Monte Carlo Methods for SLAM with Data Association Uncertainty

Constantin Berzan
Stuart J. Russell

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2015-191
http://www.eecs.berkeley.edu/Pubs/TechRpts/2015/EECS-2015-191.html

August 13, 2015

Acknowledgement

Thanks to Stuart Russell and RUGS.

Monte Carlo Methods for SLAM with Data Association Uncertainty

by Constantin Berzan

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Stuart Russell
Research Advisor

(Date)

* * * * * * *

Professor Trevor Darrell
Second Reader

(Date)

Abstract

We work towards solving the Simultaneous Localization and Mapping (SLAM) problem using a Probabilistic Programming System (PPS). After surveying existing SLAM methods, we choose FastSLAM as the most promising candidate. FastSLAM uses ad-hoc methods for data association, and does not enforce mutual exclusion between observations arriving at the same timestep. This leads to poor accuracy on an example dataset. We propose a new probabilistic model for SLAM that handles association uncertainty and mutual exclusion. We then propose an algorithm for doing inference in this model: FastSLAM-DA (FastSLAM with Data Association), which uses a particle filter with a custom data-driven proposal. We show that FastSLAM-DA performs well on the example where FastSLAM previously failed. However, the new algorithm produces inaccurate maps when there is a high rate of false detections. To remedy this, we propose FastSLAM-DA-RM (FastSLAM with Data Association and Resample-Move), which adds MCMC moves on the recent association variables. We show that FastSLAM-DA-RM performs well where FastSLAM-DA previously failed. Our two new algorithms use no heuristics other than custom proposals, so they are suitable for implementation in a PPS. As a step in this direction, we implement a general-purpose resample-move particle filter in the BLOG PPS, and demonstrate it on a simplified SLAM problem.

Table of Contents

1 Introduction 1
2 Background and Related Work 3
2.1 The BLOG Probabilistic Programming System 3
2.2 SLAM Problem Definition 4
2.3 Paradigms for Solving SLAM 5
2.4 Data Association in SLAM 7
3 FastSLAM and Its Limitations 10
3.1 The FastSLAM Probabilistic Model 10
3.2 The FastSLAM Algorithm 11
3.3 Heuristics and Approximations 13
3.4 `mutex`: A Dataset with High Observation Noise 15
3.5 Error Metrics for Measuring SLAM Accuracy 16
3.6 Results on the `mutex` Dataset 17
4 FastSLAM-DA: Sampling Data Associations 21
4.1 A Probabilistic Model for SLAM with Association Uncertainty . . 21
4.2 A Smart Proposal for Associations 26
4.3 The FastSLAM-DA Algorithm 29
4.4 Results on the `mutex` Dataset 30
4.5 `highfa`: A Dataset with a High Rate of False Detections 33
4.6 Results on the `highfa` Dataset 34
5 FastSLAM-DA-RM: Changing the Past 38
5.1 An MCMC Algorithm for Associations 38
5.2 Evaluating the MCMC Target Distribution 42
5.3 Verifying the MCMC Sampler 43
5.4 The FastSLAM-DA-RM Algorithm 44
5.5 Results on the `highfa` Dataset 46
6 Towards an Implementation in BLOG 49
6.1 `gridworld`: A One-dimensional SLAM Problem 50
6.2 BLOG Resample-Move Results on `gridworld` 53
7 Conclusions and Future Work 57

1 Introduction

AI systems use probabilistic models to deal with the uncertainty in their environment. A probabilistic model is often described using a combination of mathematical notation and natural language. The model is then converted manually into code that performs inference in that specific model. Writing inference code by hand is error prone and time consuming, and this process has to be repeated every time the model changes (e.g., when the agent gets a new sensor, or the researcher refines the environment model).

A Probabilistic Programming System (PPS) aims to facilitate the model-building process described above. First, a PPS provides a formal language for specifying probabilistic models in a compact and unambiguous way. Second, a PPS provides a set of generic inference procedures that work on large classes of models. With a sufficiently advanced PPS, the researcher should be able to quickly write a model for a new problem, run inference, and then refine the model and run inference again, all while having to write little or no custom inference code.

For a PPS to be useful in the real world, its language must be expressive enough to support the types of models that researchers want to write, and its automatic inference procedures must be general enough to perform efficient inference in those models. In the past decade we have seen the emergence of very expressive PPSs [19,29], but efficient inference continues to be a challenge. Hence, one research avenue for advancing PPSs is to examine a sufficiently complicated class of models, and see what it would take to do efficient inference in these models using a PPS.

In this thesis, we will examine a set of probabilistic models that arise from the Simultaneous Localization and Mapping (SLAM) problem in robotics. In SLAM, a robot is navigating in an unknown environment. The robot seeks to build a map of the environment, and simultaneously localize itself within this map. SLAM is a key requirement for building autonomous robots [30, 41]. There are many existing solutions to SLAM (which we will review in the next section), and our goal is not to beat these existing solutions. Instead, our goal is to understand how to solve SLAM using a PPS, and to illuminate the building blocks that a PPS needs in order to do efficient inference in non-trivial temporal models.

In section 2 we provide the background for our work. We discuss the building blocks available in a PPS and the main existing methods for solving SLAM. We also review the literature on data association in SLAM, which will become a central topic of this thesis. In section 3 we discuss FastSLAM, the existing SLAM algorithm which we will build upon. We point out the ad-hoc methods that FastSLAM uses for data association and map management. These ad-hoc methods make FastSLAM unsuitable for a PPS, and cause poor accuracy on a dataset with large association uncertainty.

We begin to address FastSLAM's limitations in section 4. We present a probabilistic model for SLAM with data association uncertainty and mutual exclusion constraints between observations arriving at the same timestep. We then develop a new algorithm for performing inference in this model: FastSLAM-DA

(FastSLAM with Data Association), which is based on particle filtering with a data-driven proposal. This new algorithm beats FastSLAM on a dataset with large association uncertainty, but produces inaccurate maps if there is a high rate of false detections. We address the latter limitation in section 5, by developing a new algorithm called FastSLAM-DA-RM (FastSLAM with Data Association and Resample-Move). This algorithm uses a resample-move particle filter, which augments FastSLAM-DA with MCMC moves on the recent association variables. This improvement allows FastSLAM-DA-RM to produce accurate maps even when there is a high rate of false detections.

In section 6 we discuss the building blocks that we would need to add to a PPS in order to support algorithms like FastSLAM-DA and FastSLAM-DA-RM. We take the first step in this direction by implementing a general-purpose resample-move particle filter in the BLOG PPS, and showing that it beats the regular particle filter on a simplified grid-world SLAM problem. In section 7 we summarize our conclusions and discuss ideas for future work.

This thesis makes the following contributions:

1. A probabilistic model for SLAM with data association uncertainty (section 4.1), which removes the need for many of the heuristics and approximations in FastSLAM.
2. Two algorithms for performing inference in this model: FastSLAM-DA (section 4) and FastSLAM-DA-RM (section 5), and examples in different regimes where one algorithm performs better than the others.
3. A general-purpose implementation of resample-move particle filtering in the BLOG PPS (section 6).

2 Background and Related Work

Our work is at the intersection of probabilistic programming, SLAM, and data association. In this section we place our work in context, by briefly surveying these three fields and reviewing the relevant literature. In section 2.1 we discuss our target PPS, BLOG, and why it guides us towards a sampling-based SLAM solution. In section 2.2 we define the SLAM problem in probabilistic terms. In section 2.3 we describe the three main paradigms for solving SLAM, and our reasons for focusing on particle-based methods. In section 2.4 we discuss data association in SLAM, which will be one of the main concerns of this thesis.

2.1 The BLOG Probabilistic Programming System

Bayesian Logic (BLOG) [29] is a PPS consisting of a powerful language to describe probabilistic models and a set of generic algorithms to perform inference in such models.

The BLOG language provides a compact way to define probability distributions over complicated outcome spaces, such as worlds with an unknown number of objects. BLOG also supports first-order-logic relationships, unlike earlier systems like BUGS [26] and JAGS [37], which only supported propositional models using a fixed set of variables.

For atemporal models, BLOG provides three generic inference algorithms: a rejection sampler, a likelihood-weighting sampler, and a Metropolis–Hastings (MH) sampler, which is a flavor of Markov chain Monte Carlo (MCMC). Good introductions to these types of samplers exist elsewhere [4, 24]. BLOG's MH sampler allows the user to plug in a custom proposal distribution. The default proposal simply picks a variable from the model uniformly at random and proposes a new value for that variable given the values of its parents.

For temporal models, BLOG provides a generic particle filtering algorithm. Additionally, BLOG provides a Liu-West filtering algorithm [25] for temporal models with static parameters. Both of these algorithms are based on the Sequential Monte Carlo (SMC) framework, which represents a probability distribution using a set of samples and efficiently updates this representation in an online fashion as new observations arrive [12, 14].

For our purposes, it is important to keep in mind that all of BLOG's inference algorithms are based on sampling: simple methods like rejection sampling and likelihood weighting, and more powerful and general methods like MCMC and SMC. Other PPSs such as Church, Venture, and Anglican also perform inference using MCMC and closely related techniques [27, 43, 44]. This puts a strong constraint on our desired goal: if we want to solve SLAM using BLOG, we need to express SLAM as a probabilistic model in BLOG, and then perform inference in this model using some kind of sampling-based method. We could extend BLOG to support Rao-Blackwellization, or conjugate analysis like in BUGS and JAGS [26,37], or EM and numerical optimization like in AutoBayes [16], but this would require a substantial re-engineering effort, since BLOG was not designed

to provide these techniques as building blocks. We will keep these constraints in mind as we consider various approaches to SLAM, below.

Our work was originally motivated by the PPAML SLAM challenge problem [2]. To our knowledge, there have not been any attempts to solve SLAM with a PPS before this challenge problem.

2.2 SLAM Problem Definition

We start by describing the SLAM problem from a probabilistic perspective. We divide time into discrete chunks called timesteps. For simplicity, we assume that the duration of a timestep is fixed, but it is easy to extend the models below to the case where timesteps have different durations.

We represent the pose (location) of the robot at timestep t by a random variable x_t. For example, a car's pose might contain an x coordinate, a y coordinate, and an angle for the orientation. We assume that the pose at the initial timestep is known:

$$x_1 = x_{\text{init}}$$

At each subsequent timestep, the robot receives controls u_t. If we had a perfect model of the robot's dynamics, we could predict the pose x_t precisely based on the previous pose x_{t-1} and the controls u_t. But usually the dynamics model is not perfectly accurate, so we get a probability distribution over the pose x_t, instead of an exact value:

$$p(x_t|x_{t-1}, u_t)$$

We represent the true map of the environment by θ. The map can include features such as landmarks, walls, etc. The robot has a set of sensors through which it can observe its environment. At each timestep, the observations are given by:

$$p(y_t|x_t, \theta)$$

The goal of SLAM is to infer the trajectory of the robot and the true map of the environment from the observations received. In the *offline SLAM problem*, we collect all the observations and then estimate the posterior over the entire trajectory and map:

$$p(x_{1:t}, \theta|u_{1:t}, y_{1:t})$$

In the *online SLAM problem*, we process observations one at a time, and estimate the posterior over the current pose and map, rather than the full trajectory:

$$p(x_t, \theta|u_{1:t}, y_{1:t})$$

Besides the offline vs. online distinction, the SLAM problem can be further categorized along the following dimensions [41]:

- Volumetric SLAM (where the map is an occupancy grid) vs. feature-based SLAM (where the map is a set of sparse features extracted from the raw observations).

– Topological SLAM (where the map captures qualitative relationships between places, such as "room A is next to room B") vs. metric SLAM (where the map captures exact distances between features).
– SLAM with known data association (where it is known a priori which landmark generated which observation, e.g., because the landmarks have unique colors) vs. SLAM with unknown data association (where there is ambiguity about which landmark generated which observation).
– SLAM in a static environment (where the true map doesn't change) vs. SLAM in a dynamic environment (where the true map changes over time).
– SLAM that requires loop closure (i.e., the robot reaches a previously visited location after a long loop, and needs to recognize that it is back on familiar territory), vs. SLAM that does not require loop closure.
– Active SLAM (where the robot actively explores the environment in an attempt to build an accurate map) vs. passive SLAM (where the SLAM algorithm is observing but not controlling the motion of the robot).
– Single-robot SLAM vs. multi-robot SLAM (where multiple robots cooperate to build a common map and help localize each other).

In the rest of this thesis we will focus on online, feature-based SLAM with unknown data association. The map will be static and metric, there will be a single robot involved, and the robot will not perform active exploration.

2.3 Paradigms for Solving SLAM

There are three main paradigms for solving SLAM [41]: Extended Kalman Filters, Graph-Based Methods, and Particle Methods. In this section we will examine each of these paradigms and consider whether they are suitable for a PPS-based solution to SLAM.

The **Extended Kalman Filter** (EKF) approach tracks the joint distribution of the robot pose and the landmark locations as a single multivariate Gaussian distribution with mean μ and covariance Σ. As the robot moves and new observations arrive, the mean and covariance of this distribution are updated using the EKF formulas obtained by linearizing the dynamics model and the observation model. The main advantage of EKF SLAM is that it tracks the covariance between robot pose and landmark locations jointly, so that when the pose uncertainty decreases (e.g., because the robot sees an old landmark again), the landmark location uncertainty decreases as well. An example of this can be seen in the Probabilistic Robotics textbook [40, figure 10.3]. However, updating the quadratic covariance matrix can be expensive, and it is sometimes necessary to split the map into submaps that are updated separately [41].

EKF SLAM requires knowing the data association (which landmark generated which observation). When the data association is unknown, EKF SLAM uses heuristics to "guess" the correct data association. We will discuss some of these data association heuristics in the next section. EKF SLAM is very sensitive to data association errors. A single data association error can make the EKF map estimate diverge, causing the pose estimate to diverge as well [6]. It

would be possible to duplicate the EKF and track multiple hypotheses whenever there is data association ambiguity, but this has not been pursued because of the large computational cost [6].

The mathematics for EKF SLAM assumes that the number of landmarks is known and fixed. To support a variable number of landmarks in practice, EKF SLAM adds unexplained observations to a provisional landmark list [11, 41]. When a provisional landmark accumulates enough observations to be considered confirmed, EKF SLAM simply enlarges the μ vector and the Σ matrix to include the new landmark.

Graph-based methods take a completely different approach. These methods build a sparse graph, with nodes for the robot pose at each timestep and for the location of each landmark. The edges in this graph encode soft geometric constraints between the nodes. We can think of the graph as a spring-mass model, whose state of minimal energy is exactly the MAP solution to SLAM [41]. Graph-based methods are inherently offline: first they accumulate all the observations into a graph, then they use optimization techniques to find the full trajectory and map that best explain the observations.

Since graph-based methods have access to the full history of observations, they can compute the probability that two features (nodes in the graph) have the same world coordinates (i.e., represent the same landmark). This allows graph-based methods to perform data association iteratively, either in a greedy way [40, chapter 11], or using RANSAC or branch-and-bound [41].

One sub-flavor of graph-based methods is scan-matching, which tries to match laser readings from one timestep to the next one, therefore inferring the displacement in the robot's location between the two timesteps. Related to scan-matching are visual SLAM methods, which do the same timestep-to-timestep matching using images instead of laser readings [39].

Particle-based methods such as FastSLAM are a third major category of SLAM solutions. These methods are based on particle filtering, which approximate a probability distribution using a set of samples called particles. Particle-based methods are typically used to solve the online SLAM problem. The key insight in FastSLAM is that the SLAM posterior factorizes into independent landmark location estimation problems, if we condition on the robot poses [13,30–32]. This means that instead of maintaining a Gaussian distribution over all the landmarks jointly, like EKF SLAM, we can just store an individual Gaussian distribution for each landmark. This insight allows FastSLAM to scale to much larger maps than the EKF methods.

Like EKF methods, particle-based methods require knowing the data association. If the data association is unknown, particle methods use heuristics to "guess" the correct data association. We will discuss some of these heuristics in the next section. Particle-based SLAM is more robust to data association errors than EKF SLAM, because each particle makes its own data association decision, so it is possible to recover from an association error, as long as at least some particles got the correct association.

Also similar to the EKF methods, the mathematics for the particle-based methods does not handle the uncertainty about the number of landmarks in the environment. Instead, FastSLAM uses heuristics to decide when to add and when to remove a landmark from the map. We will discuss these heuristics in more detail in section 3.

Of the three paradigms described in this section, particle-based methods are the most suitable for a PPS-based SLAM solution, because particle filtering is one of the building blocks available in a PPS. In contrast, EKF methods and graph-based methods do not fit into the sampling paradigm of a PPS. Moreover, we would like our PPS-based SLAM solution to handle data association uncertainty in a principled way, and particle-based methods are an elegant way to do this, since each particle can carry its own data association hypothesis.

In the rest of this thesis we will focus on a specific particle-based method (FastSLAM) aiming to understand the heuristics that this method uses, so that we can work towards a PPS SLAM solution.

2.4 Data Association in SLAM

Multiple authors have argued that data association is a central problem in SLAM [5, 6, 35]. Good surveys of data associations in SLAM are given in Tim Bailey's PhD thesis [5, chapter 3] and a tutorial article by Bailey and Durrant-Whyte [6]. Data association approaches fall into two categories: individual association and batch association.

In **individual association**, each observation–landmark association is considered independently. An observation is assigned to a particular landmark if the distance between them is below a fixed threshold. This approach is also known as nearest-neighbor association, or maximum-likelihood association. This is the simplest possible data association method, and the one used in early EKF SLAM [11] and FastSLAM [30] implementations. As we illustrate in Figure 1, individual association fails when the pose uncertainty is larger than the distance between landmarks, when false detections occur near the landmark locations, or when an observation could plausibly be associated to more than one landmark [33].

A simple way to extend FastSLAM to cases with large association uncertainty is as follows: When there are multiple plausible association hypotheses, split each particle into one particle per hypothesis [35]. The particles with the wrong association hypotheses will die eventually, because they will have lower weights than the particles with the correct association hypothesis. Another approach to reducing association uncertainty is to associate landmarks not only based on their location, but also based on a "signature" or "fingerprint", such as color, shape, or image patch [9, 17]. Yet another approach is to perform data association lazily, by maintaining a tree of possible association decisions, rather than a single association hypothesis [22]. Each node in the tree stores the likelihood of the association decisions up to that point, and tree nodes are expanded in a manner similar to A* search.

In **batch association**, the observation–landmark associations for all observations received at a single timestep are considered together. The two main approaches are Joint Compatibility Branch and Bound [33] and Combined Constraint Data Association [5], which are based on tree search and graph search respectively. These methods are not based on sampling, and it is hard to see how they could be integrated with a PPS. In sections 4 and 5 we will present sampling-based methods for batch association that could be implemented in a PPS.

The data association problem has been studied extensively in the context of Multi-Target Tracking (MTT). In MTT, we have targets moving independently of each other in a region of surveillance. We get noisy observation of their locations over some period of time, and the goal is to infer the trajectories of the targets. The MTT problem is similar to the SLAM problem: the SLAM landmarks can be seen as targets that simply do not move. Additionally, in MTT the observer is typically stationary, whereas in SLAM the observer moves. These differences are important, because in MTT the target locations are independent, whereas in SLAM they are correlated through the unknown robot pose.

Well-studied approaches to MTT include Multiple Hypothesis Tracking (MHT) [38], which maintains multiple Kalman Filters for the different plausible hypotheses, Joint Probabilistic Data Association (JPDA) [7, 8], which computes the probability of all target-observation pairs, and often requires approximations to reduce the exponential number of association hypotheses considered, and MCMC Data Association (MCMCDA) [36], which uses a reversible-jump MCMC to sample from the posterior over associations and elegantly handles the case where the number of targets is unknown. MCMCDA will be an important component of the algorithm we will develop in section 5. MCMC approaches to MTT have also been extended to the case where targets interact [23].

Another source of ideas for data association techniques is the Structure From Motion (SFM) problem in computer vision. In SFM, we have a collection of photographs of the same scene (e.g., the Coliseum in Rome), and the goal is to reconstruct the 3D geometry of the scene. In the process, we end up also estimating the pose of the camera that took each photograph. The SFM problem is similar to the SLAM problem: the scene we are trying to reconstruct corresponds to the map in SLAM, and the pose of the camera in each photograph corresponds to the pose of the robot at each timestep. The main difference is that in SLAM, the robot pose is constrained by the dynamics model, whereas in SFM, the camera locations are completely unconstrained. For this reason, the data association problem (also known as the correspondence problem in computer vision) is much harder in SFM than it is in SLAM.

There are efficient MCMC methods for sampling the correspondence between features in a pair of images [10]. SFM scales to very large applications, such as reconstructing the 3D geometry of an entire city from hundreds of thousands of photographs [3]. Achieving this kind of scale requires careful engineering and many different layers of approximations to keep the amount of computation tractable.

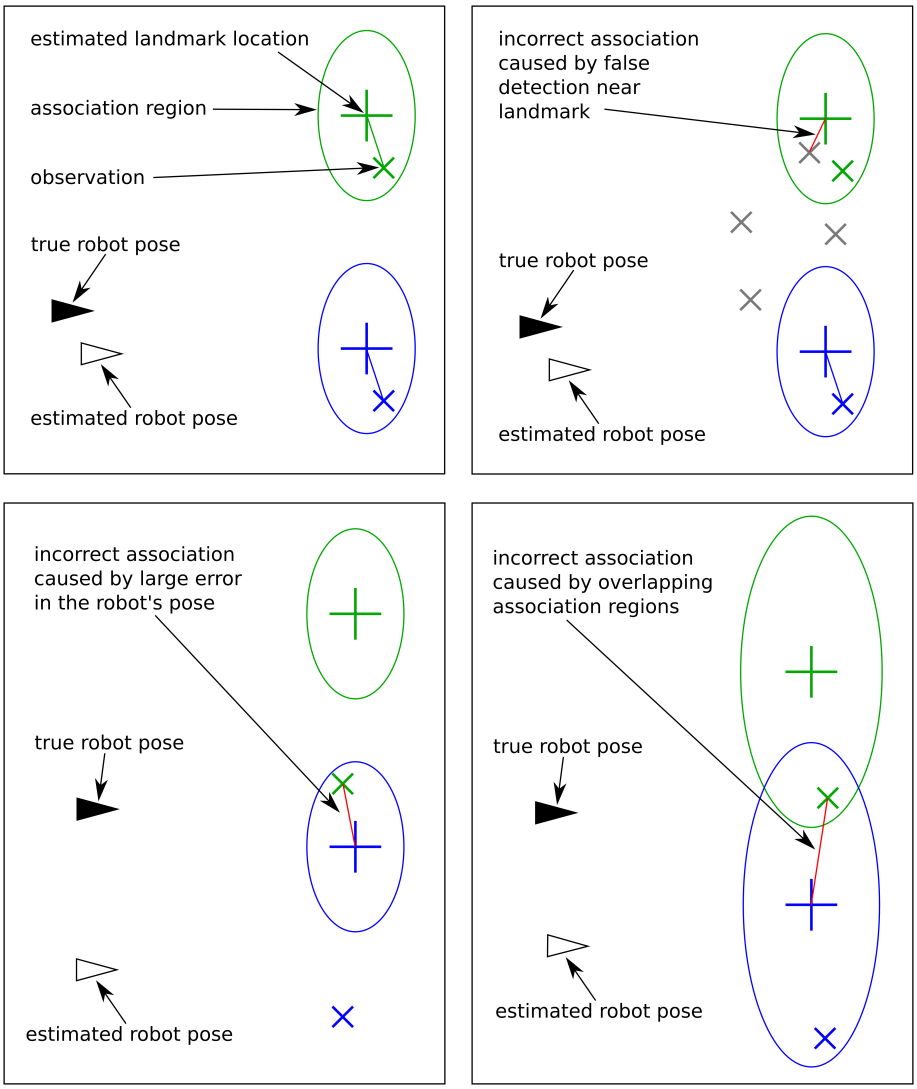


Fig. 1: Examples of individual nearest-neighbor association. Each landmark (+) has an association region (ellipse), based on the uncertainty in that landmark's location and the noise in the observation model. Each observation (×) is associated to its nearest landmark, if the observation falls in that landmark's association region. If there are multiple observations falling in the same landmark's association region, the nearest is chosen. This strategy may choose incorrect associations if there are many false alarms (top right), if there is large pose uncertainty (bottom left), or if an observation falls in the association region of multiple landmarks (bottom right).

3 FastSLAM and Its Limitations

In this section we aim to understand FastSLAM and the limitations that get in the way of implementing it in a PPS. In section 3.1 we describe the probabilistic model used by FastSLAM, and in section 3.2 we discuss the inference algorithm. In section 3.3 we summarize the heuristics that FastSLAM uses and the approximations it makes. To illustrate FastSLAM's limitations, we design an environment with large data association uncertainty, which we present in section 3.4. We discuss the error metrics used for SLAM in section 3.5, and evaluate FastSLAM in the aforementioned environment in section 3.6. Its poor accuracy points to improvements we will make in section 4.

3.1 The FastSLAM Probabilistic Model

FastSLAM uses a probabilistic model that is very similar to the general SLAM model discussed in section 2.2. The initial pose is known:

$$x_1 = x_{\text{init}}$$

The pose at the next timestep is obtained by evaluating the dynamics model and adding some Gaussian noise:

$$x_t | x_{t-1}, u_t \ \sim \ \mathcal{N}(f(x_{t-1}, u_t), \Sigma_{\text{pose}}) \tag{1}$$

where the dynamics model $f(x_{t-1}, u_t)$ is a deterministic function that takes the previous pose x_{t-1} and the controls u_t and returns the new pose of the robot.

The robot's sensors typically produce multiple observations at each timestep; for example, there are multiple obstacles visible in a single laser rangefinder scan. FastSLAM considers these observations one at a time. This means that a single sensor reading with 5 observations corresponds to 5 separate timesteps in FastSLAM, 4 of which have zero time elapsed between them and leave the pose unchanged.

The variable ω_t denotes the landmark that generated the observation y_t. Each observation is equally likely a priori to be generated by any of the N_{t-1} landmarks known up to that point:

$$\omega_t \sim \text{Uniform}(\{1, ..., N_{t-1}\})$$

The map θ is a list of landmark locations θ_j, each of which has a uniform prior over W, the world region that the robot is operating in:

$$\theta_j \sim \text{Uniform}(W)$$

Each observation is obtained by evaluating the observation model and adding some Gaussian noise:

$$y_t | x_t, \omega_t, \theta \ \sim \ \mathcal{N}(g(x_t, \theta_{\omega_t}), \Sigma_{\text{obs}})$$

where the observation model $g(x_t, \theta_j)$ is a deterministic function that takes the pose x_t and the landmark location θ_j and returns the observation corresponding to that landmark from that pose.

Taken together, the above equations describe the probabilistic model used by FastSLAM. We can visualize this model as a Dynamic Bayesian Network (DBN), shown in Figure 2a. For a particular association $\omega_{1:t}$, we can illustrate the model more clearly by drawing a separate node for each landmark, as shown in Figure 2b. Since the controls u_t are always observed, we do not add them as nodes in the graphical model, but instead encode them into the probability distributions on the edges $x_{t-1} \to x_t$.

3.2 The FastSLAM Algorithm

The key insight in the FastSLAM algorithm [30] is that given the observations $y_{1:t}$, the associations $\omega_{1:t}$, and the robot poses $x_{1:t}$, the problems of estimating the landmark locations θ_j become decoupled from each other. This can be easily seen from Figure 2b: θ_1 and θ_2 are conditionally independent given $x_{1:t}$, $\omega_{1:t}$, and $y_{1:t}$.

FastSLAM works by running a particle filter to track the pose of the robot, x_t. Instead of storing a concrete value for each landmark's location θ_j, FastSLAM approximates each landmark location distribution using a Gaussian:

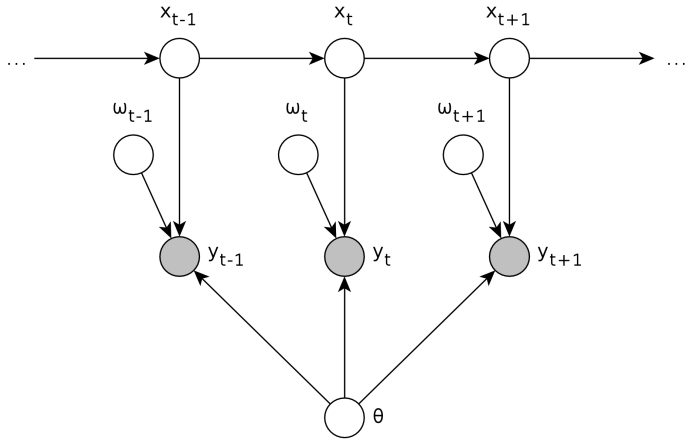
$$p(\theta_j | x_{1:t}, \omega_{1:t}, y_{1:t}) \approx \mathcal{N}(\mu_{j,t}, \Sigma_{j,t})$$

and stores only the sufficient statistics $\mu_{j,t}$ and $\Sigma_{j,t}$ for each landmark j. When a new observation for landmark j arrives, FastSLAM updates these sufficient statistics in closed form, which is equivalent to running the measurement update in an Extended Kalman Filter. (If the observation model is linear in θ_j, then no linearization is required, and we can use a plain Kalman Filter.)

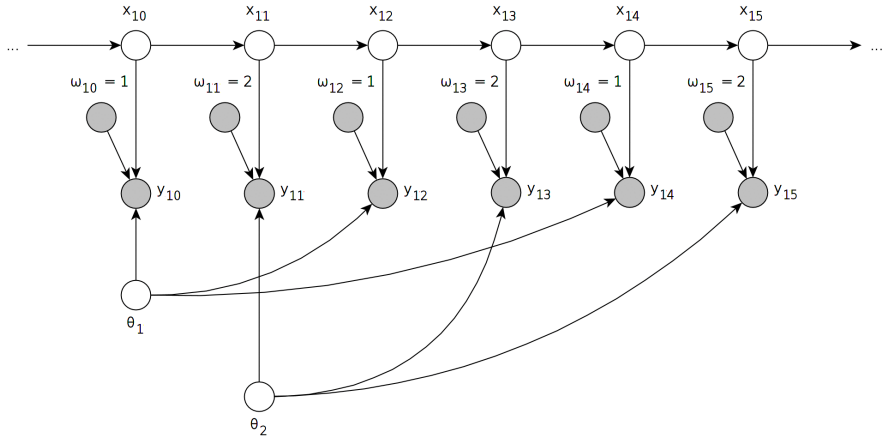
At timestep t, each particle stores a concrete value for the pose x_t, and a distribution for each landmark: $\mu_{j,t}$, $\Sigma_{j,t}$. Thus FastSLAM is a Rao-Blackwellized particle filter [30, 31], where the pose distribution is represented using particles and the landmark location distributions are represented analytically.

The final question is that of data association: How do we determine which landmark generated each observation? The FastSLAM model captures the source of observation y_t using the association variable ω_t. The FastSLAM algorithm simply imputes concrete values for ω_t using the maximum likelihood principle. When a new observation y_t comes in, FastSLAM computes the likelihood that it was generated by each known landmark, and sets ω_t to the landmark with the highest likelihood. If the highest likelihood is below a threshold τ_{new}, FastSLAM adds a new landmark to the map, and associates the observation to that new landmark.

Since every observation is associated to an existing landmark or to a new landmark, any false detections will cause spurious landmarks to be added to the map. To remedy this, FastSLAM keeps track of a "seen count" c_j for each landmark j in each particle. When processing a batch of observations from the



(a) Three time slices in the FastSLAM DBN. The x nodes are the poses, ω are the associations, y are the observations, and θ is the map, shown here as a single node.



(b) A particular association ω in the FastSLAM DBN. The landmark locations θ_1 and θ_2 are shown as separate nodes.

Fig. 2: The FastSLAM DBN.

same sensor reading, c_j is incremented for the landmarks that were seen and decremented for the landmarks that were in the field of view but were not seen. If c_j drops below zero, FastSLAM removes landmark j from the map. (This assumes that there are no detection failures. If the probability of detection p_{det} is less than one, we can modify the counters c_j to maintain the log-odds of a landmark's existence [40].) This ad-hoc method allows FastSLAM to remove landmarks that were added to the map because of false detections.

Figure 3 shows one timestep of the FastSLAM algorithm. The Probabilistic Robotics textbook [40, chapter 13] provides more detailed pseudocode for the algorithm, which is a good blueprint for a first implementation. (Make sure to check the online errata, since the printed version has a few typos.)

3.3 Heuristics and Approximations

FastSLAM is very successful and robust in practice, but it uses some ad-hoc techniques that make it difficult to implement in a PPS. In this section we summarize these heuristics and approximations, so that we have a clear idea of what we need to change in order to make FastSLAM suitable for a PPS.

- **Maximum-likelihood data association.** FastSLAM associates each observation greedily to the landmark that most likely generated it, and does not consider other association hypotheses.
- **Mutual exclusion constraint not enforced.** When multiple observations arrive at the same timestep, FastSLAM processes them sequentially and independently from each other. This means that multiple simultaneous observations can be associated to the same landmark, which is logically inconsistent with the fact that at each timestep, each landmark produces at most one observation. FastSLAM implementations either ignore this constraint, or use a heuristic "repair" step to fix inconsistent associations [40, section 7.8].
- **Hard threshold for adding a new landmark.** When an observation's best association has likelihood under a certain threshold, FastSLAM uses that observation to initialize an EKF for a new landmark, and does not consider other possible explanations for that observation.
- **Heuristics for incorporating negative information.** FastSLAM maintains a "seen count" c_j for each landmark j. When the landmark is observed, its counter is incremented. When the landmark is in the field of view but it is not observed, its counter is decremented. When the counter drops below zero, the landmark is deleted from the map. This heuristic allows FastSLAM to remove false landmarks from the map. However, this does not fully incorporate negative information. When not observing a landmark, our belief that it lies outside the field of view should increase. But FastSLAM does not update a landmark's location belief in the absence of an observation.
- **Other heuristics for map management.** When the rate of false detections is high, FastSLAM requires additional heuristics to prevent polluting the map with false landmarks. It is common to keep a provisional map [40, section 13.6], and promote landmarks from the provisional map to the main map only after they have been observed a sufficient number of times.

The heuristics described above are necessary for a practical implementation of FastSLAM. Unfortunately, they are not suitable for a PPS solution to SLAM, because they do not have clean probabilistic interpretations in a generative model. Moreover, in cases where there is significant association uncertainty, these heuristics actually hurt the accuracy of FastSLAM, as we show next.

Inputs:

- A list B_{t-1} of N particles approximately distributed according to $p(x_{1:t-1} \mid y_{1:t-1})$.
- The new observations y_t.
- Parameters for the model: controls, noises.

Output:

- A list B_t of N particles approximately distributed according to $p(x_{1:t} \mid y_{1:t})$.

Algorithm:

1. **Advance step**
   For each particle i:
   Propose the new pose $x_t^{(i)}$ from the prior (equation 1).
   Set the association $\omega_t^{(i)}$ to the landmark that maximizes the likelihood of the observation:

$$\omega_t^{(i)} \;=\; \arg\max_{j=1}^{N_{t-1}} p(y_t^{(i)} | x_t^{(i)}, \omega_t^{(i)} = j, \theta^{(i)})$$

   If the observation likelihood with the best association is less than τ_{new}, create a new landmark by setting $\omega_t^{(i)} = N_{t-1} + 1$.
   Compute the weight of the particle, which is equal to the observation likelihood:

$$w_t^{(i)} \;=\; p(y_t^{(i)} | x_t^{(i)}, \omega_t^{(i)}, \theta^{(i)})$$

2. **Resample step**
   Resample N particles from B_t in proportion to their weights.
3. **Exact step**
   For each particle, update the sufficient statistics for the location distribution of the landmark $\theta_j^{(i)}$, where $j = \omega_t^{(i)}$ is the landmark to which the latest observation was associated.
4. **Map management step**
   (runs once after every batch of observations from a single sensor reading)
   Update "seen counts" for each particle i:
   Increment $c_j^{(i)}$ for the landmarks j that received observations.
   Decrement $c_j^{(i)}$ for the landmarks j that were in the field of view, but did not receive observations.
   Delete from the map any landmarks j with $c_j^{(i)} < 0$.

Fig. 3: The FastSLAM algorithm for timestep t.

3.4 `mutex`: A Dataset with High Observation Noise

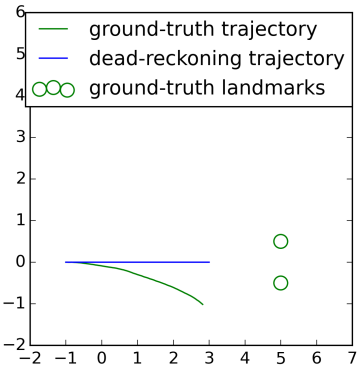
To show the limitations of FastSLAM, we design an environment with large data-association uncertainty. The `mutex` dataset represents a robot moving in a 2D environment. First, we will describe the motion model. The state vector has 3 dimensions: x, y, and the orientation γ. The controls are the steering α and velocity v at every timestep, obtained from the robot's odometry sensors. (Thus it would be more accurate to call this an odometry model, rather than a dynamics model.) The robot moves according to the kinematic model used in the PPAML SLAM challenge problem [2, 21]. There is independent Gaussian noise on each of the 3 state components, with standard deviations $\sigma_x = 0.001$, $\sigma_y = 0.001$, and $\sigma_\gamma = 0.03$ respectively. We chose a higher amount of rotational noise to increase localization uncertainty.

Figure 4a shows the true trajectory of the robot, as well as the dead-reckoning trajectory obtained by applying the controls with no noise. We can see that because of the high rotational noise, the true trajectory diverges quite a bit from the dead-reckoning trajectory. Figure 4b shows several trajectories sampled from the motion prior, i.e., trajectories obtained by applying the controls with noise. We can see that after a few timesteps, there is a lot of uncertainty about the robot's position and orientation. The hope is that SLAM will be able to reduce this uncertainty by localizing the robot with respect to landmarks in the environment.

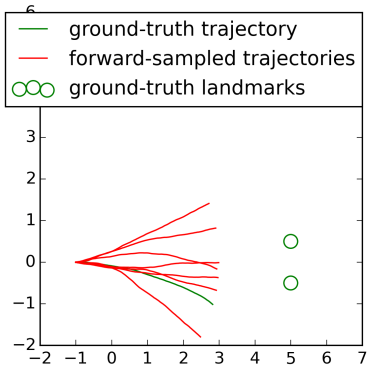
We will now describe the observation model. There are two circular landmarks in the environment, indicated by the green circles in 4a. Each landmark generates a point observation d_x, d_y indicating its location relative to the robot. (This is equivalent to converting range-and-bearing observations into Cartesian coordinates. Typically range-and-bearing observations are extracted from the readings of some other sensor, such as a laser rangefinder. To keep this example as simple as possible, we assume that this extraction step has been performed already, and the point observations d_x, d_y are given to the SLAM algorithm directly.)

For simplicity, we assume that both landmarks are detected in every timestep and that there are no false detections elsewhere in the viewing area. There is independent Gaussian noise on each of the 2 dimensions of the observations, with standard deviations $\sigma_{d_x} = 0.2$ and $\sigma_{d_y} = 0.2$ respectively. The high observation noise introduces a lot of data-association uncertainty. Figure 4c shows the observations plotted with respect to the ground-truth trajectory, and figure 4d shows the ground-truth data association.

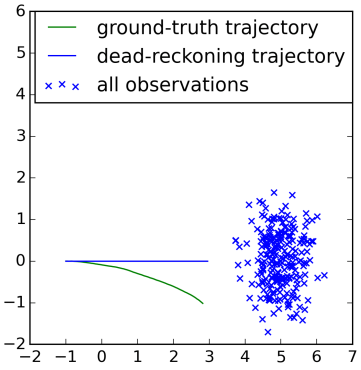
Figure 4c makes it intuitively clear why this is a difficult dataset. The observations appear to form a single cluster, and the only indication that there are actually two landmarks is that at every timestep, we get two observations instead of one. But FastSLAM cannot take advantage of this knowledge, because it processes each observation individually and does not enforce the mutual exclusion constraint.



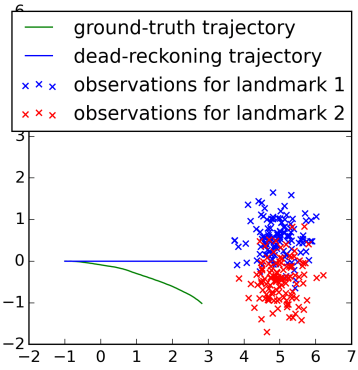
(a) The ground-truth and dead-reckoning trajectories.



(b) Some trajectories sampled from the motion prior.



(c) Observations from both landmarks, without color coding.



(d) Observations color-coded by their source landmark.

Fig. 4: The `mutex` dataset, with high rotational noise and high observation noise, resulting in large association uncertainty.

3.5 Error Metrics for Measuring SLAM Accuracy

We use a set of common metrics for evaluating SLAM accuracy throughout this thesis. To obtain the predicted trajectory, we take the mean of the filtering distribution at each timestep. At timestep t, the predicted pose is:

$$\hat{x}_t = \frac{1}{N} \sum_{i=1}^{N} x_t^{(i)}$$

Then if $x_{1:T}$ represents the ground-truth trajectory, we compute the error as:

$$\text{trajectory error} = \sqrt{\sum_{t=1}^{T} (x_t - \hat{x}_t)^2}$$

To obtain the predicted map, we simply take the map from an arbitrary particle. Since all particles have equal weights after the resample step, this is equivalent to taking one sample from the posterior distribution over maps. By the end of a SLAM run, this posterior distribution is usually peaked, so a single sample represents it well. (In cases where there is still a lot of map uncertainty at the end of the run, it would be better to take multiple samples from the posterior, i.e., to consider the maps from multiple particles instead of just one.)

For the map accuracy we report two values: the number of spurious landmarks and a measure of map dissimilarity. The number of spurious landmarks is the difference between the number of landmarks in the SLAM map and the number of landmarks in the ground-truth map. (So if the SLAM map has more landmarks than the true map, this number will be positive, and if the SLAM map has fewer landmarks than the true map, this number will be negative. Zero is the best value.)

The map dissimilarity measure is a bit more tricky, since it has to compare maps with different numbers of landmarks. Let P be the set of ground-truth landmarks, and $\hat{P}$ be the set of predicted landmarks from the SLAM map. We will define a grid of points L that cover the map. At each of these grid points $q \in L$, we will quantify the "influence" of P or $\hat{P}$ on that point:

$$\mu_q(P) = \max_{p \in P} \left(\exp\left(\frac{-||q - p||_2}{2} \right) \right)$$

The total map error is simply the sum of "influence" differences at each point on the grid:

$$\text{map error} = \sum_{(x,y) \in L} |\mu_{x,y}(P) - \mu_{x,y}(\hat{P})|$$

The PPAML documentation [1] provides additional details and an example.

3.6 Results on the `mutex` Dataset

Figure 5 shows the results of running FastSLAM on the `mutex` dataset for 10 trials using 1000 particles. (We verified that increasing the number of particles beyond 1000 does not help.) We show results for different values of the τ_{new} parameter, which controls how willing FastSLAM is to create new landmarks. (FastSLAM creates a new landmark for an observation when its best association has log-likelihood less than τ_{new}.)

When τ_{new} is very small, FastSLAM always prefers to associate observations to existing landmarks. Figure 6a shows a typical map produced in this scenario. The observations for both landmarks are "mixed" into a single landmark, resulting in an inaccurate map. The resulting trajectory is still very good, showing

that the localization part of FastSLAM is robust to this kind of error. (It is like looking at two sources of light without your glasses: You perceive them as a single source of light, but you can still localize yourself with respect to them.)

When τ_{new} is slightly higher, FastSLAM creates a new landmark for an observation that is sufficiently far from existing landmarks. Figure 6b shows a typical map produced in this case. Observations get "mixed" into one landmark, as before, but then a distant observation creates a second landmark. After that, FastSLAM begins associating observations to both landmarks, but because of false data associations in the past, the landmark locations are incorrect, causing the trajectory to drift.

When τ_{new} is even higher, FastSLAM is even more willing to create new landmarks instead of associating observations to existing landmarks. Figure 6c shows a typical map produced in this case, where the observations are split into three landmarks, and the resulting confusion again causes the trajectory to drift.

When τ_{new} is extremely high, FastSLAM always prefers to create a new landmark, rather than associate an observation to an existing landmark. This means that each landmark is observed only once. The map-management heuristics kick in and delete the landmark from the map. Thus, numerous landmarks are added and then swiftly removed from the map, never reaching the threshold to be confirmed. We end up with an empty map, as shown in Figure 6d. Since we have no landmarks to help localize the robot, all particles have equal weights, so the mean trajectory is the same as the dead-reckoning trajectory (because we are just averaging zero-mean noise).

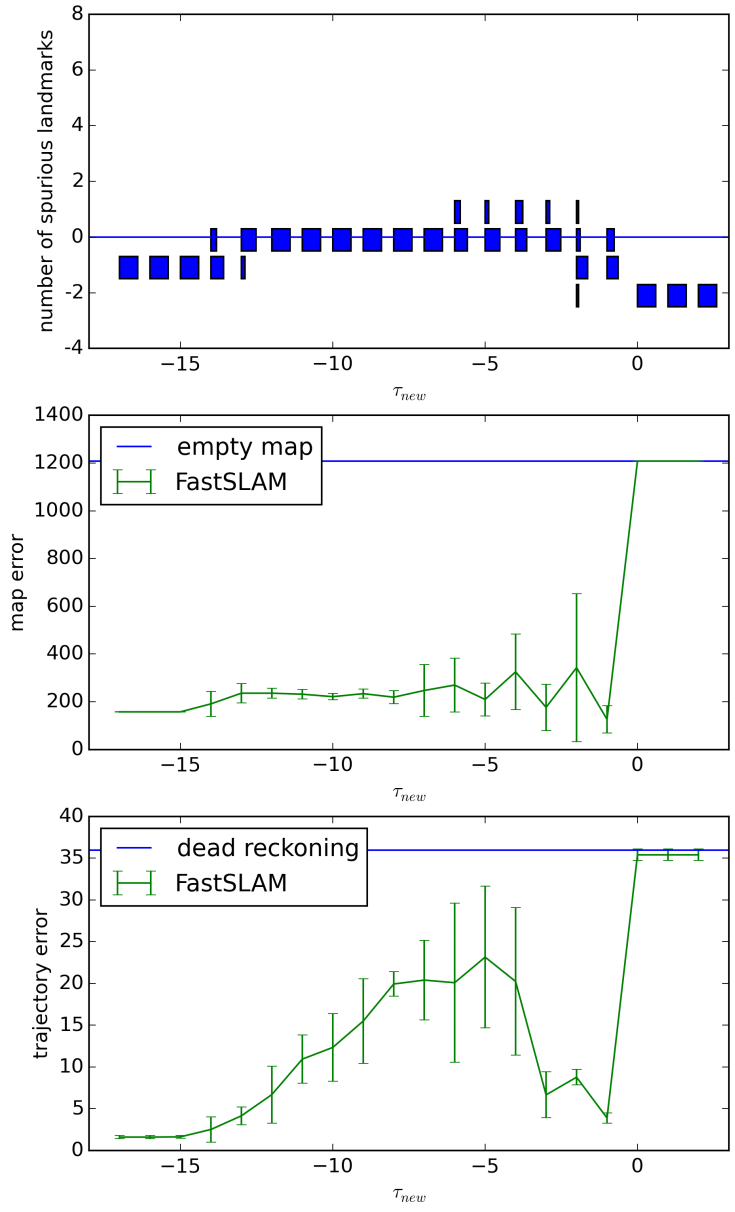
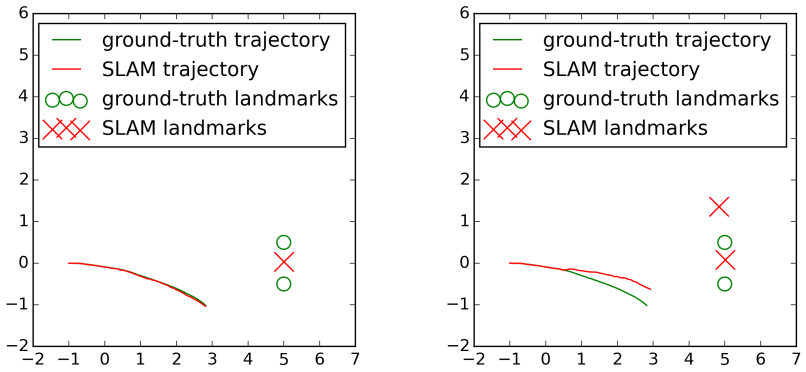
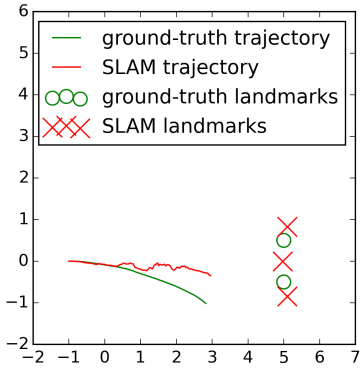


Fig. 5: FastSLAM results on the `mutex` dataset, with 1000 particles and different values for the τ_{new} parameter. Top: histogram showing number of spurious (positive) or undetected (negative) landmarks. Center: average and standard deviation of map error. Bottom: average and standard deviation of trajectory error. All statistics are based on 10 trials.

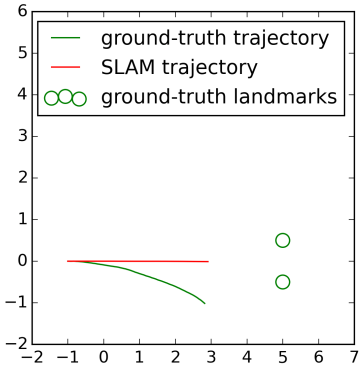


(a) $\tau_{\text{new}} = -16$

(b) $\tau_{\text{new}} = -9$



(c) $\tau_{\text{new}} = -4$



(d) $\tau_{\text{new}} = 0$

Fig. 6: Typical maps produced by FastSLAM on the `mutex` dataset, for different values of the τ_{new} parameter.

4 FastSLAM-DA: Sampling Data Associations

In the last section we described FastSLAM and demonstrated its limitations on an example dataset. FastSLAM has limited accuracy because it does not enforce the mutual exclusion constraint between observations. Additionally, FastSLAM uses heuristic methods for data association and map management, which make it unsuitable for implementation in a PPS.

In this section we present a new algorithm, FastSLAM-DA (FastSLAM with Data Association), which addresses these limitations. In section 4.1 we describe a probabilistic model that makes it possible to enforce the mutual exclusion constraint. Then in section 4.2 we explain why the forward-sampling proposal is ineffective in this model, and suggest a smart (data-driven) proposal instead. We describe the full FastSLAM-DA algorithm in section 4.3. In section 4.4 we evaluate the results of FastSLAM-DA on the `mutex` dataset, and show that it beats FastSLAM. Finally, in sections 4.5 and 4.6 we show an example dataset for another regime, where FastSLAM-DA does not do so well. This paves the way for the improvements that we will make in section 5.

4.1 A Probabilistic Model for SLAM with Association Uncertainty

We modify the FastSLAM probabilistic model by handling multiple observations at the same timestep, which allows us to enforce the mutual exclusion constraint. We also have a different probability distribution on the association variables, which allows us to handle new landmarks without relying on ad-hoc methods. The initial pose is known, as before:

$$x_1 = x_{\text{init}} \tag{2}$$

The pose at the next timestep is given by the dynamics model with Gaussian noise, as in FastSLAM:

$$x_t | x_{t-1}, u_t \ \sim \ \mathcal{N}(f(x_{t-1}, u_t), \Sigma_{\text{pose}}) \tag{3}$$

We now describe the observation model. Suppose that at timestep t we received K_t observations: $y_t = (y_t[1], ..., y_t[K_t])$. Furthermore, suppose that we already knew the association vector $a_t = (a_t[1], ..., a_t[K_t])$ indicating the source of every observation. For an observation coming from a previously existing landmark θ_j, we denote $a_t[k] = j$, and the observation is given by the Gaussian observation model:

$$y_t[k] \mid x_t, \theta_j \ \sim \ \mathcal{N}(g(x_t, \theta_j), \Sigma_{\text{obs}}) \tag{4}$$

For an observation coming from a landmark that appears for the first time at this timestep, we denote $a_t[k] = +$, and the observation is uniform over the visible region R:

$$y_t[k] \mid x_t \ \sim \ \text{Uniform}(R(x_t)) \tag{5}$$

For a false detection, we denote $a_t[k] = 0$, and the observation is again uniform over the visible region:

$$y_t[k] \mid x_t \sim \text{Uniform}(R(x_t)) \tag{6}$$

We now need to specify a generative process for the association vectors a_t. Informally, at timestep t, each of the landmarks existing at the previous timestep produces an observation with probability p_{det}, otherwise it produces no observation (missed detection). In addition, a number of new landmarks m_t appear (possibly zero), and each of them produces an observation. Finally, a number of false detections f_t occur (possibly zero), and each of them also causes an observation.

Formalizing this generative process involves a bit of gnarly notation. We have to go through this exercise to ensure that we have a valid probabilistic model. (Afterwards, we will show an example to make things more clear.) Suppose there are N_{t-1} landmarks existing at the previous timestep. At timestep t, for each pre-existing landmark $i \in \{1, ..., N_{t-1}\}$, there is a variable

$$d_{i,t} \sim \text{Bernoulli}(p_{\text{det}})$$

that indicates whether that landmark was detected. Let

$$S_t^{\text{old}} = \{i | i \in \{1, ..., N_{t-1}\} \text{ s.t. } d_{i,t} = 1\}$$

be the set of previously existing landmarks detected at timestep t. The number of new landmarks is

$$m_t \sim \text{Poisson}(\lambda_{\text{new}})$$

where λ_{new} is the rate of new landmarks appearing. Let

$$S_t^{\text{new}} = \{+_1, ..., +_{m_t}\}$$

be the set of new landmarks appearing at timestep t, where $+_i$ are arbitrary unique labels. The number of false detections is

$$f_t \sim \text{Poisson}(\lambda_{\text{false}})$$

where λ_{false} is the rate of false detections occurring. Let

$$S_t^{\text{false}} = \{0_1, ..., 0_{f_t}\}$$

be the set of "false landmarks" at timestep t, where 0_i are arbitrary unique labels different from the $+_i$ labels in S_t^{new}. The set of sources producing observations at timestep t is then the union of these three sets:

$$S_t = S_t^{\text{old}} \cup S_t^{\text{new}} \cup S_t^{\text{false}}$$

To generate the association vector a_t, we choose a random permutation of S_t:

$$a_t | S_t \sim \text{Uniform}(\text{Permutations}(S_t))$$

and drop the subscripts on the labels $+_i$ and 0_i.

The generative process described above sounds complicated, but we can simplify the notation by defining the association variable $\omega_t = (N_{t-1}; a_t)$, which contains the number of pre-existing landmarks and the association vector at timestep t. A value ω_t determines the values of variables $d_{i,t}$, m_t, f_t, as well as the sets S_t, S_t^{old}, S_t^{new}, and S_t^{false}. For example, consider the value $\omega_t = (5; 0, 2, 0, +, 4)$. We immediately read off $N_{t-1} = 5$ and $a_t = [0, 2, 0, +, 4]$. Since we know N_{t-1}, we have that $S_t^{\text{old}} = \{1, 2, 3, 4, 5\}$. By seeing which past landmarks (positive integers) appear in a_t, we get $d_{1,t} = 0$, $d_{2,t} = 1$, $d_{3,t} = 0$, $d_{4,t} = 1$, and $d_{5,t} = 0$. By counting the number of new landmarks (+s) and false detections (0s) in a_t, we get $m_t = 1$ and $f_t = 2$. From here, we get that $S_t^{\text{new}} = \{+_1\}$ and $S_t^{\text{false}} = \{0_1, 0_2\}$, and $S_t = \{1, 2, 3, 4, 5, +_1, 0_1, 0_2\}$. And finally, $N_t = N_{t-1} + m_t = 6$.

The probability distribution for the association variable ω_t can now be expressed as:

$$\begin{aligned} p(\omega_t|\omega_{t-1}) &= \left(\prod_{i=1}^{N_{t-1}} p(d_{i,t})\right) p(m_t)p(f_t)p(a_t|S_t) \\ &= (p_{\text{det}})^{n_t^{\text{det}}} \cdot (1-p_{\text{det}})^{n_t^{\text{undet}}} \cdot \left(\frac{\lambda_{\text{new}}^{m_t}}{m_t!} e^{-\lambda_{\text{new}}}\right) \cdot \left(\frac{\lambda_{\text{false}}^{f_t}}{f_t!} e^{-\lambda_{\text{false}}}\right) \cdot \frac{1}{|S_t|!} \end{aligned} \quad (7)$$

where $n_t^{\text{det}} = |S_t^{\text{old}}|$ is the number of previous landmarks that were detected, and $n_t^{\text{undet}} = N_{t-1} - n_t^{\text{det}}$ is the number of previous landmarks that were not detected.

Suppose that we have K_t observations at timestep t. It is possible for the generative process on associations to produce an association vector a_t that expects a different number of observations, i.e. $K_t \neq \text{length}(a_t)$. In that case, we simply set the probability of observations to zero. Therefore the probability of the observations is:

$$\begin{aligned} &p\left(y_t = (y_t[1], ..., y_t[K_t]) \mid x_t, \omega_t = (N_{t-1}; a_t), \theta\right) \\ &\qquad = \begin{cases} \prod_{k=1}^{K_t} p(y_t[k] \mid x_t, a_t[k], \theta) & \text{if } K_t = \text{length}(a_t) \\ 0 & \text{if } K_t \neq \text{length}(a_t) \end{cases} \end{aligned} \quad (8)$$

Each of the terms $p(y_t[k] \mid x_t, a_t[k], \theta)$ is given by equation 4, 5, or 6, depending on the association $a_t[k]$.

Figure 7 shows the DBN for the probabilistic model we just described. As in FastSLAM, we omit the controls u_t, which are always observed. For the association variables, we include only ω_t, since as we have shown, $d_{i,t}$, m_t, f_t can be determined unambiguously from ω_t. The conditional probability distributions on the nodes of this DBN are given by equations 3, 7, and 8. The initial pose is given by equation 2. The only distributions left to specify are the association prior $p(\omega_1)$ and the location prior $p(\theta_j)$ for each landmark j. The association

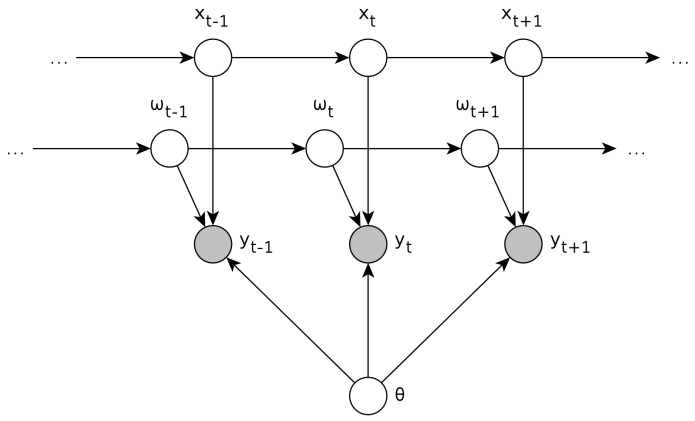


Fig. 7: The FastSLAM-DA DBN, showing three time slices. The x nodes are the poses, ω are the associations, y are the observations, and θ is the map, shown here as a single node.

prior at the first timestep can be expressed using equation 7, if we define the dummy variable $\omega_0 = (0;)$:

$$P(\omega_1) = P(\omega_1|\omega_0)$$

Alternatively, if we expect a large number of landmarks to be visible from the beginning, we can define ω_1 using a different rate λ_{init} that is higher than λ_{new}.

In our model, whenever the association variable creates a new landmark (i.e. there is a + in a_t), a new node θ_j for the location of that landmark is added at timestep t, where j is the next available index. The prior on landmark locations is uniform over the 2D world that the robot is operating in:

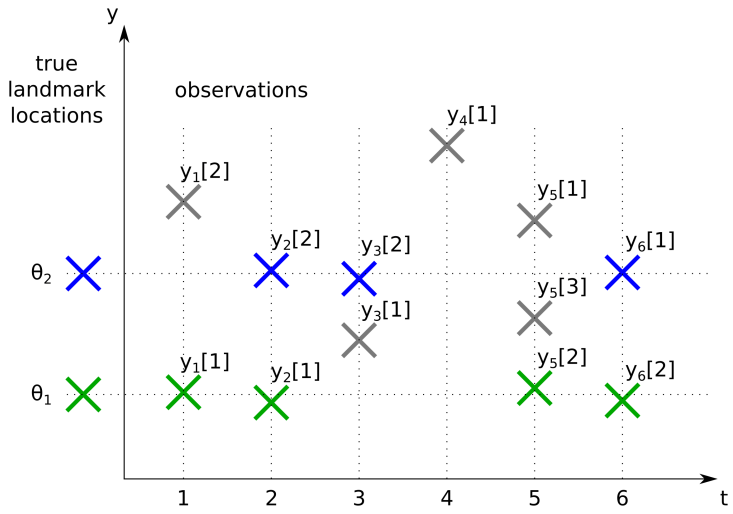
$$\theta_j \sim \text{Uniform}(W)$$

Alternatively, if the visible region is much smaller than the entire world, we can define $\theta_j|x_t \sim \text{Uniform}(R(x_t))$, where the function R gives the visible region from a particular pose. In this case, we would have to add an edge from the x_t node to any landmarks created at timestep t.

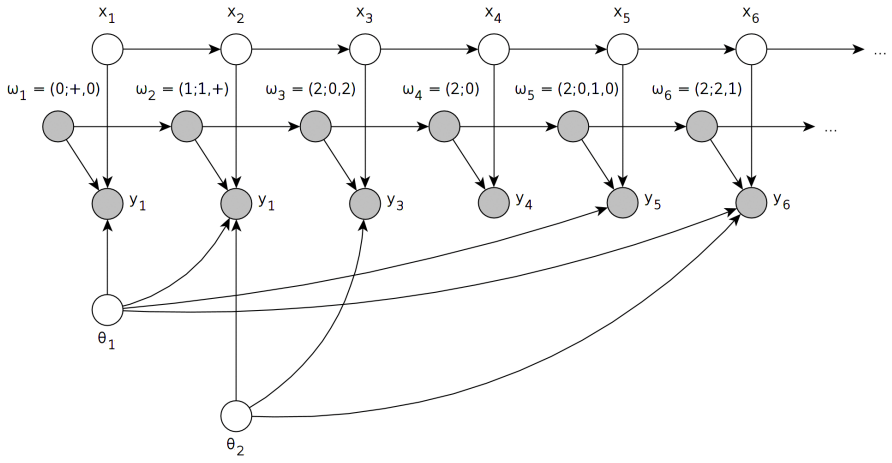
To gain a concrete understanding of our probabilistic model of associations, we show a small example. For illustration purposes, we assume that the robot is stationary, and the observations are one-dimensional. Suppose we receive the observations shown in Figure 8a, where the colors indicate the correct association. Then the true association hypothesis is as shown in Table 1. The DBN for this specific example is shown in Figure 8b. The nodes θ_1 and θ_2 are created at timesteps 1 and 2 respectively, when the association variable contains +s. Once a landmark node θ_j is created, it will exist for all future timesteps.

This model automatically enforces the mutual exclusion constraint. An association such as $\omega_t = (4; 1, 1, 2, 3)$, which assigns both $y_t[1]$ and $y_t[2]$ to landmark

θ_1, has probability zero. This is because the association vector $a_t = [1, 1, 2, 3]$ cannot arise from the generative process described above, since the set of sources S_t cannot contain the element 1 more than once.



(a) Example scenario with two landmarks. The colors indicate the correct association. False detections are shown in gray. The observations at each timestep are in an arbitrary order (i.e. they are not ordered by distance / angle / etc).



(b) The FastSLAM-DA DBN for the correct association hypothesis, showing the landmark locations θ_1 and θ_2 as separate nodes.

Fig. 8: Example illustrating how the association variables work.

timestep	association	interpretation
$t = 1$	$\omega_1 = (0; +, 0)$	there are 0 previous landmarks; $y_1[1]$ comes from a new landmark θ_1; $y_1[2]$ is a false detection
$t = 2$	$\omega_2 = (1; 1, +)$	there is 1 previous landmark; $y_2[1]$ comes from the existing landmark θ_1; $y_2[2]$ comes from a new landmark θ_2
$t = 3$	$\omega_3 = (2; 0, 2)$	there are 2 previous landmarks; $y_3[1]$ is a false detection; $y_3[2]$ comes from the existing landmark θ_2
$t = 4$	$\omega_4 = (2; 0)$	there are 2 previous landmarks; $y_4[1]$ is a false detection
$t = 5$	$\omega_5 = (2; 0, 1, 0)$	there are 2 previous landmarks; $y_5[1]$ is a false detection; $y_5[2]$ comes from the existing landmark θ_1; $y_5[3]$ is a false detection
$t = 6$	$\omega_6 = (2; 2, 1)$	there are 2 previous landmarks; $y_6[1]$ comes from the existing landmark θ_2; $y_6[2]$ comes from the existing landmark θ_1

Table 1: The correct association hypothesis for the example in Figure 8a.

4.2 A Smart Proposal for Associations

The obvious inference algorithm for our new model would be a Rao-Blackwellized particle filter similar to FastSLAM. We would maintain Rao-Blackwellized distributions for each landmark θ_j in each particle. In the advance step, we would sample the pose x_t and the association ω_t from their transition models. In the resample step, we would use the observations y_t to compute the weight of each particle.

In the FastSLAM model, where each DBN time slice contains a single observation, this would be a feasible approach. Indeed, it is possible to spawn a new particle for every possible data-association hypothesis [35], and let the particles with poor data association die off naturally because they have lower weights. If there are N known landmarks, then for each observation there are $N + 2$ hypotheses: one for associating the observation to each of the N existing landmarks, one for associating the observation to a new landmark, and one for treating the observation as a false detection.

But in the FastSLAM-DA model, each DBN time slice contains all observations received at a particular timestep. The number of association hypotheses to consider grows exponentially with the number of known landmarks. If there are N_{t-1} known landmarks and K_t observations, and ignoring the possibility of false detections and new landmarks, there are $\frac{N_{t-1}!}{(N_{t-1}-K_t)!}$ association hypotheses. The transition model $P(\omega_t|\omega_{t-1})$ places equal mass on these hypotheses, since

all landmarks are equally likely to generate observations, and the observations are equally likely to arrive in any order. If we proposed associations ω_t from the transition model, for $N_{t-1} = 6$ and $K_t = 6$, only 1 in $6! = 720$ particles will have the correct association, and all other particles will have low weights and will be killed in the resample step. This will make the particle filter perform very poorly, since it loses all particle diversity at every timestep.

To find a way around this problem, recall that the particle filtering framework is not restricted to proposing from the transition model, but instead allows other proposal distributions [12]. In our model, the most general proposal takes the form

$$q(x_t, \omega_t \mid x_{1:t-1}, \omega_{1:t-1}, y_{1:t})$$

and the weight w (not to be confused with the association ω) of particle i is computed as

$$w^{(i)} \propto \frac{p(x_t^{(i)}|x_{t-1}^{(i)}) \ \cdot \ p(\omega_t^{(i)}|\omega_{t-1}^{(i)}) \ \cdot \ p(y_t|x_{1:t}^{(i)}, \omega_{1:t}^{(i)}, y_{1:t-1})}{q(x_t^{(i)}, \omega_t^{(i)} \mid x_{1:t-1}^{(i)}, \omega_{1:t-1}^{(i)}, y_{1:t})}$$

To keep the notation light, we will omit the $^{(i)}$ superscript where it does not lead to confusion. The forward-sampling ("bootstrap filter") proposal is

$$q_{\text{forward}}(x_t, \omega_t \mid x_{1:t-1}, \omega_{1:t-1}, y_{1:t}) \ = \ p(x_t|x_{t-1}) \cdot p(\omega_t|\omega_{t-1})$$

which makes the weight formula simplify to

$$w_{\text{forward}} \propto p(y_t|x_{1:t}, \omega_{1:t}, y_{1:t-1})$$

As we have shown above, this proposal ignores the latest observations y_t, which leads to many poor associations being proposed. This in turn leads to a loss of particle diversity, and poor accuracy of the particle filter.

We now define our smart proposal as an alternative to the forward-sampling proposal. We will propose the pose x_t by forward sampling, as before. But when we propose the association ω_t, we will take into account the latest observations y_t, by preferring associations that lead to a high observation likelihood. If we have K_t observations at timestep t, our smart proposal takes the form

$$q_{\text{smart}}(x_t, \omega_t \mid x_{1:t-1}, \omega_{1:t-1}, y_{1:t}) \ = \ p(x_t|x_{t-1}) \cdot \prod_{k=1}^{K_t} \pi(\omega_t[k] \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) \quad (9)$$

Therefore the weight of a particle can be computed as

$$w_{\text{smart}} \propto \frac{p(\omega_t|\omega_{t-1}) \ \cdot \ p(y_t|x_{1:t}, \omega_{1:t}, y_{1:t-1})}{q_{\text{smart}}(x_t, \omega_t \mid x_{1:t-1}, \omega_{1:t-1}, y_{1:t})} \quad (10)$$

The only term left to specify is the proposal for an individual association, $\pi(\omega_t[k] \mid x_{1:t}, \omega_{1:t-1}, y_{1:t})$. The variable $\omega_t[k]$ may take any value from the set

$C_t = \{1, ..., N_{t-1}, +, 0\}$, corresponding to an association to an existing landmark, a new landmark, and a false detection, respectively. We define

$$\pi(\omega_t[k] = c \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) \;=\; \frac{\text{score}(\omega_t[k] = c \mid x_{1:t}, \omega_{1:t-1}, y_{1:t})}{\sum_{c' \in C_t} \text{score}(\omega_t[k] = c' \mid x_{1:t}, \omega_{1:t-1}, y_{1:t})}$$

For an association to a past landmark, the score is given by the likelihood of that landmark generating the observation $y_t[k]$:

$$\text{score}(\omega_t[k] = j \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) \;=\; p(y_t[k] \mid x_t, \omega_t[k] = j, x_{1:t-1}, \omega_{1:t-1}, y_{1:t-1})$$

To compute this likelihood we use $\alpha_{t-1}[j]$, the Rao-Blackwellized distribution parameters (mean and variance) for the position of landmark j after incorporating all observations associated to that landmark in timesteps $1 : t-1$. Therefore, with a slight abuse of notation, we can say:

$$\text{score}(\omega_t[k] = j \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) \;=\; p(y_t[k] \mid x_t, \alpha_{t-1}[j])$$

For the remaining two hypotheses (new landmark and false detection), the scores are given by respective constants, which are tunable parameters of the proposal:

$$\begin{aligned} \text{score}(\omega_t[k] = + \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) &= \exp(\phi_{\text{new}}) \\ \text{score}(\omega_t[k] = 0 \mid x_{1:t}, \omega_{1:t-1}, y_{1:t}) &= \exp(\phi_{\text{fa}}) \end{aligned}$$

Our smart proposal in equation 9 proposes an association for each observation independently of the others. This means that it can propose associations that violate the mutual exclusion constraint. This is not a problem, since in equation 10, the term $p(\omega_t|\omega_{t-1})$ will be zero for any ω_t that violates the mutual exclusion constraint. Therefore, particles that violate the mutual exclusion constraint will automatically get a weight of zero.

The smart proposal is data-driven, meaning that it takes into account the latest observations y_t when proposing the association ω_t. A greedy data-driven proposal has been suggested before for multi-target tracking [15]. The greedy proposal considers targets in a fixed order, and enforces the mutual exclusion constraint. This means that if observation $y_t[k]$ can be explained equally well by targets θ_i and θ_j with $i < j$, the greedy proposal will be biased in favor of θ_i, since it considers θ_i before θ_j. By proposing associations independently from each other and then assigning weight zero to associations violating the mutual exclusion constraint, our proposal avoids this bias.

Our proposal can be qualitatively summarized as follows: If there is a single "obvious" association for observation $y_t[k]$, that association will have a high likelihood score, so it will be chosen with high probability. If there are multiple plausible associations for the observation, the proposal will choose among them in proportion to their likelihood scores. If there are no plausible associations, then all past landmarks will have low scores, and the proposal will choose between the + (new landmark) and 0 (false detection) hypotheses, in accordance to the relative magnitude of the parameters ϕ_{new} and ϕ_{fa}.

4.3 The FastSLAM-DA Algorithm

The FastSLAM-DA algorithm is an application of Rao-Blackwellized particle filtering [31] on the probabilistic model described in section 4.1, using the smart proposal described in section 4.2. Figure 9 shows one timestep of the algorithm.

Recall that it is possible for the proposal to suggest associations that violate the mutual exclusion constraint. This means that after advancing N particles, some of them have a weight of zero, and do not survive the resampling step. We modify the textbook particle filter slightly, to prevent this loss of particle diversity. In particular, instead of simply advancing the N existing particles, we repeat the entire advance step until we have at least N resulting particles with non-zero weight. This is equivalent to making k copies of each particle before the advance step. It is easy to show that this maintains the correctness of the particle filter: Since the particles have uniform weights before the advance step, the original N particles and the copied kN particles represent the same distribution.

It is possible to view our advance step as a rejection sampler, which samples from the proposal again and again until it has at least N non-zero-weight particles. One concern is that there could be too many rejections, which would slow down the particle filter. Rejections occur especially when observations are close to each other, thus "competing" for being explained by the same landmark. In our case, the observations are extracted from laser rangefinder data, so two observations are rarely very close to each other. (If two sources were that close to each other, one of them would occlude the other, and we would only get one observation.) This explains why we haven't seen large amounts of rejections in our experiments with FastSLAM-DA. If a different observation model were used, we might need some other technique to avoid excessive rejections.

FastSLAM-DA differs from FastSLAM in the following ways:

- FastSLAM performs heuristic data association (choose the landmark with maximum likelihood, and use a hard to decide when to add a new landmark). FastSLAM-DA samples data associations instead.
- FastSLAM processes one observation at a time, and does not enforce the mutual exclusion constraint. FastSLAM-DA processes all observations at a timestep together, and disallows associations that violate the mutual exclusion constraint.
- FastSLAM uses map-management heuristics to remove spurious landmarks from the map. FastSLAM-DA does not use such heuristics, instead relying on particle diversity to prune away map hypotheses that are not supported by the data. Particles with spurious landmarks will have smaller weights, because those landmarks are not detected in subsequent timesteps.

FastSLAM-DA would be straightforward to implement in a PPS that supported Rao-Blackwellized particle filtering with an interface for specifying a custom proposal. This is a big improvement over FastSLAM, which cannot be implemented in a PPS because of its ad-hoc heuristic techniques.

Inputs:

- A list B_{t-1} of N particles approximately distributed according to $p(x_{1:t-1}, \omega_{1:t-1} \mid y_{1:t-1})$.
- The new observations y_t.
- Parameters for the model: controls, noises, etc.
- Parameters for the proposal: ϕ_{new} and ϕ_{fa}.

Output:

- A list B_t of N particles approximately distributed according to $p(x_{1:t}, \omega_{1:t} \mid y_{1:t})$.

Algorithm:

1. **Advance step**
   Let B_t be an empty list.
   While length$(B_t) < N$:
   For each particle $(x_{1:t-1}^{(i)}, \omega_{1:t-1}^{(i)})$ in B_{t-1}:
   Propose $(x_t^{(i)}, \omega_t^{(i)})$ according to equation 9.
   Compute the weight $w_t^{(i)}$ according to equation 10.
   If $w_t^{(i)} > 0$, add the new particle to B_t.
2. **Resample step**
   Resample N particles from B_t in proportion to their weights.
3. **Exact step**
   Update the Rao-Blackwellized distributions on landmark locations in each particle, according to any new observations that were associated to those landmarks.

Fig. 9: The FastSLAM-DA algorithm for timestep t.

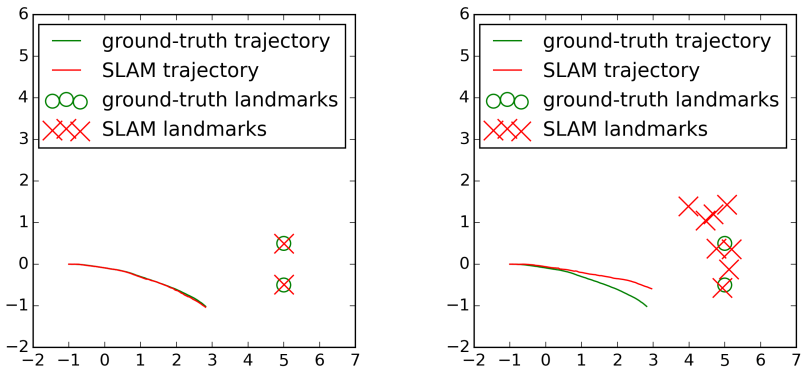
4.4 Results on the `mutex` Dataset

In section 3.4, we introduced the `mutex` dataset, on which FastSLAM does not do very well because it ignores the mutual exclusion constraint and uses a hard threshold to decide when to add new landmarks to the map. We expect FastSLAM-DA to do better on this dataset, because it chooses associations by sampling, and because it handles mutual exclusion properly. In this section we investigate the accuracy of FastSLAM-DA on the `mutex` dataset.

Figure 11 shows the results of running FastSLAM-DA with 1000 particles on the `mutex` dataset. (We verified that there is no significant improvement in accuracy from using more than 1000 particles.) Since this dataset has no false detections, we set $\phi_{\text{fa}} = -\infty$, which causes false-detection hypotheses to never be

proposed. We vary ϕ_{new} to see how it influences accuracy. We superimpose these results on the FastSLAM results from Figure 5 for comparison. (The FastSLAM parameter τ_{new} and the FastSLAM-DA parameter ϕ_{new} have slightly different semantics, due to the normalization constants elided when computing particle weights. For this reason we consider a large range for these parameters, to see how the two algorithms behave across this range.)

We can clearly see that FastSLAM-DA achieves a trajectory accuracy that is as good as FastSLAM with the best settings of τ_{new}, while producing maps that are much more accurate. The increased map accuracy is thanks to FastSLAM-DA's ability to enforce the mutual exclusion constraint. FastSLAM-DA works well for many settings of the ϕ_{new} parameter, and only fails when ϕ_{new} is so high that the "new landmark" hypotheses wins over the "associate to existing landmark" hypothesis. Figure 10 shows some typical maps produced by FastSLAM-DA. With a good setting of ϕ_{new}, FastSLAM-DA recovers a near-perfect map and trajectory (Figure 10a). With a setting of ϕ_{new} that is too high, FastSLAM-DA adds too many false landmarks to the map, and loses its ability to localize the robot accurately (Figure 10b).



(a) $\phi_{\text{new}} = -8$ (b) $\phi_{\text{new}} = +2$

Fig. 10: Typical maps produced by FastSLAM-DA on the `mutex` dataset, for different values of the ϕ_{new} parameter.

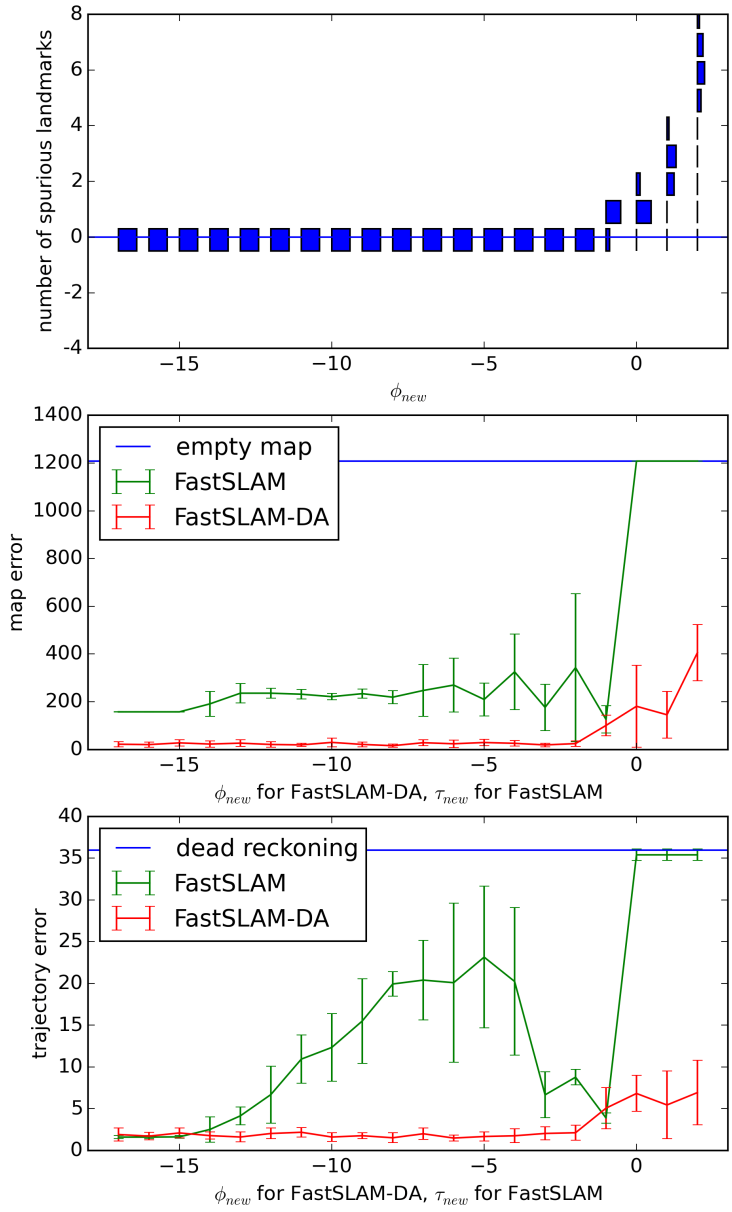


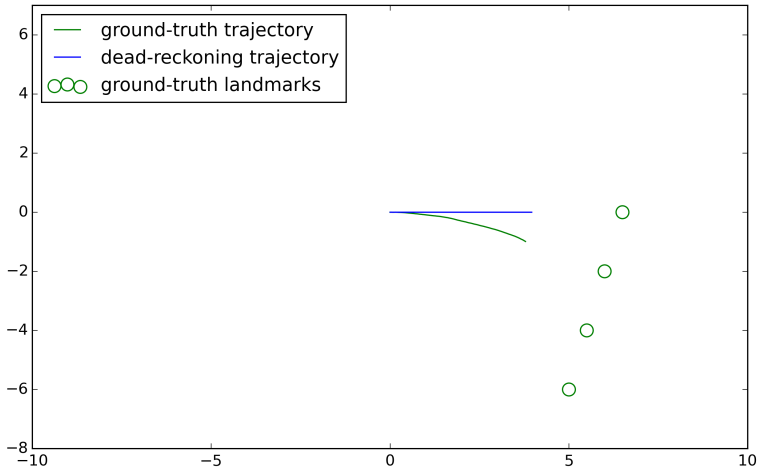
Fig. 11: FastSLAM-DA results on the `mutex` dataset, with 1000 particles and different values for the ϕ_{new} parameter. Top: histogram showing number of spurious (positive) or undetected (negative) landmarks. Center: average and standard deviation of map error. Bottom: average and standard deviation of trajectory error. All statistics are based on 10 trials. We also show the FastSLAM results for comparison.

4.5 `highfa`: A Dataset with a High Rate of False Detections

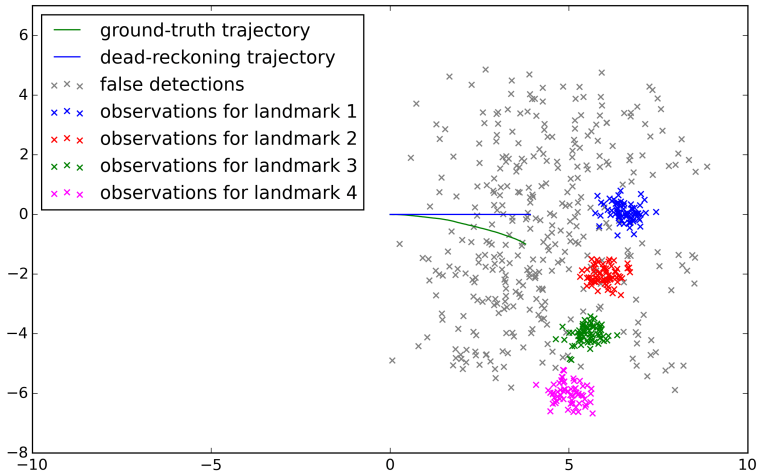
In the `mutex` dataset, the data-association uncertainty comes from high rotational noise in the dynamics model and high observation noise in the sensor model. But there is another scenario with high association uncertainty: when there are many false detections, a SLAM algorithm might add false landmarks to the map. To explore whether this is a problem, we design an environment with a high rate of false detections.

The `highfa` dataset is generated with the same dynamics model as the `mutex` dataset. We reduce the observation noise from $\sigma_{d_x} = \sigma_{d_y} = 0.2$ to $\sigma_{d_x} = \sigma_{d_y} = 0.1$. In addition, we generate false detections with a Poisson rate of $\lambda_{\text{fa}} = 5.0$ per timestep. In the real world, these false detections may be caused by laser sensor errors, or by temporary obstructions, such as people walking in front of the robot. When a false detection occurs, it may occlude any landmarks that are behind it.

There are four landmarks in the `highfa` dataset, shown in Figure 12a together with the ground-truth and dead-reckoning trajectories. The high rate of false detections causes a lot of data-association uncertainty, since any set of false detections that cluster together in time and space could be confused for a landmark. Figure 12b shows the observations plotted with respect to the ground-truth trajectory, and the colors show the ground-truth data association.



(a) The ground-truth and dead-reckoning trajectories.



(b) Observations color-coded by their source landmark.

Fig. 12: The `highfa` dataset, with high rotational noise and a high rate of false detections, resulting in large association uncertainty.

4.6 Results on the `highfa` Dataset

First we consider FastSLAM on the `highfa` dataset. As expected, FastSLAM does very poorly, because each false detection adds a new landmark to the map. Many of these landmarks eventually get deleted, because their "seen counts" c_j drop below zero, but some of them remain in the final map. It is possible

to obtain reasonable maps using FastSLAM only by introducing additional heuristics. For example, we could post-process the resulting maps by removing all landmarks that were observed fewer than 20 times. Or we could introduce a provisional map separate from the main map, as mentioned in section 3.3.

We now investigate the accuracy of FastSLAM-DA on the `highfa` dataset, as a function of the parameters ϕ_{new} and ϕ_{fa}, and the number of particles used. First, we fixed the number of particles to 1000. Running a preliminary grid search, we obtained reasonable results with $\phi_{\text{fa}} = -5$ and $\phi_{\text{new}} = -15$. We then altered these parameters one at a time, to see how they influence accuracy.

The ϕ_{new} parameter controls how willing FastSLAM-DA is to explain an observation by creating a new landmark. Figure 13 shows how this parameter influences the accuracy of FastSLAM-DA. If ϕ_{new} is too low, then FastSLAM-DA explains all observations as false detections, and fails to localize the robot. If ϕ_{new} is too high, FastSLAM-DA explains some false detections as landmarks, and adds these false landmarks to the map. We found no value of ϕ_{new} that achieved a satisfactory balance between these two behaviors. FastSLAM-DA either added spurious landmarks, or failed to add true landmarks, as shown in the examples in Figure 14.

The ϕ_{fa} parameter controls how willing FastSLAM-DA is to disregard an observation as a false detection. Extreme values of this parameter caused bad accuracy (empty map, or map full of false landmarks), and we found no value of ϕ_{fa} that gave us results better on average than those shown in Figure 13.

Using the best values found so far, $\phi_{\text{fa}} = -5$ and $\phi_{\text{new}} = -15$, we increased the number of particles up to 50000, but we did not see improved accuracy beyond 1000 particles. Shouldn't the particles with the correct map survive, and the ones with an incorrect map die during the resampling step?

To understand why FastSLAM-DA on this dataset nearly always produces maps with spurious landmarks, consider the case in Figure 15. In this simplified example, the observations are one-dimensional, and the robot is stationary. There is a single true landmark, shown in green. In timesteps 1:3, three false detections (shown in red) occur relatively near one another. Our model assigns very high probability to the hypothesis that these three detections come from a second landmark. In fact, after timestep 3, the hypothesis that there is a single landmark has probability 0.0000011, and the hypothesis that there are two landmarks has probability 0.9999989. This means that even with one million particles, on average only one particle will carry the one-landmark hypothesis.

After seeing additional observations in timesteps 4:10, we fail to see any more observations supporting the existence of the second landmark. Our model adjusts its belief accordingly, assigning probability 0.187 to the two-landmark hypothesis and 0.813 to the one-landmark hypothesis. However, if after timestep 3, all particles contain two landmarks, then there is no way for the particle filter to return to the one-landmark hypothesis. In FastSLAM, the map management heuristics would remove the false landmark some timesteps later, but in FastSLAM-DA, there is no mechanism to do so. We will address this limitation in the next section.

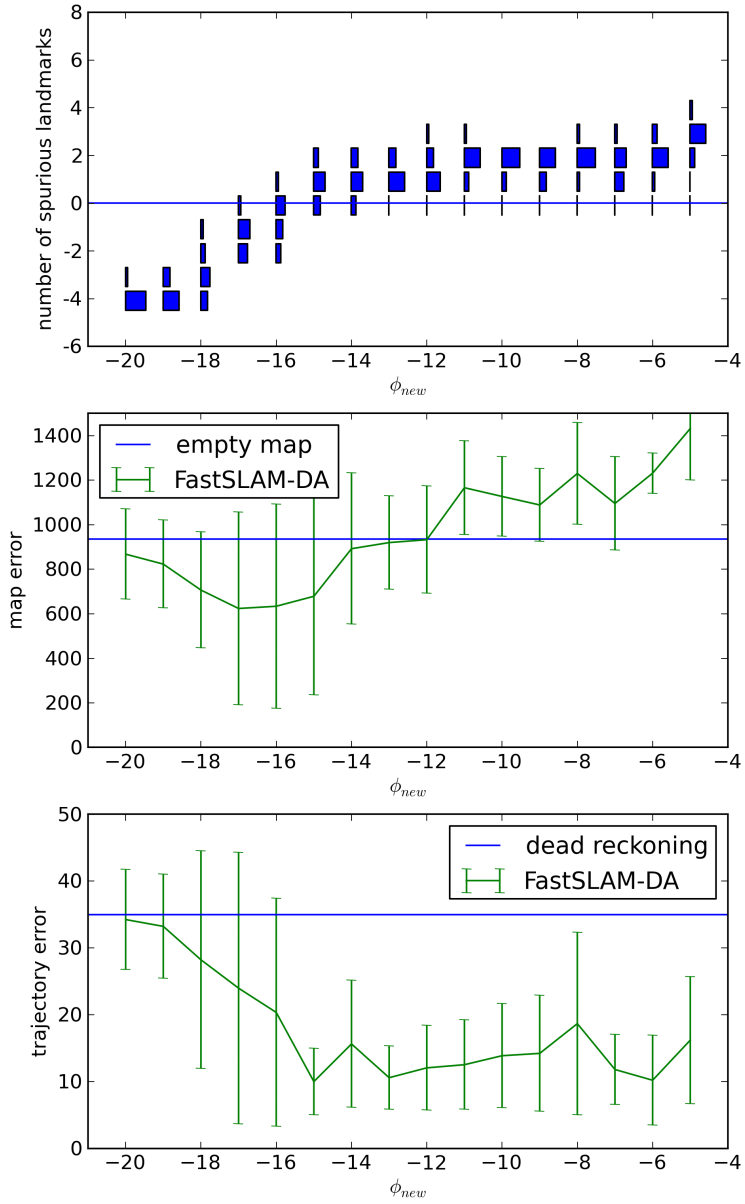
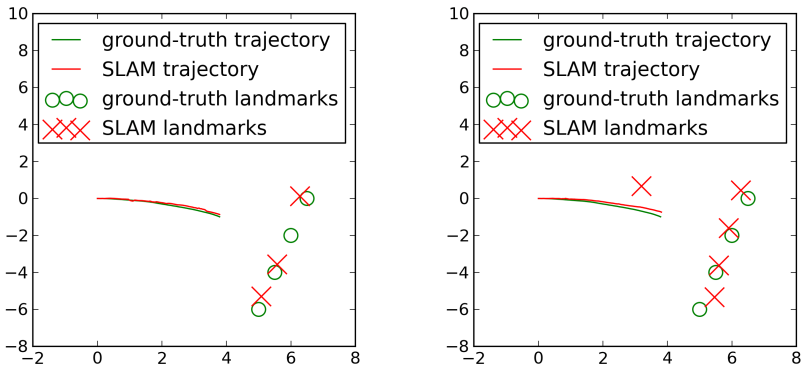


Fig. 13: FastSLAM-DA results on the `highfa` dataset, with 1000 particles, $\phi_{\text{fa}} = -5$, and different values for ϕ_{new} parameter. Top: histogram showing number of spurious (positive) or undetected (negative) landmarks. Center: average and standard deviation of map error. Bottom: average and standard deviation of trajectory error. All statistics are based on 10 trials.



(a) $\phi_{\text{new}} = -16$ (b) $\phi_{\text{new}} = -15$

Fig. 14: Typical maps produced by FastSLAM-DA on the `highfa` dataset, with a good setting of ϕ_{new}.

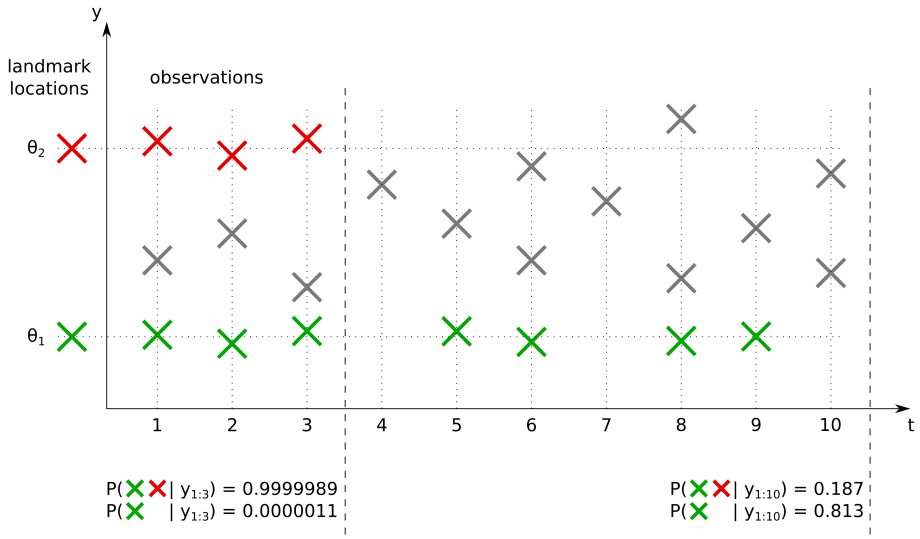


Fig. 15: An example showing how FastSLAM-DA adds a false landmark and is then unable to remove it. The true landmark is shown in green, and the false landmark is shown in red.

5 FastSLAM-DA-RM: Changing the Past

In the last section we described FastSLAM-DA, an algorithm that fixes many of the limitations in FastSLAM. FastSLAM-DA achieves better accuracy on the `mutex` dataset, and it removes the data-association and map-management heuristics. This makes it more suitable for implementation in a PPS. On the other hand, we have shown that FastSLAM-DA produces inaccurate maps on a dataset with a high rate of false detections. In this section, we present an algorithm that removes the latter limitation.

FastSLAM-DA relies on particle diversity to prune away inaccurate maps. As shown at the end of section 4.6, in a scenario with many false detections, it is possible for FastSLAM-DA to reach a state where all particles have a false landmark in their map. Subsequently, FastSLAM-DA has no way to remove that false landmark. This is an instance of sample degeneracy, a common problem in particle filters [14]. Intuitively, the particle filter cannot change its mind about decisions made in the past. When processing timestep t in particle i, the particle filter samples $x_t^{(i)}$ and $\omega_t^{(i)}$, but it cannot alter $x_{1:t-1}^{(i)}$ or $\omega_{1:t-1}^{(i)}$. This is illustrated in Figure 16a.

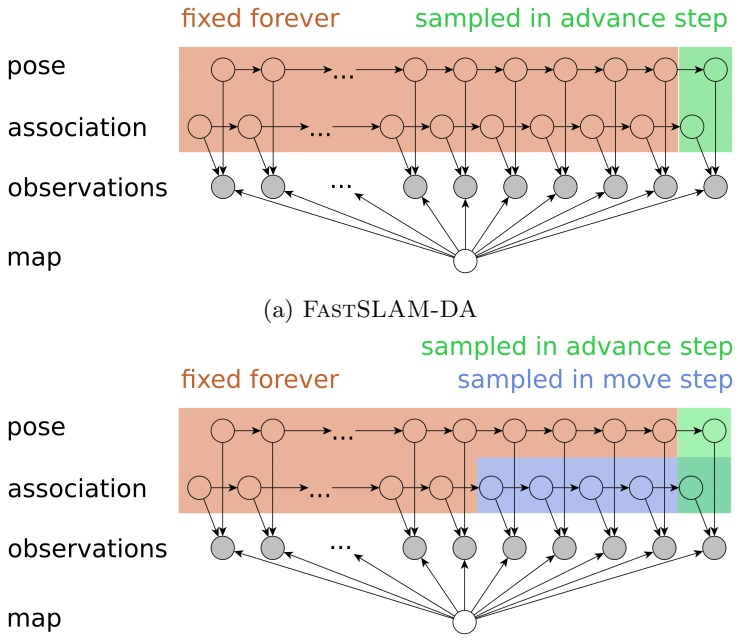
One way to avoid degeneracy in a particle filter is to introduce MCMC moves. After the resample step, the particles are approximately distributed according to $p(x_{1:t}, \omega_{1:t}|y_{1:t})$. If we pass the particles through an MCMC transition kernel with stationary distribution $p(x_{1:t}, \omega_{1:t}|y_{1:t})$, they will still be approximately distributed according to the same distribution, thus maintaining the correctness of the particle filter. The MCMC moves introduce some diversity in the particles, which can alleviate the degeneracy problem. This technique is known as Resample-Move [14, 18].

Our new algorithm, FastSLAM-DA-RM (FastSLAM with Data Association and Resample-Move), is obtained by applying the Resample-Move technique to the FastSLAM-DA algorithm. Instead of designing an MCMC to alter the full trajectory $x_{1:t}$ and association history $\omega_{1:t}$, we will follow the existing work on MCMC for data association [36], and only allow changing the association over a moving window of L timesteps: $\omega_{t-L+1:t}$. This is illustrated in Figure 16b.

In section 5.1 we describe the proposal in our MCMC algorithm, which is based on a set of reversible move pairs. In section 5.2 we show how to evaluate the target distribution in order to compute the acceptance ratio. In section 5.3 we verify the correctness of our MCMC algorithm on a small example where it is possible to compute the exact posterior by exhaustive search. Finally, we present the FastSLAM-DA-RM algorithm in section 5.4 and evaluate its results on the `highfa` dataset in section 5.5.

5.1 An MCMC Algorithm for Associations

The state of our Markov chain is given by the association variables in the past L timesteps: $s = \omega_{t-L+1:t}$. The target probability distribution is $\pi(s) = p(x_{1:t}, \omega_{1:t}|y_{1:t})$. (We will show exactly how to evaluate this target distribution in the next section.) Our MCMC algorithm is a standard Metropolis–Hastings



(a) FastSLAM-DA

(b) FastSLAM-DA-RM

Fig. 16: The nodes whose values change at the current timestep.

(MH) sampler [4] that uses a proposal distribution $q(s'|s)$. The basic algorithm is shown in Figure 17.

The meat of the MCMC algorithm is in the proposal distribution $q(s'|s)$. Inspired by existing work on MCMC for multi-target tracking [36], we will define a set of reversible move pairs that change the association $\omega_{t-L+1:t}$. We refer to the timesteps $1 : t - L$ as the *past* and to the timesteps $t - L + 1 : t$ as the *present*. This allows us to categorize landmarks into past-only, present-only, and past-and-present, as shown in Figure 18. We define the following move pairs, which we illustrate in Figure 19.

- **Birth:** Pick a group of false detections from the present and associate them to a new landmark. We select the false detections sequentially, just like the birth move in the MCMCDA algorithm [36].
  Death: Pick a present-only landmark and delete it, associating all its observations to false detections.
- **Push:** Pick a false detection o and associate it to an existing landmark l. This is possible only if the landmark l has no other observation at the same timestep as o.
  Pop: Pick an observation associated to a landmark and associate it to a false detection. Since a landmark with a single observation is indistinguishable

Input:

– An initial state $s^{(i)}$.

Output:

– The next state $s^{(i+1)}$.

Algorithm:

1. Sample a proposed state $s' \sim q(s'|s^{(i)})$.
2. Compute the acceptance ratio $\alpha = \min\left(1, \frac{\pi(s') \cdot q(s^{(i)}|s')}{\pi(s^{(i)}) \cdot q(s'|s^{(i)})}\right)$.
3. Sample a random value $u \sim \text{Uniform}(0, 1)$.
   If $u < \alpha$, accept the proposed state, and return $s^{(i+1)} = s'$.
   Otherwise, reject the proposed state, and return $s^{(i+1)} = s^{(i)}$.

Fig. 17: The Metropolis–Hastings sampler.

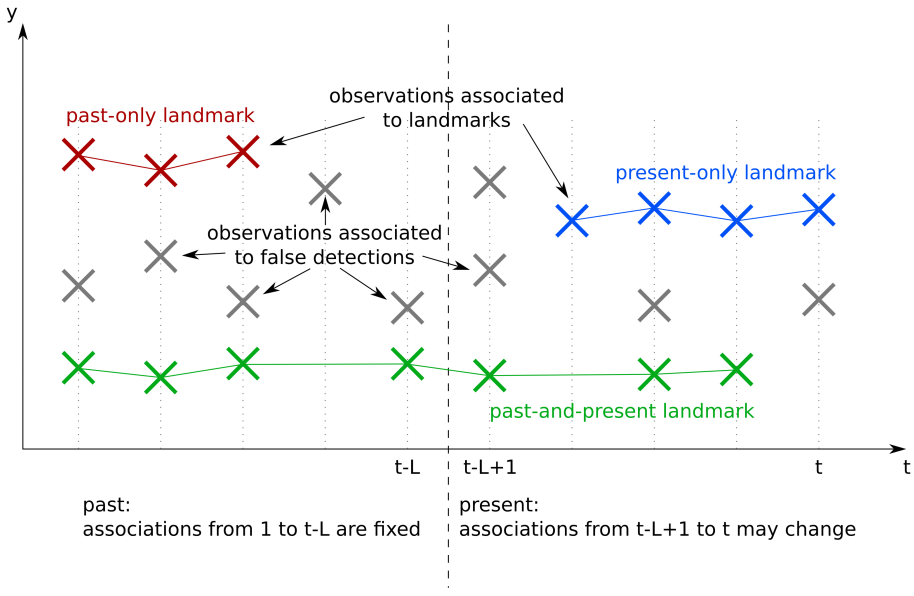


Fig. 18: An example association hypothesis, showing a past-only landmark, a present-only landmark, and a past-and-present landmark.

from a false detection, this move is possible only if the original landmark has at least 3 observations.

– **Merge:** Pick two landmarks l_1 and l_2, and merge them into a single landmark. One landmark must be present-only, while the other may be present-only or past-and-present. The two landmarks must have observations at disjoint timesteps.
  Split: Pick a landmark l and split it into two landmarks l_1 and l_2, by assigning each (present) observation in l randomly to either l_1 or l_2. Both l_1 and l_2 must end up with at least two observations each.
– **PastMerge:** Pick a present-only landmark and a past-only landmark, and merge them into a single past-and-present landmark.
  PastSplit: Pick a past-and-present landmark and split it into a past-only landmark and a present-only landmark.

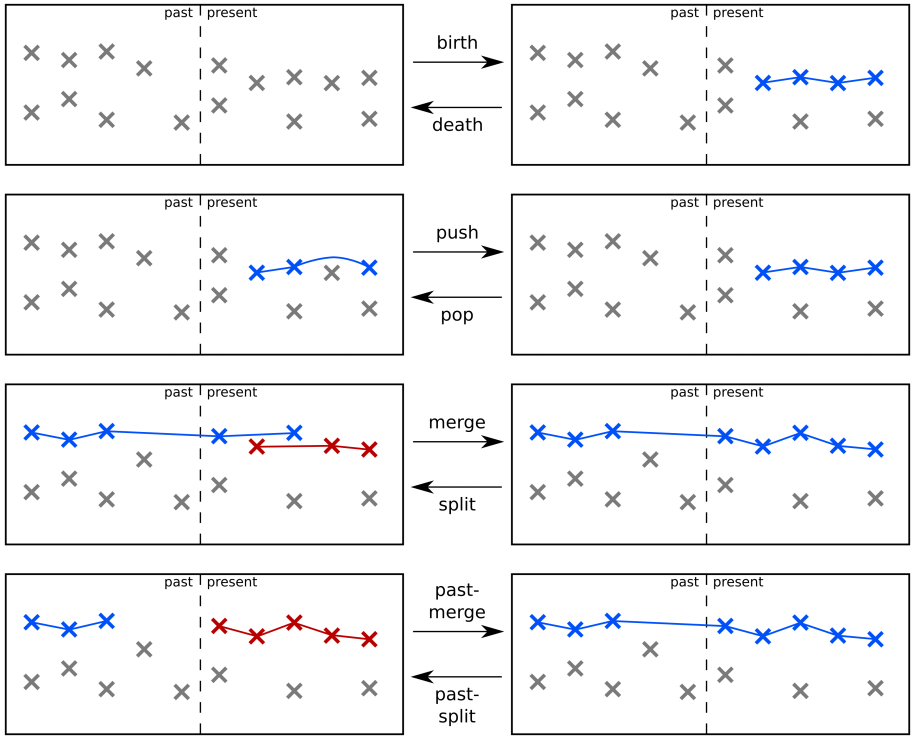


Fig. 19: Reversible move pairs for the MCMC proposal.

For the present timesteps $t - L + 1 : t$, we store all of the observations and robot poses. For the past timesteps $1 : t - L$, we only store a list of landmarks and the parameters (mean and covariance) of their location distribution. Thus,

the space complexity of our algorithm is linear in L and constant with respect to the total number of timesteps t. Because past landmarks and present landmarks are represented differently in our implementation, we found it easier to have Merge, Split, PastMerge, and PastSplit moves, instead of just having Merge and Split moves that handled both past and present landmarks.

Our MCMC proposal $q(s'|s)$ first picks a type of move, and then generates a move of that type. If there are no valid moves of that type, the proposal defaults to a **NoopMove**, which makes no changes to the current association. A NoopMove has the acceptance ratio $\alpha = 1$, so it is always accepted.

5.2 Evaluating the MCMC Target Distribution

To finish the description of our MCMC algorithm, we need to specify how we evaluate the target distribution $\pi(s) = p(x_{1:t}, \omega_{1:t}|y_{1:t})$ up to a normalization constant. Since the MCMC is only allowed to change $\omega_{t-L+1:t}$, the target distribution simplifies as follows:

$$\begin{aligned} p(x_{1:t}, \omega_{1:t}|y_{1:t}) &= p(x_{1:t}, \omega_{1:t-L}|y_{1:t}) \cdot p(\omega_{t-L+1:t}|x_{1:t}, \omega_{1:t-L}, y_{1:t}) \\ &\propto \qquad\qquad p(\omega_{t-L+1:t}|x_{1:t}, \omega_{1:t-L}, y_{1:t}) \end{aligned}$$

since the first term is a constant with respect to $\omega_{t-L+1:t}$. By applying Bayes' rule and then taking advantage of the conditional independences in our model, we can decompose this target distribution into the product of an association prior term and an observation likelihood term:

$$\begin{aligned} &p(\omega_{t-L+1:t}|x_{1:t}, \omega_{1:t-L}, y_{1:t}) \\ &\qquad \propto p(\omega_{t-L+1:t}|x_{1:t}, \omega_{1:t-L}, y_{1:t-L}) \cdot p(y_{t-L+1:t}|x_{1:t}, \omega_{1:t}, y_{1:t-L}) \\ &\qquad \propto p(\omega_{t-L+1:t}|\omega_{t-L}) \cdot p(y_{t-L+1:t}|x_{1:t}, \omega_{1:t}, y_{1:t-L}) \end{aligned}$$

The association prior term decomposes into a product, where each term can be evaluated using equation 7:

$$p(\omega_{t-L+1:t}|\omega_{t-L}) = \prod_{i=t-L+1}^{t} p(\omega_i|\omega_{i-1})$$

The observation likelihood term also decomposes into a product, and by the conditional independences in our model we get:

$$\begin{aligned} p(y_{t-L+1:t}|x_{1:t}, \omega_{1:t}, y_{1:t-L}) &= \prod_{i=t-L+1}^{t} p(y_i|x_{1:t}, \omega_{1:t}, y_{1:i-1}) \\ &= \prod_{i=t-L+1}^{t} p(y_i|x_{1:i}, \omega_{1:i}, y_{1:i-1}) \end{aligned}$$

Each term in the above product can be rewritten by reintroducing the Rao-Blackwellized variable θ, and marginalizing over it:

$$\begin{aligned} p(y_i|x_{1:i}, \omega_{1:i}, y_{1:i-1}) &= \int_\theta p(y_i, \theta|x_{1:i}, \omega_{1:i}, y_{1:i-1}) \, \mathrm{d}\theta \\ &= \int_\theta p(\theta|x_{1:i}, \omega_{1:i}, y_{1:i-1}) \; p(y_i|\theta, x_{1:i}, \omega_{1:i}, y_{1:i-1}) \, \mathrm{d}\theta \\ &= \int_\theta p(\theta|x_{1:i-1}, \omega_{1:i-1}, y_{1:i-1}) \; p(y_i|x_i, \omega_i, \theta) \, \mathrm{d}\theta \end{aligned}$$

where we got the second line using the chain rule of probability, and the third line by applying the conditional independences in our model. In the final integral, the first term $p(\theta|x_{1:i-1}, \omega_{1:i-1}, y_{1:i-1})$ is the Gaussian distribution over landmark locations, after incorporating observations up to timestep $i-1$. The second term $p(y_i|x_i, \omega_i, \theta)$ is simply the observation probability in our model, given by equation 8. Therefore, we can evaluate the integral in closed form, since it is equivalent to computing the observation likelihood in the EKF for each landmark that appears in ω_i.

5.3 Verifying the MCMC Sampler

To verify the implementation of the MCMC sampler, we picked a small example with a stationary robot, 10 timesteps, and 12 observations, similar to Figure 15. Every possible association hypothesis is a partition of the set of observations, so there are at most $B_{12} = 4,213,597$ possible associations, where B_i is the i^{th} Bell number. Some of these associations are invalid in our model, because they put two observations at the same timestep in the same partition, thus violating the mutual exclusion constraint.

We exhaustively listed all valid associations (there were $2,504,665$ of them), and computed the target probability $p(x_{1:t}, \omega_{1:t}|y_{1:t})$ for each association up to a proportionality constant, as described in the previous section. Normalizing the resulting distribution, we obtained that hypothesis A with one track accounted for 0.81329 of the probability mass, and hypothesis B with two tracks accounted for 0.18671 of the probability mass. All other hypotheses had negligible mass (less than 10^{-6}).

We then ran on our MCMC sampler on the same example, starting from an initial state that associated every observation to a false detection. Figure 20 plots the (unnormalized) log probability of the current state of the Markov chain for the first 5000 iterations. In this small example, the chain converges to the true posterior very quickly (in less than 100 iterations), and then efficiently switches between the two modes of the posterior. Running 10 trials with one million iterations, the MCMC visited state A with average frequency 0.81324 and state B with average frequency 0.18676 (the standard deviation was 0.00168). These numbers match the exact posterior very closely, indicating that the MCMC samples from the correct distribution.

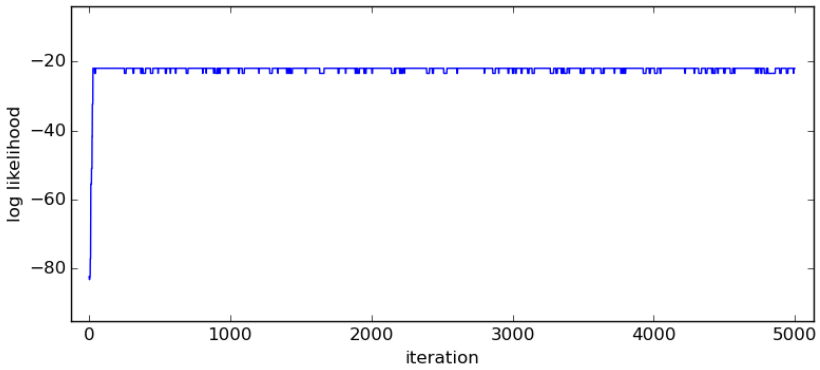


Fig. 20: The mixing behavior of our MCMC sampler on a small example.

5.4 The FastSLAM-DA-RM Algorithm

The FastSLAM-DA-RM algorithm is obtained by taking FastSLAM-DA and adding MCMC moves after the resample step, using the MCMC sampler described in the sections above. In the particle filtering part, we still use the smart proposal from section 4.2. Since a landmark with a single observation is indistinguishable from a false detection, we can reduce the number of parameters by setting $\phi_{\text{new}} = -\infty$. This means that the proposal will associate observations to existing landmarks or to false detections, but it will never create new landmarks. The job of creating new landmarks falls on the MCMC in the move step. Figure 21 shows one timestep of this algorithm.

Both FastSLAM-DA and FastSLAM-DA-RM are true online algorithms, meaning that their time and space complexity is bounded and does not grow with the total number of timesteps t. In FastSLAM-DA, the variables from timestep $t-1$ are "forgotten" as soon as timestep t is processed. After timestep t, each FastSLAM-DA particle contains:

- The robot pose x_t.
- The association ω_t.
- The mean and covariance for the landmark location distributions, incorporating observations up to the current timestep: $p(\theta|x_{1:t}, \omega_{1:t}, y_{1:t})$.

Since FastSLAM-DA-RM allows changing the recent past, the particle definition is more complicated. After timestep t, each FastSLAM-DA-RM particle contains:

- The robot pose for the current window: $x_{t-L+1:t}$.
- The association for the current window: $\omega_{t-L+1:t}$.
- The mean and covariance for the landmark location distributions, incorporating all observations before the current window: $p(\theta|x_{1:t-L}, \omega_{1:t-L}, y_{1:t-L})$.

Inputs:

- A list B_{t-1} of N particles approximately distributed according to $p(x_{1:t-1}, \omega_{1:t-1} \mid y_{1:t-1})$.
- The new observations y_t.
- Parameters for the model: controls, noises, etc.
- Parameters for the proposal: ϕ_{fa}.
- Parameters for the move step: the window length L and the number M of MH iterations in each move step.

Output:

- A list B_t of N particles approximately distributed according to $p(x_{1:t}, \omega_{1:t} \mid y_{1:t})$.

Algorithm:

1. **Advance step**
   Let B_t be an empty list.
   While length$(B_t) < N$:
   For each particle $(x_{1:t-1}^{(i)}, \omega_{1:t-1}^{(i)})$ in B_{t-1}:
   Propose $(x_t^{(i)}, \omega_t^{(i)})$ according to equation 9.
   Compute the weight $w_t^{(i)}$ according to equation 10.
   If $w_t^{(i)} > 0$, add the new particle to B_t.
2. **Resample step**
   Resample N particles from B_t in proportion to their weights.
3. **Move step**
   Pass each particle through M iterations of the MCMC kernel with target distribution $p(x_{1:t}, \omega_{1:t} | y_{1:t})$.
4. **Exact step**
   Update the Rao-Blackwellized distributions on landmark locations in each particle, according to which observations were associated to those landmarks.

Fig. 21: The FastSLAM-DA-RM algorithm for timestep t.

- The observations for the current window: $y_{t-L+1:t}$. These are needed for evaluating the target distribution in the move step.

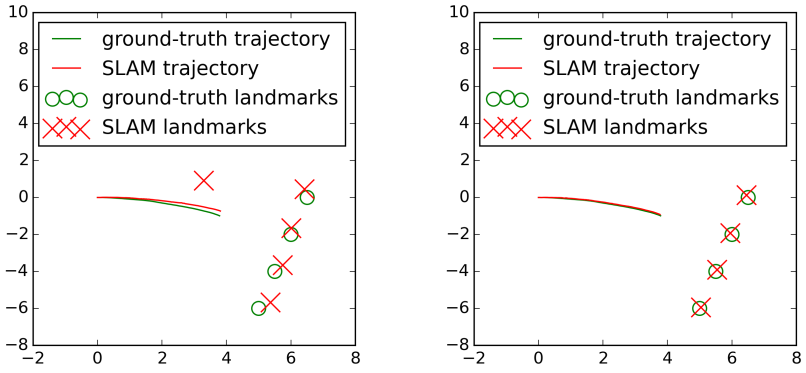
When processing timestep $t+1$, the window is extended to $t-L+1 : t+1$. After the move step, the landmark location distribution is updated from $p(\theta | x_{1:t-L}, \omega_{1:t-L}, y_{1:t-L})$ to $p(\theta | x_{1:t-L+1}, \omega_{1:t-L+1}, y_{1:t-L+1})$, and the variables x_{t-L+1}, ω_{t-L+1}, y_{t-L+1} are "forgotten", since they can never be changed in the

future. This reduces the window to $t - L + 2 : t + 1$, maintaining the window length invariant for the next timestep.

5.5 Results on the `highfa` Dataset

FastSLAM-DA-RM has too many parameters to do a grid search over their entire space. We set $\phi_{\text{new}} = -\infty$ as described in the previous section, and $\phi_{\text{fa}} = -5$, which achieved the best results for FastSLAM-DA. We then fixed the number of particles to 100, and did a preliminary grid search on the remaining two parameters. We got good results with a window of length $L = 30$ and $M = 20$ MH iterations.

Recall from section 4.6 that the main limitation of FastSLAM-DA on the `highfa` dataset is its inability to remove spurious landmarks. Thanks to the move step, FastSLAM-DA-RM is able to remove spurious landmarks, as long as their observations are all in the sliding window. The window length L directly controls how far into the past FastSLAM-DA-RM can look. Figure 23 shows how this parameter influences the accuracy of FastSLAM-DA-RM. As expected, if the window is long enough, FastSLAM-DA-RM easily beats FastSLAM-DA in terms of map accuracy, and the trajectory accuracy improves a bit as well. Figure 22 shows some typical maps produced by FastSLAM-DA-RM. With an insufficiently large window, FastSLAM-DA-RM performs similarly to FastSLAM-DA (Figure 22a). With a large enough window, FastSLAM-DA-RM removes the spurious landmark and produces a near-perfect map and trajectory (Figure 22b).



(a) $L = 20$ (b) $L = 30$

Fig. 22: Typical maps produced by FastSLAM-DA-RM on the `highfa` dataset, for different window lengths L.

The effect of M, the number of MH iterations per move step, was less pronounced. Accuracy suffered with less than 10 iterations, but was essentially stable for any $M \geq 10$. This shows that the move step can achieve its purpose of removing spurious landmarks even with very few MCMC iterations. This makes sense, because the initial state of the MCMC is near the mode of the target distribution, so no burn-in is necessary. In a scenario like Figure 15, all that is needed is for the MCMC to propose and accept a Death move for the false landmark.

One might argue that FastSLAM-DA-RM should get better as we increase M, because the move step would fix other association errors, besides just removing false landmarks. While it is true that the move step can improve the association $\omega_{t-L+1:t}$, it cannot change the trajectory $x_{t-L+1:t}$. Therefore, besides removing false landmarks, the best that the move step can do is to slightly improve the accuracy of the landmark location beliefs. We conclude that on `highfa`, the main benefit of adding MCMC moves is the ability to "look back in time" and remove landmarks that seemed plausible in the past, but are no longer supported by evidence.

Finally we considered the effect of the number of particles N. As expected, map and trajectory accuracy improved with more particles, reaching a plateau for $N \geq 100$. In contrast, FastSLAM-DA reached a plateau for $N \geq 1000$. This shows that fewer particles are required when using resample-move. However, a FastSLAM-DA-RM iteration is more expensive than a FastSLAM-DA iteration, because it performs M MH iterations in an inner loop. FastSLAM-DA is recommended on datasets where "changing the past" is not necessary, since it is faster and much easier to implement than FastSLAM-DA-RM.

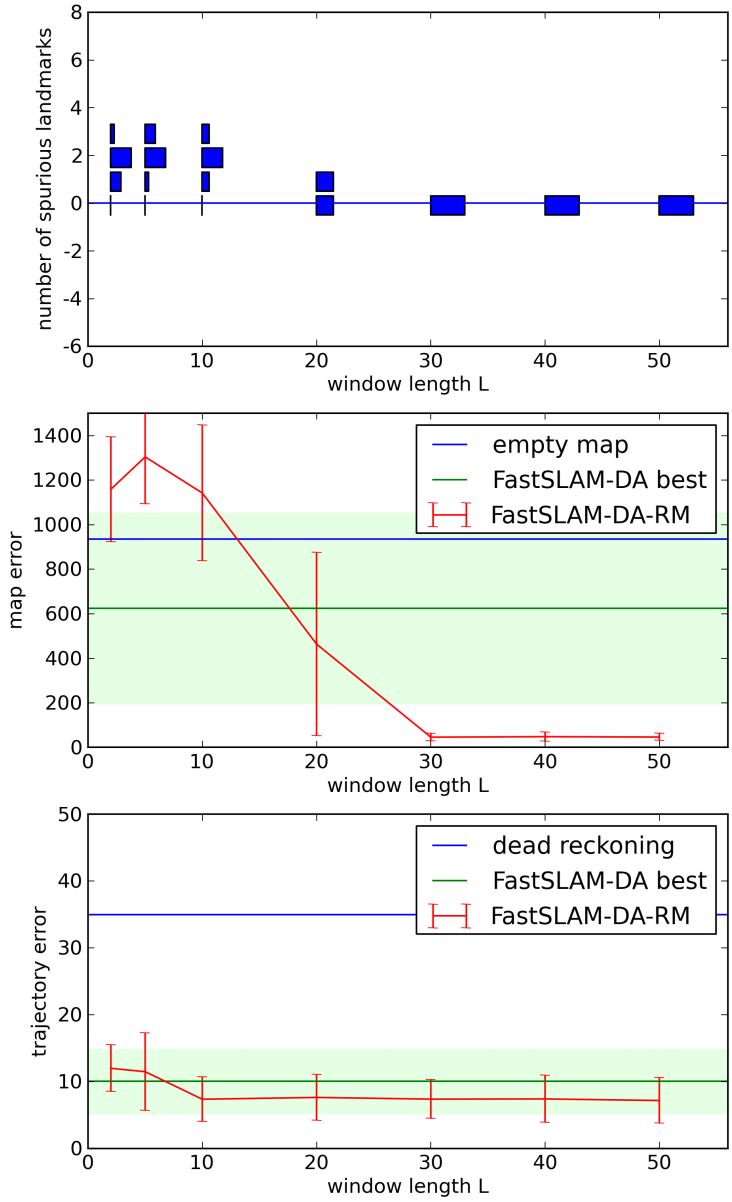


Fig. 23: FastSLAM-DA-RM results on the `highfa` dataset, with 100 particles, $\phi_{\text{new}} = -\infty$, $\phi_{\text{fa}} = -5$, $M = 20$ MH iterations per move step, and different values for the window length L. Top: histogram showing number of spurious (positive) or undetected (negative) landmarks. Center: average and standard deviation of map error. Bottom: average and standard deviation of trajectory error. All statistics are based on 10 trials. We also show the best result from FastSLAM-DA for comparison.

6 Towards an Implementation in BLOG

To implement FastSLAM-DA in BLOG, we would need the PPS to support the following features:

- **Particle filtering with a custom proposal distribution.** The user should be able to plug in a custom proposal distribution for use with the particle filter. The particle filter then needs to compute the particle weights correctly for the custom proposal.
- **Rao-Blackwellization of variables.** This means that each particle stores sufficient statistics for the distribution of the Rao-Blackwellized variables, instead of storing concrete values for them. We need to be able to initialize these sufficient statistics, update them when conditioning on new evidence, and compute the likelihood of the evidence conditioned on the Rao-Blackwellized variables.

To implement FastSLAM-DA-RM in BLOG, we would need an additional feature:

- **Resample-Move particle filtering.** This means adding a move step after the resample step of the particle filter. The move step could use the MCMC sampler built into BLOG, or it could allow the user to specify a custom MCMC sampler to use.

BLOG already has support for custom proposals in the particle filter. The user can implement a custom proposal by creating a subclass of `Sampler` and defining methods to generate a sample from the proposal distribution, and to evaluate the likelihood of a sample according to the proposal distribution. The user can then run the particle filter with the custom sampler by passing the `samplerClass` property at construction time.

Adding support for Rao-Blackwellization in BLOG would be a large effort. We could require the user to write the Rao-Blackwellization code (such as the integrals in section 5.2) themselves, but then there would be little benefit from using BLOG, since the user would still have had to write most of the inference code by hand. The alternative is to have BLOG automatically determine how to initialize and update the sufficient statistics for the Rao-Blackwellized variables, how to sample child variables when one or more parents are Rao-Blackwellized, and how to evaluate the likelihood of child variables when one or more parents are Rao-Blackwellized. We would need to build a library of combinations of distributions where these operations are possible in closed form, e.g., Gaussians for the continuous case and categorical distributions for the discrete case. Because of the amount of effort involved, we decided that adding Rao-Blackwellization to BLOG is out of scope for this thesis.

We thus decided to focus on the third component: adding Resample-Move particle filtering to BLOG. We created a new inference engine `ResampleMovePF`, which behaves just like the `ParticleFilter`, except that it passes the particles through an MCMC transition kernel after the resample step. We use the

Metropolis–Hastings (MH) sampler built into BLOG for the MCMC transition kernel. By default, the MH sampler uses a proposal that picks a variable in the DBN uniformly at random, and then samples a new value for it based on the values of its parents. Thanks to this default proposal, the user can try Resample-Move on their model without having to write any custom inference code. If the parent-sampling proposal is insufficient, the user can specify a custom proposal for the MH sampler, by creating a subclass of the `AbstractProposer` class, and implementing methods to generate the next sample and compute its acceptance ratio.

The move step in `ResampleMovePF` has the ability to change any variable defined in the model. In contrast, the move step in FastSLAM-DA-RM only changes some of the variables (the associations ω) at some timesteps (at most L timesteps into the past). We can achieve the same behavior with `ResampleMovePF` using a custom proposal that only changes a subset of the variables. If the proposal guarantees never to change variables from more than L timesteps ago, we can extend `ResampleMovePF` to "forget" older variables, thus operating in constant space, like FastSLAM-DA-RM.

FastSLAM-DA and FastSLAM-DA-RM require Rao-Blackwellization for the landmark location variables. Parent-sampling these variables is hopeless, since most samples would place the landmarks in locations that do not agree with the observations. For this reason, we cannot demonstrate FastSLAM-DA or FastSLAM-DA-RM in BLOG until we have support for Rao-Blackwellization. However, we can demonstrate the benefit of Resample-Move on a toy SLAM problem that does not require Rao-Blackwellization. In section 6.1 we define the toy problem and express it as a BLOG model. Then in section 6.2 we show results comparing the existing `ParticleFilter` with the new `ResampleMovePF` on this BLOG model.

6.1 `gridworld`: A One-dimensional SLAM Problem

Our grid-world example is based on the one-dimensional SLAM problem described by Murphy and Russell [31]. We have a one-dimensional grid world with n_{cells} cells, each of which is either black (0) or white (1). When the robot is in cell i, it observes the true color of cell i with probability $1 - p_{\text{obs fail}}$, and the opposite color with probability $p_{\text{obs fail}}$. The robot tries to move left or right at each timestep, according to a fixed control policy. If the move is valid (i.e. it does not cause the robot to fall off the edge of the world), it succeeds with probability $1 - p_{\text{move fail}}$ and fails with probability $p_{\text{move fail}}$. If the move is invalid, it fails with probability one. When a move fails, the robot stays in its current cell. When a move succeeds, the robot moves one cell to the right or left, depending on the current controls.

In our particular example, we set $n_{\text{cells}} = 8$ and used the map shown in Figure 24a. We set $p_{\text{move fail}} = 0.05$ and $p_{\text{obs fail}} = 0.2$. We fixed a control policy as shown in Figure 24b, and generated a scenario by sampling from the generative process described above. In this scenario there are four observation failures and one move failure, as shown in Figure 24b.

cell	0	1	2	3	4	5	6	7
true color	0	1	0	1	0	1	0	1

(a) The map.

timestep	controls	true loc	true color	obs color	comments
0	-	0	0	0	initial loc known
1	→	1	1	0	obs failure
2	→	2	0	0	
3	→	2	0	0	move failure
4	→	3	1	1	
5	→	4	0	0	
6	→	5	1	1	
7	→	6	0	0	
8	→	7	1	1	
9	←	6	0	0	
10	←	5	1	0	obs failure
11	←	4	0	1	obs failure
12	←	3	1	1	
13	→	4	0	0	
14	→	5	1	1	
15	→	6	0	0	
16	→	7	1	0	obs failure
17	→	7	1	1	
18	←	6	0	0	
19	←	5	1	1	
20	←	4	0	0	
21	←	3	1	1	
22	←	2	0	0	
23	←	1	1	1	
24	←	0	0	0	
25	←	0	0	0	

(b) The control policy and observations.

Fig. 24: The `gridworld` SLAM problem.

The goal of SLAM is to infer the true map and the location of the robot at each timestep. It is easy to formulate this problem as an inference problem in a BLOG model. First we write the model encoding the generative process for our world, as shown in Figure 25. We pick a map prior that says each cell is a priori equally likely to be black or white, independently of all other cells. The prior on

the controls is a `DontCare`, a special "distribution" indicating that all control values will be observed. We then describe the specifics of our inference problem, as shown in Figure 26. We observe the controls and noisy colors, and query the location at each timestep and the color at every cell.

```
type Control;
distinct Control Left, Right;

random Integer color(Integer c) ~ Bernoulli(0.5);

random Integer noisyColor(Timestep t) ~
    Categorical({
        color(location(t)) -> 1 - p_obs_fail,
        (1 - color(location(t))) -> p_obs_fail
    });

fixed Integer left_of(Integer c) =
    if c == min_cell then min_cell
    else c - 1;

fixed Integer right_of(Integer c) =
    if c == max_cell then max_cell
    else c + 1;

random Integer location(Timestep t) ~
    if (t == first_timestep) then start_cell
    else if control(t) == Left then Categorical({
        left_of(location(t - 1)) -> 1 - p_move_fail,
        location(t - 1) -> p_move_fail
    }) else Categorical({
        right_of(location(t - 1)) -> 1 - p_move_fail,
        location(t - 1) -> p_move_fail
    });

random Control control(Timestep t) ~ DontCare();
```

Fig. 25: The BLOG model for the `gridworld` problem.

```
fixed Timestep first_timestep = @0;
fixed Integer start_cell = 0;
fixed Integer min_cell = 0;
fixed Integer max_cell = 7;
fixed Real p_move_fail = 0.05;
fixed Real p_obs_fail = 0.2;

obs control(@1) = Right;
obs control(@2) = Right;
obs control(@3) = Right;
...
obs control(@25) = Left;

obs noisyColor(@0) = 0;
obs noisyColor(@1) = 0;
obs noisyColor(@2) = 0;
...
obs noisyColor(@25) = 0;

query location(@0);
query location(@1);
query location(@2);
...
query location(@25);

query color(0);
query color(1);
query color(2);
...
query color(7);
```

Fig. 26: The BLOG evidence and queries for the `gridworld` problem.

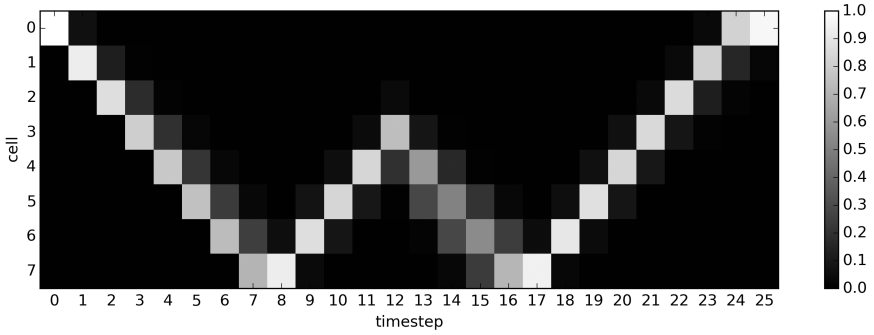
6.2 BLOG Resample-Move Results on `gridworld`

The `gridworld` example is small enough that we can perform filtering by exact inference. There are $2^8 = 256$ possible maps and 8 possible locations for the robot, so there are only 2048 possible worlds. We perform exact inference by keeping track of the probability that we could be in each of those possible worlds in each timestep, which is equivalent to doing a forward pass in a discrete Hidden Markov Model (HMM). We show the exact location posterior at each timestep

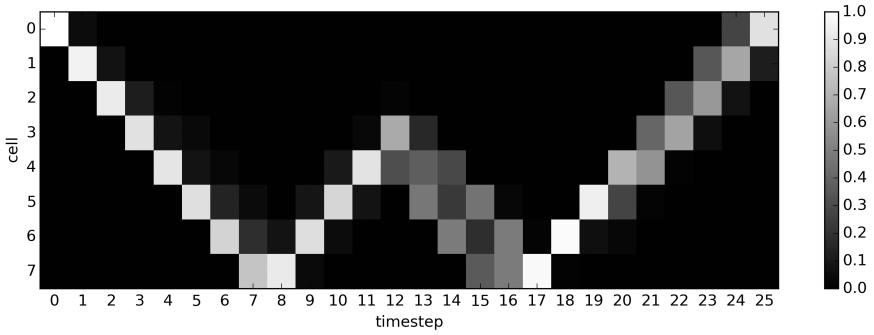
in Figure 27a, and the exact map color marginals at each timestep in Figure 28a.

With 100 particles, the built-in `ParticleFilter` in BLOG fails to track the true map and location posterior, as shown in Figures 27b and 28b. In this problem, the map is a static (atemporal) parameter. Once a particle samples the color for every cell, those colors will be fixed for the entire lifetime of the particle. Since particles get resampled, the total number of map hypotheses considered by the particle filter decreases. This is another form of the degeneracy problem that we encountered in section 5.

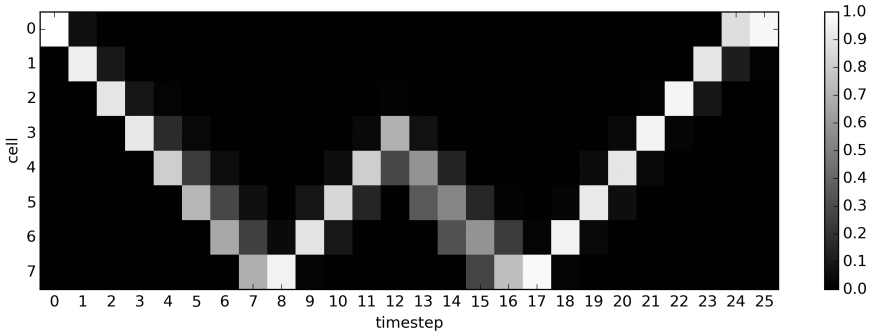
The new `ResampleMovePF` addresses the degeneracy problem by resampling the cell colors in the move step. This is similar to "changing the past" in FastSLAM-DA-RM, except that instead of sampling the association variables $\omega_{t-L+1:t}$, we sample the atemporal variables `color(0)`, ..., `color(7)`. With 100 particles and 10 MH iterations per move step, `ResampleMovePF` in BLOG successfully tracks the exact posterior, as shown in Figures 27c and 28c.



(a) Exact inference.

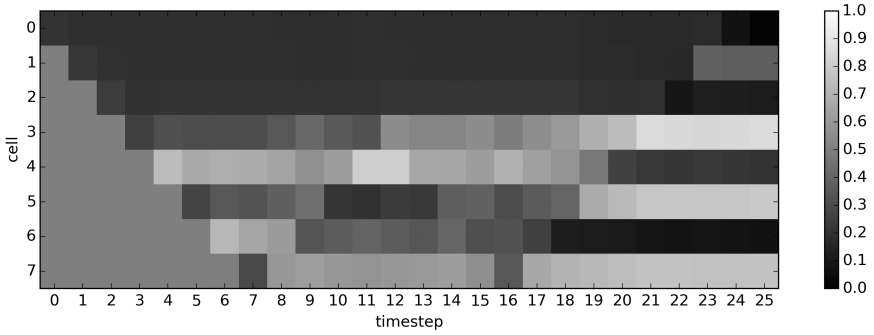


(b) PF with 100 particles.

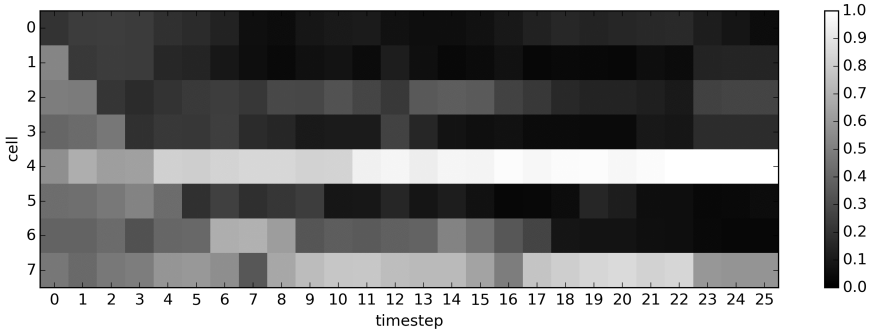


(c) Resample-move PF with 100 particles and 10 MH iterations per move step.

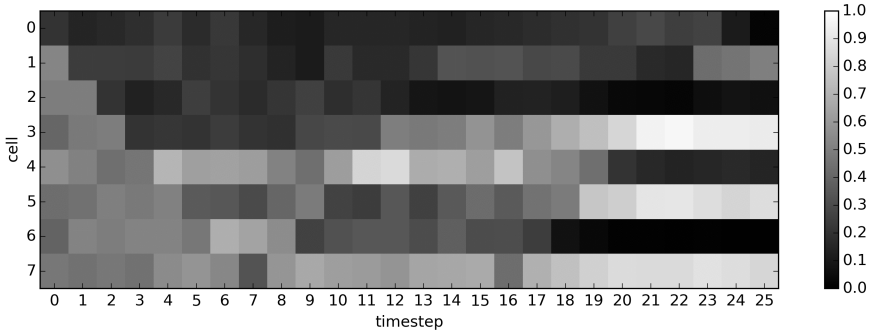
Fig. 27: Robot location belief at each timestep.



(a) Exact inference.



(b) PF with 100 particles.



(c) Resample-move PF with 100 particles and 10 MH iterations per move step.

Fig. 28: Cell color belief marginals at each timestep.

7 Conclusions and Future Work

Our goal was to understand how to solve SLAM using a PPS. We started by surveying existing SLAM approaches, and focusing on FastSLAM as the most promising candidate (section 2). Upon closer examination of FastSLAM, we learned that it uses ad-hoc techniques for data association, which impede a PPS implementation. We also showed that FastSLAM performs poorly on the `mutex` dataset, because of its inability to enforce the mutual exclusion constraint (section 3).

Our first contribution was a new probabilistic model for SLAM with data association uncertainty (section 4.1). This model handles the mutual exclusion constraint, and the uncertainty over the number of landmarks in the environment. We developed FastSLAM-DA, an algorithm for performing inference in this model (the rest of section 4). This new algorithm fits cleanly into the SMC framework, without requiring ad-hoc heuristics like FastSLAM. We showed that FastSLAM-DA beats FastSLAM on the `mutex` dataset. On the other hand, FastSLAM-DA adds false landmarks to the map on the `highfa` dataset, where there is a high rate of false detections. To fix this, we developed FastSLAM-DA-RM, an algorithm based on resample-move particle filtering, which performs MCMC moves on the recent association variables (section 5). This final algorithm beats FastSLAM-DA on the `highfa` dataset. The two algorithms, FastSLAM-DA and FastSLAM-DA-RM, form our second contribution. Our third contribution is a general-purpose resample-move particle filter in BLOG (section 6). This is a first step towards being able to run algorithms like FastSLAM-DA and FastSLAM-DA-RM in BLOG.

One avenue for future work is to extend our model by capturing more relevant aspects of the environment. Currently our model does not account for occlusion or out-of-sight landmarks. This means that we "pay" (in terms of likelihood) for not detecting a landmark, even when it is occluded or behind us. The obvious way to handle occlusion and out-of-sight landmarks is to add edges from θ and x_t to ω_t (see Figure 7). However, this would mean adding an edge from a Rao-Blackwellized variable to a non-Rao-Blackwellized variable, which makes inference intractable [31]. Adding support for occlusion in a principled way while keeping inference tractable is an open question.

Another avenue for future work is to extend FastSLAM-DA-RM so that the move step changes not only the associations $\omega_{t-L+1:t}$, but also the poses $x_{t-L+1:t}$. Currently the move step allows recovering from association errors that have spread through all particles. If we allowed the move step to alter the poses, it would also allow the algorithm to recover from pose errors that have spread through all particles. This could allow the algorithm to run with fewer particles overall. Another intriguing idea is to switch to a Decayed MCMC framework [28], where the MCMC would do all the work, and there would be no need for a particle filter with a smart proposal.

It would also be beneficial to evaluate our algorithms on larger-scale datasets, especially ones collected from real robots. FastSLAM-like algorithms often have difficulties closing loops in large environments, i.e., recognizing that they are in a

place that they have seen a long time ago. This problem has given rise to dozens of ad-hoc solutions in different SLAM domains [20, 22, 34, 42, and the references therein]. It would be interesting to solve the loop closure problem in a principled Bayesian way, without relying on ad-hoc heuristics.

Finally, to truly solve SLAM in BLOG, we would need to add support for Rao-Blackwellized variables in the PPS. It is an open question to define the boundary between what the PPS can do automatically, and what requires input from the user (e.g., in the form of hand-written code to compute integrals). Even in simple cases where all the DBN nodes are linear-Gaussian, extending the BLOG inference engines to support Rao-Blackwellization requires a substantial effort.

References

1. Map Dissimilarity Quantification for the PPAML SLAM Challenge Problem. `http://ppaml.galois.com/wiki/attachment/wiki/CP1QuadRotor/MapDissimilarityQuantification.pdf?format=raw`. Accessed: 2015-07-30.
2. PPAML Challenge Problem: SLAM. `http://ppaml.galois.com/wiki/attachment/wiki/CP1QuadRotor/QuadRotorProblem.pdf?format=raw`. Accessed: 2015-07-15.
3. Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building Rome in a Day. *Communications of the ACM*, 54(10):105–112, 2011.
4. Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An Introduction to MCMC for Machine Learning. *Machine learning*, 50(1-2):5–43, 2003.
5. Tim Bailey. *Mobile robot localisation and mapping in extensive outdoor environments*. PhD thesis, University of Sydney, 2002.
6. Tim Bailey and H. Durrant-Whyte. Simultaneous Localization and Mapping (SLAM): Part II. *Robotics Automation Magazine, IEEE*, 13(3):108–117, Sept 2006.
7. Yaakov Bar-Shalom. *Tracking and data association*. Academic Press Professional, Inc., 1987.
8. Ingemar J Cox. A review of statistical data association techniques for motion correspondence. *International Journal of Computer Vision*, 10(1):53–66, 1993.
9. Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. MonoSLAM: Real-time single camera SLAM. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.
10. Frank Dellaert, Steven M Seitz, Charles E Thorpe, and Sebastian Thrun. EM, MCMC, and chain flipping for structure from motion with unknown correspondence. *Machine learning*, 50(1-2):45–71, 2003.
11. MWMG Dissanayake, Paul Newman, Steven Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (SLAM) problem. *Robotics and Automation, IEEE Transactions on*, 17(3):229–241, 2001.
12. Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential Monte Carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.
13. Arnaud Doucet, Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pages 176–183, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.
14. Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12:656–704, 2009.
15. Daniel Duckworth. Monte Carlo Methods for Multiple Target Tracking and Parameter Estimation. Master's thesis, EECS Department, University of California, Berkeley, May 2012.
16. Bernd Fischer and Johann Schumann. AutoBayes: A system for generating data analysis programs from statistical models. *Journal of Functional Programming*, 13(03):483–508, 2003.
17. Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International Journal of Computer Vision*, 94(3):335–360, 2011.

18. Walter R Gilks and Carlo Berzuini. Following a moving target: Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society. Series B, Statistical Methodology*, pages 127–146, 2001.
19. Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A Language for Generative Models. In *Proc. of Uncertainty in Artificial Intelligence*, 2008.
20. Karl Granström, Jonas Callmer, Fabio Ramos, and Juan Nieto. Learning to detect loop closure from range data. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 15–22. IEEE, 2009.
21. Jose Guivant, Eduardo Nebot, and Stephan Baiker. Autonomous navigation and map building using laser range sensors in outdoor applications. *Journal of Robotic Systems*, 17(10):565–583, 2000.
22. Dirk Hähnel, Sebastian Thrun, Ben Wegbreit, and Wolfram Burgard. Towards lazy data association in SLAM. In *Robotics Research. The Eleventh International Symposium*, pages 421–431. Springer, 2005.
23. Zia Khan, Tucker Balch, and Frank Dellaert. MCMC-based particle filtering for tracking a variable number of interacting targets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(11):1805–1819, 2005.
24. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.
25. Jane Liu and Mike West. Combined parameter and state estimation in simulation-based filtering. In *Sequential Monte Carlo methods in practice*, pages 197–223. Springer, 2001.
26. David Lunn, David Spiegelhalter, Andrew Thomas, and Nicky Best. The BUGS project: Evolution, critique and future directions. *Statistics in medicine*, 28(25):3049, 2009.
27. Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A Higher-order Probabilistic Programming Platform with Programmable Inference. *arXiv preprint arXiv:1404.0099*, 2014.
28. Bhaskara Marthi, Hanna Pasula, Stuart Russell, and Yuval Peres. Decayed MCMC Filtering. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 319–326. Morgan Kaufmann Publishers Inc., 2002.
29. Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic Models with Unknown Objects. In *Proc. 19th International Joint Conference on Artificial Intelligence*, pages 1352–1359, 2005.
30. Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI/IAAI*, pages 593–598, 2002.
31. Kevin Murphy and Stuart Russell. Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks. In *Sequential Monte Carlo methods in practice*, pages 499–515. Springer, 2001.
32. Kevin P Murphy. Bayesian Map Learning in Dynamic Environments. In *Neural Information Processing Systems (NIPS)*, pages 1015–1021, 1999.
33. José Neira and Juan D Tardós. Data association in stochastic mapping using the joint compatibility test. *Robotics and Automation, IEEE Transactions on*, 17(6):890–897, 2001.
34. Paul Newman, David Cole, and Kin Ho. Outdoor SLAM using visual appearance and laser ranging. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1180–1187. IEEE, 2006.

35. Juan Nieto, Jose Guivant, Eduardo Nebot, and Sebastian Thrun. Real time data association for FastSLAM. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 1, pages 412–418. IEEE, 2003.
36. Songhwai Oh, Stuart Russell, and Shankar Sastry. Markov chain Monte Carlo data association for general multiple-target tracking problems. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 1, pages 735–742. IEEE, 2004.
37. Martyn Plummer et al. JAGS: A program for analysis of Bayesian graphical models using Gibbs sampling. In *Proceedings of the 3rd International Workshop on Distributed Statistical Computing*, volume 124, page 125, 2003.
38. Donald B Reid. An algorithm for tracking multiple targets. *Automatic Control, IEEE Transactions on*, 24(6):843–854, 1979.
39. Geraldo Silveira, Ezio Malis, and Patrick Rives. An efficient direct approach to visual SLAM. *Robotics, IEEE Transactions on*, 24(5):969–979, 2008.
40. Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT press, 2005.
41. Sebastian Thrun and John J Leonard. Simultaneous Localization and Mapping. In *Springer Handbook of Robotics*, pages 871–889. Springer, 2008.
42. Brian Williams, Mark Cummins, José Neira, Paul Newman, Ian Reid, and Juan Tardós. A comparison of loop closing techniques in monocular SLAM. *Robotics and Autonomous Systems*, 57(12):1188–1197, 2009.
43. David Wingate, Andreas Stuhlmueller, and Noah D Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *International Conference on Artificial Intelligence and Statistics*, pages 770–778, 2011.
44. Frank Wood, Jan Willem van de Meent, and Vikash Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International conference on Artificial Intelligence and Statistics*, pages 2–46, 2014.