B. Evaluation Metric

To evaluate the result of the SLAM algorithm after execution, we compare the final map estimate of the algorithm with a ground truth map retained by the executed test case. Since a small error in the pose estimate of the vehicle may lead to a translation or rotation of its local coordinate system, a naive comparison of absolute landmark positions is undesirable for evaluation, as it would potentially create a large error even if the resulting map is internally consistent with the ground truth map. Fig. 4 illustrates this problem. In the left picture, almost all error in the SLAM estimate is caused by a rotational misalignment at one single point. The right picture shows a chaotic error that can be expected only from an incorrect SLAM implementation. Since we are interested in detecting faulty implementations, we focus on the latter type of error. To correct for translation and rotation, we employ the Kabsch algorithm [39]. We then use the root-mean-square (RMS) error between the corrected map and the ground truth for algorithms that preserve landmark correspondences. To support algorithms that do not preserve landmark correspondences, our implementation integrates the earth mover's distance [40] as an alternative.

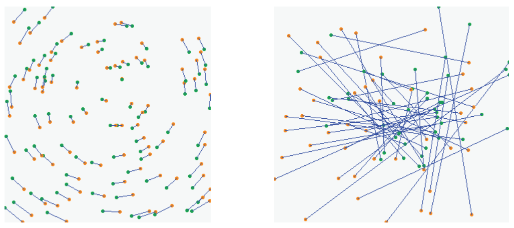


Figure 4: Rotational and chaotic errors (orange: landmark, green: SLAM estimate, blue: correspondance)

C. Setup

We generate our *systematic* test suite such that it covers all equivalence classes with the default ranges from Table I. We achieve this with a total of five test cases. A sample visualisation of the resulting test cases is shown in Fig. 5. It demonstrates the diversity of test cases our approach generates. Notable is the difference between the loop traversal shown in Cases 1 and 4 and the random traversal in Cases 2, 3 and 5, as well as the effect of different map sizes. While in Cases 2 and 5, the vehicle mostly drives back and forth between the landmarks with many opportunities to correct flawed observations, Case 1 only has the vehicle revisit a landmark a few times.

Our approach is based on two key concepts. One is the structuring of the SLAM problem's input space with our parameter definitions and the other is the enforcement of high coverage with regard to our equivalence class partitioning. To be able to evaluate both components, we use two separate control suites as comparison. To evaluate the effectiveness of our parameter definitions, we contrast our approach against a control suite with fixed parameter values (*fixed*), i.e. all of its test cases are generated from the same default parameter values. This comparison shows the effect of varying the parameters we identify. To evaluate the effectiveness of our coverage-driven test generation, we use a control suite with random parameter values (*random*), i.e. each parameter value is randomly chosen for each test case. This shows the effect of enforcing a high coverage of the input space. Both control suites also contain five test cases. They are intended to be approximations of the current standard approach of executing the algorithm under test on a public data set or in an experimental environment. As such an experimental setup contains some diversity in environmental conditions, but is not constructed systematically to cover the input space, we expect it to lie somewhere between the *fixed* and *random* suites.

All test suites are run in simulation on the variations of the SLAM algorithm described above. We have repeated each experiment three times (Evaluation Run 1, 2, 3). In each run, new test cases were generated for each suite for a total of 15 test cases per test suite generation method. Each evaluation run took approximately 8 to 9 hours (see [36]).

D. Results

Table II shows the RMS errors resulting from executing the test cases on the original implementation of the SLAM algorithm (*nonfaulty baseline*) and on the three faulty versions. RMS errors above our thresholds for fault detection represent test cases capable of detecting the respective fault and are highlighted in green. The *Detected Faults* column shows which percentage of total test cases detected the respective fault and the *by Suite* column shows the percentage of faults that the test suite detected in total over the three runs.

Comparing the *fixed* suite to the *systematic* and *random* ones shows the effectiveness of the parameter definitions. While the *fixed* suite managed to achieve a 100 % detection rate for the rotation and sign errors, this is likely caused by the high similarity between all test cases in this suite due to the fixed parameter values. Varying parameter values in the other suites led to a greater diversity of test cases. This caused slightly lower detection rates for the rotation and sign errors, but also the inclusion of some test cases that managed to detect the naive resampling error, which the *fixed* suite could not detect in any run.

Comparing the *random* to the *systematic* test suite shows the effect of our coverage-driven equivalence class partitioning. As can be seen in Table II, the behavior of both suites is similar. In both suites, most test cases detect the rotation and sign errors, while some test cases detect the resampling error. However, the *systematic* suite is more consistent in

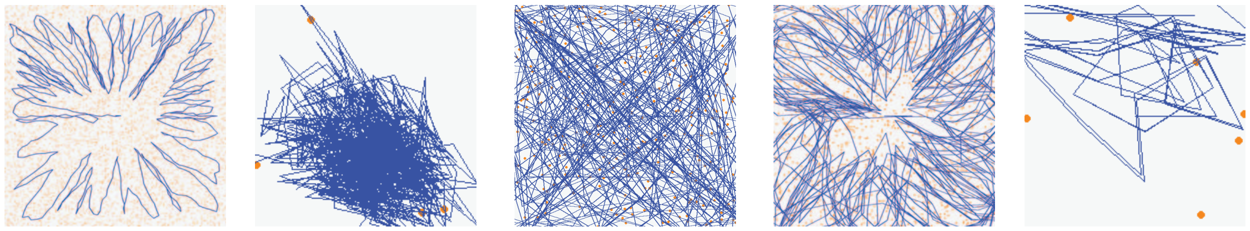

(a) Case 1 **(b)** Case 2 **(c)** Case 3 **(d)** Case 4 **(e)** Case 5

Figure 5: Visualization of our test suite from Evaluation Run 1 (orange: landmark, blue: vehicle path)

Suite	Algorithm	Evaluation Run 1					Evaluation Run 2					Evaluation Run 3					Detected	by
	Test Case	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	**Faults**	**Suite**
Systematic	Nonfaulty Baseline	5616	40	55	544	42	55	859	28	201	47	2285	3951	37	5982	69	-	100 %
	Naive Resampling	**9038**	44	88	368	48	52	434	29	**849**	56	1127	**8548**	37	**13640**	80	27 %	
	Faulty Rotation	6922	**627**	**2118**	**4048**	**1232**	**1770**	**3977**	**675**	**5320**	**585**	**7919**	3998	**1292**	6020	**2657**	80 %	
	Sign Error	**7694**	**755**	**2509**	**3849**	**920**	**2022**	**3861**	**798**	**5867**	**3307**	**6772**	4084	**1671**	6018	**2879**	87 %	
Fixed	Nonfaulty Baseline	43	47	40	72	46	56	43	101	41	60	47	45	48	57	38	-	67 %
	Naive Resampling	68	51	58	138	44	92	44	103	67	47	58	42	40	44	50	0 %	
	Faulty Rotation	**601**	**501**	**557**	**601**	**548**	**625**	**566**	**2239**	**583**	**487**	**533**	**631**	**479**	**574**	**611**	100 %	
	Sign Error	**3242**	**3083**	**3156**	**3206**	**3026**	**3323**	**3248**	**3088**	**2994**	**2985**	**3253**	**3337**	**3085**	**3176**	**3254**	100 %	
Random	Nonfaulty Baseline	233	216	7462	1465	7606	102	96	4295	160	304	577	2959	1363	47	3773	-	78 %
	Naive Resampling	377	199	**13267**	1639	**17827**	93	118	5293	169	140	772	3422	1426	85	4441	13 %	
	Faulty Rotation	**3206**	**497**	7099	**6014**	7625	**1532**	**2515**	**5535**	**869**	**2222**	**3853**	**7653**	1241	247	**6303**	80 %	
	Sign Error	**3495**	**2798**	7822	**6123**	8069	**1543**	**2392**	**5828**	**1890**	**2674**	**4522**	**7502**	**7041**	**1133**	**3254**	87 %	

Table II: RMS errors (rounded to mm) and error detection capability (green)

this regard. While the *systematic* suite generated four test cases over the three runs that managed to detect the resampling error, at least one for each run, the *random* suite only generated two such test cases, both in the first run. In the second and third run, only the *systematic* suite managed to detect all faults. This is likely caused by the enforced coverage of the input space, which leads to the inclusion of diverse test cases in every suite.

Our experiments indicate that both components of our approach are effective for a systematic test approach. They further show that this systematic approach is able to detect more errors than random approaches, especially in hard-to-test corner cases like the naive resampling implementation. Overall, the results indicate a significant advantage of the full coverage on the equivalence classes compared to random approaches.

VIII. Conclusion

In this paper, we have presented a systematic approach for SLAM testing. We have identified parameters to structure the SLAM problem's input space, defined equivalence classes for each parameter in a declarative style and developed a method for generating test cases with a high coverage of the input space. Based on this method, we have implemented a freely available test case generator and demonstrated the applicability of our approach for the FastSLAM algorithm. Our experimental results indicate that our systematic approach is more effective and more consistent than random testing at finding errors. In particular, our approach has proven advantageous for the detection of errors in typically hard-to-test corner cases.

As there is little research in this area, this paper is intended as a first step in the direction of a systematic method for SLAM testing. While our experimental results are promising, a more extensive evaluation with other SLAM algorithms is subject to future work. Further future research directions include the definition of a test oracle for automated test evaluation, a deeper analysis of potential faults (e.g. hardware limitations or sensor faults), and to investigate each parameter's individual effect for a more deliberate and effective test case selection.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, August 2005, ISBN 0-262-20162-3.

[2] G. J. Myers, T. Badgett, and C. Sandler, *The art of software testing*, 3rd ed. John Wiley & Sons, 2012, ISBN 978-1-118-03196-4.

[3] A. Bhat and S. Quadri, "Equivalence class partitioning and boundary value analysis-A review," in *Intl. Conf. on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2015, pp. 1557–1562.

[4] W.-l. Huang and J. Peleska, "Exhaustive model-based equivalence class testing," in *IFIP International Conference on Testing Software and Systems*. Springer, 2013, pp. 49–64.

[5] A. Sakai, D. Ingram, J. Dinius, K. Chawla, A. Raffin, and A. Paques, "PythonRobotics: a Python code collection of robotics algorithms," *CoRR*, vol. abs/1808.10703, 2018.

[6] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," in *National Conference on Artificial Intelligence*, 2002, pp. 593–598.

[7] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte Carlo Localization for Mobile Robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2. IEEE, 1999, pp. 1322–1328.

[8] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges," in *International Joint Conference on Artificial Intelligence (IJCAI)*, vol. 3, 2003.

[9] C. Stachniss, U. Frese, and G. Grisetti, "OpenSLAM," 2018. [Online]. Available: https://openslam-org.github.io/

[10] A. Howard and N. Roy, "The Robotics Data Set Repository (Radish)," 2003. [Online]. Available: https://dspace.mit.edu/handle/1721.1/62236

[11] T. Li, D. Ho, C. Li, D. Zhu, C. Wang, and M. Q.-H. Meng, "HouseExpo: A Large-scale 2D Indoor Layout Dataset for Learning-based Algorithms on Mobile Robots," 2019. [Online]. Available: http://arxiv.org/abs/1903.09845

[12] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, no. 4, pp. 387 – 407, 2009.

[13] G. Grisetti, C. Stachniss, and W. Burgard, "Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling," in *IEEE Intl. Conf. on Robotics and Automation*, 2005.

[14] ——, "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34 – 46, 2007.

[15] T.-W. Huang, C.-C. Hsu, W.-Y. Wang, and J. Baltes, "ROSLAM—A Faster Algorithm for Simultaneous Localization and Mapping (SLAM)," *Advances in Intelligent Systems and Computing*, vol. 447, pp. 65–74, 2017.

[16] H. Luo, C. Pape, and E. Reithmeier, "Hybrid Monocular SLAM Using Double Window Optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, 2021.

[17] L. Somlyai and Z. Vámossy, "ISVD-Based Advanced Simultaneous Localization and Mapping (SLAM) Algorithm for Mobile Robots," *Machines*, vol. 10, 6 2022.

[18] T. Zhang, C. Liu, J. Li, M. Pang, and M. Wang, "A New Visual Inertial Simultaneous Localization and Mapping (SLAM) Algorithm Based on Point and Line Features," *Drones*, vol. 6, 1 2022.

[19] Y. Chen, J. Tang, C. Jiang, L. Zhu, M. Lehtomäki, H. Kaartinen, R. Kaijaluoto, Y. Wang, J. Hyyppä, H. Hyyppä, H. Zhou, L. Pei, and R. Chen, "The Accuracy Comparison of Three Simultaneous Localization and Mapping (SLAM)-Based Indoor Mapping Technologies," *Sensors*, vol. 18, no. 10, 2018.

[20] F. Amigoni, S. Gasparini, and M. Gini, "Good Experimental Methodologies for Robotic Mapping: A Proposal," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 4176 – 4181.

[21] F. Amigoni, V. Castelli, and M. Luperto, "Improving Repeatability of Experiments by Automatic Evaluation of SLAM Algorithms," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[22] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Lujan, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2015-June. IEEE, 5 2015, pp. 5783–5790.

[23] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Lujan, S. Furber, A. J. Davison, P. H. J. Kelly, and M. F. P. O'Boyle, "SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 5 2018, pp. 3637–3644.

[24] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, O. M. F.P., A. J. Davison, K. P. H.J., G. Riley, B. Lennox, M. Lujan, and S. Furber, "SLAMBench 3.0: Systematic Automated Reproducible Evaluation of SLAM Systems for Robot Vision Challenges and Scene Understanding," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 5 2019, pp. 6351–6358.

[25] O. Bühler and J. Wegener, "Evolutionary functional testing," *Computers & Operations Research*, vol. 35, no. 10, 2008, sI: Search-based Software Engineering.

[26] D. S. Rodrigues, M. E. Delamaro, C. G. Corrêa, and F. L. S. Nunes, "Using Genetic Algorithms in Test Data Generation: A Critical Systematic Mapping," *ACM Computing Surveys*, vol. 51, no. 2, pp. 41:1 – 41:23, 2018.

[27] U. Witteck, D. Grießbach, and P. Herber, "Equivalence Class Definition for Automated Testing of Satellite On-Board Image Processing," in *Software Technologies (ICSOFT)*, vol. 1250, 2019, pp. 3 – 25.

[28] ——, "A Genetic Algorithm with Tournament Selection for Automated Testing of Satellite On-Board Image Processing," in *International Conference on Software Technologies (ICSOFT)*, 2020, pp. 128 – 135.

[29] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, pp. 1–29, 2011.

[30] D. R. Kuhn, R. N. Kacker, and Y. Lei, *Introduction to combinatorial testing*. CRC press, 2013.

[31] J. Tao, Y. Li, F. Wotawa, H. Felbinger, and M. Nica, "On the industrial application of combinatorial testing for autonomous driving functions," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2019, pp. 234–240.

[32] M. Luperto, V. Castelli, and F. Amigoni, "Predicting Performance of SLAM Algorithms," 9 2021. [Online]. Available: http://arxiv.org/abs/2109.02329

[33] E. Piazza, P. U. Lima, and M. Matteucci, "Performance Models in Robotics With a Use Case on SLAM," *IEEE Robotics and Automation Letters*, vol. 7, pp. 4646–4653, 4 2022.

[34] Z. Javed and G.-W. Kim, "PanoVILD: a challenging panoramic vision, inertial and LiDAR dataset for simultaneous localization and mapping," *The Journal of Supercomputing*, vol. 78, pp. 8247–8267, 4 2022.

[35] Y. Chang, K. Ebadi, C. E. Denniston, M. F. Ginting, A. Rosinol, A. Reinke, M. Palieri, J. Shi, A. Chatterjee, B. Morrell, A. akbar Agha-mohammadi, and L. Carlone, "LAMP 2.0: A Robust Multi-Robot SLAM System for Operation in Challenging Large-Scale Underground Environments," *IEEE Robotics and Automation Letters*, vol. 7, pp. 9175–9182, 10 2022.

[36] P. Tasche and P. Herber, "Coverage-Driven SLAM Testing," Dataset on 4TU.ResearchData. [Online]. Available: https://dx.doi.org/10.4121/21946514

[37] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[38] M. Papadakis, M. Kintis, J. Zhang, Y. Jia, Y. Le Traon, and M. Harman, "Mutation testing advances: an analysis and survey," in *Advances in Computers*. Elsevier, 2019, vol. 112, pp. 275–378.

[39] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A*, vol. 32, no. 5, pp. 922–923, 1976.

[40] Y. Rubner, C. Tomasi, and L. Guibas, "A metric for distributions with applications to image databases," in *IEEE Intl. Conf. on Computer Vision*, 1998, pp. 59–66.