ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

INSTITUT FÜR INFORMATIK

Arbeitsgruppe Autonome Intelligente Systeme

Prof. Dr. Wolfram Burgard

Learning Sample-Based Maps for Mobile Robots

Master Thesis

Daniel Meyer-Delius di Vasto

September 2005 – May 2006

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Außerdem erkläre ich, dass die Masterarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

(Daniel Meyer-Delius di Vasto)
Freiburg, den April 12, 2007

Acknowledgment

First of all, I would like to thank Prof. Wolfram Burgard for giving me the opportunity to work in his research group. I would like to thank him for his guidance, support, and motivation.

Many thanks to the members of the group: Cyrill Stachniss, Christian Plagemann, Rudolph Triebel, Patrick Pfaff, Kai Arras and Giorgio Grisetti. Thanks to Axel Rottmann for his advises, ideas, and proof-reading part of this work. A very special thanks to Óscar Martínez Mozos for his guidance and friendship.

La vida no es la que uno vivió, sino la
que uno recuerda y cómo la recuerda
para contarla.

Gabriel García Márquez

Contents

1. Introduction

Mobile robots differentiate themselves from other types of robots in being able to go from one place to another in order to execute a given task. During the last decade, mobile robots have performed successfully in a wide range of different environments such as indoor, outdoor, underwater, and even on other planets. For most robotic applications a model of the environment is a fundamental part of the system. A representation of how the world looks like is necessary for performing basic tasks such as localization, path planing, and exploration. Without a model of the environment those tasks would be impossible, limiting the practical applications of such a robot.

The way in which the environment is represented has an important impact on the performance of the robot. Accurate maps are fundamental for navigation. One way to describe the environment is to use a detailed geometrical description of all the objects in it. These spatial representations can be very accurate and are well suited for various important tasks like motion control and accurate localization. A fundamental question when representing the environment geometrically is the choice of geometrical primitive to be used. Using lines, for example, imposes a linear structure on the underlying environment. This is well suited for some environments such as an office, but can be inappropriate for others. Points are the most general geometrical primitive. Using points allows different environments to be accurately represented without imposing any geometrical structure on them.

To construct a map, the information about the environment perceived by the robot is used. This information can be, for example, the distance to the objects detected by the robot's sensors while moving through the environment. Using the distance measurements directly in the way they are produced by the sensors is straight-forward and general since it does not rely on the environment having some specific features. By converting these measurements into a set of points in an absolut coordinate system a *sample-based* map is constructed. Such a map constitutes a point-based geometrical representation of the environment where each point or sample corresponds to a measurement made by the robot. Thus, beside their accuracy and generality, sample-based maps are also consistent with the observations.

This thesis investigates the idea of using samples to model the environment and presents different techniques for generating sample-based maps from the distance measurements acquired by the robot. We seek to find an efficient representation to accurately describe the environment. Obviously, if all the measurements acquired by the robot are used, the resulting map would be the best representation of the environment given that data. Distance measurements, however, come in large amounts and may lead to too large models. Additionally, not every sample contributes in the same way to the representation, and we may be interested in representing the environment using fewer samples. Thus, our goal is to find a subset of the complete dataset to efficiently represent the environment.

The contribution of this thesis are the various approaches to generate sample-based geometrical maps from range measurements gathered with a mobile robot as an efficient representation of the environment. Sample-based maps are general in the sense that they are not limited to a specific type of environment and by using points as primitives for the representation do not impose any structure to the environment that is being represented. Additionally, sample-based maps are consistent with the data since they do not contain spurious points. Every point in a sample based map can be explained by an existing measurement.

1.1. Related Work

In the robotic literature many different strategies have been proposed to learn efficient geometric representations from range data. One of the first strategies was described by Crowley [1989] who uses a Kalman filter to fit lines on range measurements obtained from sonar sensors. In the paper by Gonzales et al. [1994], point clusters are computed from each range scan based on the distance between consecutive points. Linear regression is then applied to fit lines to these clusters. Arras and Siegwart [1997] use a hierarchical clustering approach to extract lines from the points obtained from laser data. The strategy proposed by Leonard et al. [2001] uses a Hough transform to extract linear features from a sequence of consecutive sonar measurements. The approach presented by Schröter et al. [2002] clusters scans using the split-and-merge algorithm and combines nearby segments using a weighted variant of linear regression.

In a recent work, Sack and Burgard [2004] present two approaches for extracting lines from laser data. They use the Hough transform and the EM algorithm to extract lines out of the set of points obtained from the range measurements. Both approaches work on the complete dataset in contrast to techniques that work on individual scans. Similar to these approaches, our techniques work also on the complete dataset. However, we do not depend on the extraction of features to construct a representation of the environment.

The EM algorithm has also being used by Liu et al. [2001] to learn planar structures from 3D data, and by Anguelov and colleagues [2004] apply the EM algorithm to cluster different types of objects like walls and doors from sequences of range data. Burgard et al. [1999] use the EM algorithm for learning maps using sonar data, and Bennewitz et al. [2002] to learn motion behaviors of persons. The k-means and fuzzy k-means algorithms we used in our work to improve the quality of a sample-based map are instances of the EM algorithm.

A different approach for improving a given model is to use the simulated annealing algorithm. This algorithm was used by Kirkpatrick et al. [Kirkpatrick *et al.*, 1983] to solve optimization problems associated to the design of integrated circuits like component placement and wiring. They also apply the algorithm to the classic optimization problem of the traveling salesman. In our work we state the problem of generating a sample-based map as an optimization problem and apply the simulated annealing algorithm to find the solution.

Beside lines, other geometrical primitives have being explored for describing the environment. González-Baños and Latombe [2000] use polylines as primitives for the representation. A polyline is a sequence of line segments that are connected at their endpoints. In their work, polylines are extracted from range scans exploiting the order of the individual laser beams. Veeck and Burgard [2004] describe an approach for learning polyline maps that operates on an arbitrary set of points and does not assume any ordering of the points.

In the computer graphics field, Levoy and Whitted [1985] proposed points as a universal meta-primitive for geometric modeling and rendering applications for 3D geometry. Pauly et al. [2002] explore the usage of points as primitives for modeling three-dimensional objects, and presents several techniques for modifying and reducing the size of the original set of points. However, these techniques are approximative in the sense that the resulting set of points are not necessarily a subset of the original dataset. Alexa et al. [2001] works on a set of samples that is a subset of the original dataset.

This thesis investigates the idea of using samples to efficiently represent the environment and presents various approaches to generate sample-based maps from the range measurements gathered with a mobile robot. The rest of this thesis is organized as follows. Chapter 2 introduces sample-based maps. It describes how the range measurements are used to obtain the points used in the representation and presents the evaluation criteria for the models used throughout the rest of the work. Chapter 3 presents several techniques used to reduce the number of points in the original dataset. In Chapter 4 we present the k-means and fuzzy k-means algorithms as maximum-likelihood optimization techniques for improving the quality of a given model. This chapter also describes the simulated annealing algorithm as a stochastic solution for the optimization problem. In Chapter 5 we present several experiments designed to evaluate the different algorithms presented in this work. Finally, in Chapter 6 we present the conclusions of out work.

2. Sample-Based Maps

A sample-based map is a spatial representation of the environment that uses points as model primitives. These points are obtained by projecting the range measurements of the robot's sensors into a global Cartesian coordinate system. The resulting set of points constitutes a two-dimensional spatial model of the environment and represents a two-dimensional floor plan, or a two-dimensional slide of the three-dimensional environment. Using points as the model's primitives complex environments can be accurately represented and no structure is imposed on the underlying environment. Besides its generality and accuracy, sample-based maps are also consistent with the data. There are no spurious points in the representation, since every point can be associated to an originating measurement.

Throughout this work, we assume the range measurements from which the samples are obtained are generated using a laser-range finder. Laser-range finders are very common in robotics and currently state-of-art for distance measurements given their high accuracy. At each time t, a laser-range finder generates a set of M range measurements $z_t = \{z_t^1, \ldots, z_t^M\}$ referred to as *scan*. Each measurement $z_t^i = (d_t^i, \varphi_t^i)$ in a scan, corresponds to a laser beam and consists of a distance measurement d_t^i, and direction φ_t^i relative to the robot's orientation.

A measurement z_t^i can be projected into a point $x = (p_x, p_y)$ in Cartesian coordinates using the following equation

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \begin{pmatrix} r_x^t \\ r_y^t \end{pmatrix} + \begin{pmatrix} d_t^i \cos(r_\theta - \varphi_t^i) \\ d_t^i \sin(r_\theta - \varphi_t^i) \end{pmatrix}, \tag{2.1}$$

where r_x^t and r_y^t correspond to the x-y coordinates of the robot at the moment t of the measurement, and r_θ^t corresponds to its bearing. Figure 2.1 shows a complete scan consisting of 361 measurements covering a 180 degree area in front of the robot. The figure also shows the points generated by projecting the measurements using Eq. (2.1).

According to Eq. (2.1), in order to project a measurement z_t^i, the pose of the robot $(r_x^t, r_y^t, r_\theta^t)$ at the moment of taking the measurement must be known. Unfortunately, this is, in general, not the case. The problem of building a map without knowing the pose of the robot, is known as the *simultaneous localization and mapping* problem (*SLAM*), and is one of the most fundamental problems in robotics. An extensive literature for the field of SLAM can be found in [Thrun, 2002]. We do not address SLAM is our work, and it is assumed that the scans are *aligned*. That is, the pose of the robot at the moment of making a measurement has already being estimated using some scan matching technique.

For the purpose of creating a map, the robot travels through the environment gathering a set of scans $z_1, \ldots, z_K$. Knowing the pose of the robot, all these scans, can be projected using

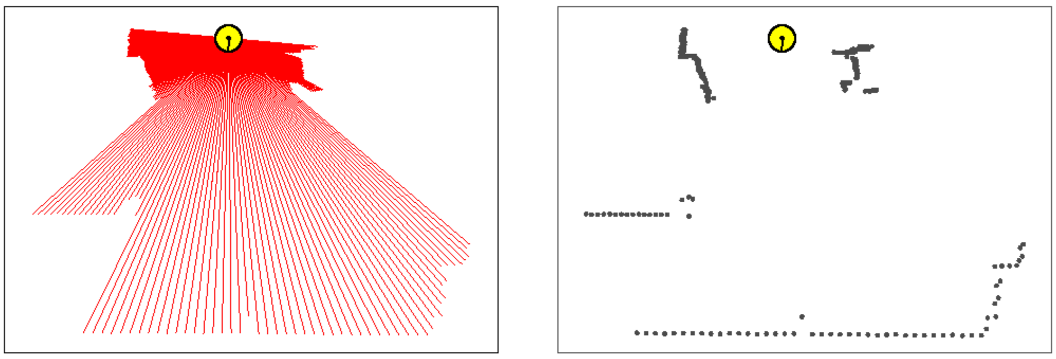


Figure 2.1.: The left image shows a complete scan consisting of 361 range measurements with an angular resolution of 0.5 degrees. The points obtained by projecting the range measurements into Cartesian coordinates are shown on the right image.

Eq. (2.1) producing a set $\mathcal{D} = \{x_1, \ldots, x_n\}$ of points. We refer to these points as *samples*. The measurement of the environment can be viewed as a sampling process, in which discrete points are selected from a continuous space. The set of samples $\mathcal{D}$ creates a two-dimensional spatial representation of the environment as shown in Figure 2.2. The line in the figure indicates the trajectory of the robot, and the points indicate the positions of the robot at the moment of making a measurement.

The way in which the robot gradually moves through the environment, and the frequency at which scans are made, cause consecutive scans to partially overlay. Consecutive scans measure to some extend the same region in the environment. The set of samples $\mathcal{D}$ obtained by projecting all the gathered measurements, can be used as a representation of the environment. However, such a representation would contain, in general, many redundant samples. Additionally, since measurements come in large numbers, such a representation may be too large. Instead, we are interested in an efficient representation that uses a less redundant and reduced set of $\mathcal{D}$. Throughout this thesis, we present several techniques for selecting a good subset of $\mathcal{D}$ to represent the environment according to some evaluation criterion. The next section describes two evaluation functions used to measure the goodness of a given subset.

2.1. Evaluation Criteria

In the next section, we present two evaluation functions that can be used to compare different subsets of samples. We would like to be able to select the best subset for a given dataset. Therefore, we need an evaluation criterion to compare the subsets with. Throughout this work, the *likelihood of the data* and the *sum of squared error* are used as evaluation functions for the subsets. The likelihood of the data is usually preferred given its solid probabilistic framework, whereas the sum of square error has the advantage of being easier to compute.

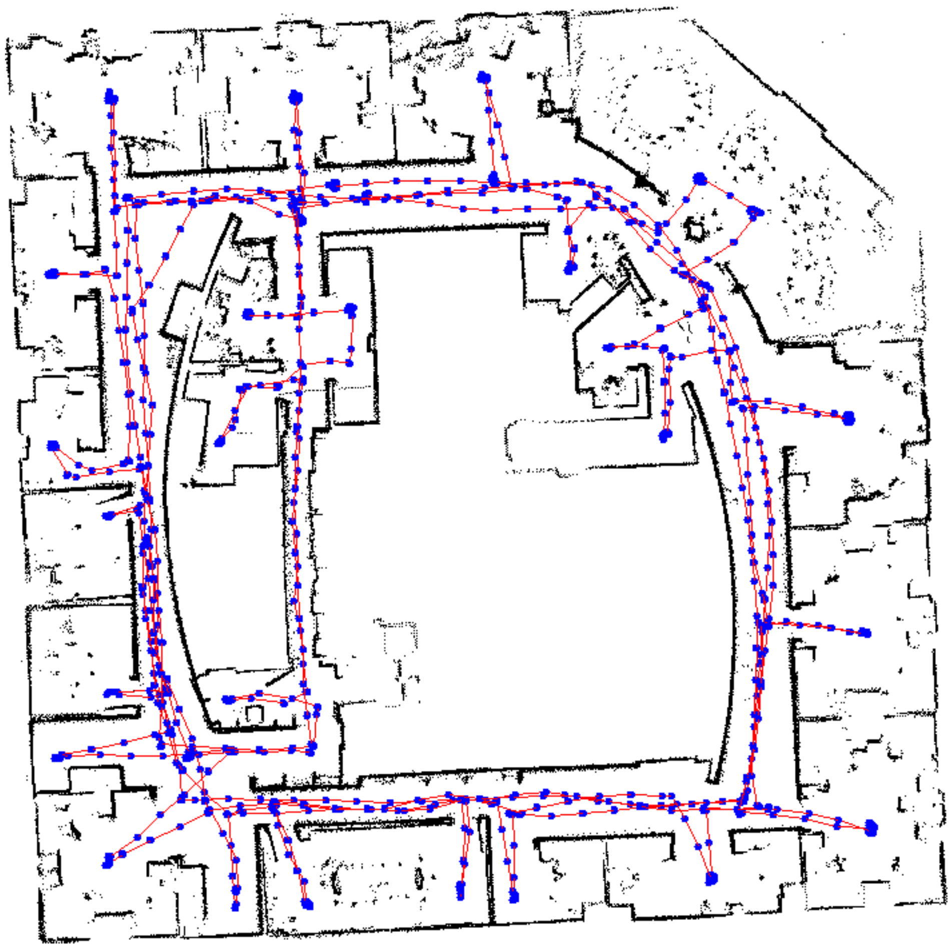


Figure 2.2.: Representation of the Intel Research Lab in Seattle based on the projection of range measurements gathered by a robot while moving through the environment. A total of 910 scans were made, each scan consisting of 180 measurements. Only measurements shorter than 10 meters were taken into account resulting in 155648 samples. The line indicates the trajectory of the robot, and the points indicate the positions where a scan was made.

2.1.1. Likelihood of the Data

A subset $\mathcal{X}$ of samples out of the complete dataset $\mathcal{D}$ can be interpreted as a probabilistic model. The parameters of this model are denoted by $\Theta = \{\theta_1, \ldots, \theta_k\}$ where k is the number of samples in $\mathcal{X}$, and θ_j represents the parameters associated to component ω_j for $j = 1, \ldots, k$. Each sample $x_i \in \mathcal{D}$ is assumed to be generated by first selecting a model component ω_j with probability $P(\omega_j)$, and then generating x_i according to a certain probability distribution $p(x_i|\omega_j, \theta_j)$. Throughout this work, the following assumptions about the probabilistic structure of the model are made:

1. The number k of components ω_j in the model Θ is known.
2. Each model component ω_j has the same prior probability $P(\omega_j) = 1/k,\ j = 1, \ldots, k$.
3. The probability $p(x_i|\omega_j, \theta_j)$ for sample x_i of being generated by component ω_j is normally distributed, $p(x_i|\omega_j, \theta_j) \sim N(\mu_j, \Sigma_j)$.

The complete parameter vector Θ can be written as $\Theta = \{(\mu_1, \Sigma_1), \ldots, (\mu_k, \Sigma_k)\}$ according to these assumptions. Each μ_j in Θ corresponds to a sample x_j in the subset $\mathcal{X}$. Additionally. if we assume that each normal distribution in Θ has the same symmetric covariance matrix Σ, and the standard deviation σ is known. That is, $\Sigma_j = \sigma^2 I$, for all $j = 1, \ldots, k$. Then, the parameter vector can simply be written as $\Theta = \{\mu_1, \ldots, \mu_k\}$. The probability density function for a sample x_i of being generated by component ω_j is then given by

$$p(x_i|\omega_j, \theta_j) \quad = \quad \frac{1}{\sqrt{2\pi}\sigma} \exp\Big[- \frac{(x_i - \mu_j)^2}{2\sigma^2} \Big], \tag{2.2}$$

and the probability density function for a sample x_i is given by the following mixture density:

$$p(x_i|\Theta) \quad = \quad \sum_{j=1}^{k} p(x_i|\mu_j, \theta_j) P(\omega_j). \tag{2.3}$$

Since the prior probability $P(\omega_j)$ is the same for all k components in the model, replacing $p(x_i|\omega_j, \theta_j)$ in Eq. (2.3) with Eq. (2.2), we obtain

$$p(x_i|\Theta) \quad = \quad \frac{1}{k\sqrt{2\pi}\sigma} \sum_{j=1}^{k} \exp\Big[- \frac{(x_i - \mu_j)^2}{2\sigma^2} \Big]. \tag{2.4}$$

Treating a sample-based map as a probabilistic model, enables us to cope with measurement and pose estimation errors in an elegant way. The points in the dataset that belong to the model are considered the "real" points. The remaining points are the result of Gaussian errors in the measurements and pose estimation centered around these "real" points. The standard deviation σ of this error, is related to the accuracy of the range sensor and pose estimation. Small values for σ indicate a small measurement error and a high accuracy in pose estimation.

Large values for σ, on the other hand, should be use for large measurement errors and pour pose estimations.

As a result of the Gaussian interpretation of the model, every sample x_i in the complete dataset $\mathcal{D}$ is assumed to be drawn independently according to $p(x_i|\omega_j, \theta_j) \sim N(\mu_j, \Sigma_j)$. If $\mathcal{D}$ contains n samples, then the likelihood of the model Θ with respect to the samples in $\mathcal{D}$ is given by

$$L(\mathcal{D}|\Theta) \;=\; \prod_{i=1}^{n} p(x_i|\Theta). \tag{2.5}$$

An usual practice when dealing with likelihoods, is to use the logarithm instead. The logarithm is a strictly monotonic function, so, the greater the logarithm of the likelihood, the greater the likelihood itself. The logarithm of the likelihood can be computed as

$$\begin{aligned}\ln L(\mathcal{D}|\Theta) &= \ln \prod_{i=1}^{n} p(x_i|\Theta) \\ &= \sum_{i=1}^{n} \ln p(x_i|\Theta).\end{aligned} \tag{2.6}$$

For convenience throughout this work we will refer to the logarithm of the likelihood of the data $\mathcal{D}$ given a model Θ as log-likelihood, or simply as likelihood, and we will denote it as $LL(\mathcal{D}|\Theta)$. Substituting $p(x_i|\Theta)$ in Eq. (2.6) with Eq. (2.4) we obtain

$$LL(\mathcal{D}|\Theta) \;=\; \sum_{i=1}^{n} \ln \left[\frac{1}{k\sqrt{2\pi}\sigma} \sum_{j=1}^{k} \exp\left[-\frac{(x_i - \mu_j)^2}{2\sigma^2} \right] \right]. \tag{2.7}$$

The likelihood is a measure of the goodness of the model. It indicates, for a fixed dataset $\mathcal{D}$, that the model Θ for which $L(\mathcal{D}|\Theta)$ is large is more likely to be the true model. It is important to consider the value of σ at the moment of interpreting the likelihood values. If the value of σ is too large, every model will have high likelihood, and the difference between the likelihood of different models will be small. The opposite is also true. If the value of σ is almost zero, every model will have an almost identical low likelihood. Figure 2.3 plots the likelihood of too different models for different values of σ. When the values of σ are nearly zero or too large, the likelihood of the models become almost identical.

2.1.2. Sum of Squared Error

Another way to evaluate the quality of a given subset $\mathcal{X}$ for a given set of samples $\mathcal{D}$ is the sum of squared error $E(\mathcal{D}|\mathcal{X})$ defined as follows

$$E(\mathcal{D}|\mathcal{X}) \;=\; \sum_{i=1}^{n} (x_i - x_i^*)^2, \tag{2.8}$$

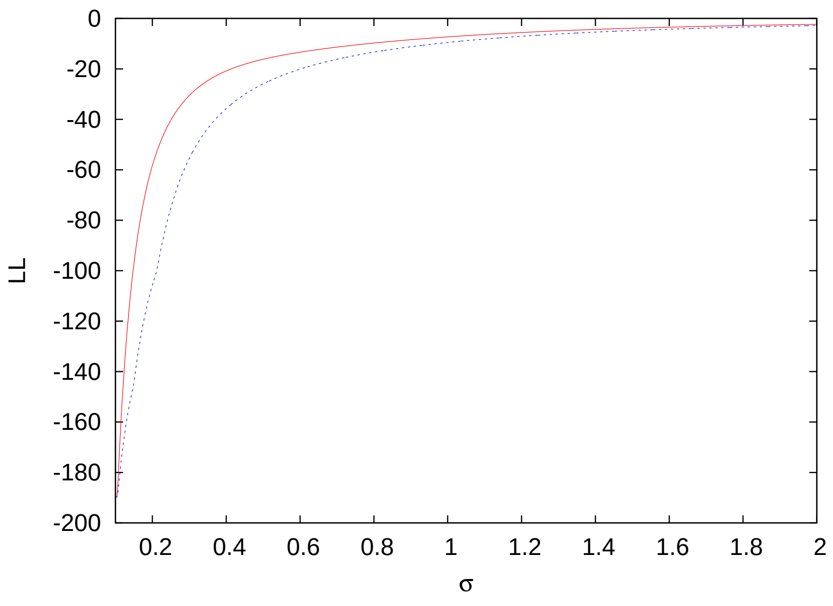


Figure 2.3.: Plot of the likelihood value of two different models for different values of σ. Observe how when the value of σ is too large or almost zero, the likelihood of the two models are almost identical.

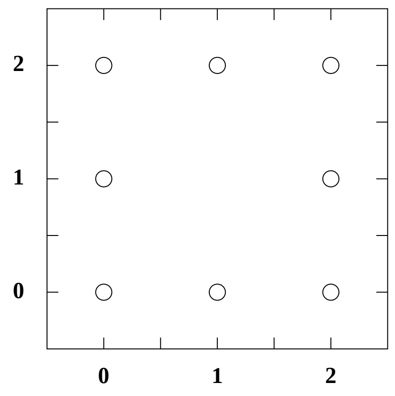
where n is the number of samples in $\mathcal{D}$, $(x_i - x_i^*)$ is the Euclidean distance between x_i and x_i^* and

$$x_i^* = \underset{x_j}{\operatorname{argmin}} \left(x_i - x_j\right)^2 \qquad \forall x_j \in \mathcal{X}. \tag{2.9}$$

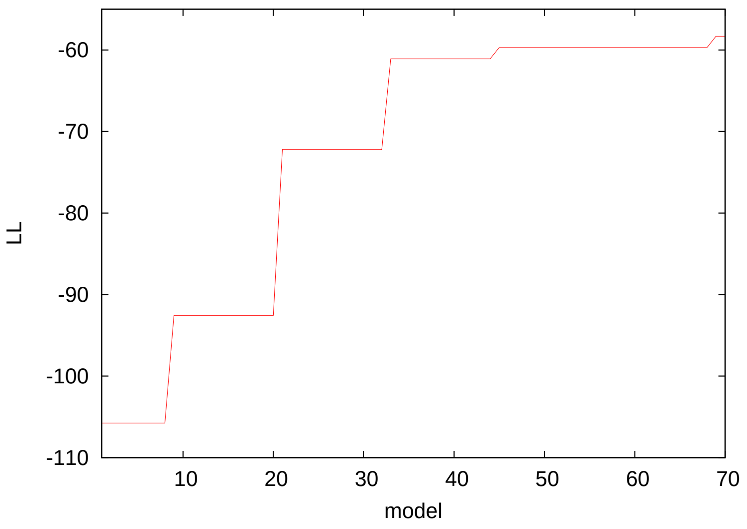
This function measures the total squared error incurred in representing the n samples in $\mathcal{D}$ with the k samples in $\mathcal{X}$.

Both the model likelihood and the sum of squared error are strongly related. From Eq. (2.7) it can be seen that the likelihood of a sample x_i is large when the squared error $\left(x_i - \mu_j\right)^2$ is small. The key difference between the two evaluation functions lies in the fact that for each sample x_i in $\mathcal{D}$ the sum of squared error considers only the closest sample x_i^* in $\mathcal{X}$ to calculate the error, whereas the likelihood of the data takes all the samples in $\mathcal{X}$ into account. That is, the likelihood of the data evaluates the subset $\mathcal{X}$ as a whole, and therefore provides more information about the quality of the representation.

To illustrate the two evaluation functions, we will used the toy dataset presented in Figure 2.4(a). The dataset consist of 8 distinct samples represented with empty circles. There are $\binom{8}{4} = 70$ possible ways of selecting models with 4 points out of a dataset with 8 points. For each of these models, we computed the likelihood according to Eq. (2.7). Figure 2.4(b) plots these likelihoods sorted in ascending order. Figure 2.5(a) shows the maximum-likelihood model. The filled circles represent the points in the model. Figure 2.5(a) shows one of the minimum-likelihood models. Figure 2.6(a) plots the sum of squared errors for all the possible models. As expected, models with high likelihoods have a corresponding low sum of squared errors, and models with low likelihood values have a corresponding high sum of squared errors. This can be clearly seen in Figure 2.6(b) where both likelihood LL and sum of squared error E spaces are aligned for comparison. It can also be observe that small differences in LL are not reflected in the values of E.



(a)



(b)

Figure 2.4.: Figure 2.4(b) shows the sorted likelihood of all the 70 models of size 4 for the dataset shown in Figure 2.4(a).

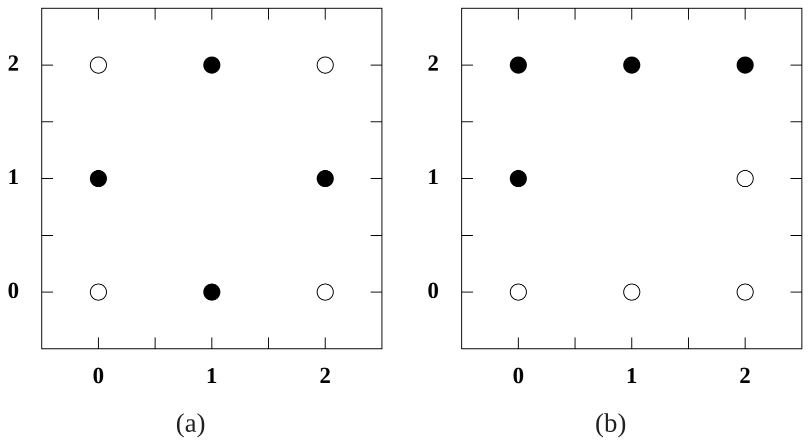


Figure 2.5.: Figure 2.5(a) shows the maximum-likelihood model for the dataset shown in Figure 2.4(a). One of the minimum-likelihood models is shown in Figure 2.5(b).

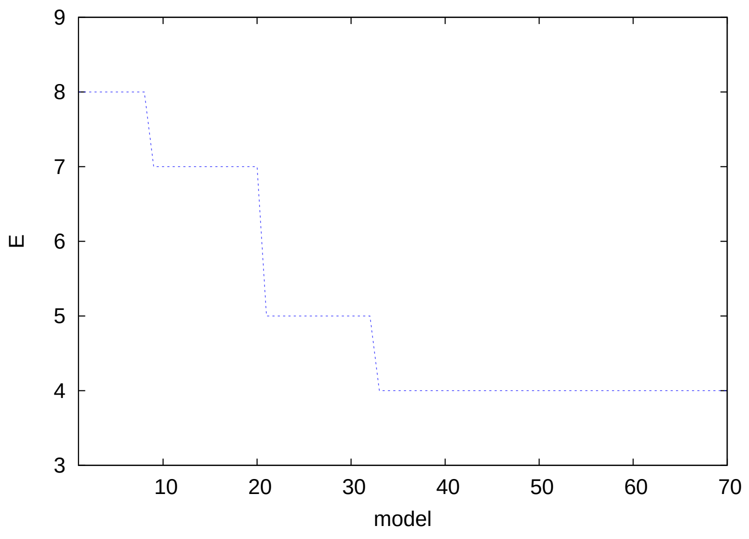
2.2. Estimating the Size of the Model

Another key issue when selecting a model for a given dataset, is the number of model components. Both LL and E are directly affected by the size of the model. In general, the likelihood of the model increases as new components are added to the model, while E decreases as the size of the model increases. Clearly, a model which has a component for each data point, would be an optimal fit for the dataset. We are interested in finding a balance between the number of components and the quality of the model.

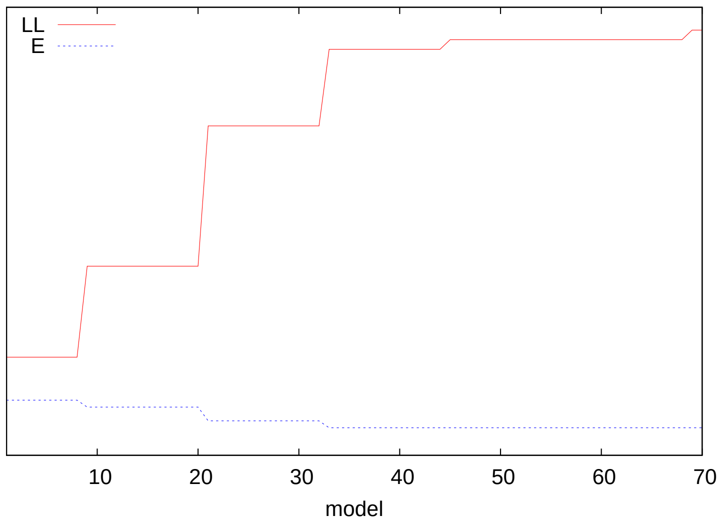
A commonly used measure that balances the accuracy of the model with its complexity is the *Bayesian Information Criterion* [Schwarz, 1978]

$$BIC_E(\mathcal{D}|\Theta) \quad = \quad \alpha E(\mathcal{D}|\Theta) + k \log n,$$

where k is the number of components in the model Θ, and n is the total number of points in the dataset. The constant α is a scaling factor for the error measure E. The BIC_E represents a compromise between the approximation error, the size of the model, and the size of the dataset. When using the BIC_E criterion to determine the size of a model, we choose the size k^* for which the BIC_E is minimal. According to the BIC_E, adding a point to a model with a minimal BIC_E value would increase the complexity of the model without increasing its accuracy. Removing a point from such a model would decrease its accuracy. Figure 2.7 plots the BIC_E value for a dataset with 10687 points using three different values for α. Points where added to the models incrementally, by selecting the point with the greatest distance to all other points already in the model. Observe how for small values for α, the resulting models are smaller, since more weight is given to the complexity of the model. Large values of α give more weight to the error measure and less to the complexity of the model, generating then, larger models.



(a)



(b)

Figure 2.6.: Figure 2.6(a) shows the sorted sum of squared error of all the 70 models of size 4 for the dataset shown in Figure 2.4(a). Figure 2.6(b) compares the likelihood and the sum of squared error for these models.

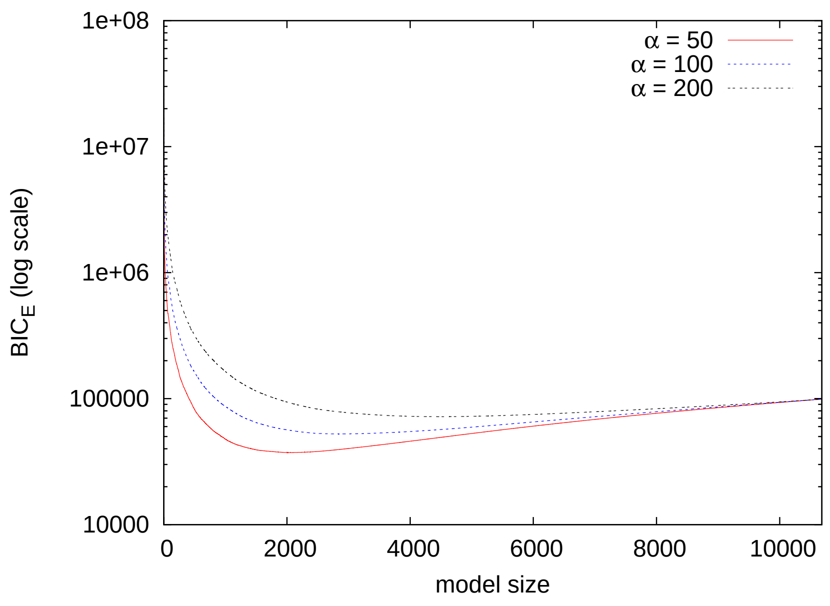


Figure 2.7.: BIC_E values using three different values for α as a function of the size of the model for a dataset of 10687 points. For $\alpha = 50$, 100, and 200, the optimal model sizes are 2048, 2861, and 4550 respectively.

3. Dataset Simplification

In the previous chapter we introduced sample-based maps as a representation of the environment that uses samples as model primitives. These samples were obtained by projecting the end point of the range measurements made by the robot into a two-dimensional global Cartesian system. We argument that given the redundancy involved in the data acquisition process, using all the projected samples to represent the environment would produce an unnecessarily large model. In this chapter we present various techniques for selecting a subset of samples from the complete dataset. These techniques are quite simple, and are aimed at reducing the number of points in the dataset, rather than at finding a good subset according to some evaluation criteria. In our work, the techniques presented in this chapter are usually used to generate an initial model that will be improved later, using the iterative improvement algorithms discussed in the next chapter. Still, the resulting set of samples constitutes a representation of the environment and can be perfectly used as a map. Dataset simplification or *downsampling* techniques are also important since, given the high dimensionality of the problem of learning a map from the data, reducing the size of the data is often of utmost importance in order to solve this problem in a reasonable time.

All the algorithms presented throughout this chapter create a set $\mathcal{X}$ of samples that is a subset of the complete dataset $\mathcal{D}$. In Section 3.1 we present two algorithms based on a grid representation of the environment. These algorithms divide the space into cells of equal size, and then replace all the samples that fall within each cell with a common representative. The representatives from all the non-empty cells constitute the resulting set of samples. This basic approach is extended in Section 3.1.1 to associate an occupancy probability to each cell. This probability value indicates how likely it is for the cell to be occupied and can be used to filter out samples caused by spurious measurements. In Section 3.2 we present a clustering algorithm that groups together samples that lie close to each other. The resulting set of samples is created by selecting a representative sample from each cluster. Finally, in Section 3.3 we present an incremental approach, in which samples are added to the resulting set by selecting the point that has the greatest distance to all the points already in the set.

3.1. Grid-based Sampling

Grid-based sampling techniques divide the space into cells of equal size and then replace all the samples that fall within the same cell with a common representative. The resolution of the grid, that is the size of the cells, affects the number of points that will contain the resulting set. A fine-grained grid will produce a larger set, than a coarse-grained grid. However, the exact

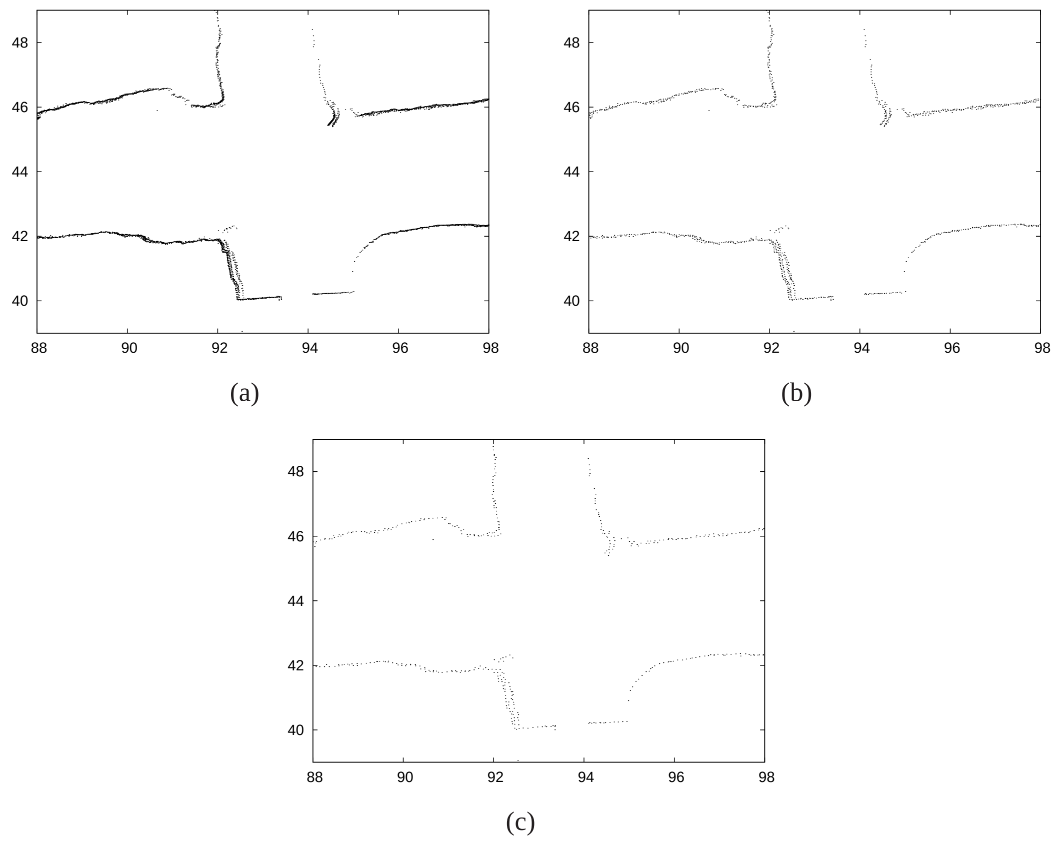


Figure 3.1.: Grid-based sampling. Figure 3.1(a) shows a fraction of the original map of the Bruceton Mine containing 129936 samples. The map in Figure 3.1(b) contains 35702 samples and was generated using a grid resolution of 5 centimeters. The map in Figure 3.1(c) contains 15688 samples and was generated using a grid resolution of 10 centimeters.

number of points the will contain the resulting set can not be directly specified. The size of the cells, can be chosen as to represent the error associated with the measurements. The greater the error, the bigger the cells can be without loosing too much information. Figure 3.1 shows a part of a map generated using two different grid resolutions. The complete dataset contains 129936 samples and is shown in Figure 3.1(a). The map shown in Figure 3.1(b) was generated using a grid resolution of 5 centimeters and contains 35702 samples. The map in Figure 3.1(c) was generated using a grid resolution of 10 centimeters and contains 15688 samples. For this specific environment using a 5 and 10 centimeter grid resolution the dataset was reduced by 87.93% and 72.52% respectively. How much the dataset is reduced depends not only on the resolution of the grid but also in the sampling density. For a fixed grid resolution, dataset with higher sample densities will suffer a larger reduction than datasets with lower densities.

All the points that fall within the same cell are replaced by a common representative. The representative sample of each non-empty cell is selected by first calculating the center of mass of all the samples in the cell. Let $\mathcal{C}_{i,j} = \{x_1, \ldots, x_m\}$ be the set of all samples that fall within the cell indexed by $\langle i, j \rangle$. The center of mass $\bar{c}_{i,j}$ of the cell $\langle i, j \rangle$ is computed as

$$\bar{c}_{i,j} = \frac{1}{|\mathcal{C}_{i,j}|} \sum_{x_i \in \mathcal{C}_{i,j}} x_i. \tag{3.1}$$

Since $\bar{c}_{i,j}$ does not necessarily belong to the dataset, the point $c' \in \mathcal{C}_{i,j}$ that is closest to $\bar{c}_{i,j}$ is used instead. If the center of mass were to be used as cell representative, the resulting representation would contain spurious samples. These samples would be inconsistent with the data since they would not correspond to any measurement. The general grid-based sample approach is described in Algorithm 3.1. The input of this algorithm is the set of all samples $\mathcal{D}$, and the resolution r of the grid. In line 1, a grid for the dataset is generated. Lines 2 through 6 select samples from the grid, by first computing the center of mass $\bar{c}$ of each non-empty cell and then selecting the point in $\mathcal{C}_{i,j}$ closest to $\bar{c}$. Despite its simplicity, it is difficult to describe the time complexity of the grid-based sampling algorithm. The number of operations needed depends on four factors: the number n of samples in $\mathcal{D}$, the resolution of the grid r, the number of samples that fall within each cell, and the size of the underlying environment. Still, the results of our experiments have shown that the determinant factor in the time complexity of the algorithm is the size of the dataset n.

Algorithm 3.1 Grid-based Sampling

```
Input:  set of samples $\mathcal{D} = \{x_1, \ldots, x_n\}$,
        grid resolution $r$.
1: generate grid for $\mathcal{D}$ with resolution $r$
2: for each cell $\langle i, j \rangle$ do
3:     if cell $\langle i, j \rangle$ is not empty then
4:         $\bar{c} = \frac{1}{|\mathcal{C}_{i,j}|} \sum_{x_i \in \mathcal{C}_{i,j}} x_i$
5:         $c'$ = closest point for $\bar{c}$ in $\mathcal{C}_{i,j}$
6:         add $c'$ to $\mathcal{X}$
7:     end if
8: end for
9: return $\mathcal{X}$
```

3.1.1. Occupancy Grid Sampling

Until now we have implicitly assumed that at the moment of gathering the measurements the robot is the only moving object in the environment. This strong assumption can be justified, for example, if the robot is allowed to measure the environment under controlled conditions. This is, however, not always possible and we must take into account the dynamic nature of the environment at the moment of building a map of it. Failing to do this, will produce a representation that, though consistent with the data, will not be consistent with the environment. For example, if the measurements corresponding to a person walking by the robot are included in the representation, this person will be considered as an structural part of the environment in the final map (see Figure 3.3(a)).

Dynamic objects are not the only source of spurious measurements. Other erroneous measurements can be produced by noise in the sensors. To alleviate these problems, we present an approach called *occupancy grid sampling*, which associates to each cell a value representing the probability of being occupied. These approach is based on occupancy grid maps introduced by Elfes [Elfes, 1989]. Discarding the cells with low probability of being occupied effectively eliminates samples caused by spurious measurements.

The probability value associated to each cell lies between 0 and 1, where 0 indicates that the cell is definitely free, and 1 indicates that the cell is definitely occupied. The approach used in this work to compute the occupancy probability for the cells, is known as the *counting model* [Hähnel *et al.*, 2003]. This model, associates to each cell (i, j) the probability $p_{i,j}$ for a range measurement of being reflected by the cell. The value $p_{i,j}$ is computed as

$$p_{i,j} \quad = \quad \frac{hits_{i,j}}{hits_{i,j} + misses_{i,j}}, \tag{3.2}$$

where $hits_{i,j}$ is the number of times a range measurement was reflected, or ended up in cell (i, j), and $misses_{i,j}$ is the number of times a range measurement passed through (i, j) without being reflected. To obtain the $hits_{i,j}$ and $misses_{i,j}$ values for each cell, a ray-casting operation is used. For each range measurement z_t^i, all the cells traversed by the beam are evaluated. Figure 3.2 illustrates the ray-casting operation. The gray cells are traversed by the laser beam and must be updated accordingly. The cell that contains the end point of the beam gets its $hits$ value is incremented therefore incrementing its occupancy probability. The remaining traversed cells get their $misses$ value is incremented and their occupancy probability reduced. Figure 3.2 also illustrates the importance of the grid resolution at the moment of computing the probability value associated to each cell. Large cells make the beams "thick" whereas small cells make the beams "thin". Figure 3.2(a) and Figure 3.2(b) represent the same situation using two different grid resolutions. The filled circles represent structural parts of the environment. The beam in Figure 3.2(a) is too thick and will decrease the occupancy probability of cells that represent structural parts of the environment. If the cells are too small, beams will be too "thin" and most of the cells will be visited only once. This reduces the corrective effect of integrating multiple observations at the moment of computing the occupancy values.

As in the grid-based sampling approach described above, all the samples that fall within the same cell are replaced by a common representative. In contrast to grid-based sampling, occupancy grid sampling only takes into account the cells whose occupancy values lie above a specified threshold. This threshold ψ is a value between 0 and 1, and indicates the smallest occupancy value a cell must have in order to be considered at the moment of sampling. All the samples in each non-empty cell whose occupancy values is greater than the specified threshold, are replaced by a common representative. This representative is selected in the same way as in

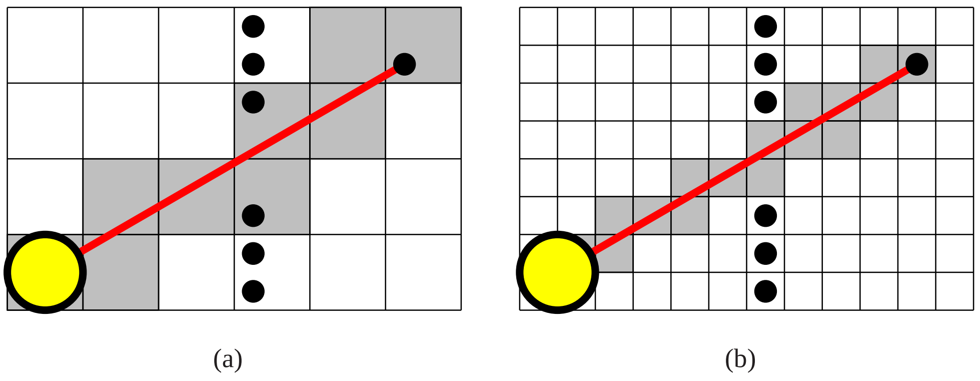


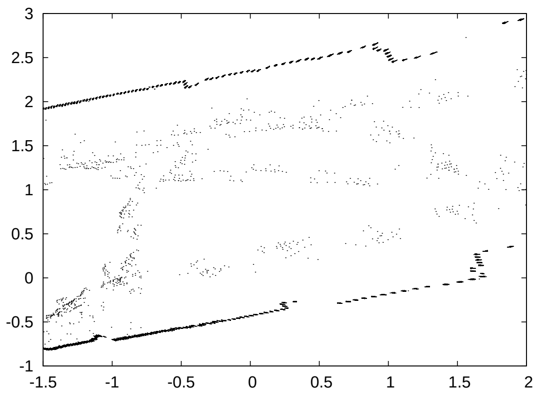
Figure 3.2.: Illustration of the ray-casting operation for two different grid resolutions. Observe how a coarse resolution (a) makes the beam "thick" and a fine resolution makes the beam "thin"

the grid-based sampling algorithm described above. First the center of mass $\bar{c}_{i,j}$ for the cell is computed and then its closest point c' in the cell is selected. Figure 3.3(b) shows the result of applying occupancy grid sampling on the dataset shown in Figure 3.3(a). The samples were gathered in the corridor of the building 79 at the University of Freiburg. The spurious samples were caused by a person moving around the robot during the data acquisition process.

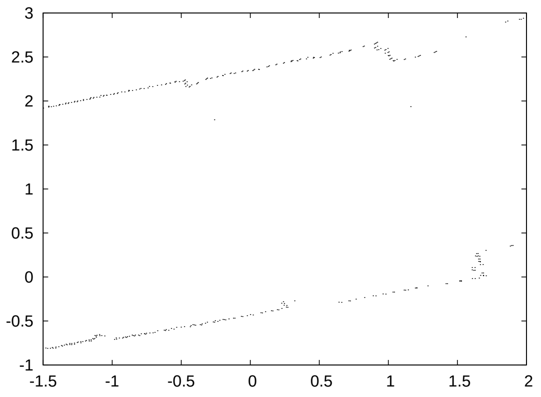
The general occupancy grid sampling approach is described in Algorithm 3.2. This algorithm requires as input the set of aligned scans $z_1, \ldots, z_K$, as well as the resolution r of the grid and the minimum occupancy threshold ψ. The measurements are needed in order to calculate the occupancy probability for the cells. Lines 2 through 8 of the algorithm generate the projected points, add them to the grid, and update the occupancy value of the cells. Lines 9 through 15 select the samples from the grid, taking only into account the cells whose occupancy value lies above ψ. The determinant factor in the time complexity of the occupancy grid sampling algorithm is the ray-casting operation. The number of times the ray-casting operation must be carried out is given by $S * M$, where S is the number of scans gathered during data acquisition, and M is the number of beams per scan. The number of cells that must be updated for each beam depends on the underlying environment and the resolution of the grid.

3.2. Incremental Clustering

A different approach for reducing the number of samples in a dataset is to used a clustering technique. These techniques, divide the dataset into groups or clusters of similar samples. By selecting a representative sample from each cluster, the original dataset can then be represented in a simplified way. For our specific case, the samples in the dataset are grouped using the distance to the other samples as similarity measure. Samples that lie close to each other will be grouped together into a single cluster.



(a)



(b)

Figure 3.3.: The map shown in Figure 3.3(a) presents spurious samples caused by a person walking around the robot during data acquisition. The map shown in Figure 3.3(b) was generated using occupancy grid sampling with a grid resolution of 1 cm and an occupancy threshold of 0.5.

Algorithm 3.2 Occupancy Grid Sampling

```
Input:  set of aligned scans $z_1, \dots, z_K$,
        grid resolution $r$,
        occupancy threshold $\psi$.
 1: for each scan $z_t$ do
 2:   for each beam $z_t^i$ in scan $z_t$ do
 3:     project beam $z_t^i$ into a point $x$ and add it to its corresponding cell $\langle i, j \rangle$
 4:     update all cells traversed by beam $z_t^i$
 5:   end for
 6: end for
 7: for each cell $\langle i, j \rangle$ do
 8:   if cell $\langle i, j \rangle$ is not empty and $p_{i,j} >= \psi$ then
 9:     $\bar{c} = \frac{1}{|\mathcal{C}_{i,j}|} \sum_{x_i \in \mathcal{C}_{i,j}} x_i$
10:     $c' =$ closest point for $\bar{c}$ in $\mathcal{C}_{i,j}$
11:     add $c'$ to $\mathcal{X}$
12:   end if
13: end for
14: return $\mathcal{X}$
```

The incremental clustering algorithm [Pauly *et al.*, 2002] starts by a choosing randomly a sample x_0 from $\mathcal{D}$ creating a cluster $\mathcal{C}_0 = \{x_0\}$ with a single sample. The algorithm then successively adds to $\mathcal{C}_0$ the nearest neighbors of x_0 in $\mathcal{D}$ until no more neighbors are found within a maximum allowed cluster radius r_{max}. The next cluster $\mathcal{C}_1$ is then created by selecting the nearest neighbor of x_0 in $\mathcal{D}$ excluding all the samples that have already been assigned to $\mathcal{C}_0$. This process is repeated until all the samples in $\mathcal{D}$ have been assigned to a cluster. As representative for each cluster, the sample used for initialization is selected. Figure 3.4 illustrates how incremental clustering groups the samples. The circular regions represent the different clusters. The numbers indicate the order in which the clusters where created. This order depends on the sample selected to initialize the first cluster. Since this sample is selected randomly, incremental clustering will produce different results even if the dataset $\mathcal{D}$ and the maximum cluster radius r_{max} remain fixed on repeated executions of the algorithm. These results may differ in the samples selected and also in the resulting number of samples.

The incremental clustering algorithm is described in Algorithm 3.3. The r_{max} parameter specifies the maximum radius of the clusters and therefore indirectly influences the number of clusters that will be created. In general, using a small value for r_{max} will produce more clusters than using a larger value. However, the exact number of clusters that will be produce, can not be specified directly. Line 2 of the algorithm select a random sample out of $\mathcal{D}$ and uses it *seed* to create the initial cluster. The while-loop in lines 4 through 14 keeps adding to the last created cluster the nearest neighbors of the cluster's seed. After no more samples can be found within the specified radius r_{max}, a new cluster is created in lines 9 through 11,

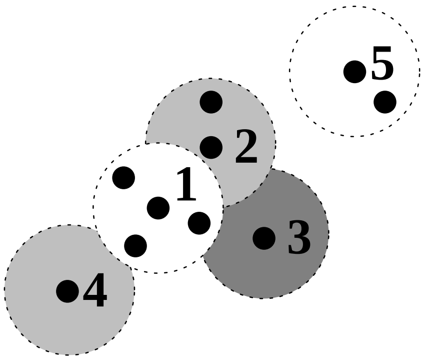


Figure 3.4.: Diagram illustrating how incremental clustering partitions the samples. The circular regions represent the different clusters. The numbers indicate the order in which the clusters where created.

using as seed the nearest unassigned neighbor of the previous seed. The time complexity of the algorithm is governed by the nearest neighbors searches. Using a kd-tree [Bentley, 1975] as data structure, each nearest neighbor search can be carried out in $O(\log n)$. The algorithm can be optimized by exploiting the structure of the kd-tree and searching for multiple nearest neighbors in each search. The general time complexity of the incremental clustering algorithm depends mostly on the distribution of the samples in $\mathcal{D}$, the maximum radius of the clusters r_{max}, and the structure of the underlying environment. In the worst case where all samples are separated by more than r_{max}, the algorithm has a time complexity of $O(n \log n)$.

3.3. Farthest Point Sampling

The last algorithm presented in this chapter is called farthest-point sampling. This sampling technique generates a model in an incremental way. The first point is selected randomly. Further points are added by selecting the point with the greatest distance to all those already in the model. The sampling process terminates when a specified number of points is reached.

Farthest point sampling is an approximative strategy for finding a model that minimizes the sum of squared error criterion E, discussed in Chapter 2, and repeated here for convenience

$$E(\mathcal{D}|\mathcal{X}) \quad = \quad \sum_{i=1}^{n} (x_i - x_i^*)^2, \tag{3.3}$$

where n is the number of samples in $\mathcal{D}$ and

$$x_i^* \quad = \quad \operatorname*{argmin}_{x_j} (x_i - x_j)^2 \qquad \forall x_j \in \mathcal{X}. \tag{3.4}$$

Algorithm 3.3 Incremental Clustering

```
Input:  set of samples $\mathcal{D} = \{x_1, \dots, x_n\}$,
        maximal radius $r_{max}$.
 1: $i = 0$
 2: randomly select $x_i \in \mathcal{D}$ and create cluster $\mathcal{C}_i = \{x_i\}$
 3: add $x_i$ to $\mathcal{X}$
 4: while unassigned samples remain do
 5:   $x'$ = nearest unassigned neighbor of $x_i$ in $\mathcal{D}$
 6:   if distance from $x_i$ to $x' \leq r_{max}$ then
 7:     assign $x'$ to $\mathcal{C}_i$
 8:   else
 9:     $i = i + 1$
10:     $x_i$ = nearest unassigned neighbor of $x_{i-1}$ in $\mathcal{D}$
11:     create cluster $\mathcal{C}_i = \{x_i\}$
12:     add $x_i$ to $\mathcal{X}$
13:   end if
14: end while
15: return $\mathcal{X}$
```

At the moment of adding a sample to the model, farthest-point sampling adds the sample x_{fp} that satisfies

$$\begin{aligned} x_{fp} &= \underset{x_i}{\operatorname{argmax}} \, (x_i - x_i^*)^2 \qquad \forall x_i \in \mathcal{D} \\ &= \underset{x_i}{\operatorname{argmax}}[\underset{x_j}{\operatorname{argmin}} \, (x_i - x_j)^2] \qquad \forall x_i \in \mathcal{D}, x_j \in \mathcal{X} \end{aligned} \tag{3.5}$$

In other words, farthest-point sampling adds the point $x_i \in \mathcal{D}$ that has the largest distance to its closest sample $x_j \in \mathcal{X}$. The farthest point sampling approach is described in Algorithm 3.4. Lines 4 through 10 compute for each $x_i \in \mathcal{D}$ the distance to its closest sample $x_j \in \mathcal{X}$, keeping track at the same time of the largest distance. This is repeated for each of the k samples that is added to $\mathcal{X}$. Using a kd-tree as data structure, the time complexity of a closest point search in the model is $O(\log k)$. Before adding a new point to the model, a closest point search must be carried out for each one of the n points in the data set. Since k points must be added to the model, the final time complexity of the algorithm is $O(kn \log k)$. In comparison, a greedy strategy that always adds the point that minimizes the error E, would have a time complexity of $O(kn^3)$. The greedy approach requires $(n - i)(n - i - 1)$ calculations of the error, when adding the i-th component to the model. Computing the error E has a time complexity of $O(n \log k)$, thus, generating a model of size k in a greedy way has a total time complexity of $O(kn^3 \log k)$. The implementation of the farthest-point algorithm used in this work is actually faster since each time a sample x_i is added to $\mathcal{X}$, only the distance of the samples that have a x_i as closest sample is recomputed. However, here we described the algorithm in its simplest form for clarity and to be able to analyze its time complexity formally.

Algorithm 3.4 Farthest-Point Sampling

```
Input:  set of points $\mathcal{D} = \{x_1, \ldots, x_n\}$,
        size $k$ of the looked-for model.
 1: select a random point $x_r$ from $\mathcal{D}$ and add it to $\mathcal{X}$
 2: while size of $\mathcal{X} \leq k$ do
 3:     $d_{max} = -\infty$
 4:     for $i = 1, \ldots, n$ do
 5:         $x_i^*$ = closest point for $x_i$ in $\mathcal{X}$
 6:         if $(x_i - x_i^*)^2 > d_{max}$ then
 7:             $d_{max} = (x_i - x_i^*)^2$
 8:             $x_{fp} = x_i$
 9:         end if
10:     end for
11:     add $x_{fp}$ to $\mathcal{X}$
12: end while
13: return $\mathcal{X}$
```

Figure 3.5 plots the likelihood LL of the a model obtained using farthest-point sampling and a model obtain the greedy approach as a function of the size of the model. When the models are small, the likelihood of the model obtained using the greedy strategy increases more rapid than the likelihood of the model obtained using farthest-point sampling. But as the models reach a certain size the likelihood of both models becomes almost identical. This is a property of the likelihood function and is independent of the strategy used to generate the model. Large models are all almost equally likely. x

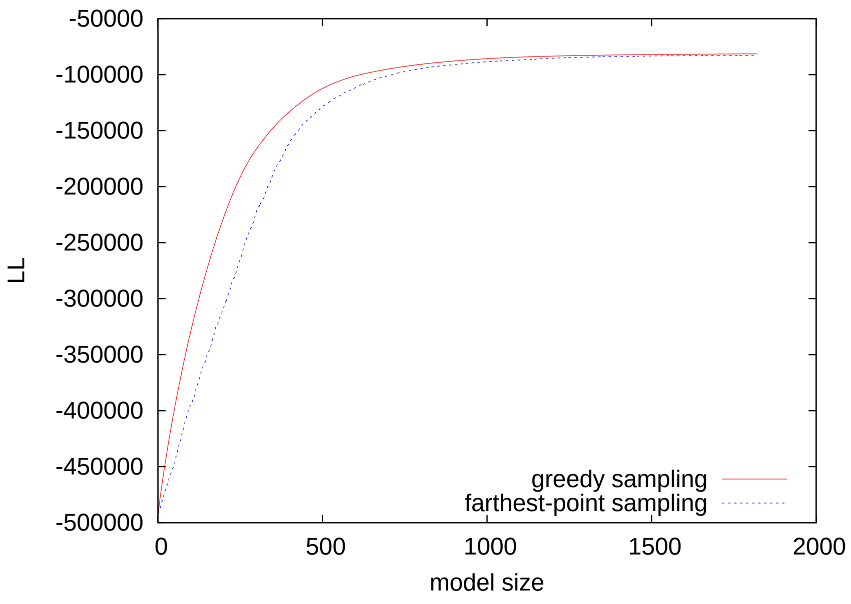


Figure 3.5.: Likelihood of a model generated using farthest-point sampling and another using a greedy algorithm as a function of the size of the model.

4. Iterative Optimization

Learning a sample-based map can be stated as the problem of finding the best subset of samples for a given dataset according to a given evaluation criterion. Since the number of possible subsets is extremely large an exhaustive search in which each possible subset is evaluated is only feasible for very small problems. Even if we restrict ourselves to the case where the size k of the model is known, there are $\binom{n}{k}$ possible ways to select k out of n samples. For a trivial problem where $n = 30$ and $k = 15$ we would have to evaluate over 155 million subsets in order to find the optimal one. For real datasets where n may be over a million samples we need to rely on approximative methods for finding a model for the data. The dataset simplification techniques presented in the previous chapter generate a subset of samples from a given dataset, starting with an empty set and adding samples to it according to some algorithm-dependent built-in heuristic. In contrast to that, we consider in this chapter the problem of iteratively improving an already given subset.

We present two different approximative iterative optimization approaches. In Section 4.1 we describe a maximum-likelihood strategy based on local derivatives of the likelihood function. We first describe analytically the idea behind maximum-likelihood estimation and then present the *k-means* and *fuzzy k-means* algorithms as concrete implementations. In Section 4.2 we present a stochastic approach for solving the optimization problem known as *simulated annealing*, in which random modifications of a model are evaluate, and are accepted or rejected according to some probability.

4.1. Maximum-Likelihood Estimation

Maximum-likelihood estimation interprets a sample-based map as a probabilistic model whose parameters are fixed but unknown. The idea behind maximum-likelihood estimation is to use all the samples in the dataset to select the parameters of the model that maximize the likelihood of the data.

In Section 2.1.1 it was mentioned that a subset $\mathcal{X}$ of samples can be interpreted as a probabilistic model for the complete dataset $\mathcal{D}$. This model is represented by the parameter vector $\Theta = \{\theta_1, \ldots, \theta_k\}$ where k is the number of samples in $\mathcal{X}$ and θ_j represents the parameters associated to model component ω_j. We are assuming that each sample $x_i \in \mathcal{D}$ is generated independently by first selecting a model component ω_j with probability $P(\omega_j)$ and then producing x_i according to a Normal distribution $p(x_i|\omega_j, \theta_j) \sim N(\mu_j, \Sigma_j)$. The parameters θ_j associated to the model component ω_j are the parameters μ_j and Σ_j of the corresponding Normal distribution. We further assume that each model component ω_j has the same prior probability $P(\omega_j)$ and that each Normal distribution $p(x_i|\omega_j, \theta_j)$ has the same symmetric covariance matrix Σ with known standard deviation σ. The only unknown parameters in the model are the values of the means μ_j. Our problem is then simplified to estimating the values of the parameters $\hat{\Theta} = \{\hat{\mu}_1, \ldots, \hat{\mu}_k\}$ that maximize the likelihood of $\mathcal{D}$.

Recalling from Section 2.1.1 that given a model Θ the likelihood of the data $\mathcal{D}$ is defined as

$$L(\mathcal{D}|\Theta) = \prod_{i=1}^{n} p(x_i|\Theta), \tag{4.1}$$

$$\tag{4.2}$$

the maximum-likelihood estimation problem can then be formally stated as

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmax}}\, L(\mathcal{D}|\Theta). \tag{4.3}$$

The values of the parameters $\hat{\Theta}$ are found by the standard methods of differential calculus, computing the values for Θ where the gradient of the likelihood function $L(\mathcal{D}|\Theta)$ is zero. There may be many values of Θ that make the gradient of $L(\mathcal{D}|\Theta)$ zero. This values can represent a global maximum, a local maximum or minimum, or an inflection point of $L(\mathcal{D}|\Theta)$. Thus, each solution must be tested in order to find which one represents the global maximum.

When dealing with likelihoods, an usual practice is to used the logarithm of the likelihood instead of the likelihood itself. Thus our problem is to find the values of $\Theta = \{\theta_1, \ldots, \theta_k\}$ that make the gradient of $\ln L(\mathcal{D}|\Theta)$ zero. In our specific case, we need to find the values of the means $\mu_\iota \in \Theta$ that satisfy

$$\frac{\partial}{\partial \mu_\iota} \ln L(\mathcal{D}|\Theta) = 0, \qquad \iota = 1, \ldots, k. \tag{4.4}$$

The derivative of $\ln L(\mathcal{D}|\Theta)$ with respect to μ_ι shown in the left part of the above expression can be computed as

$$\begin{aligned} \frac{\partial}{\partial \mu_\iota} \ln L(\mathcal{D}|\Theta) &= \frac{\partial}{\partial \mu_\iota} \ln \prod_{i=1}^{n} p(x_i|\Theta) \\ &= \frac{\partial}{\partial \mu_\iota} \sum_{i=1}^{n} \ln p(x_i|\Theta) \\ &= \sum_{i=1}^{n} \frac{\partial}{\partial \mu_\iota} \ln p(x_i|\Theta) \\ &= \sum_{i=1}^{n} \frac{1}{p(x_i|\Theta)} \frac{\partial}{\partial \mu_\iota} p(x_i|\Theta). \end{aligned} \tag{4.5}$$

From Section 2.1.1 we known that

$$p(x_i|\Theta) = \sum_{j=1}^{k} p(x_i|\omega_j, \theta_j) P(\omega_j). \tag{4.6}$$

Therefore Eq. (4.5) can then be written as

$$\begin{aligned}
\frac{\partial}{\partial \mu_\iota} \ln L(\mathcal{D}|\Theta) &= \sum_{i=1}^{n} \frac{1}{p(x_i|\Theta)} \frac{\partial}{\partial \mu_\iota} \sum_{j=1}^{k} p(x_i|\omega_j, \theta_j) P(\omega_j) \\
&= \sum_{i=1}^{n} \frac{P(\omega_\iota)}{p(x_i|\Theta)} \frac{\partial}{\partial \mu_\iota} p(x_i|\omega_\iota, \theta_\iota) \\
&\overset{\text{Derivative of } \ln}{=} \sum_{i=1}^{n} \frac{p(x_i|\omega_\iota, \theta_\iota) P(\omega_\iota)}{p(x_i|\Theta)} \frac{\partial}{\partial \mu_\iota} \ln p(x_i|\omega_\iota, \theta_\iota). \qquad (4.7)
\end{aligned}$$

As in [Duda *et al.*, 2000] let us define the posterior probability of component ω_ι given x_i

$$P(\omega_\iota|x_i, \Theta) = \frac{p(x_i|\omega_\iota, \theta_\iota) P(\omega_\iota)}{p(x_i|\Theta)}, \qquad (4.8)$$

so that the derivative of $\ln L(\mathcal{D}|\Theta)$ with respect to μ_ι can be written as

$$\frac{\partial}{\partial \mu_\iota} \ln L(\mathcal{D}|\Theta) = \sum_{i=1}^{n} P(\omega_\iota|x_i, \Theta) \frac{\partial}{\partial \mu_\iota} \ln p(x_i|\omega_\iota, \theta_\iota). \qquad (4.9)$$

Since the component conditional probabilities $p(x_i|\omega_\iota, \theta_\iota)$ are Normally distributed with mean μ_ι and covariance matrix $\Sigma_\iota = \sigma^2 I$ we get

$$\begin{aligned}
\frac{\partial}{\partial \mu_\iota} \ln p(x_i|\omega_\iota, \theta_\iota) &= \frac{\partial}{\partial \mu_\iota} \ln \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{(x_i - \mu_\iota)^2}{2\sigma^2} \right] \right] \qquad (4.10) \\
&= \frac{\partial}{\partial \mu_\iota} \left[\ln \left[\frac{1}{\sqrt{2\pi}\sigma} \right] - \frac{(x_i - \mu_\iota)^2}{2\sigma^2} \right] \qquad (4.11) \\
&= -\frac{1}{2\sigma^2} \frac{\partial}{\partial \mu_\iota} (x_i - \mu_\iota)^2 \\
&= \frac{(x_i - \mu_\iota)}{\sigma^2}. \qquad (4.12)
\end{aligned}$$

Substituting the above expression in Eq. (4.9) we get

$$\frac{\partial}{\partial \mu_\iota} \ln L(\mathcal{D}|\Theta) = \sum_{i=1}^{n} P(\omega_\iota|x_i, \hat{\Theta}) \frac{(x_i - \hat{\mu}_\iota)}{\sigma^2}. \qquad (4.13)$$

According to expression (4.4) the derivatives $\ln L(\mathcal{D}|\Theta)$ with respect to μ_ι must be zero at the values of μ_ι that maximize the likelihood of the data. We can multiply Eq. (4.13) by σ^2 and rearrange the terms to obtain the solution

$$\hat{\mu}_\iota = \frac{\sum_i^n P(\omega_\iota|x_i, \hat{\Theta}) x_i}{\sum_i^n P(\omega_\iota|x_i, \hat{\Theta})}. \qquad (4.14)$$

Expression (4.14) leads to a set of k interlaced nonlinear equations which usually do not have a unique solution. These solutions correspond to the values of the model Θ for which the gradient of the likelihood function $L(\mathcal{D}|\Theta)$ is zero and it is necessary to test each solution to see whether it corresponds to the global maximum. One technique that can be used to obtain a solution, is to use an initial estimate for the priors $P(\omega_\iota|x_i, \hat{\Theta})$ and then use Eq. (4.14) to iteratively update the estimates. This technique can be viewed as an instance of the *Expectation-Maximization* algorithm which can be shown to converge to a local optimum [Dempster *et al.*, 1977]. In the following section we present the k*-means* and *fuzzy* k*-means* algorithms as two concrete implementations of this strategy.

4.1.1. The k-Means Algorithm

Expression (4.14) can not be used to obtain $\hat{\Theta} = \{\hat{\mu}_1, \dots, \hat{\mu}_k\}$ explicitly, but can be used to improve an initial estimate $\Theta = \{\mu_1, \dots, \mu_k\}$ as follows

$$\hat{\mu}_\iota \quad = \quad \frac{\sum_i^n P(\omega_\iota|x_i, \Theta)x_i}{\sum_i^n P(\omega_\iota|x_i, \Theta)} \qquad \iota = 1 \dots, k. \tag{4.15}$$

If we approximate the posterior $P(\omega_\iota|x_i, \Theta)$ as

$$P(\omega_\iota|x_i, \Theta) \quad = \quad \begin{cases} 1 & \text{if } ||x_i - \mu_\iota|| < ||x_i - \mu_{\iota'}|| \text{ for all } \iota \neq \iota' \\ 0 & \text{otherwise}, \end{cases} \tag{4.16}$$

the posterior $P(\omega_\iota|x_i, \Theta)$ is 1 when ω_ι is the closest model component for x_i. Then Eq. (4.15) gives $\hat{\mu}_\iota$ as the average or mean of the samples that have ω_ι as closest component. If we define $\mathcal{D}_\iota$ as the set of all samples that have ω_ι as closest model component, then Eq. (4.15) can be stated as

$$\hat{\mu}_\iota \quad = \quad \frac{1}{|\mathcal{D}_\iota|} \sum_{x_i \in \mathcal{D}_\iota} x_i. \tag{4.17}$$

Starting with an initial model $\Theta^0 = \{\mu_1^0, \dots, \mu_k^0\}$ we can divide the dataset $\mathcal{D}$ into k partitions $\mathcal{D}_1^0, \dots, \mathcal{D}_k^0$ where each $\mathcal{D}_j^0$ is the set containing all samples $x_i \in \mathcal{D}$ that have ω_j as closest model component. Using $\mathcal{D}_1^0, \dots, \mathcal{D}_k^0$ we can compute an improved model $\Theta^1 = \{\mu_1^1, \dots, \mu_k^1\}$ according to Eq. (4.17). The the newly computed model Θ^1 is used to partition $\mathcal{D}$ again and the resulting partitions are then used to computed an improved model Θ^2. This process is repeated until no more changes take place between two consecutive iterations. The resulting procedure is known as the k*-means* algorithm [MacQueen, 1967]. In its original version, the k-means algorithm computes each $\hat{\mu}_j$ as the mean of the samples that have ω_j as closest model component. The resulting set of points is not necessarily a subset of $\mathcal{D}$ since the means may not correspond to an existing sample. In order to solve this problem, after the algorithm has converged each one of the computed means $\hat{\mu}_j$ is replaced by its closest sample in $\mathcal{D}$.

The k-means procedure is presented in Algorithm (4.1). Lines 4 through 7 associate to each sample x_i its closest component in Θ. In line 9 the estimates $\hat{\mu}_j$ are computed as specified in

Algorithm 4.1 k-Means

```
Input:  set of points $\mathcal{D} = \{x_1, \dots, x_N\}$,
        initial model $\Theta = \{\mu_1, \dots, \mu_k\}$
 1: repeat
 2:   $changes = 0$
 3:   $\mathcal{D}_j = \emptyset, \quad j = 1, \dots, k$
 4:   for all $x_i \in \mathcal{D}$ do
 5:     $\mu_j$ = nearest neighbor of $x_i$ in $\Theta$
 6:     add $x_i$ to $\mathcal{D}_j$
 7:   end for
 8:   for all $\mu_j \in \Theta$ do
 9:     $\hat{\mu} = \frac{1}{|\mathcal{D}_j|} \sum_{x_i \in \mathcal{D}_j} x_i$
10:     if $\bar{\mu} \neq \mu_j$ then
11:       $\mu_j = \hat{\mu}$
12:       $changes++$
13:     end if
14:   end for
15: until $changes = 0$
16: for all $\mu_j \in \Theta$ do
17:   $\mu'$ = nearest neighbor of $\mu_j$ in $\mathcal{D}$
18:   $\mu_j = \mu'$
19: end for
20: return $\Theta$
```

Eq. (4.15) based on the posterior defined in Eq. (4.16). Lines 16 through 19 assign to each $\mu_j \in \Theta$ its closest sample in $\mathcal{D}$. By using a kd-tree as data structure for the model nearest neighbor searches can be carried out in $O(\log k)$, where k is the number of components in Θ. The partitioning of the samples has a complexity of $O(n \log k)$. Computing the improved values for the parameters of the model has a complexity of $O(k)$. This two steps are executed T times until the algorithm converges. Finally, replacing all the computed means with its closest point in $\mathcal{D}$ has a time complexity of $O(k \log n)$. The total time complexity of the algorithm is then $O(Tn \log k)$, where T is the number of iterations needed for convergence, n is the number of samples in the dataset, and k is the number of components in the model.

As most greedy strategies the k-means algorithm converges to a local optimum. There is no guarantee that a global optimum will be found. The quality of the model produced by the k-means algorithm will depend strongly on the starting model. Consider for example the model shown in Figure 4.1(a). The filled circles represents the samples that belong to the model. This configuration would induce the partitioning shown in Figure 4.1(b), where the rectangles represent the different partitions. The filled circles in Figure 4.1(c) represent the improved model parameters. The next iteration does not change the model and therefore the algorithm

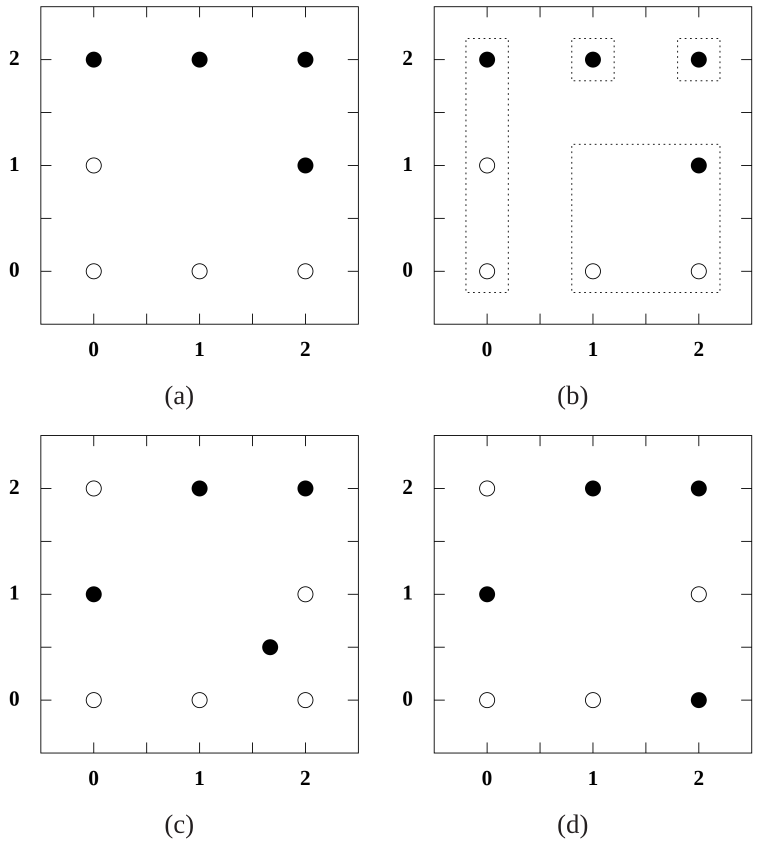


Figure 4.1.: Illustration of the k-means procedure. Figure (a) shows the initial model. The filled circles represent the samples that belong to the model. Figure (b) shows the partitioning induced by the model shown in (a). The filled circles in Figure (c) represent the improved values for the model components. Figure (d) shows the final resulting model. For this particular configuration, the algorithm terminates after just 2 iterations.

terminates. The final model is shown in Figure 4.1(d). If a different initial model is used, the algorithm may produce a different result. A common strategy to deal with this problem is to run the algorithm several times using different initial models and keep the best result.

4.1.2. The Fuzzy k-Means Algorithm

The k-means algorithm presented above associates each sample $x_i \in \mathcal{D}$ to exactly one component ω_j in the model Θ. It partitions $\mathcal{D}$ into k disjunct sets $\mathcal{D} = \mathcal{D}_1 \cup \ldots \cup \mathcal{D}_k$, where k is the number of components in Θ. In a more general version of the algorithm, known as *fuzzy k-means*, samples can be associated to more than one component according to a degree of membership. From a clustering perspective the fuzzy k-means algorithm produces *fuzzy* clusters that do not poses a defined border.

The membership of sample x_i to a component ω_j is given by the posterior $P(\omega_j|x_i, \Theta)$ as defined in Eq. (4.8). From Section 2.1.1 we know that

$$p(x_i|\Theta) \quad = \quad \sum_{\iota=1}^{k} p(x_i|\omega_\iota, \theta_\iota)P(\omega_\iota). \tag{4.18}$$

Substituting the above equation in Eq. (4.8) we get

$$P(\omega_j|x_i, \Theta) \quad = \quad \frac{p(x_i|\omega_j, \theta_j)P(\omega_j)}{\sum_{\iota=1}^{k} p(x_i|\omega_\iota, \theta_\iota)P(\omega_\iota)}. \tag{4.19}$$

Since we are assuming that every component ω_j has the same prior probability $P(\omega_j)$, the degree of membership of sample x_i to ω_j can be computed as

$$P(\omega_j|x_i, \Theta) \quad = \quad \frac{p(x_i|\omega_j, \theta_j)}{\sum_{\iota=1}^{k} p(x_i|\omega_\iota, \theta_\iota)}. \tag{4.20}$$

Now, according to the maximum-likelihood estimation strategy the improved value for the parameters μ_ι of the model are computed as

$$\hat{\mu}_\iota \quad = \quad \frac{\sum_i^n P(\omega_\iota|x_i, \Theta)x_i}{\sum_i^n P(\omega_\iota|x_i, \Theta)}. \tag{4.21}$$

This gives us the general strategy of the fuzzy k-means procedure. As in the k-means algorithm, we start with an initial model $\Theta^0 = \{\mu_1^0, \ldots, \mu_k^0\}$. Then for each sample $x_i \in \mathcal{D}$ the degree of membership to each component $\omega_j \in \Theta^0$ is computed. These values are used to calculate the improved values for the parameters according to Eq. (4.21). The new improved model Θ^1 is then used as starting point for the next iteration. The process is repeated until the difference in the value of the parameters is smaller than a given threshold ϵ.

The fuzzy k-means procedure is described in Algorithm 4.2. Lines 4 through 8 compute the normalizing constant $\eta_i = \sum_{j=1}^{k} p(x_i|\omega_j, \theta_j)$ for each sample $x_i \in \mathcal{D}$. The normalizing

Algorithm 4.2 Fuzzy k-Means

```
Input:  set of points $\mathcal{D} = \{x_1, \ldots, x_n\}$,
        initial model $\Theta = \{\mu_1, \ldots, \mu_k\}$
 1: repeat
 2:     $changes = 0$
 3:     $\eta_j = 0, \quad j = 1, \ldots, n$
 4:     for all $x_i \in \mathcal{D}$ do
 5:         for all $\mu_j \in \Theta$ do
 6:             $\eta_i = \eta_i + p(x_i|\omega_j, \Theta)$
 7:         end for
 8:     end for
 9:     for all $\mu_j \in \Theta$ do
10:         $\hat{\mu} = 0$, $\rho = 0$
11:         for all $x_i \in \mathcal{D}$ do
12:             $P(\omega_j|x_i, \Theta) = p(x_i|\omega_j, \Theta)/\eta_i$
13:             $\hat{\mu} = \hat{\mu}_j + P(\omega_j|x_i, \Theta)x_i$
14:             $\rho = \rho + P(\omega_j|x_i, \Theta)$
15:         end for
16:         $\hat{\mu} = \hat{\mu}/\rho$
17:         if $|\bar{\mu} - \mu_j| \geq \epsilon$ then
18:             $\mu_j = \bar{\mu}$
19:             $changes + +$
20:         end if
21:     end for
22: until $changes = 0$
23: for all $\mu_j \in \Theta$ do
24:     $\mu'$ = nearest neighbor of $\mu_j$ in $\mathcal{D}$
25:     $\mu_j = \mu'$
26: end for
27: return $\Theta$
```

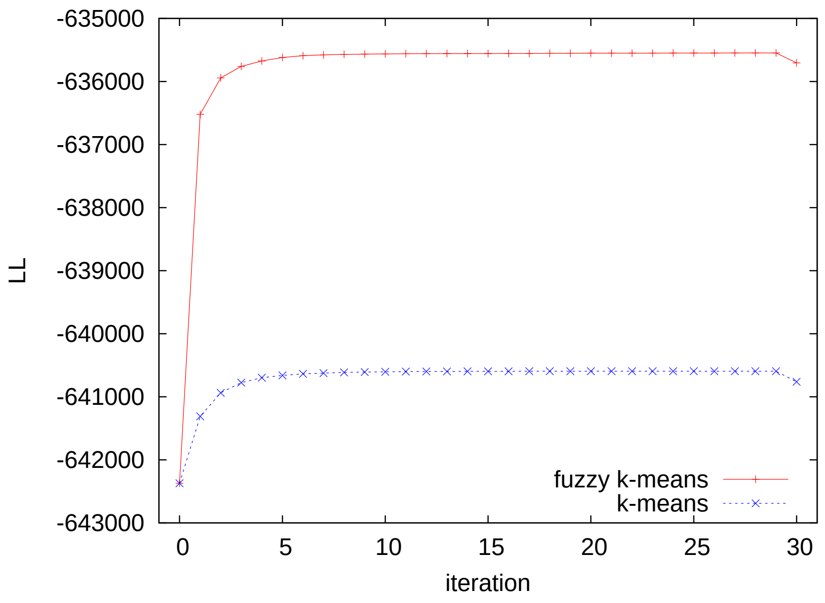


Figure 4.2.: Behavior of the Likelihood of a model as it is iteratively improved by the k-means and fuzzy k-means algorithm. For this example, both algorithms converged after the same number of iterations. The decrease in the likelihood at the last iteration is caused by the replacement of the means by their closest component in the dataset.

constants are used in line 12 to compute the posterior $P(\omega_j|x_i, \Theta)$ according to Eq. (4.20). Lines 10 through 16 reestimate the value of every model component according to Eq. (4.15). Just as in the k-means algorithm, the resulting set of points is not necessarily a subset of $\mathcal{D}$. After the algorithm converges, lines 23 through 26 replace the computed components with their closest point in $\mathcal{D}$. Computing the normalizing constants has a time complexity of $O(nk)$ and computing the improved values of the parameters of the model has too a time complexity of $O(nk)$. This two steps are repeated T times until convergence. Finally, replacing each computed means with its closest point in $\mathcal{D}$ has a total time complexity of $O(k \log n)$. The total time complexity of the fuzzy k-means algorithm is $O(Tkn)$, where T is the number of iterations required for convergence, n the number of samples in $\mathcal{D}$, and k the number of components in Θ.

Figure 4.2 shows the general behavior of the likelihood of a model as the k-means and fuzzy k-means algorithm progress. In the example shown in the figure both algorithms converged after the same number of iterations. The decrease in the likelihood at the last iteration is caused by the replacement of the means by their closest component in the dataset.

For both algorithms, the number of iterations required for convergence depends on the dataset and the model used to initialize the algorithm. The running times for the fuzzy k-means algorithm are in general longer than for the k-means algorithm due to the computation of the posteriors. This is specially critical for large datasets and large models. However, fuzzy

k-means yields better results in general, and has a more elegant probabilistic framework than the k-means algorithm. The fuzzy k-means just like the k-means algorithm suffers too from the local optima problem, and the results also depend on the starting model.

In the next section we present a different optimization strategy known as *simulated annealing* in which random modifications of a model are evaluated, and are accepted or rejected according to some probability. The stochastic nature of this algorithm is what helps it to overcome the local optima problem.

4.2. Optimization by Simulated Annealing

Annealing is a process used in physics in which some material, like a metal, is first melted and then slowly cooled until it solidifies. By gradually lowering the temperature the process allows the material to reach a state of thermodynamic equilibrium. If this is not done, the material will solidify in an unstable state and will produce, for example, a weak metal.

Simulated annealing [Kirkpatrick *et al.*, 1983] is motivated by these ideas from the field of physics. It is an iterative improvement algorithm that starts with a given model Θ_0. In each step i the current model Θ_i is modified by removing a randomly chosen component and replacing it with a different randomly chosen component outside the model. Remember that a model $\Theta = \{\mu_1, \ldots, \mu_k\}$ for a dataset $\mathcal{D}$ is nothing more than a different interpretation for a subset $\mathcal{X} = \{x_1, \ldots, x_k\}$ of the samples in $\mathcal{D}$. Each $\mu_i \in \Theta$ represents a $x_j \in \mathcal{D}$. Thus, in each step i a randomly chosen component $\mu_j \in \Theta_i$ is replaced with a component in $\{\mathcal{D} - \Theta_i\}$ to obtain a new model $\bar{\Theta}_i$. The new model $\bar{\Theta}_i$ is then evaluated and the resulting change in the likelihood of the model $\Delta LL(\bar{\Theta}_i, \Theta_i)$ is computed using the following equation:

$$\Delta LL(\bar{\Theta}, \Theta) \quad = \quad LL(\mathcal{D}|\bar{\Theta}) - LL(\mathcal{D}|\Theta), \tag{4.22}$$

where $LL(\mathcal{D}|\Theta)$ is the log-likelihood of the data in $\mathcal{D}$ given the model Θ as described in Section 2.1.1. If the new model $\bar{\Theta}_i$ is better, the change $\Delta LL(\bar{\Theta}_i, \Theta_i)$ will be positive and the new model is accepted. On the other hand, if the resulting model is worst or equal, the new model is accepted according to the following probability:

$$\exp\left[\Delta LL(\bar{\Theta}_i, \Theta_i)/t_i\right], \tag{4.23}$$

where t_i is called the *temperature* of the system, and is a control parameter in the same unit as LL. The temperature t_i determines the probability of accepting a lower-likelihood model at step i. The model selected at step i is then used as starting model for the next step $i + 1$. The initial temperature t_0 is set at some high, problem-specific value, and is decreased according to some predefined schedule.The algorithm terminates when the temperature reaches a specified lower bound t_f.

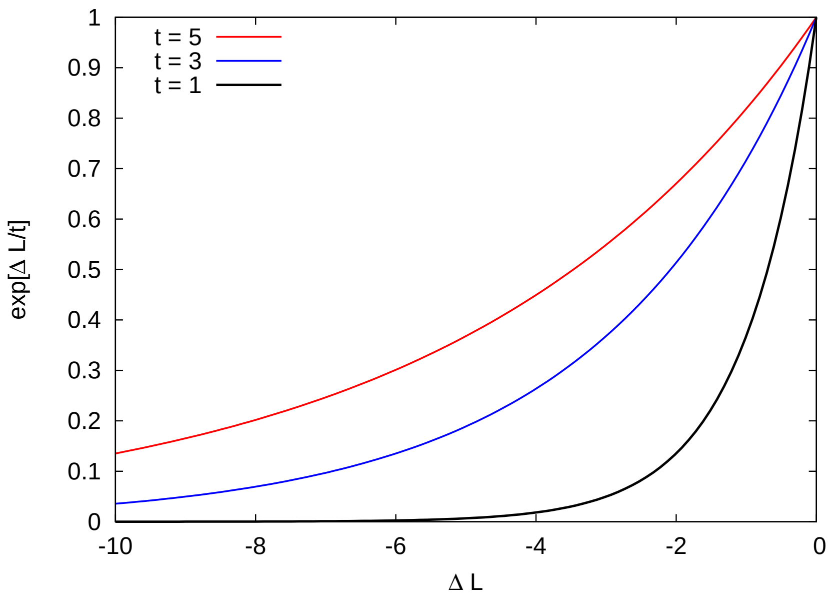
During the first iterations of the algorithm when the temperature is high, the probability of accepting a lower-likelihood model is also high. As the algorithm progresses and the temperature decreases, the probability of accepting a lower-likelihood model decreases. At lower temperatures the algorithm behaves in a greedy way, having a stronger preference for higher-likelihood models. The fact that models with lower likelihood values have a positive probability of being accepted is what allows the algorithm to escape from local optima.

In Figure 4.3(a) the acceptance probability $\exp[\Delta LL/t]$ is shown as a function in the change of the likelihood of two models $\Delta LL(\bar{\Theta}, \Theta)$. Negative values for $\Delta LL(\bar{\Theta}, \Theta)$ indicate that the model $\bar{\Theta}$ to be evaluated has a lower likelihood than the current model Θ. The probability of accepting a lower-likelihood model depends on how big the change $\Delta LL(\bar{\Theta}, \Theta)$ is. Models that produce big changes have a lower probability of being accepted than models that produce only small changes. As the temperature t decreases the probability of accepting a lower-likelihood model also decreases. Figure 4.3(b) shows the acceptance probability as a function of the temperature t.

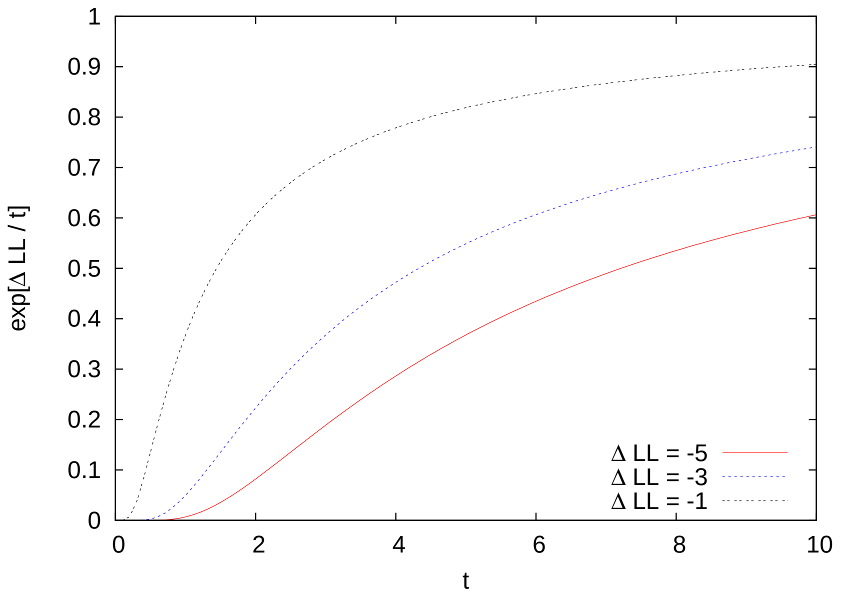
The simulated annealing procedure is described in Algorithm 4.3. In line 3 the current model Θ_i is modified by removing one randomly chosen component and replacing it with a different randomly chosen component outside Θ_i. Line 4 evaluates the new model $\bar{\Theta}_i$. This new model is accepted if its likelihood is higher than the likelihood of the current model Θ_i. If its likelihood is not higher, the the new model is accepted according to the acceptance probability for the computed change $\Delta LL(\bar{\Theta}_i, \Theta_i)$ and the current temperature t_i. The acceptance of a model is implemented by randomly selecting a positive real number $rand[0, 1)$ smaller than 1. If the acceptance probability $\exp[\Delta LL(\bar{\Theta}_i, \Theta_i)/t_i]$ is larger than $rand[0, 1)$ then $\bar{\Theta}_i$ is accepted. The function $t(.)$ is called the *annealing schedule* and controls the way in which the temperature is decreased as a function of the number of iterations. The annealing schedule determines the starting temperature $T(0) = t_0$, the number of models that are to be evaluated before decreasing the temperature, and the rate at which this temperature decreases. The time complexity of the algorithm is $O(Tnk)$ where T is the number of iterations needed until the final temperature t_f is reached, n is the number of samples in the dataset $\mathcal{D}$, and k is the number of components in the model. The factor $O(nk)$ in the complexity of the algorithm is caused by the calculation of the likelihood of the models. The time complexity can be reduced by exploiting the fact that the models to be compared are only different by one component. This can be used to compute the likelihood in a more efficient way by only considering the local changes.

4.2.1. Annealing Schedule

It remains to describe how the annealing schedule $t(.)$ actually looks like. It has already being mentioned that the initial temperature t_0 is set to some high problem-specific value and then decreased until a final temperature t_f is reached. In general, the initial temperature must be high enough so that every model has a positive probability of being accepted. The decrease in temperature must be slow enough to allow the algorithm to escape a local optima, and the final temperature t_f must be low enough to minimize the probability of accepting a model different than a global optima in the final steps of the iteration. Beside these general guidelines, there is no specific strategy for coming up with a schedule other than by tuning the parameters in a problem-specific way.



(a)



(b)

Figure 4.3.: Figure 4.3(a) shows the acceptance probability $\exp\left[\Delta LL/t\right]$ as a function of the change in the likelihood $\Delta LL(\bar{\Theta}, \Theta)$ for three different values of the temperature t. Figure 4.3(b) shows the acceptance probability as a function of the temperature t for three different values of ΔLL.

Algorithm 4.3 Simulated Annealing

```
Input:  set of points $\mathcal{D} = \{x_1, \ldots, x_n\}$,
        initial model $\Theta_0 = \{\mu_1, \ldots, \mu_k\}$,
        annealing schedule $t(.)$,
        final temperature $t_f$
 1: $i = 0$
 2: while $t(i) \geq t_f$ do
 3:     generate random model $\bar{\Theta}_i$ from $\Theta_i$
 4:     if $\Delta LL(\bar{\Theta}_i, \Theta_i) > 0$ or $\exp\left[\Delta LL(\bar{\Theta}_i, \Theta_i)/t(i)\right] > rand[0, 1)$ then
 5:         $\Theta_{i+1} = \bar{\Theta}_i$
 6:     else
 7:         $\Theta_{i+1} = \Theta_i$
 8:     end if
 9:     $i = i + 1$
10: end while
11: return $\Theta_i$
```

When selecting reasonable parameters for the annealing schedule, it is important to consider the way the algorithm explores the likelihood space. In each step i, our algorithm evaluates models by removing a component from the current model Θ_i and replacing it with a different component outside the model. Thus the temperature values must be selected based on the difference ΔLL between the likelihood of models that differentiate each other by only one component. The way in which the algorithm explores the likelihood space leads to the two following observations. First, the algorithm moves slowly through the likelihood space since the transitions are between models that are identical but for a single component. This makes the local likelihood landscapes shallow. Secondly, models with a high likelihoods lie far apart in the likelihood space from models with low likelihoods.

Based on these observations a low-temperature schedule is chosen to prevent the algorithm to wander too deep into low-likelihood regions. The initial temperature t_0 is set empirically by studying the local landscape for different randomly chosen models and observing the average ΔLL. At the beginning, the algorithm should behave in a random way with a slight preference for high-likelihood models. The degree of randomness is controlled by the initial temperature. As the algorithm progresses, the temperature is decreased according to $t(i + 1) = \alpha t(i)$ with $0 < \alpha < 1$. The parameter α controls how gradual the decrease in temperature is. Since the algorithm moves slowly through the likelihood space, it is important to allow enough iterations to take place so that the high likelihood regions can be reached. For this, the temperature is not decreased after every iteration but we let the algorithm evaluate a number m of models before decreasing the temperature.

Figure 4.4 shows the likelihood of a model as its modified by the simulated annealing algorithm. It can be observed how the likelihood of the current model can decrease during some iterations. This occurs when a lower-likelihood model is accepted. In general, however, the

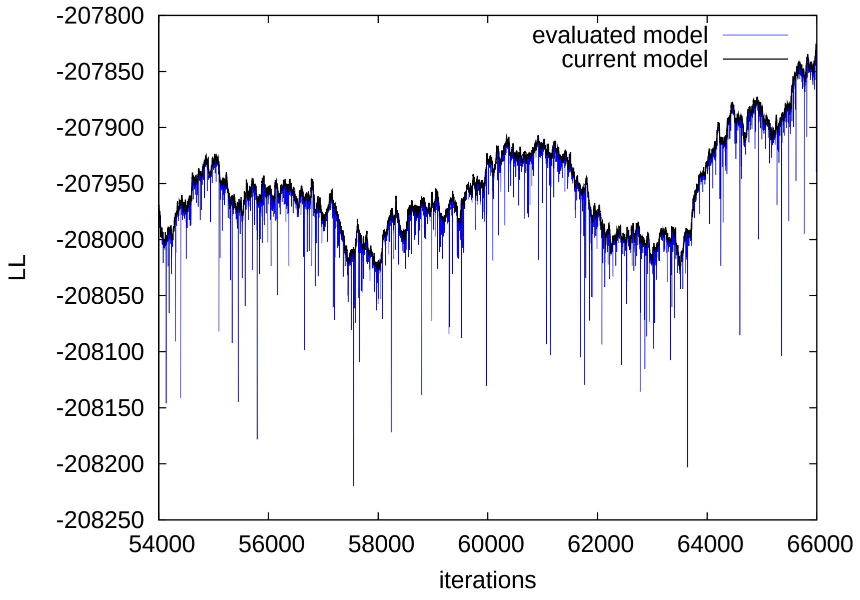


Figure 4.4.: Likelihood of a model as the simulated annealing progresses. The figure also shows the likelihood of the models that were evaluated on each iteration.

likelihood tends to increase. The figure shows how the simulated annealing algorithm moves through the likelihood space. Also shown in the figure is the likelihood of the models that were evaluated on each iteration.

According to the annealing schedule, the number of iterations needed for the algorithm to terminate can be computed as

$$m\lceil \log_\alpha(t_f/t_0)\rceil, \tag{4.24}$$

where m is the number of models to be evaluated before every temperature reduction and $\lceil\log_\alpha(t_f/t_0)\rceil$ is the number of times the initial temperature t_0 must be decreased until the final temperature t_f is reached. According to (4.24), high values for t_0, α, m, and a low t_f cause the algorithm to evaluate a large number of models. This increase the probability of finding a good solution. However, computational resources must be taken into account at the moment of selecting the schedule parameters since the number of models to evaluate can be restrictively large. A compromise must be reached between the probability of finding a good model, and the needed computing time.

The simulated annealing algorithm like the k-means and fuzzy k-means algorithm is susceptible to the initial model. Different initial models produce different results. However, the effect of the initial model is not as large as for the other two algorithms.

5. Experimental Results

The algorithms presented throughout this work have been implemented and evaluated using various datasets. These correspond to different environments and were gathered using real robots. A description of the datasets used can be found in Appendix A. The measurements were recorded and aligned using the Carnegie Mellon robot navigation toolkit (CARMEN) [Roy *et al.*, 2003]. For clarity only the results for two different datasets are presented throughout this chapter. However, the observations and analysis of the results are general and include the results obtained using other datasets as well. These other results are presented in Appendix B. The datasets used for the results presented in this chapter where gathered at the Intel Research Lab in Seattle and at the 4th floor of the Sieg Hall at the University of Washington.

The Intel Research Lab is shown in Figure 5.1 and has a size of 29m$\times$29m. The measurements were gathered using a Pioneer 2 robot equipped with a SICK laser range finder sensor. A total of 910 scans were taken with 180 range measurements per scan. Only measurements under 10 meters were considered at the moment of projecting the samples. The resulting dataset has a total of 155648 samples and has a density of 181.41 samples per squared meter. The 4th floor of the Sieg Hall at the University of Washington shown in Figure 5.2 has a size of 50m$\times$12m. A total of 241 scans where taken with 361 range measurements per scan. The corresponding dataset consist of 83892 samples and has a density of 92.53 samples per squared meter. Here too, only ranges under 10 meters where used to obtain the samples.

5.1. Dataset Simplification

The first experiment is designed to evaluate the four different dataset simplification techniques presented in Chapter 3. To compare the quality of the models, the likelihood of the data LL is used as evaluation criterion. As mentioned in Chapter 2 larger models have in general higher likelihoods. Therefore, in order to make the comparisons fairer, we compare models having the same number of components. To do this, we defined *reference sizes* for the models to be compared. These sizes correspond to the size of the models obtained with grid-based sampling using different grid resolutions. We used 4 different resolutions for the different datasets used in the experiments.

Once the reference sizes were set, we used the remaining dataset simplification techniques to generate models according to these reference sizes. With incremental clustering and occupancy grid sampling, the size of the resulting model has to be controlled by adjusting the input parameters of the algorithms. These parameters are the maximum allowed radius for incre-

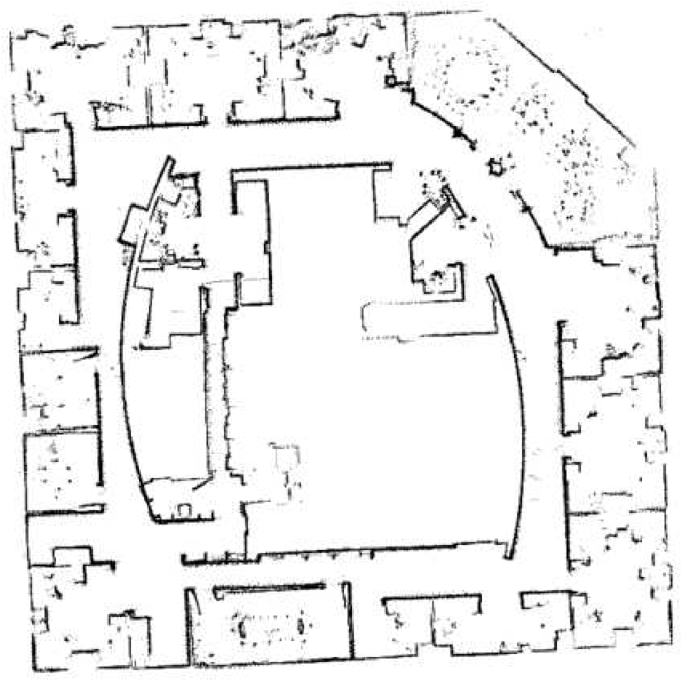


Figure 5.1.: Intel Research Lab in Seattle.

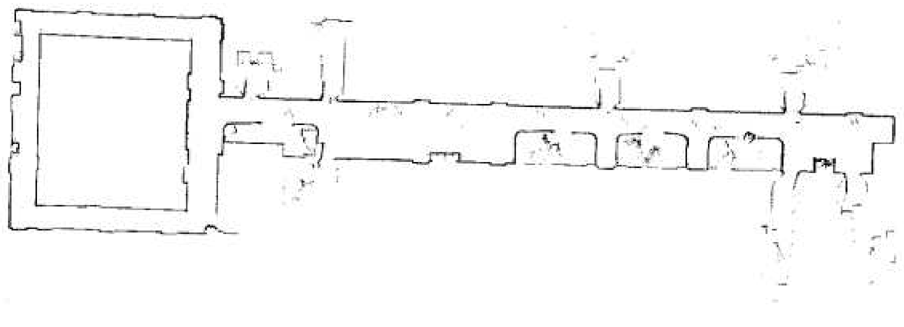


Figure 5.2.: Sieg Hall at the University of Washington.

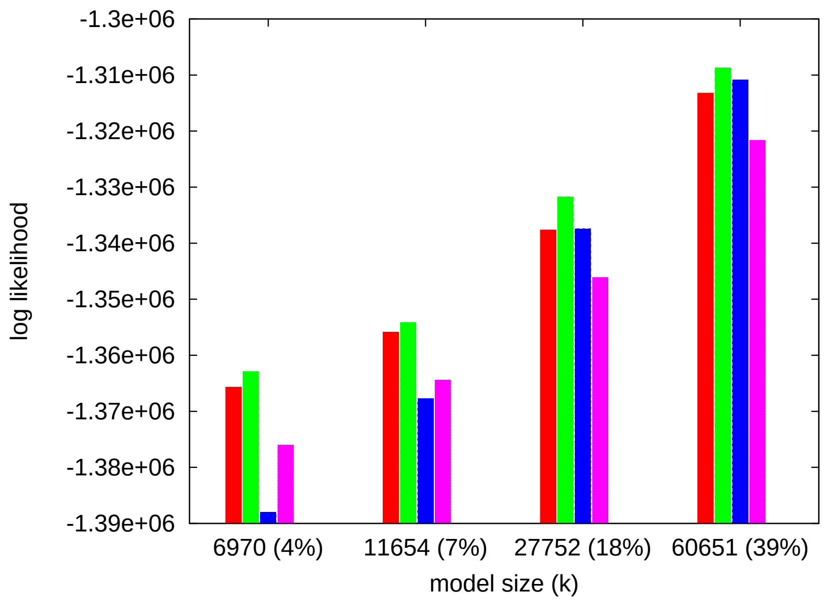
mental clustering, and the resolution of the grid and the occupancy threshold for occupancy grid sampling. The degree in which the size of the resulting model can be controlled adjusting the values of these parameters is limited and the sizes of the resulting models were not always the desired reference sizes. Farthest-point sampling is the only downsampling strategy for which the size of the resulting model can be explicitly specified. Therefore no parameter had to be tuned in order to obtain models of the desired size.

Incremental clustering and farthest-point sampling are stochastic algorithms since they produce different models even when given the same input parameters. Incremental clustering not only produces different models, but also produces models of different sizes even for a fixed maximum allowed radius. To deal with the stochastic nature of these two algorithms the average of the results of several runs was used for the evaluations.

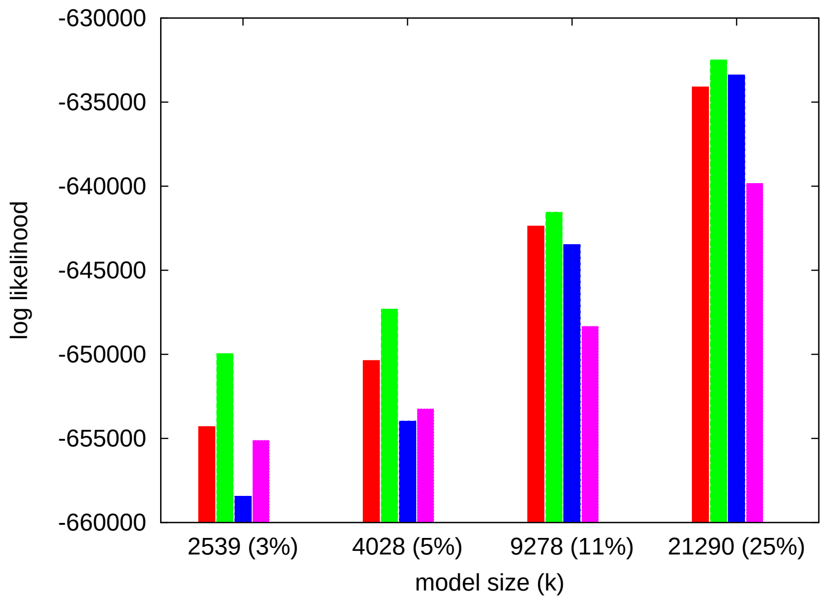
Figure 5.3 shows the likelihood of the models obtained using the different simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). From all these techniques, incremental clustering produces the best results overall. Farthest-point sampling produces relatively good models for large reference sizes, but as the size decreases the quality of the resulting models deteriorates in contrast to the results of the other algorithms. Grid-based sampling produces good models for both large and small sizes. The likelihood of the models obtained with occupancy grid sampling are in general relatively low. This is to be expected since the samples that are discarded by this algorithm are usually outsiders which are important for the reduction of the sum of squared error of the model, ultimately associated to the likelihood of the model.

In general the quality of the models obtained with the different techniques depends on how large the models is in proportion to the complete dataset. Larger models have higher likelihoods and the difference between the results of the different algorithms decreases as the size of the models increases. This can be see in Figure 5.4 which shows the likelihood as a function of the size of the model for the incremental clustering and farthest-point sampling algorithms. The standard deviation shown on the figure was augmented by a factor of 10 for displaying purposes. Observe how it increases as the size of the models decreases. This shows that the difference in the likelihood values of the produced models is small for large models and increases as the size of the models decreases. We observe on the graphic that the incremental clustering algorithm produces not only better models than farthest-point sampling but the variance in the quality of the produced models is smaller. The memory requirements needed to produced large models with grid-based sampling and occupancy grid sampling prevented us from using these two algorithms for this comparison.

Besides the quality of the resulting models the execution time of the algorithms is also an important factor when comparing the different simplification techniques. We evaluated the execution times of the different techniques running them on a standard PC with a 2.8 GHz processor. Figures 5.5 and 5.6 show the execution times for the Intel Research Lab and Sieg Hall dataset respectively. It is clear that farthest-point sampling is, by far, the slowest algorithm. For clarity, the figures also show the execution times of the other three algorithms in detail. It can be observe that the grid-based sampling algorithm is the fastest one. Its execution time is almost constant. In contrast occupancy-grid sampling has the overhead of the ray-

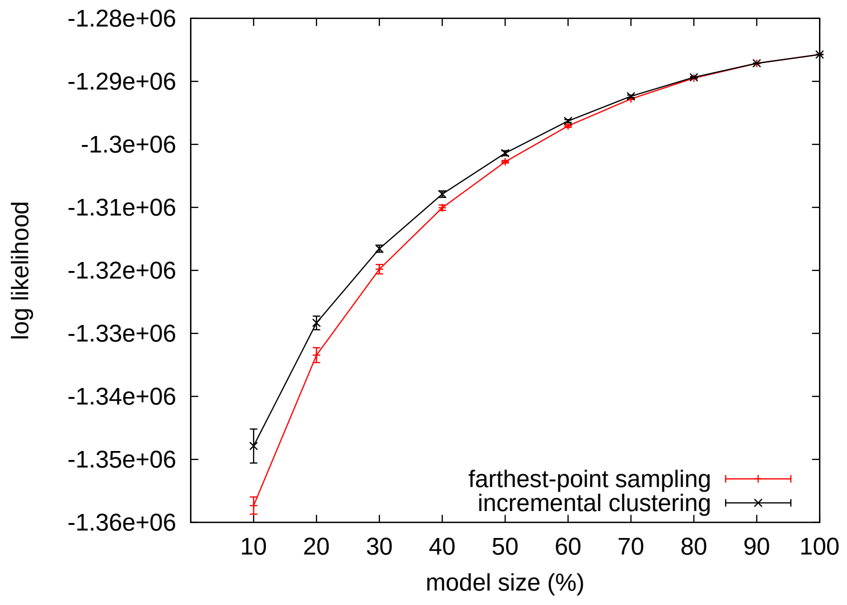


(a) Intel Research Lab

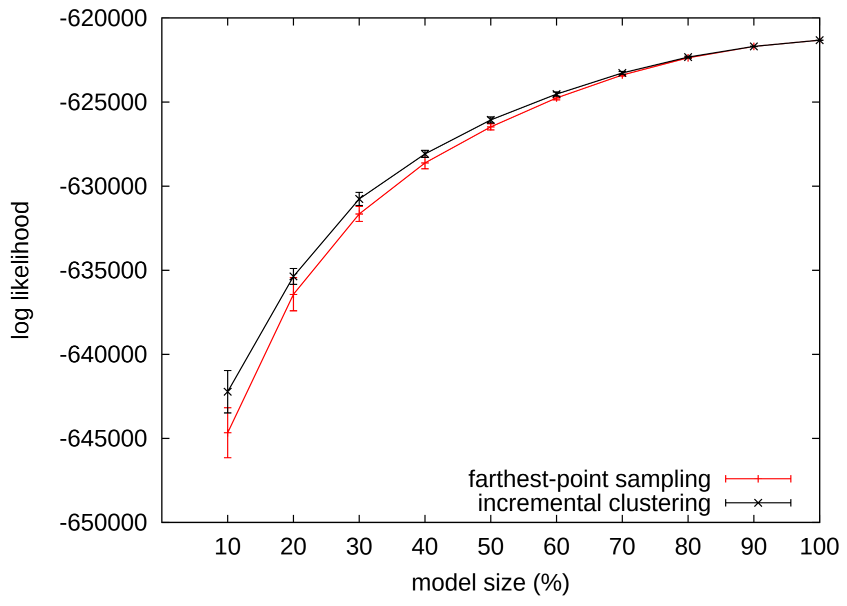


(b) Sieg Hall

Figure 5.3.: Likelihood of the models obtained for the Intel Research Lab and Sieg Hall datasets. For each dataset, models of four different sizes were compared. The models were generated using the different simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). The size of the compared models is indicated as number of points and as percentage of the complete dataset.



(a) Intel Research Lab



(b) Sieg Hall

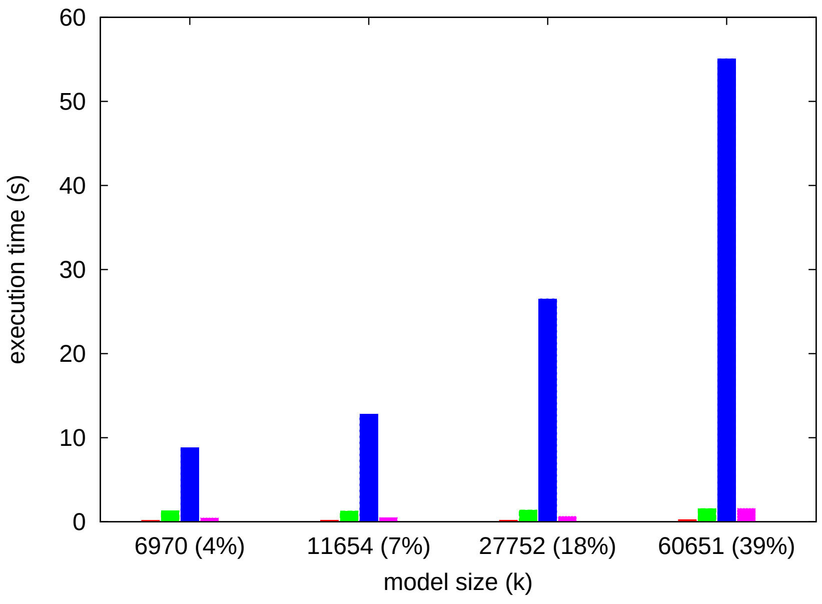
Figure 5.4.: Likelihood of the models obtained with farthest-point sampling (fps) and incremental clustering (ic) as a function of the size of the model. The standard deviation shown on the figure was augmented by a factor of 10 for displaying purposes.

casting operation which is clearly affected by the resolution of the grid. The execution times of the incremental clustering algorithm depend both on the size of the complete dataset and the size of the resulting model. The results of this experiment supports the theoretical analysis about the time complexity of the dataset simplification algorithms presented in Chapter 3. As a reminder let n and k be the size of the dataset and the model respectively. The farthest-point algorithm has a time complexity of $O(kn \log k)$, incremental clustering is $O(k \log n)$, and grid-based sampling and occupancy grid sampling are both $O(n)$ although occupancy grid sampling has the additional overhead of the ray-casting operation. Remember that this complexity analysis was very rough since the actual execution times depend greatly on the distribution of the samples in the dataset and on the underlying structure of the environment.

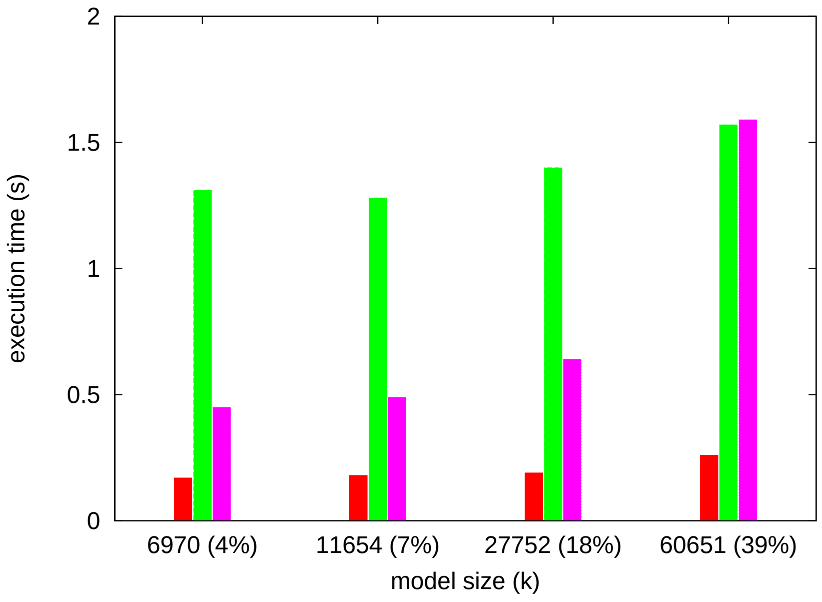
5.2. Evaluation of the k-Means Algorithm

The second experiment is designed to show the effect of the k-means algorithm for improving the quality of a given model. We use the models obtained using the different dataset simplification techniques as starting point for the k-means algorithm. Figure 5.7(b) shows the likelihood of the models obtained using the k-means algorithm together with the likelihood of the models used as starting points. As can be seen on the figure, the k-means algorithm actually improves the quality of the starting models increasing its likelihood. The figure also shows the effect of the starting model on the quality of the results. For a fixed model size, different starting models produce different results. Another observation is that the improvement for lower likelihood models is in general larger than for models with higher likelihoods. This can be seen in Figure 5.8 which compares the likelihood of the models obtained using the k-means algorithm with the likelihood of the models used as starting point in function of the size of the model. Observe how as the size of the model increases the difference between the likelihood of the original model and the improved one decreases. This is a property of the likelihood function and is independent of the strategy used to generate the model. The difference in the likelihood between large models is small compared with the difference in the likelihood between smaller models independent of how the models are generated.

How much a model is improved is related to the number iterations of the algorithm. Larger improvements are in general associated to larger number of iterations. However, the amount of improvement is also affected by the distribution of the samples in the initial model. For example, the k-means algorithm converged after 54 iterations when starting with the model of size 6970 obtained using grid-based sampling for the Intel Research Lab dataset. For the same dataset and model size, k-means converged after 54 iterations too when starting with the model obtained using farthest-point sampling. In the first case the improvement in the likelihood of the model was of 23450. In the second case the improvement was of 49487, more than twice as much as in the first case. These number have no absolut meaning but show that the improvement can not be uniquely characterized by the number of iterations required for convergence. Tables 5.1 and 5.2 show the number of iterations needed by the k-means algorithm to converge for the Intel Research Lab and Sieg Hall datasets respectively. The

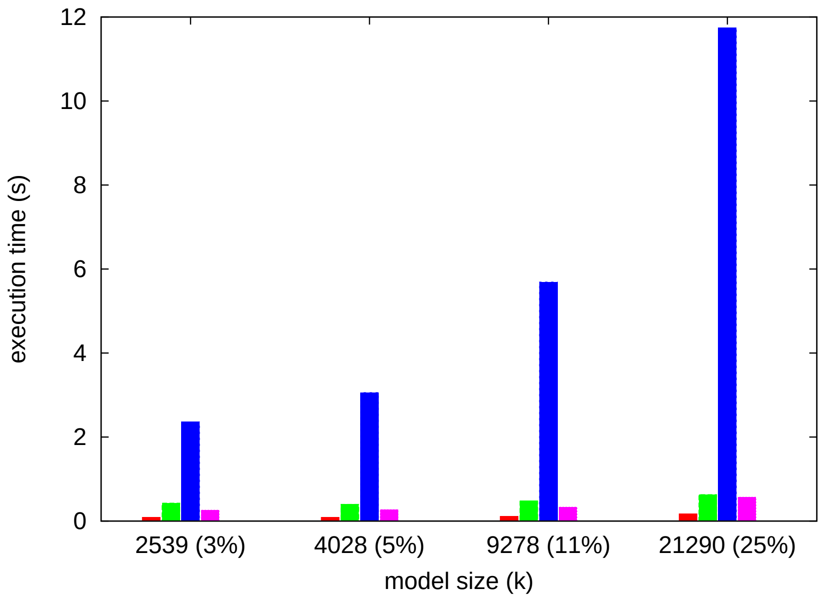


(a)

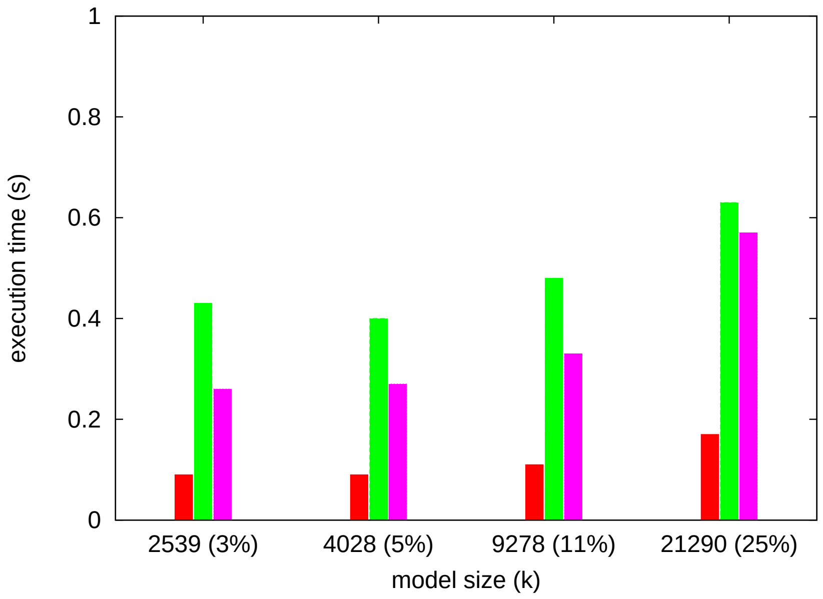


(b)

Figure 5.5.: Execution times for the Intel Research Lab dataset on a standard PC with a 2.8 GHz processor. The execution times correspond to the 4 simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). For clarity Figure 5.5(b) does not show the times for the farthest-points sampling algorithm.

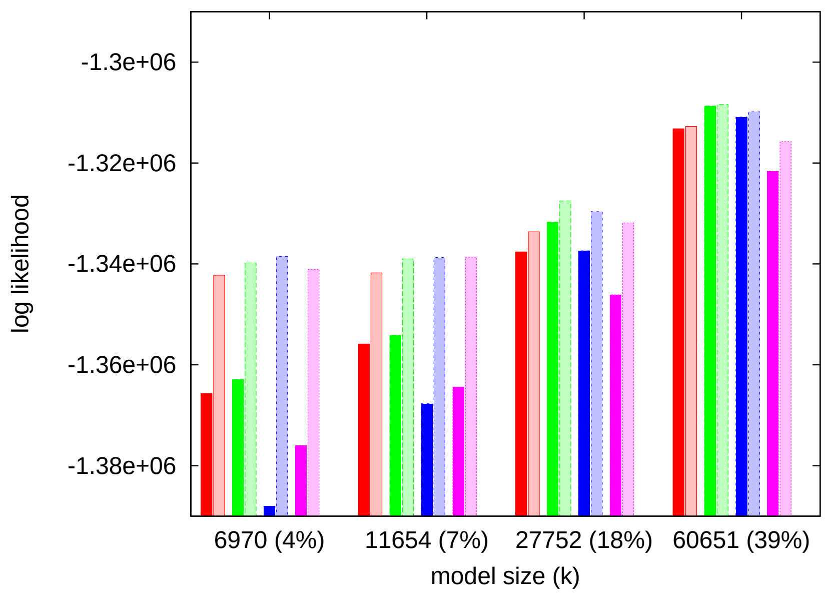


(a)

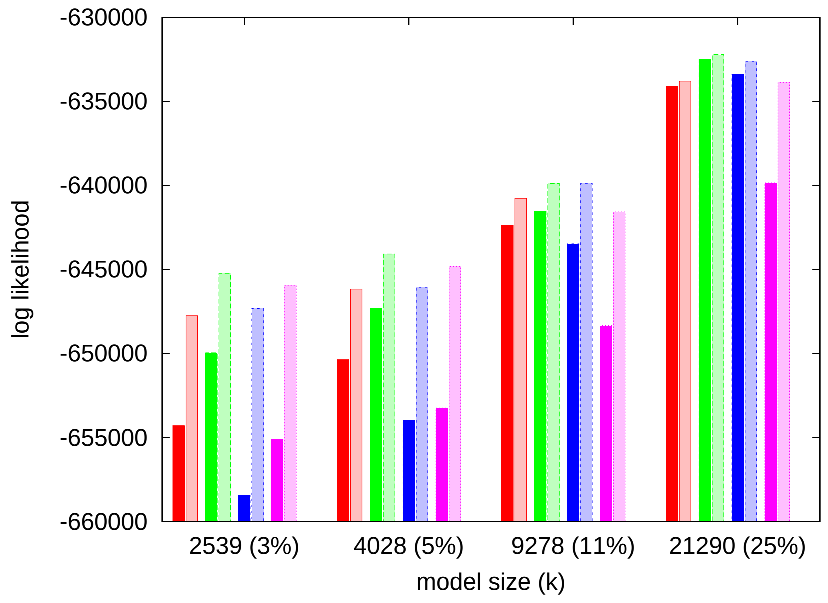


(b)

Figure 5.6.: Execution times for the Sieg Hall dataset corresponding to the 4 simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). For clarity Figure 5.6(b) does not show the times for the farthest-points sampling algorithm.

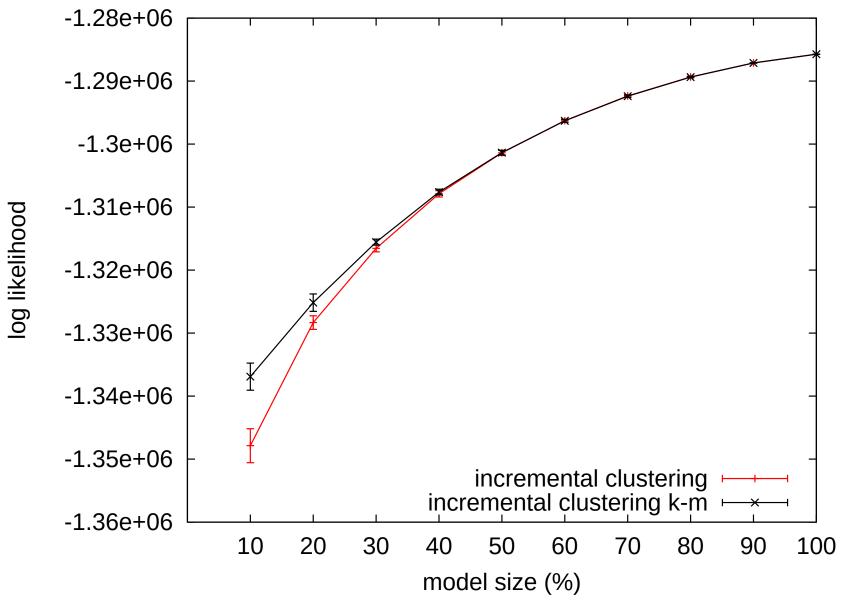


(a) Intel Research Lab

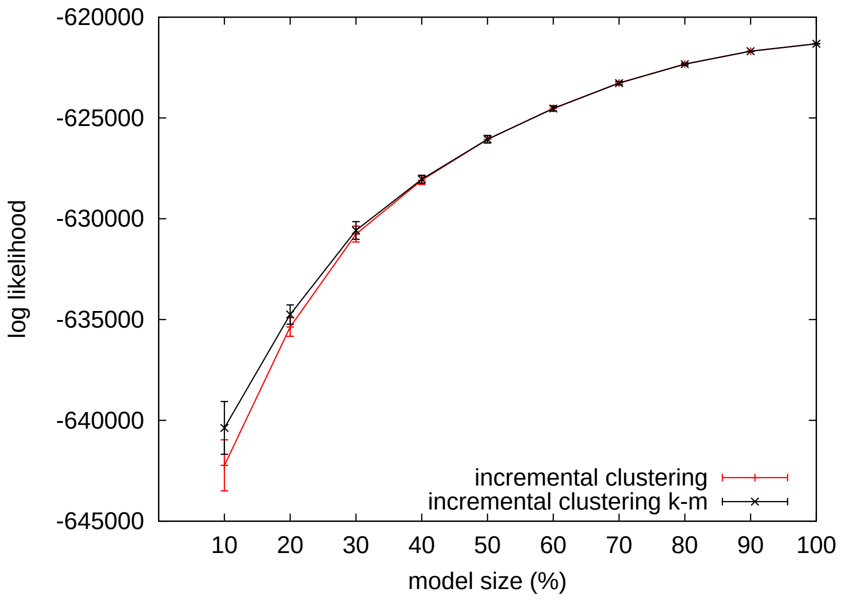


(b) Sieg Hall

Figure 5.7.: Likelihood of the models obtained using the k-means (k-m) algorithm for different starting models. Also shown are the likelihood of the models used as starting point generated with the different simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).



(a) Intel Research Lab



(b) Sieg Hall

Figure 5.8.: Likelihood of the models obtained using k-means (k-m) as a function of the size of the model. The models used as starting point where obtained using Incremental Clustering (ic).

Initial Model	6970 (4%)	11654 (7%)	27752 (18%)	60651 (39%)
gbs	54	37	24	17
ic	55	34	21	11
fps	54	35	22	13
ogs	90	41	26	26

Table 5.1.: Number of iterations needed by the k-means algorithm for the Intel Research Lab dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	2539 (3%)	4028 (5%)	9278 (11%)	21290 (25%)
gbs	72	48	29	12
ic	39	30	20	10
fps	59	38	22	14
ogs	70	42	30	18

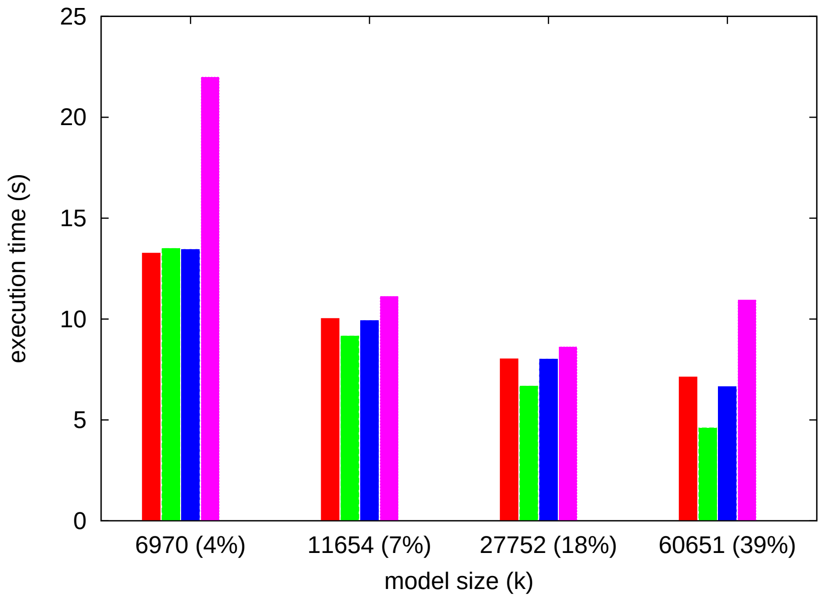
Table 5.2.: Number of iterations needed by the k-means algorithm for the Sieg Hall dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

increase in the likelihood of the models associated to these iteration values can be clearly observe in Figure 5.7.

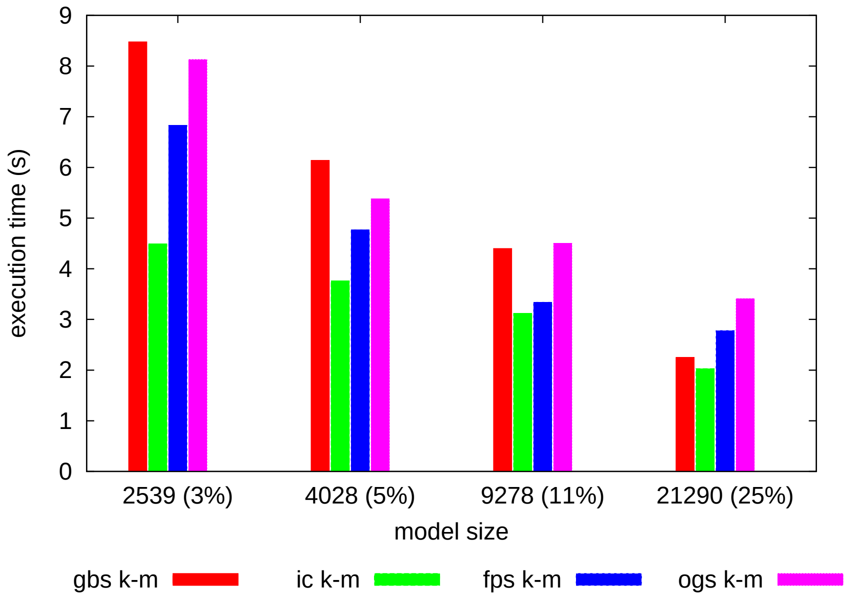
The number of iterations also affects the execution times of the algorithm. Recall from Chapter 4 that the k-means algorithm has a time complexity of $O(Tn \log k)$ where n is the number of samples in the dataset, k is the size of the model, and T is the number of iterations needed for the algorithm for convergence. Figure 5.9 shows the execution times of the k-means algorithm for the Intel Research Lab and Sieg Hall datasets. The effect of the number of iterations (Tables 5.1 and 5.2) can be clearly seen on the figure. For a given dataset and model size the highest execution times correspond to the runs that required the largest number of iterations.

5.3. Evaluation of the Fuzzy k-Means Algorithm

The third experiment is designed to demonstrate the capability of the fuzzy k-means algorithm to improve the likelihood of a given model. To evaluate the results we use as staring point the models obtained using the different dataset simplification techniques. Figure 5.10 shows the



(a) Intel Research Lab



(b) Sieg Hall

Figure 5.9.: Execution times of the k-means (k-m) algorithm on a standard PC with a 2.8 GHz processor for the starting models produced with the dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	6970 (4%)	11654 (7%)	27752 (18%)	60651 (39%)
gbs	18	24	20	12
ic	21	22	19	15
fps	20	24	19	16
ogs	27	20	34	21

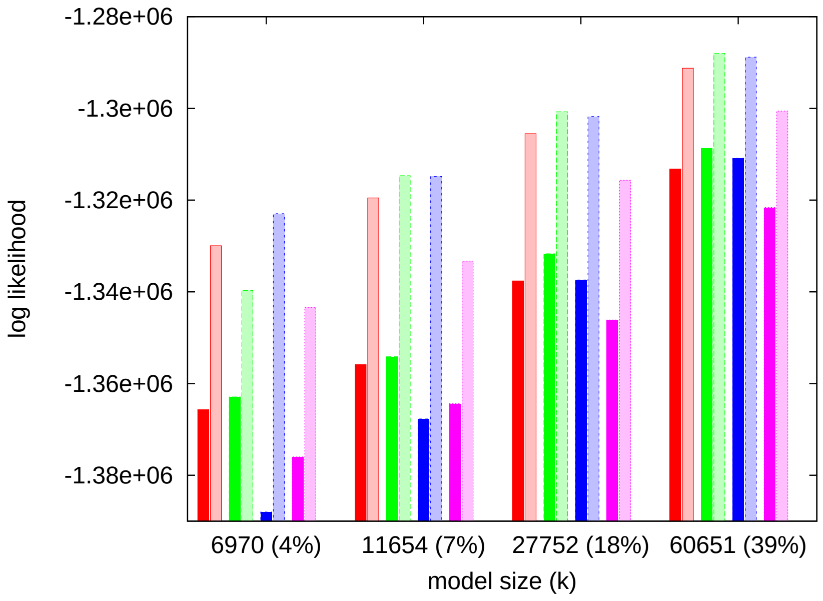
Table 5.3.: Number of iterations needed by the fuzzy k-means algorithm for the Intel Research Lab dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

likelihood of the models obtained using the fuzzy k-means algorithm. Also shown are the likelihood of the models used as starting point for the algorithm. As can be seen in the figures, using the fuzzy k-means algorithm the likelihood of the given models can be improved. It can also be seen, how the model used as starting point affects the results of the algorithm. Figures 5.11 compares the likelihood of the models obtained using the k-means algorithm with the likelihood of the models obtained using the fuzzy k-means algorithm. The same starting models were used for both algorithms. Observe that in most of the cases fuzzy k-means produces better results than the k-means algorithm. However, the models produced using k-means can be better than the ones produced with fuzzy k-means. In our specific case this can be observed when occupancy grid sampling is used to generate the starting models. This reason for this is that outliers in the original dataset have only a very small influence on the models that fuzzy k-means produces. This negatively affects on the likelihood of the resulting models. On the other hand, the k-means algorithm may include in the resulting model, samples that where previously removed by the occupancy grid sampling algorithm.

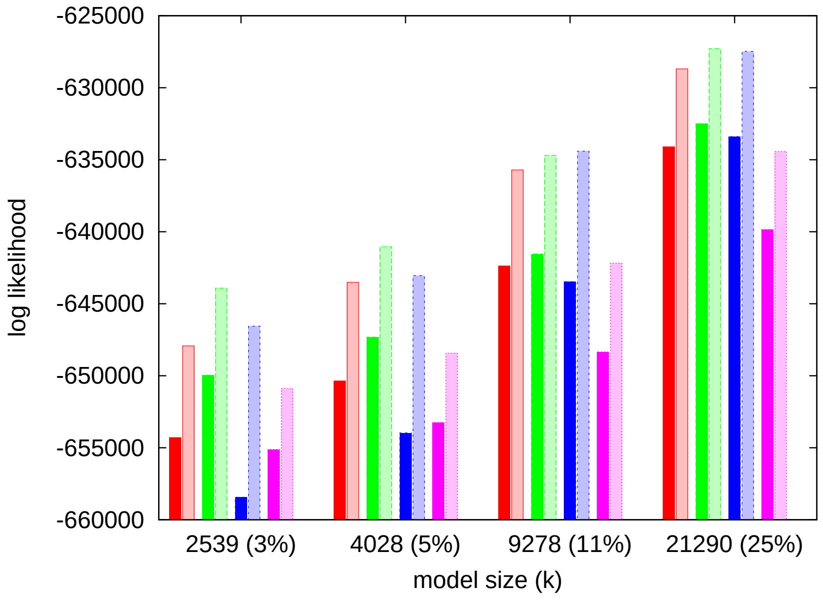
The execution times of the fuzzy k-means algorithm range from almost a minute for small datasets to well over one hour for larger ones. The time complexity of the algorithm is $O(Tkn)$ where T is the number of iterations needed by the algorithm, n the number of samples in the dataset, and k the size of the model. The number of iterations is related to the increase in the likelihood of the starting model. Larger increments in likelihood are associated to larger number of iterations. Tables 5.3 and 5.5 show the number of iterations needed by the fuzzy k-means algorithm for the Intel Research Lab and Sieg Hall datasets. Tables 5.4 and 5.6 show the corresponding execution times.

5.4. Likelihood Optimization by Simulate Annealing

This experiment is designed to illustrate how simulated annealing can be used to improve the likelihood of a model. In particular we show that simulated annealing can even improve models where the k-mean and fuzzy k-mean algorithms fail. Both these algorithms get stock

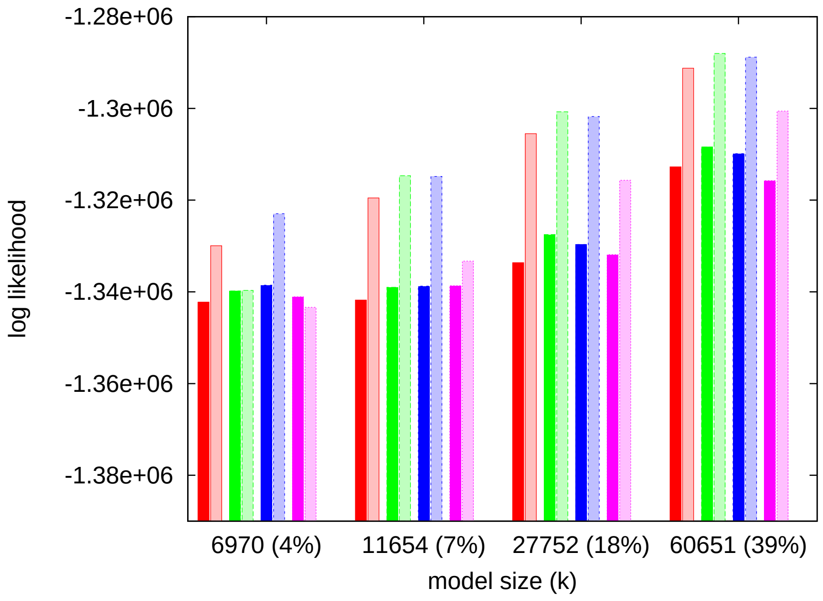


(a) Intel Research Lab

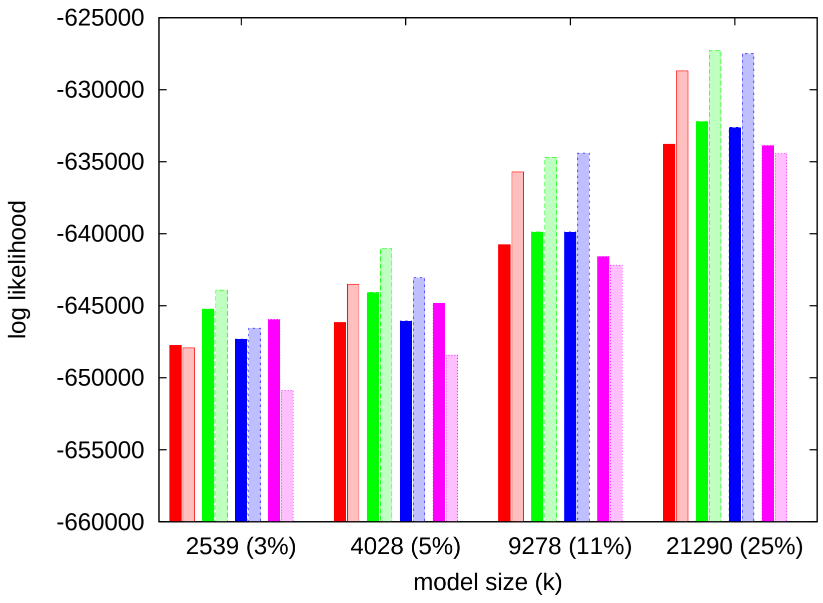


(b) Sieg Hall

Figure 5.10.: Likelihood of the models obtained using the fuzzy k-means (f) algorithm. Also shown are the likelihood of the models obtained using the different simplification techniques used as starting models: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).



(a) Intel Research Lab



(b) Sieg Hall

Figure 5.11.: Likelihood of the models obtained using the fuzzy k-means (f) algorithm compared with the likelihood of the models obtained using the k-means algorithm for the same starting models. The starting models where generated using the dataset simplification algorithms: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	6970 (4%)	11654 (7%)	27752 (18%)	60651 (39%)
gbs	7.47	17.08	35.69	80.46
ic	9.08	14.35	30.78	67.72
fps	8.41	17.22	34.91	68.36
ogs	10.76	13.13	54.80	78.38

Table 5.4.: Execution times in minutes of the fuzzy k-means for the Intel Research dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	2539 (3%)	4028 (5%)	9278 (11%)	21290 (25%)
gbs	15	16	29	12
ic	19	17	16	12
fps	17	17	13	13
ogs	20	16	25	16

Table 5.5.: Number of iterations needed by the fuzzy k-means algorithm for the Sieg Hall dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	2539 (3%)	4028 (5%)	9278 (11%)	21290 (25%)
gbs	1.60	2.72	11.66	11.68
ic	1.89	2.72	6.00	11.52
fps	1.76	2.90	4.93	12.68
ogs	2.04	2.63	9.33	14.61

Table 5.6.: Execution times in minutes of the fuzzy k-means for the Sieg Hall dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

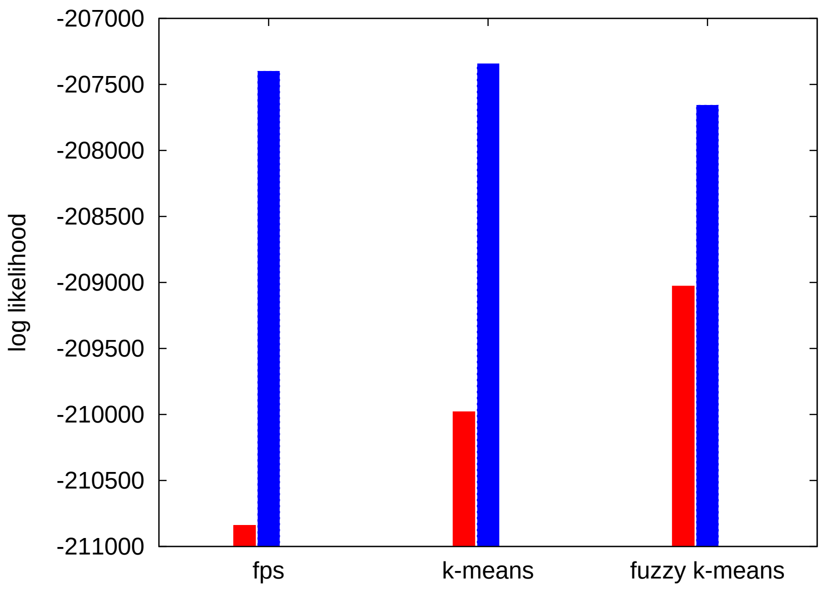


Figure 5.12.: Likelihood of the models obtained using simulated annealing (sa) for three different initial models after 110000 iterations. The first model (fps) was obtained sampling 4500 points from the Sieg Hall dataset using farthest-point sampling. The second (k-means) and third (fuzzy k-means) models were obtained with the k-means and fuzzy k-means algorithms respectively using the first model (fps) as starting initial model.

in models that constitute local optima in the likelihood space. Repeated applications of the algorithms do not provide further improvements of these models. To perform this experiment we selected an initial model and applied k-means, fuzzy k-means and simulated annealing to it. We then applied simulated annealing to the models obtained using k-means and fuzzy k-means. In order to set the initial temperature for each specific run, we ran the simulated annealing algorithm using the different starting models and observed the average difference in the likelihood during the first iterations. This value was used as initial temperature. Thus lower likelihood models would have in average a probability of approximately 37% of being accepted before the first decrease in the temperature. We set the annealing schedule as $t(i+1) = \alpha t(i)$ with $\alpha = 0.95$ and allowed 4500 iterations (the size of the model) before decreasing the temperature. The final temperature was set to 0.05. Figure 5.12 shows the likelihood of the models obtained after 110000 iterations together with the likelihood of the models that were used as starting point. As expected from the discussion in the previous experiments, k-means and fuzzy k-means improved the likelihood of the original model. In the figure it can be observed that the model used as starting point does affect much the results of the simulated annealing algorithm. Figure 5.13 shows the behavior of the likelihood values for the different initial models. The figure shows how the algorithm sometimes chooses

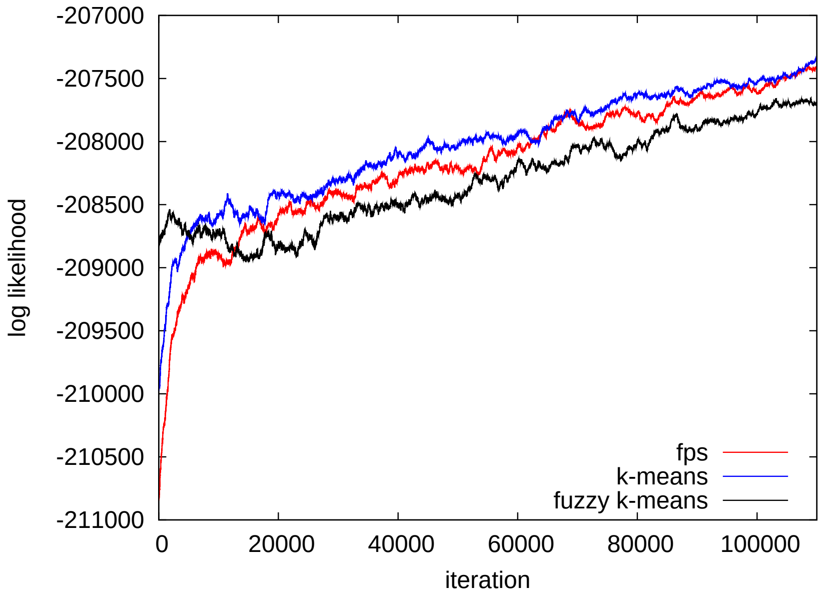
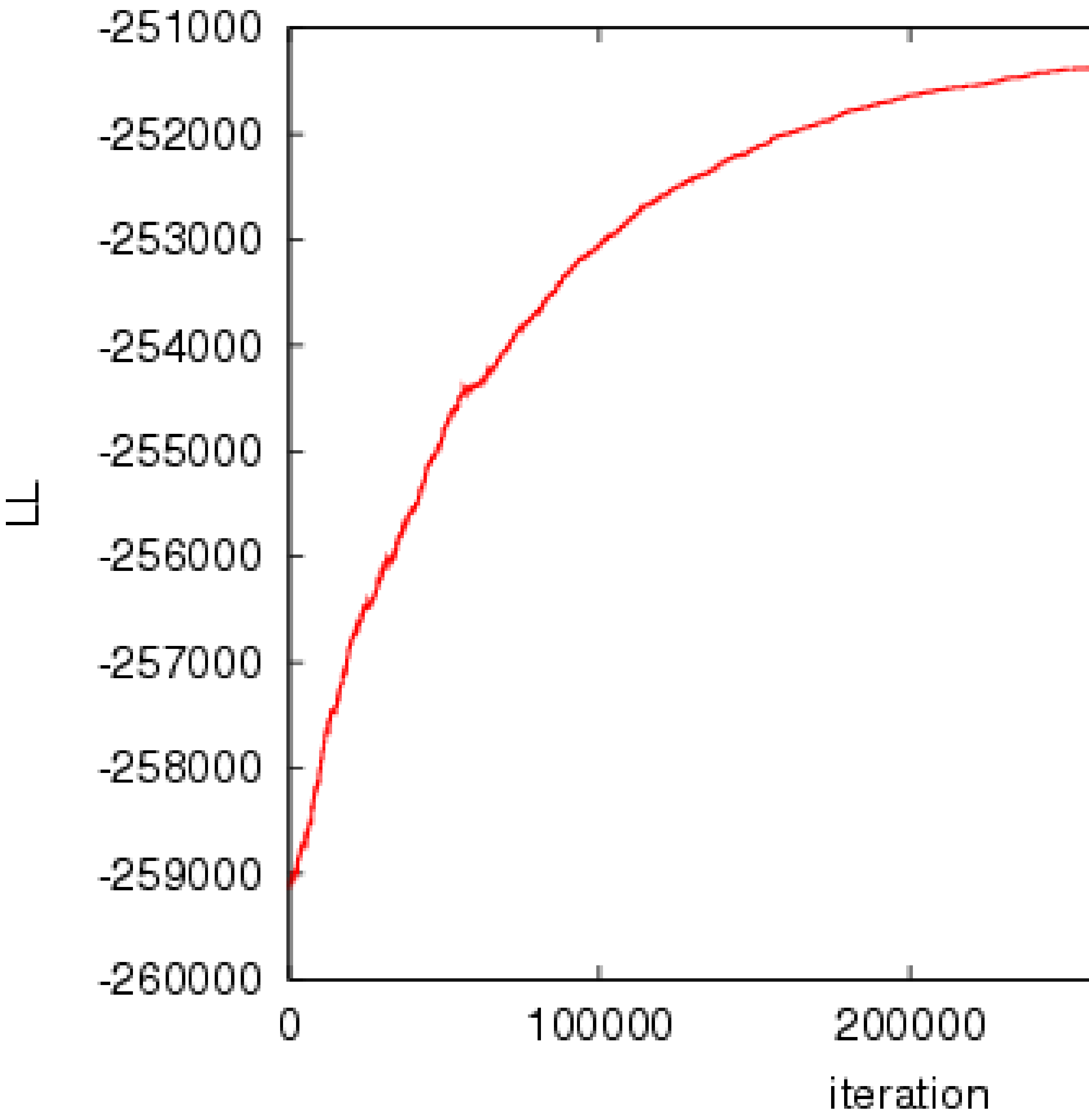


Figure 5.13.: Likelihood for three different initial models as the simulated annealing algorithm progresses. The final values together with their corresponding starting models are shown in Figure 5.12.

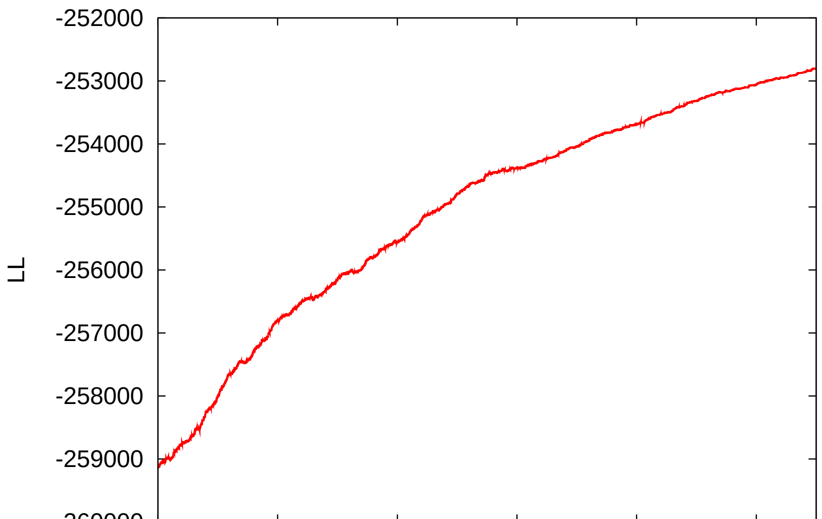
lower likelihood models. Observe how the curve goes down on the initial iterations when using as starting model the result of the fuzzy k-means algorithm. Given the specific annealing schedules used in the experiment we can calculate the number of iterations required by the algorithm to terminate for each case. Using formula (4.24) we obtain that the algorithm needs, for the best case 382500 iterations to terminate. Observe that in Figure 5.13 only the first 110000 iterations are shown, a little over the 30% of the whole execution. Each iteration of the algorithm requires approximately 2 seconds, thus the complete execution time would have being approximately 212.5 hours or almost 9 days. Figure 5.14(a) shows the behavior of the likelihood values for a completed run of the algorithm using a different initial model and greedier annealing schedule in which the initial temperature was set so that lower likelihood models would have in average a probability of approximately a 1% of being accepted before the first decrease in the temperature. Figure 5.14(b) shows only the first iterations of the algorithm. The figure shows the same number of iterations as the ones shown in Figure 5.13 for comparison. Observe how by using a greedier approach the curve does not go down as much.

5.5. Memory Requirements of the Representation

In this final experiment, we compare the memory requirements of our sample-based representation against occupancy grid maps [Elfes, 1989]. These maps divide the environment into

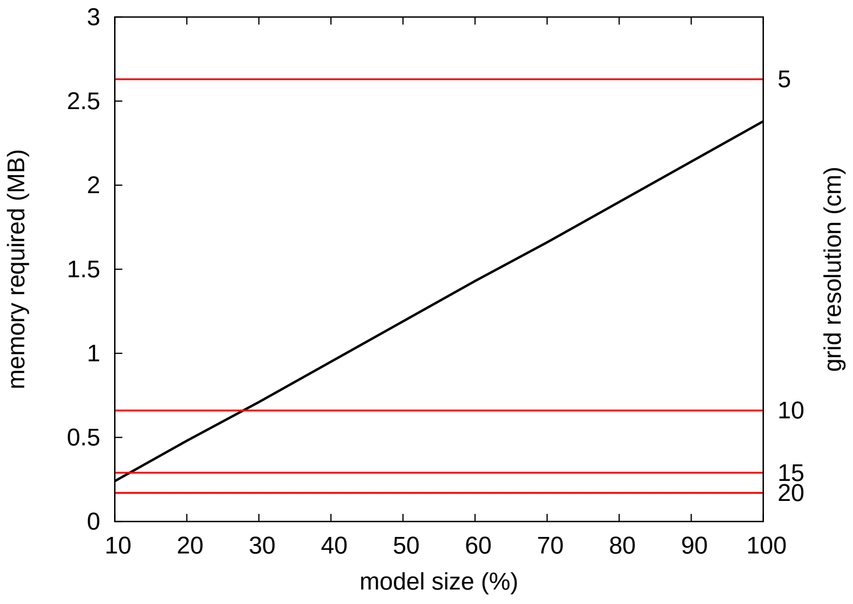

-251000
-252000
-253000
-254000
-255000
-256000
-257000
-258000
-259000
-260000
LL
0
100000
200000
iteration

(a)

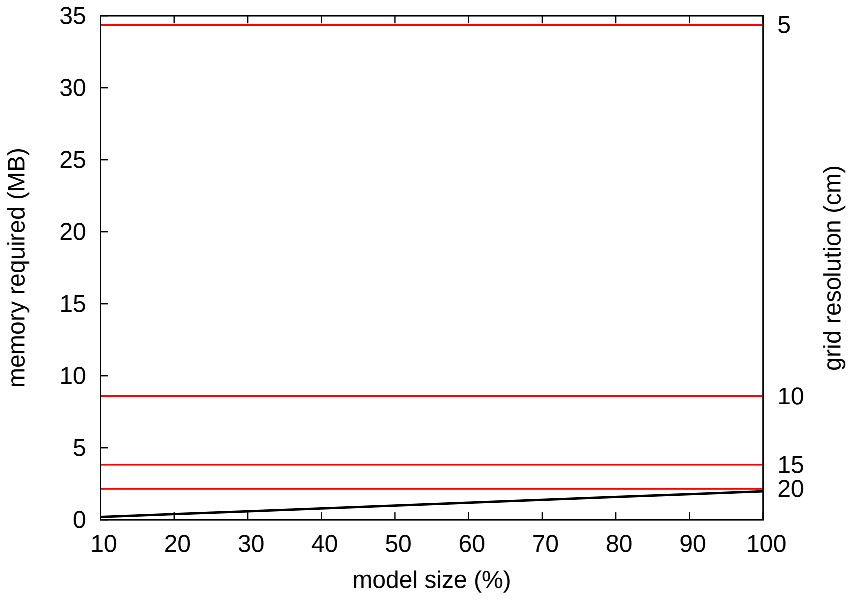

-252000
-253000
-254000
-255000
-256000
-257000
-258000
-259000
LL

cells of equal size. Each cell represents the are of the environment it covers and contains information about the occupancy probability for that area. In this experiment, the concrete implementation for the gird is based on the *counting model* [Hähnel *et al.*, 2003]. For each cell (i, j) we must store information about the number $hits_{i,j}$ of times a range measurement ended up in (i, j), and the number $misses_{i,j}$ of times a range measurement passed through (i, j) without being reflected. Thus, for each cell 2×4 bytes of memory are needed. For each point in a sample-based map we need to store its x and y coordinates. We use double precision for this in order to obtain accurate maps. Thus, for each point 2×8 bytes of memory are needed.

Sample-based maps are not affected by the size of the environment. Only the number of samples affect the memory requirements of sample-based maps. Occupancy grid maps, on the other hand, are not affected by the number of samples but by the size of the environment. Figures 5.15(a) and 5.15(b) show the memory required by a sample-based map as a function of the number of samples in the map for the Intel Research Lab and Bruceton Mine datasets. The figures also show the memory required by an occupancy grid map for 4 different grid resolutions. The Intel Research Lab is 29m$\times$29m and the corresponding dataset has 155648 samples. The Bruceton Mine is 153m$\times$73m and the corresponding dataset has 129936 samples. As can be seen in Figure 5.15(b) the size of the environment has a great impact on the memory requirements for occupancy grid maps. In Figure 5.15(a) can be observe that for some resolutions, an occupancy grid map requires less memory than a sample-based map in certain environments. However, the memory requirements of grid-based representations grow exponentially with the size of the environment, while they grow linearly with the number of samples for sample-based maps.



(a) Intel Research Lab



(b) Bruceton Mine

Figure 5.15.: The sloped line represents the memory required in megabytes (MB) by a sample-based representation in function of the number of samples in the map. Also shown is the memory required by an occupancy grid representation for 4 different grid resolutions.

6. Conclusion

A model of the environment is an essential requirement for many robotic applications. In particular, mobile robots rely on these models or maps to perform fundamental tasks like localization, path planing, and exploration. The applications of a robot that can not perform these tasks are very limited and of little practical use. It is therefore of great importance to generate accurate maps of the environment.

In this thesis we presented sample-based maps as a efficient spatial description of the environment that uses points to represent the objects detected by the robot with its sensors. Such maps can be very accurate, and are general in the sense that they are not limited to a specific type of environment. By using points as primitives for the representation these maps do not impose any structure to the environment that is being represented. Additionally, by using only points that correspond to actual measurements made by the robot, sample-based maps constitute a representation that is consistent with the data.

We presented several approaches for generating sample-based maps from the measurements gathered by the robot. In Chapter 3 we described various techniques for reducing the number of points in the representation. The first strategy divides the space into cells of equal size replacing all the points that fall within a cell with a common representative. This leads to a very fast but crude simplification of the dataset. We then extended this technique and associated to each cell a probability of being occupied. By discarding cells of low probability we effectively eliminate points generated by spurious measurements caused by noise in the sensor or by dynamic objects in the environment. We also describe an approach that builds clusters in an incremental way by repeatedly adding points to the clusters until they reach a maximum radius. This divides the dataset into a set of partitions of a specified maximum size each of which is then replaced by one representative point. The last dataset simplification technique presented repeatedly adds to the model the farthest point in the dataset. This is the point that has the greatest distance to all the points already in the model. In this way, we add to the model the point that reduces the most the spaces between the points already in the model. None of this methods attempts to find an optimal model according to a global evaluation criterion. Each technique has its own approximative heuristic. The grid-based and the incremental clustering approaches try to partition the samples according to regions, assuming that each region describes an equally important part of the environment. The farthest-point sampling approach tries to minimize the sum of squared errors incurred in representing the dataset using the points selected in the model.

The problem of generating a sample-based map can be stated as the problem of finding the best subset of points for a given dataset according to some evaluation criterion. By considering the set of points as a probabilistic model the likelihood of the data can be used as evaluation function for the models. Thus our problem can be restated as the problem of finding a subset that maximizes the likelihood of the data. Since this problem is in practice intractable, we presented in Chapter 4 three approximative optimization techniques for generating a maximum-likelihood subset. We first described the k-means and fuzzy k-means algorithms. Both of this algorithms take a given initial model and improved it iteratively by recomputing the value of its points. The k-means algorithm associates each point in the dataset to its closest point in the model and then recomputes the value of the points in the model based on all its associated points. The fuzzy k-means algorithm is a more general version of the k-means algorithm. It associates each point in the dataset to all the points in the model. These associations are weighted according to the distance between the two points. In each iteration the points in the model are recomputed as the weighted average of all the points in the dataset. The improvements obtained using fuzzy k-means proved to be, in general, better than the ones obtained with the k-means algorithm. Both of this algorithms are greedy procedures that maximize the improvement in the likelihood of a model on each iteration. These greedy approaches, however, tend to get stock in local maxima producing models that are not necessarily overall bests. To alleviate this problem, we described the simulated annealing algorithm as a stochastic approach for solving the optimization problem. Simulated annealing takes an initial model and starts evaluating random modifications accepting or rejecting them according to some probability. Since the algorithm accepts lower-likelihood models with a positive probability is able to scape local maxima, producing in general better results than the k-means and fuzzy k-means algorithms.

All the techniques presented in this thesis were implemented and evaluated using data gathered with real robots in different environments. We used the likelihood of the data as a measure of the quality of the models. Since larger models have in general a higher likelihood, we compared models of the same size to evaluate our algorithms. We first compared the different dataset simplification techniques in function of the quality of the resulting models. Our experiments showed that the incremental clustering algorithm produced the bests models. Since the clusters do not impose a rigid geometrical structure on the distribution of the points the resulting model resembles the underlying distribution of the points in a more natural way. Farthest-points sampling produces models whose points lie far apart from each other. For large simplification rates such a distribution of points produces models of relatively low likelihood. By using a fixed size partitioning of the space, the grid-based approach can not adapt to the underlying point distribution producing models that are slightly worse than the ones obtained using incremental clustering. Finally, by adding occupancy probabilities to the cells of the gird to remove spurious points we obtain maps that are more consistent with the environment. But for the likelihood of the data this spurious points are as important as any other point, so the models obtained using this occupancy grid sampling technique have in general low likelihood.

We also evaluated the iterative improvement techniques. For this experiment we used the models obtained with the different simplification techniques as initial models for the k-means, fuzzy k-means, and simulated annealing algorithms. The results showed that using these techniques we can improve the likelihood of the initial models. The experiments also demonstrated that in general the fuzzy k-means algorithm produces better results than those of the k-means algorithm specially for larger models. The models obtained using these two algorithms can be improved no further using these algorithms repeatedly since these results constitute a local maxima. We applied the simulated annealing to those models and demonstrated how this stochastic approach can escape local optima and further improve the models.

Despite the results, the presented techniques have also their limitations. The grid-based simplification techniques need to construct a grid to represent the environment with. Grids for large environments or fine-grained grids require large amounts of memory which can easily exceed the capacity of a standard PC. The execution time is also a disadvantage for some of the techniques. Fuzzy k-means can require more than an hour to produce a result when for the same initial model k-means requires a little more than 10 seconds. The simulated annealing algorithm requires the initial temperature, and annealing schedule to be specified. This has to be done according to the dataset and model size of the specific experiment. There are no parameters that work well for the general case. Additionally, simulated annealing can have extremely large execution times depending on the parameters chosen since it requires many iterations in order to obtain good models.

Representing the environment using sample-based maps has its disadvantages too. Sample-based maps, as most geometrical representations, do not model explicitely free space. Only the points were an obstacle was detected are represented and there is no way to distinguish between free and unexplored spaced. Representing unexplored space is critical for exploration tasks. Sample-based maps are also very susceptible to spurious measurements. This problem can be alleviated using the described occupancy grid sampling technique. This technique, however, can not be applied always since it requires a grid representation of the environment a suffers from the limitations previously mentioned. Another related problem is that the uncertainty in the measurements is not explicitly represented in the map.

Despite this important disadvantages and limitations, the presented techniques can be used to generate accurate, general, and consistent representations of the environment using the measurements gathered with a robot.

A. Datasets

The lines in the figures indicate the trajectory of the robot while gathering the data. The points indicate the positions where the scans were made. The sample density was calculated using a fixed-size grid with a resolution of 5 centimeters per cell. The occupied area was computed summing up the are of all the occupied cells in the grid.

Most of these datasets are freely available online on *The Robotics Data Set Repository* (Radish).

A.1. Intel Research Lab

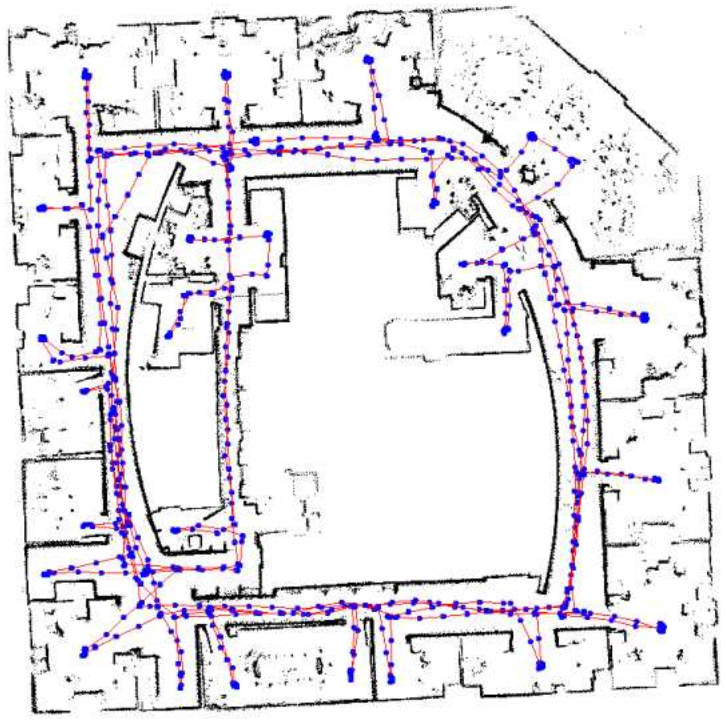


Figure A.1.: Intel Research Lab in Seattle.

- Number of Samples: 155648.

- Number of Scans: 910.
- Measurements per Scans: 180.
- Environment size: 29x29 m.
- Sample density: 1.81 samples/m^2.
- Occupied area: 69.38 m^2.
- Submitted by Dieter Fox to Radish

A.2. Austin ACES

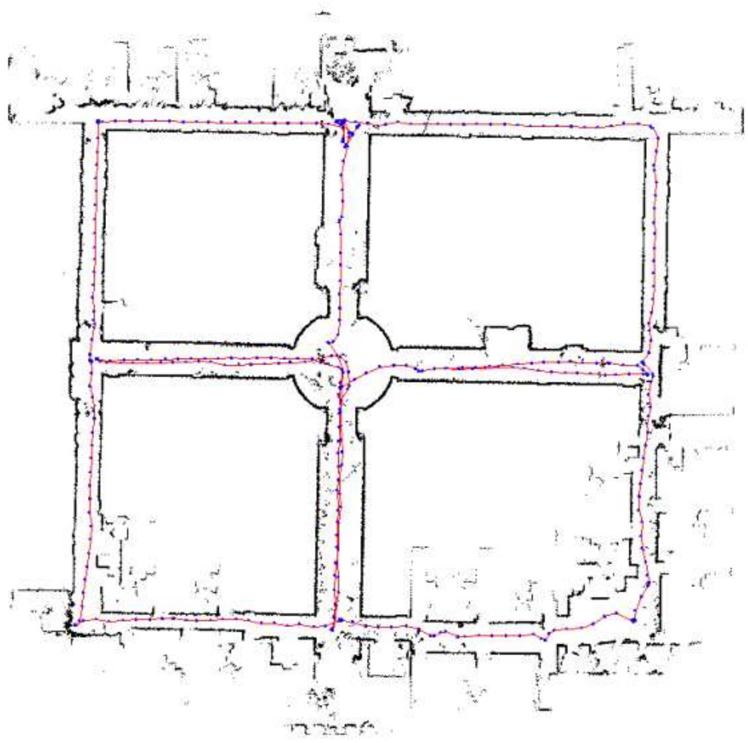


Figure A.2.: ACES building at the University of Texas Austin campus.

- Number of Samples: 73195.
- Number of Scans: 440.
- Measurements per Scans: 180.

- Environment size: 56x55 m.
- Sample density: 0.23 samples/m^2.
- Occupied area: 71.13 m^2.
- Submitted by Patrick Beeson to Radish

A.3. Bruceton Mine

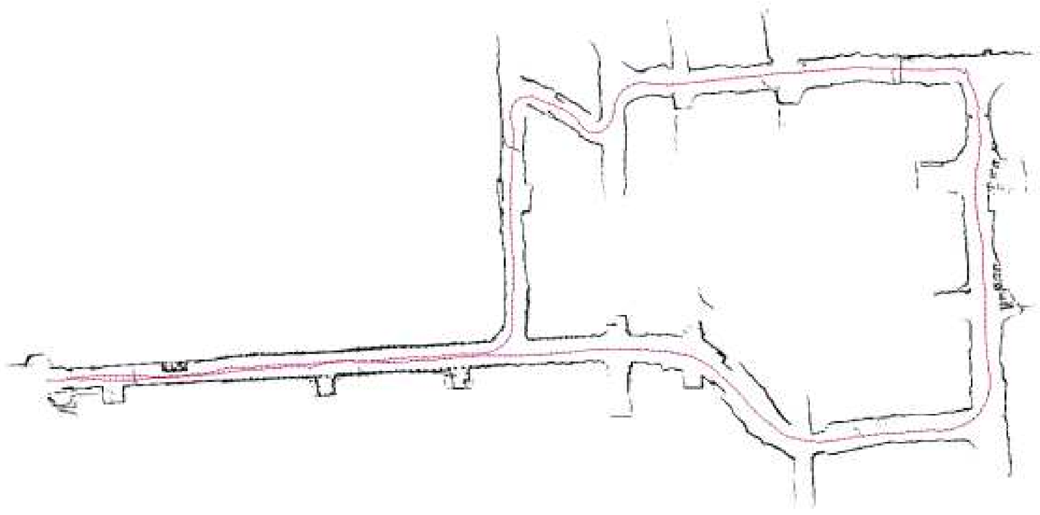


Figure A.3.: Bruceton Research Mine near Pittsburgh.

- Number of Samples: 129936.
- Number of Scans: 415.
- Measurements per Scans: 360.
- Environment size: 153x73 m.
- Sample density: 0.12 samples/m^2.
- Occupied area: 89.23 m^2.
- Provided by Cyrill Stachniss

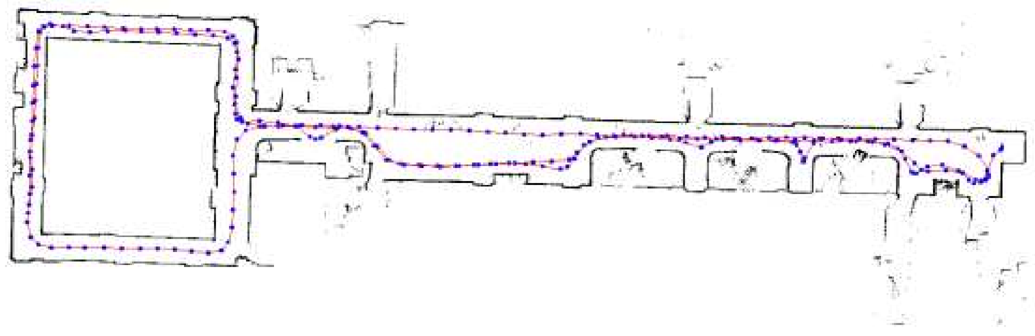
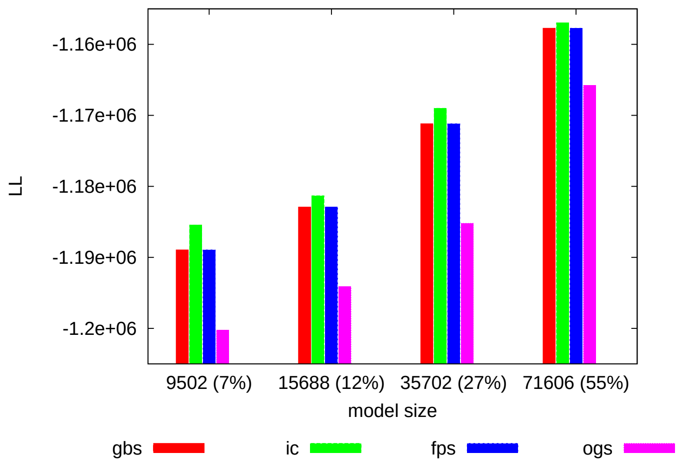


Figure A.4.: Sieg Hall at the University of Washington.

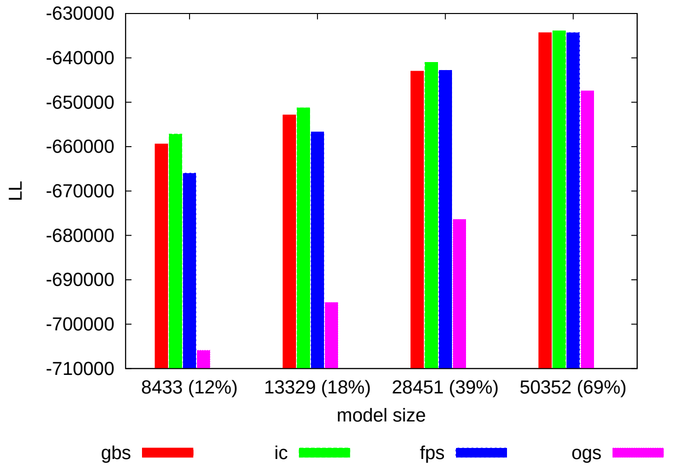
A.4. Sieg Hall

- Number of Samples: 83892.
- Number of Scans: 241.
- Measurements per Scans: 361.
- Environment size: 52x17 m.
- Sample density: 0.93 samples/m^2.
- Occupied area: 23.2 m^2.
- Provided by Cyrill Stachniss

B. Additional Experimental Results

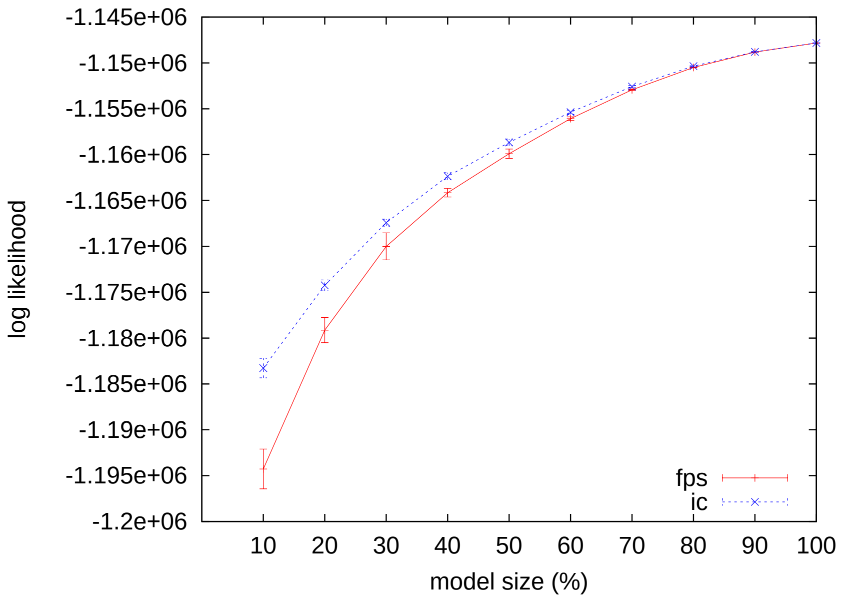


(a) Bruceton Mine

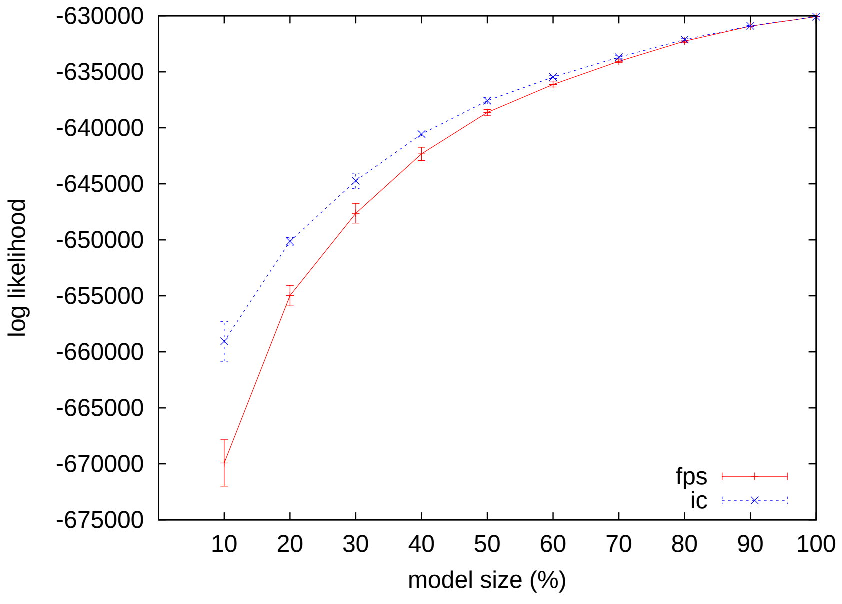


(b) Austin ACES

Figure B.1.: Likelihood of the models obtained using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

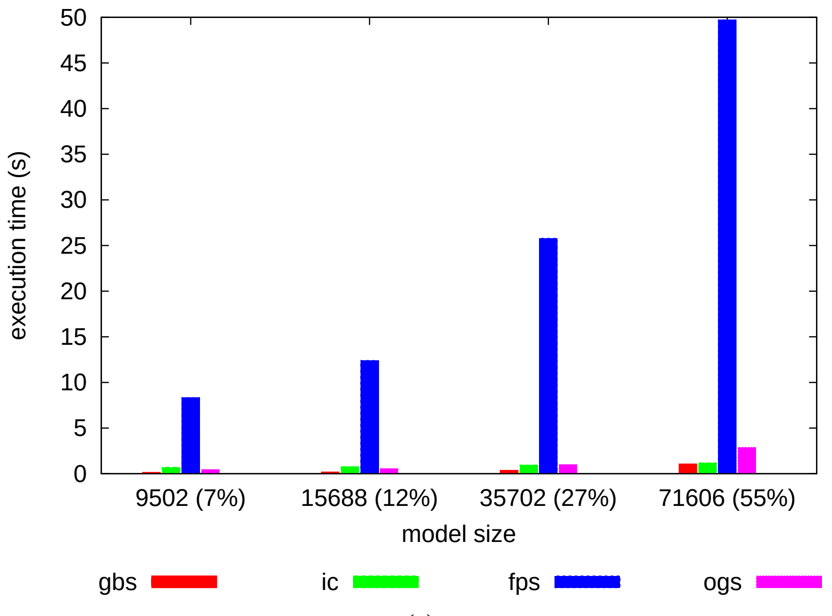


(a) Bruceton Mine

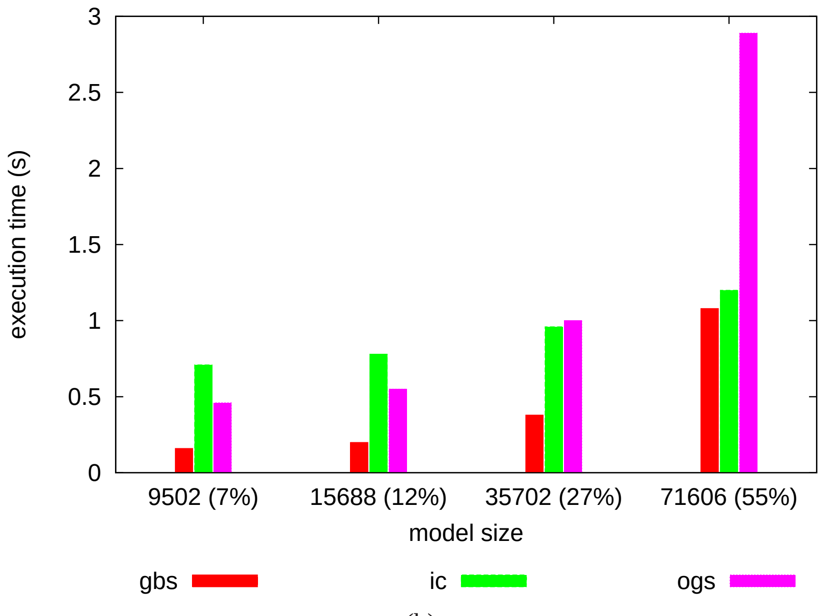


(b) Austin ACES

Figure B.2.: Likelihood of the models obtained with farthest-point sampling (fps) and incremental clustering (ic) as a function of the size of the model. The standard deviation shown on the figure was augmented by a factor of 10 for displaying purposes.

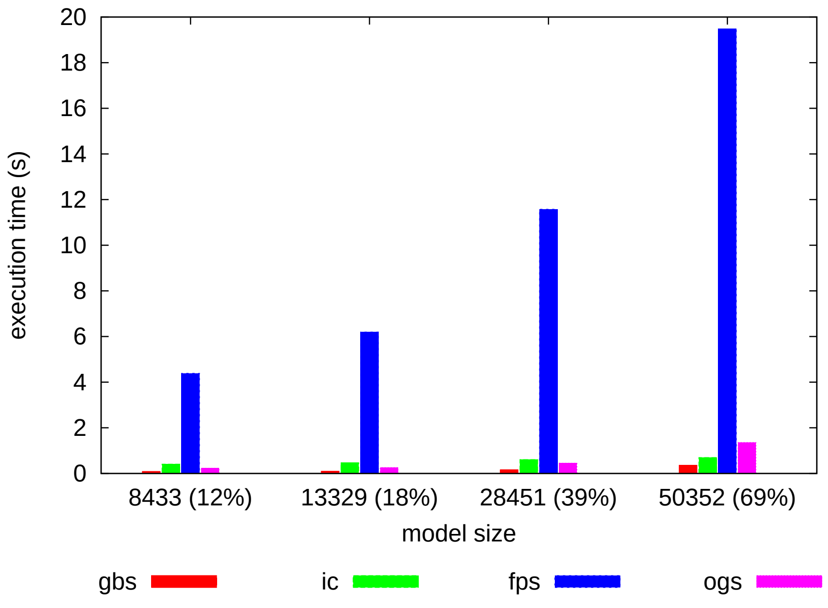


(a)

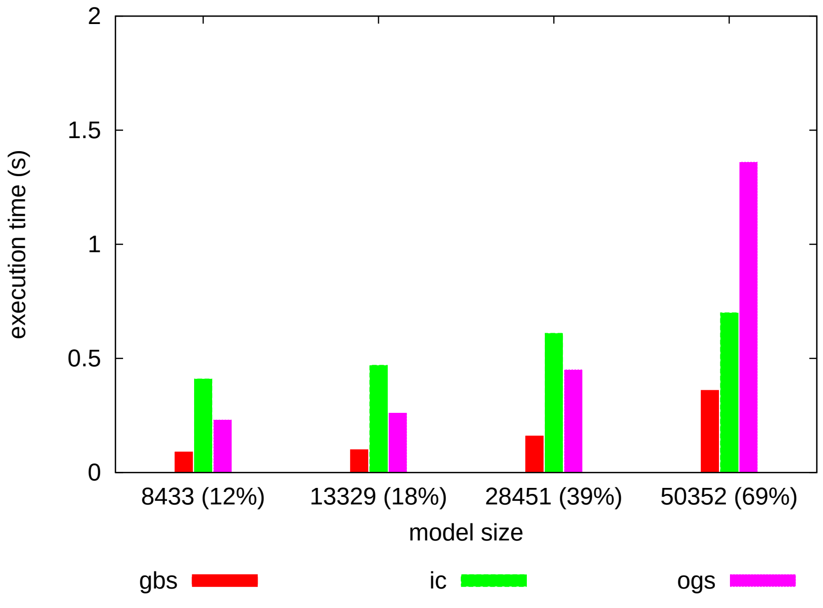


(b)

Figure B.3.: Execution times for the Bruceton Mine dataset on a standard PC with a 2.8 GHz processor. The execution times correspond to the 4 simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). For clarity Figure (b) does not show the times for the farthest-points sampling algorithm.

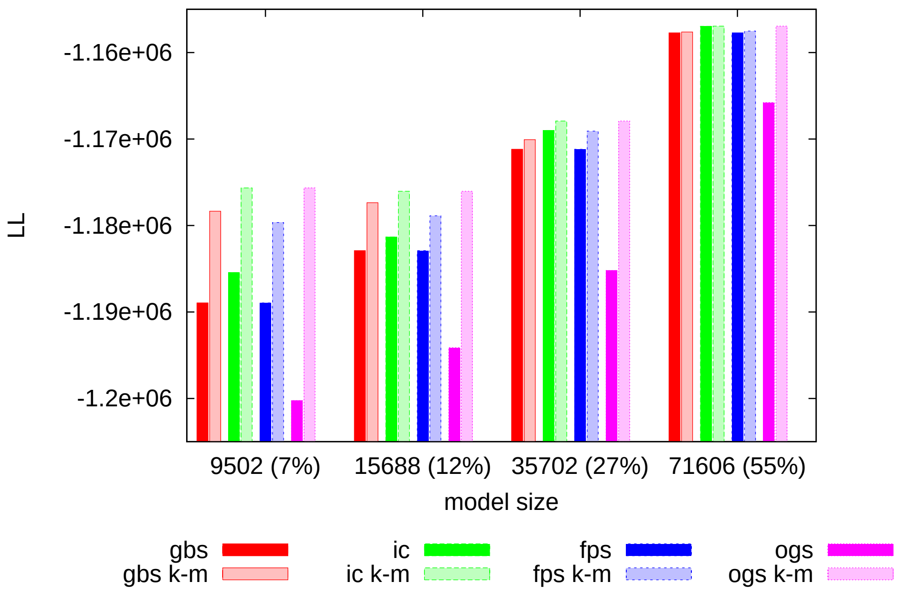


(a)

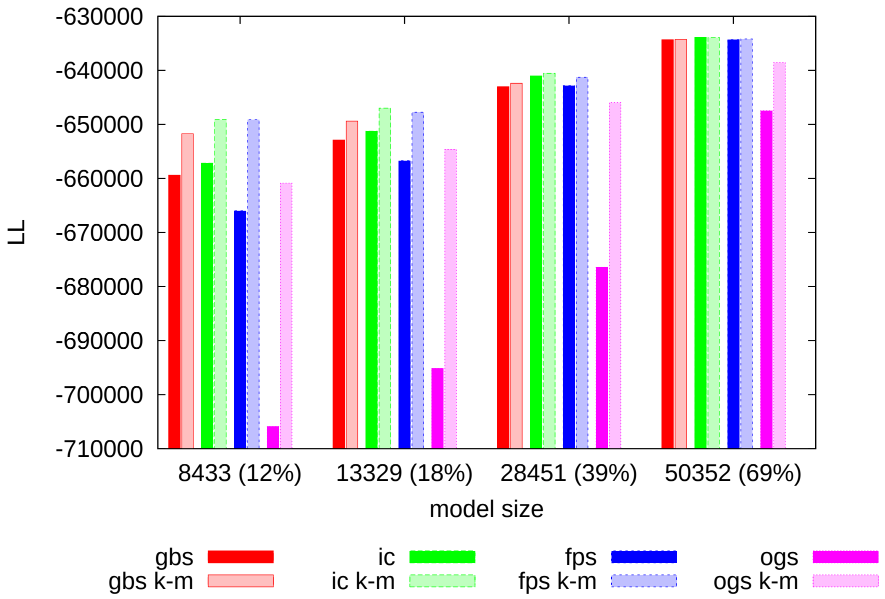


(b)

Figure B.4.: Execution times for the Austin ACES dataset corresponding to the 4 simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs). For clarity Figure (b) does not show the times for the farthest-points sampling algorithm.

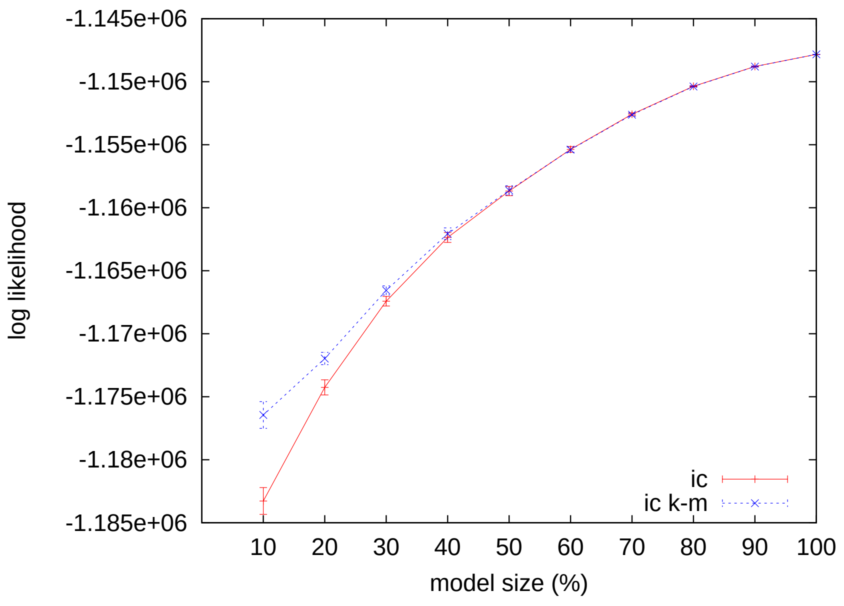


(a) Bruceton Mine

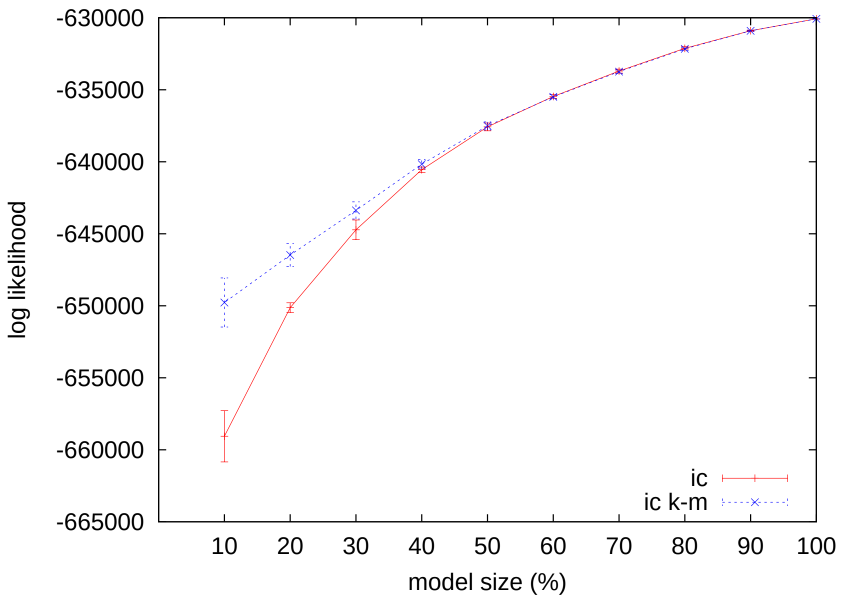


(b) Austin ACES

Figure B.5.: Likelihood of the models obtained using the k-means (k-m) algorithm for different starting models. Also shown are the likelihood of the models used as starting point generated with the different simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).



(a) Bruceton Mine



(b) Austin ACES

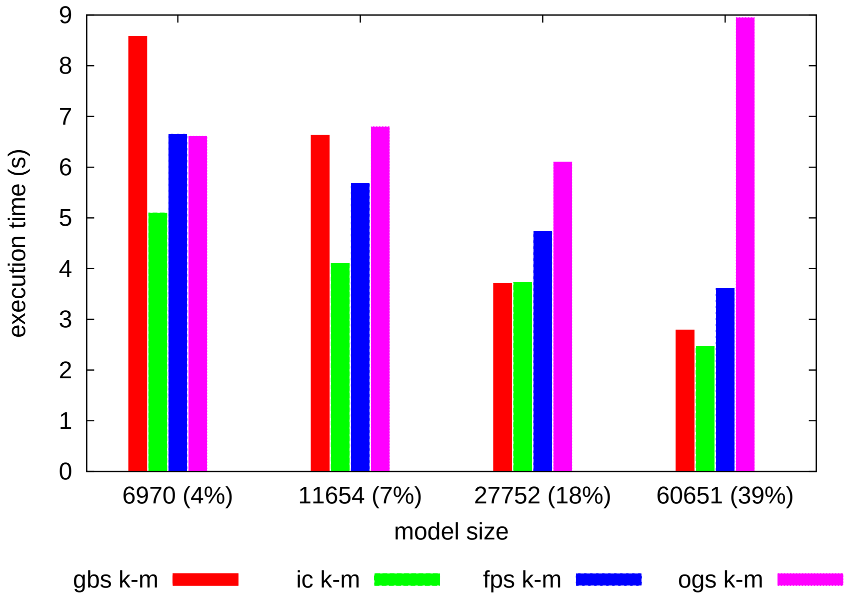
Figure B.6.: Likelihood of the models obtained using k-means (k-m) as a function of the size of the model. The models used as starting point where obtained using Incremental Clustering (ic).

Initial Model	6970 (4%)	11654 (7%)	27752 (18%)	60651 (39%)
gbs	54	37	24	17
ic	55	34	21	11
fps	54	35	22	13
ogs	90	41	26	26

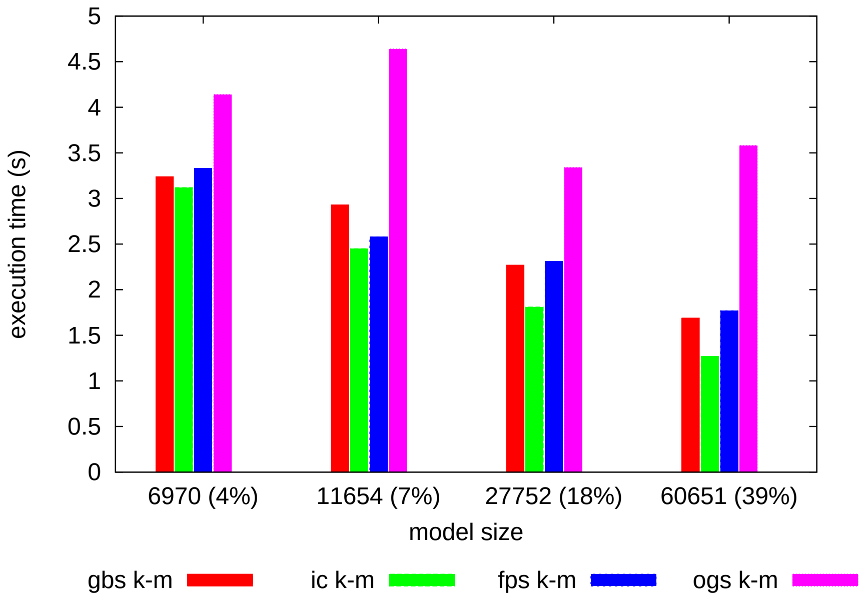
Table B.1.: Number of iterations needed by the k-means algorithm for the Bruceton Mine dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	6970 (4%)	11654 (7%)	27752 (18%)	60651 (39%)
gbs	72	48	29	12
ic	39	30	20	10
fps	59	38	22	14
ogs	70	42	30	18

Table B.2.: Number of iterations needed by the k-means algorithm for the Austin ACES dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

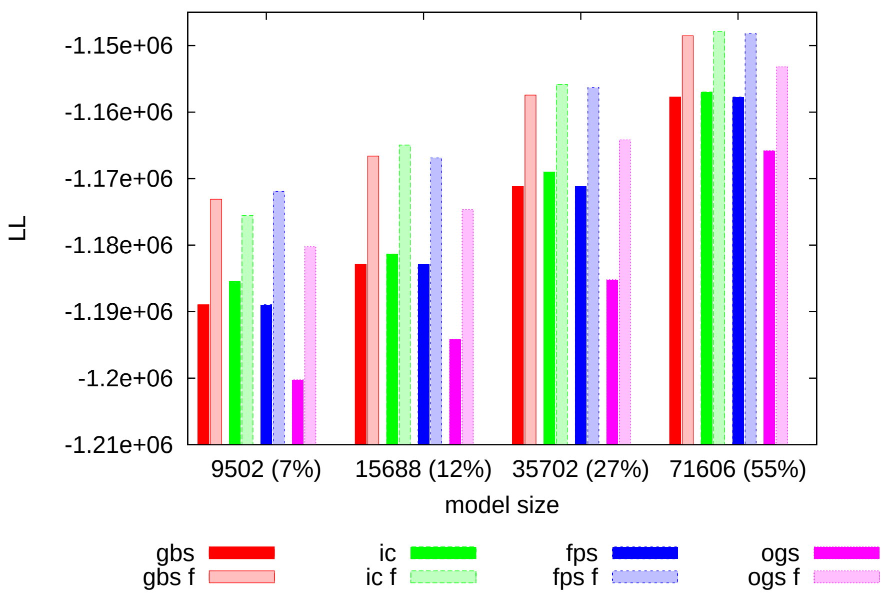


(a) Bruceton Mine

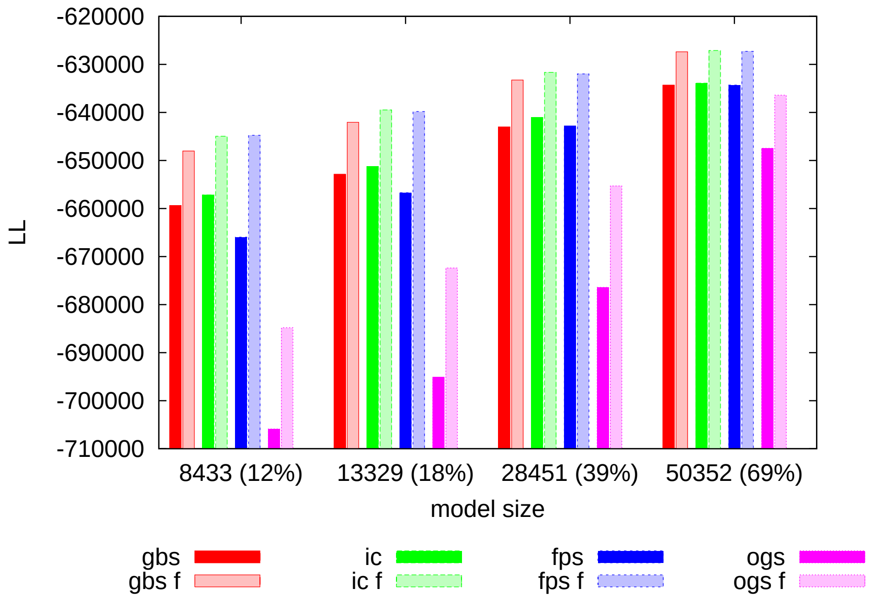


(b) Austin ACES

Figure B.7.: Execution times of the k-means (k-m) algorithm on a standard PC with a 2.8 GHz processor for the starting models produced with the dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

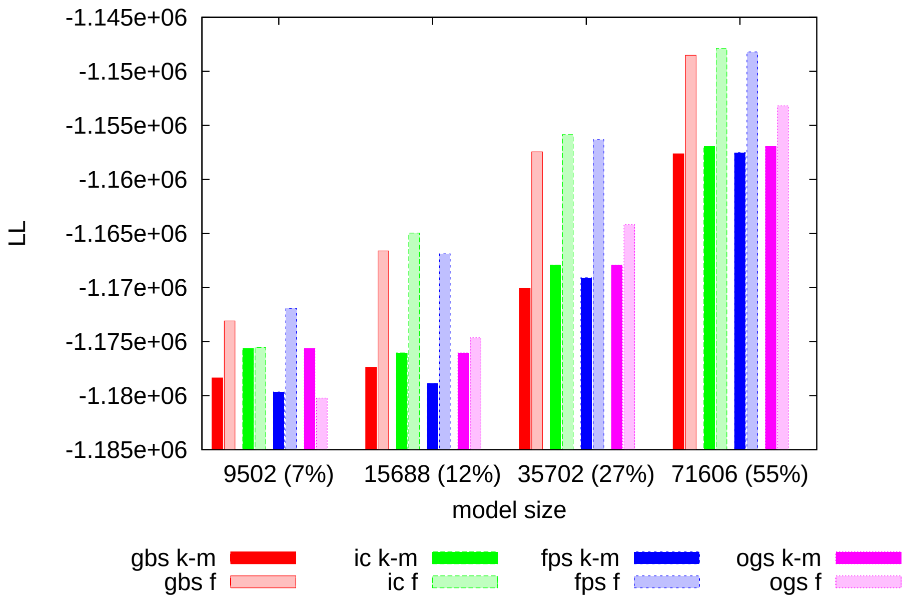


(a) Bruceton Mine

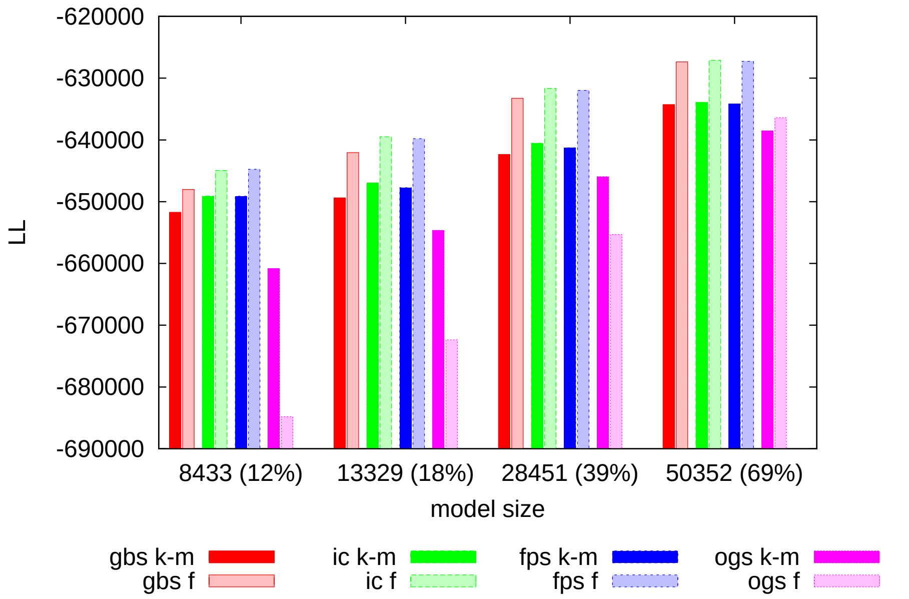


(b) Austin ACES

Figure B.8.: Likelihood of the models obtained using the fuzzy k-means (f) algorithm. Also shown are the likelihood of the models obtained using the different simplification techniques used as starting models: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).



(a) Bruceton Mine



(b) Austin ACES

Figure B.9.: Likelihood of the models obtained using the fuzzy k-means (f) algorithm compared with the likelihood of the models obtained using the k-means algorithm for the same startig models. The starting models where generated using the dataset simplification algorithms: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	9502 (7%)	15688 (12%)	35702 (27%)	71606 (55%)
gbs	21	17	13	10
ic	16	17	14	10
fps	15	20	14	11
ogs	42	22	21	36

Table B.3.: Number of iterations needed by the fuzzy k-means algorithm for the Bruceton Mine dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	9502 (7%)	15688 (12%)	35702 (27%)	71606 (55%)
gbs	6.82	8.99	15.86	25.61
ic	5.10	8.09	16.22	23.90
fps	5.23	10.90	17.56	28.46
ogs	13.35	11.41	24.74	88.72

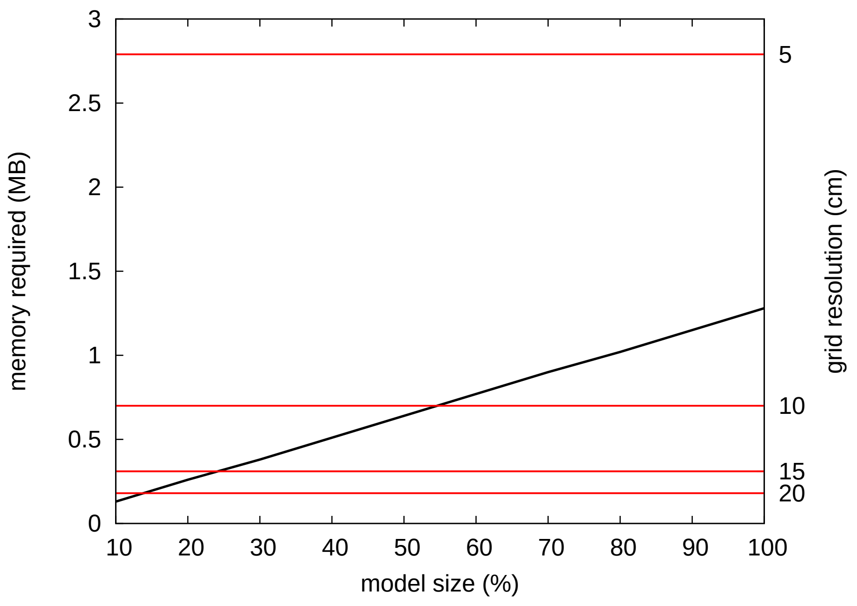
Table B.4.: Execution times in minutes of the fuzzy k-means for the Bruceton Mine dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	8433 (12%)	13329 (18%)	28451 (39%)	50352 (69%)
gbs	20	19	15	8
ic	19	18	12	9
fps	19	14	13	9
ogs	38	39	23	22

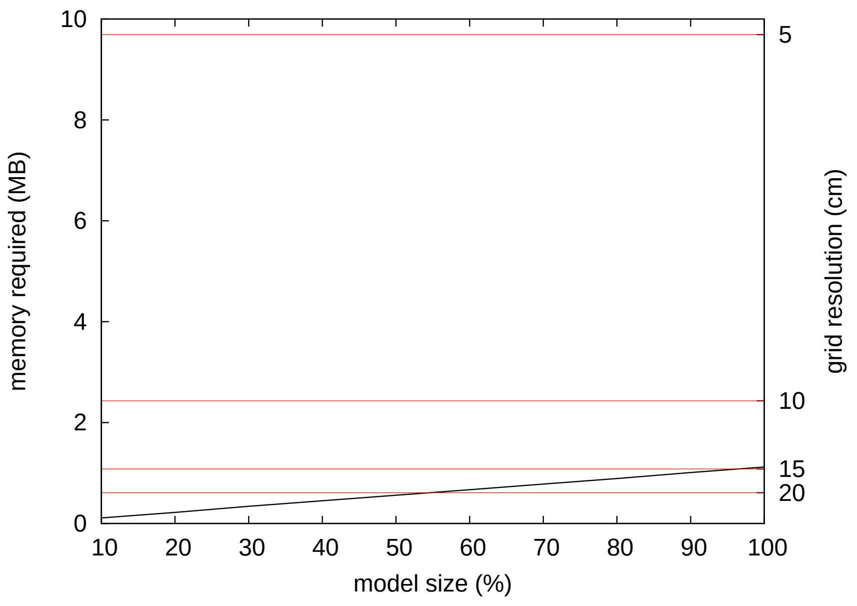
Table B.5.: Number of iterations needed by the fuzzy k-means algorithm for the Austin ACES dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).

Initial Model	2539 (3%)	4028 (5%)	9278 (11%)	21290 (25%)
gbs	3.44	5.17	9.10	9.14
ic	3.30	4.97	7.22	10.16
fps	3.43	3.76	7.82	9.86
ogs	6.51	10.69	13.66	25.98

Table B.6.: Execution times in minutes of the fuzzy k-means for the Austin ACES dataset. The initial models were generated using the different dataset simplification techniques: Grid-Based Sampling (gbs), Incremental Clustering (ic), Farthest-Point Sampling (fps), and Occupancy Grid Sampling (ogs).



(a) Sieg Hall



(b) Austin ACES

Figure B.10.: Memory required in megabytes (MB) by a sample-based representation in function of the number of samples in the map. Also shown is the memory required by an occupancy grid representation for 4 different grid resolutions. For each point in a sample-based map 2×8 bytes are needed. For each cell in an occupancy grid 2×4 bytes are needed.

List of Algorithms

Bibliography

[Alexa *et al.*, 2001] M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. David Levin, and C. T. Silva. Point set surfaces. In *IEEE Visualization*, 2001.

[Anguelov *et al.*, 2004] D. Anguelov, D. Koller, Parker E., and S. Thrun. Detecting and modeling doors with mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2004.

[Arras and Siegwart, 1997] Kai O. Arras and Roland Y. Siegwart. Feature extraction and scene interpretation for map-based navigation and map building. volume 3210, pages 42–53. SPIE, 1997.

[Bennewitz *et al.*, 2002] M. Bennewitz, W. Burgard, and S. Thrun. Using EM to learn motion behaviors of persons with mobile robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.

[Bentley, 1975] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[Burgard *et al.*, 1999] D. Burgard, W. andFox, H. Jans, C. Matenar, and S. Thrun. Sonar-based mapping with mobile robots using EM. 1999.

[Crowley, 1989] J. L. Crowley. World modeling and position estimation for a mobile robot using ultrasonic ranging. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1989.

[Dempster *et al.*, 1977] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[Duda *et al.*, 2000] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley Interscience, 2000.

[Elfes, 1989] A. Elfes. *Occupancy grids: a probabilistic framework for mobile robot perception and navigation*. PhD thesis, Electrical and Computer Engineering Department, Carnegie Mellon University, 1989.

[González-Baños and Latombe, 2000] H. González-Baños and J.-C. Latombe. Robot navigation for automatic model construction using safe regions. In *ISER*, pages 405–415, 2000.

[Gonzalez *et al.*, 1994] J. Gonzalez, A. Ollero, and A. Reina. Map building for a mobile robot equipped with a 2d laser rangefinder. In *ICRA*, pages 1904–1909, 1994.

[Hähnel *et al.*, 2003] D. Hähnel, R. Triebel, W. Burgard, and S. Thrun. Map building with mobile robots in dynamic environments. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

[Kirkpatrick *et al.*, 1983] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.

[Leonard *et al.*, 2001] J. Leonard, P Newman, R. Rikoski, J. Neira, and J. Tardós. Towards robust data association and feature modeling for concurrent mapping and localization. In *10th Internation Symposium on Robotics Research (ISRR)*, 2001.

[Levoy and Whitted, 1985] M Levoy and T. Whitted. The use of points as a display primitive. Computer Science Technical Report 85-022, UNC-Chapel Hill, 1985.

[Liu *et al.*, 2001] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun. Using EM to learn 3D models with mobile robots. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2001.

[MacQueen, 1967] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical statistics and probability*, volume 1, pages 281–297. University of California Press, Berkeley, 1967.

[Pauly *et al.*, 2002] M. Pauly, M. Gross, and L. P. Kobbelt. Efficient simplification of point-sampled surfaces. *Theor. Comput. Sci.*, 284(1):67–108, 2002.

[Roy *et al.*, 2003] N. Roy, M. Montemerlo, and S. Thrun. Perspectives on standardization in mobile robot programming. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003.

[Sack and Burgard, 2004] D. Sack and W. Burgard. A comparison of methods for line extraction from range data. In *Proc. of the 5th IFAC Symposium on Intelligent Autonomous Vehicles (IAV)*, 2004.

[Schröter *et al.*, 2002] D. Schröter, M. Beetz, and J.-S. Gutmann. Rg mapping: Learning compact and structured 2d line maps of indoor environments. In *Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication (ROMAN'02)*, 2002.

[Schwarz, 1978] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistic*, 6:461–464, 1978.

[Thrun, 2002] S. Thrun. Robotic mapping: A survey. In G. Lakemeyer and B. Nebel, editors, *Exploring Artificial Intelligence in the New Millenium*. Morgan Kaufmann, 2002.

[Veeck and Burgard, 2004] M. Veeck and W. Burgard. Learning polyline maps from range scan data acquired with mobile robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004.