Spatio-Temporal Terrain Classification for Mapping and Robot Localization

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Stefan Laible

aus Friedrichshafen

Tübingen

2016

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	19.01.2017
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

Abstract

Detection and classification of the surrounding terrain is a fundamental ability of a mobile robot in outdoor navigation to enable safe and efficient path planning. Our robot is equipped with a 3D LiDAR and a color camera, since these sensors complement each other very well. The terrain in front of the robot is divided into a grid, and each grid cell is classified individually using the sensor measurements. A new method for 3D LiDAR-based terrain classification with easy-to-compute and yet discriminative 3D features based on intensity and roughness histograms is presented. The feature extraction for one grid only takes 1.5 ms on average. For classification, we use Random forests since they outperform all other tested classifiers. When only considering the two classes of asphalt and grass the classification results are consistently above 99.9% for different tested light conditions. Camera-based texture classification with Local ternary patterns is used in addition, and for grid cells where data of both sensors are present, the results are fused accordingly.

To exploit the fact that terrain appears in contiguous areas, spatial dependencies between the individual cells of the terrain grid are taken into account by modeling the grid as a Conditional random field. An approximately optimal configuration of terrain labels is found by optimizing feature- and neighborhood-dependent energy terms using Gibbs sampling in a simulated-annealing scheme, and an efficient way for describing the neighborhood-dependent energy is presented. In our experiments considering the four terrain classes asphalt, cobblestones, grass, and gravel, using spatial dependencies improves results significantly, and we get a classification rate of 96.8%, in contrast to 81.5% when classifying grid cells individually.

As the robot moves, we constantly update a terrain map with the current classification result. In this way, we are not only able to exploit temporal dependencies, but we are building whole terrain maps of the environment. We show how to efficiently incorporate the classification result of the current terrain grid into the map, how to combine it with results from previous time steps, and how to integrate the label configuration obtained from simulated annealing. In experiments using a new data set, spatial classification yields a classification rate of 92.2%, only using temporal classification yields 96.8%, and finally spatio-temporal classification yields the best result with 98.4%. By additionally integrating information about obstacles, we can generate meaningful terrain and elevation maps of the robot's environment. Spatio-temporal classification and elevation mapping take 23.9 ms on average, which corresponds to about 41.8 Hz. This shows that our classification method is real-time capable. Finally, we show how to use these maps for a semantic localization of the robot.

Kurzfassung

Das Erkennen und Klassifizieren des umliegenden Terrains ist von großer Bedeutung für einen mobilen Roboter, der in Außenbereichen navigieren soll, da nur damit eine sichere und effiziente Pfadplanung gewährleistet werden kann. Wir haben unseren Roboter mit einem 3D LiDAR und einer Farbkamera ausgestattet, da diese Sensoren sich gegenseitig sehr gut ergänzen. Das Terrain, das vor dem Roboter liegt, wird in ein regelmäßiges Gitter unterteilt, und jede Gitterzelle wird einzeln anhand der Messdaten der Sensoren klassifiziert. Es wird eine neue Methode für 3D-LiDAR-basierte Terrainklassifikation vorgestellt, mit leicht zu berechnenden, und dennoch charakteristischen 3D-Merkmalen basierend auf Intensitäts- und Rauheits-Histogrammen. Die Merkmalsextraktion für ein Gitter benötigt durchschnittlich nur 1.5 ms. Für die Klassifikation verwenden wir Random Forests, da diese von allen von uns getesteten Klassifizierern am besten abgeschnitten haben. Werden nur die zwei Klassen Asphalt und Gras betrachtet, sind die Klassifikationsergebnisse für unterschiedliche getestete Lichtverhältnisse durchgehend über 99.9%. Zusätzlich wird eine kamerabasierte Texturklassifikation mit Local Ternary Patterns verwendet, und für Gitterzellen, bei denen Daten von beiden Sensoren zur Verfügung stehen, werden die Ergebnisse entsprechend fusioniert.

Um die Tatsache zu nutzen, dass Terrain meist in zusammenhängenden Flächen vorkommt, werden räumliche Abhängigkeiten zwischen einzelnen Zellen des Terraingitters mitberücksichtigt, indem das Gitter als Conditional Random Field modelliert wird. Durch Optimieren von Merkmals- und Nachbarschafts-abhängigen Energietermen kann eine Näherungslösung für eine optimale Konfiguration von Terrainlabels gefunden werden; dazu wird ein Gibbs Sampler in einem Simulated-Annealing[1]-Verfahren verwendet. Außerdem wird eine effiziente Möglichkeit, den Nachbarschafts-abhängigen Term zu beschreiben, vorgestellt. In unseren Experimenten, bei denen die vier Terrainklassen Asphalt, Pflastersteine, Gras und Kies betrachtet werden, verbessert die Berücksichtigung räumlicher Abhängigkeiten die Ergebnisse signifikant, und wir erhalten eine Klassifikationsrate von 96.8%, im Gegensatz zu 81.5%, wenn die Gitterzellen einzeln klassifiziert werden.

Wenn der Roboter sich bewegt, wird eine Terrainkarte mit dem jeweils aktuellen Klassifikationsergebnis aktualisiert. Auf diese Weise sind wir nicht nur in der Lage zeitliche Abhängigkeiten zu nutzen, sondern können ganze Terrainkarten der Umgebung erstellen. Wir zeigen wie das Klassifikationsergebnis des aktuellen Terraingitters effizient in die Terrainkarte integriert werden kann, wie man es mit Ergebnissen früherer Zeitschrit-

[1] dt.: simulierte Abkühlung

te kombinieren kann, und wie man die Labelkonfiguration, die das Simulated Annealing liefert, mitberücksichtigt. In Experimenten mit einem neuen Datensatz liefert die räumliche Klassifikation eine Klassifikationsrate von 92.2%, verwendet man nur die zeitliche Klassifikation erhält man 96.8%, und die räumlich-zeitliche Klassifikation schließlich liefert das beste Ergebnis mit 98.4%. Durch das Verwenden zusätzlicher Informationen über Hindernisse können wertvolle Terrain- und Höhenkarten der Umgebung des Roboters erstellt werden. Die räumlich-zeitliche Klassifikation und Höhenkartierung braucht durchschnittlich 23.9 ms, was ungefähr 41.8 Hz entspricht. Dies zeigt, dass unsere Klassifikationsmethode echtzeitfähig ist. Schließlich zeigen wir, wie diese Karten für eine semantische Lokalisierung des Roboters verwendet werden können.

Danksagung

Eine Doktorarbeit ist eine langwierige und mühselige Angelegenheit, und vieles muss man alleine meistern. Doch immer konnte ich auch auf die Unterstützung meiner Kollegen, Freunde und Familie zählen und bauen, und ich möchte mich an dieser Stelle bei ein paar Leuten bedanken.

Meinem Doktorvater Prof. Zell danke ich für die wertvollen Tipps bei Veröffentlichungen und meiner Dissertation, sowie allgemein für die kritische Auseinandersetzung mit meiner Arbeit. Prof. Schilling danke ich dafür, dass er trotz eines schon beträchtlichen Stapels an noch zu lesenden Dissertationen, die Zweitkorrektur übernommen hat. Ich bin dankbar für die finanzielle Unterstützung meiner Promotion durch ein MINT-Stipendium des Landes Baden-Württemberg in Kooperation mit der Robert Bosch GmbH, sowie für die daraus entstandene Zusammenarbeit. Ich danke meinen Studenten Richard Hanten, Holger Kaden und Marek Gardocki für die tollen Bachelor-, Studien- und Diplomarbeiten und ihr großes Engagement. Ein besonderer Dank geht auch an Lucy Blaney-Laible, Christian Widmer und Gerald Rauscher für das Korrekturlesen meiner Arbeit und ihre wertvollen Verbesserungsvorschläge, Korrekturen und Anmerkungen.

Ich danke allen meinen ehemaligen Kollegen am Lehrstuhl für eine großartige und unvergessliche Zeit. Insbesondere danke ich Klaus Beyreuther und Vita Serbakova für ihre Unterstützung, und meinen Bürokollegen und -nachbarn Yasir Niaz Khan, Gerald Rauscher und Markus Beck, sowie Timo Sachsenberg, dessen Besuche mir immer eine willkommene Abwechslung waren.

Zuletzt und besonders herzlich möchte ich meinen Eltern und Becky danken, für ihre Unterstützung und ihr Vertrauen in mich, welches zuweilen auch sehr beansprucht wurde.

Contents

When it is not in our power to determine what is true, we ought to follow what is most probable.

RENÉ DESCARTES

Chapter 1

Introduction

The reason that we can safely and efficiently navigate our everyday environment lies in our ability to perceive our surroundings. Hereby, perception not only means seeing the world, or more generally, receiving sensory data, but also processing this information and drawing conclusions, in order to understand the world around us. The same must be true for robots that have the ability to navigate autonomously. While among the early days of robotics, most robots operated inside rooms or buildings, now they "leave the house" and go outside, as exemplified in the growing fields of agricultural robots, autonomous cars, or service robots such as autonomous lawnmowers. When it comes to robot perception, localization, and navigation, there is a general difference between indoor and outdoor environments. Man-made environments like office buildings and factories tend to feature a geometric structure, which can be used for precisely locating a robot. In corridor-like environments, a 2D laser scanner is often sufficient for the robot to perform basic navigation tasks. In this case, the robot does not need a semantic understanding of its surroundings; it only distinguishes between obstacles and non-obstacles, without the need to understand what these obstacles really are. In contrast, outdoor surroundings often lack this kind of structure. In order to enable a safe and efficient navigation in such environments a comprehensive semantic perception is essential. Of particular interest is the terrain. Often the only thing a robot can see outdoors is its surrounding terrain, and the knowledge about it is crucial for safe and efficient path planning. For example, it is far easier for robots to drive on asphalt roads than on rough grassy areas. However, an autonomous lawnmower must stay on the grass, and needs to be able to detect other types of terrain. Additionally, knowledge about the surrounding terrain is also valuable for robot localization, as a supplement or alternative when GPS is too unreliable or not available at all, such as near buildings, under trees, or in any situation when there is no clear line of sight to the satellites. The accuracy requirements of such localization depend on whether the robot is to follow fixed lines, for example, a field boundary or a road, or whether it should travel long distances over an open field.

Perception in robotics is about processing sensor measurements, and in this thesis, we use a 3D LiDAR in conjunction with a color camera. These sensors complement each other very well; while the LiDAR provides range values and is less affected by different lighting conditions, the camera is able to capture the appearance of things. Neverthe-

less, no sensor is perfect: each one can give potentially noisy and faulty measurements. In the case of range measurements, for instance, we can only tell how far away things are within a certain probability, and in the case of odometry, we can only approximate where the robot moved. In addition to the unreliability of sensor measurements, there is always ambiguity involved in classifying terrain. Two patches of terrain never look exactly the same, and classifying them based on a previous learned model always involves uncertainty. Here, probability theory helps us to deal with these uncertain informations. The advantage of probability theory in the context of robotics is that the combination of different uncertain information can lead to a much higher degree of certainty.

A large part in this thesis discusses the use of probabilistic graphical models for terrain classification, where probability theory and graphical models are combined. Using these models it is possible to also consider spatial dependencies between individual patches of terrain. In some cases it can be hard, even for a human, to tell whether a tiny patch of terrain is grass or gravel, for instance, but it becomes clearer in the spatial context. In addition to these spatial dependencies, we also investigate temporal dependencies. When a robot drives over terrain, it sees the same patch of terrain in consecutive time steps, each time from a slightly different position and angle. As was already noted above, combining the individual classification results should yield a better final result. Moreover, integrating the results of each time step into a common coordinate frame lets us build whole terrain maps of the environment. By additionally integrating information about obstacles, we can generate meaningful terrain and elevation maps of the robot's environment. Finally, these maps can be used for a semantic localization of the robot.

1.1 Outline

The remaining chapters are organized as follows:

- **Chapter 2** contains a discussion of theoretical principles that are necessary for an understanding of the algorithms and mathematical models presented in this thesis, which involves Local ternary patterns, Random forests, and Monte Carlo localization. It also contains a detailed description of probabilistic graphical models, since they are fundamental to the terrain-classification method.

- **Chapter 3** presents the robot platform used for data acquisition and experiments. The main sensors are a color camera and the 3D LiDAR Nippon Signal FX6. Since this LiDAR is rather uncommon, its theory of measurement is described in more detail.

- In **Chapter 4** a new method for 3D LiDAR-based terrain classification with easy-to-compute and yet discriminative 3D features is presented. These results are then combined with camera-based texture classification in order to classify a terrain grid in front of the robot.

- To improve classification results, spatial dependencies between the individual cells of the terrain grid are taken into account in **Chapter 5** by describing the grid as a probabilistic graphical model. An approximate optimal configuration of terrain labels is found by optimizing feature- and neighborhood-dependent energy terms, and an efficient way for describing the neighborhood-dependent energy is presented.
- **Chapter 6** describes how to build spatio-temporal terrain and elevation maps of the environment by constantly integrating the current classification results into a map at each time step, and by taking into account temporal dependencies. Finally, this chapter shows how to use these maps for a semantic localization of the robot.
- The conclusions in **Chapter 7** offer a brief summary of the main results and an outlook on future work.

Chapter 2

Theoretical Principles

In this chapter the theoretical principles that are necessary for an understanding of the following chapters are discussed. The methods presented in this thesis build on these principles, modify and extend them. We start in Sec. 2.1 by formulating the fundamental classification problem using probability theory. In Sec. 2.2 we present Local binary patterns and Local ternary patterns, descriptors that are used for computing feature vectors for image regions. Random forests, the classification method that builds the foundation of all our methods, will be explained in Sec. 2.3. We then show in Sec. 2.4 how to incorporate structural dependencies in classification using graphical models. We compare Markov random fields and Conditional random fields, discuss advantages and disadvantages of the two models, and present a method for maximum-a-posteriori inference. Finally, in Sec. 2.5, we turn to the topic of robot localization and discuss Monte Carlo localization, a localization method using a particle filter.

2.1 Classification and Probability

Classification, in the sense in which we use it in this thesis, is the problem of identifying to which of a set of predefined classes an instance belongs. In our work, an instance is usually a small patch of terrain, which belongs to one of the terrain classes *asphalt*, *cobblestones*, *grass*, *gravel*, or *tiles*. In general, an instance is represented by a D-dimensional feature vector $x \in \mathbb{R}^D$. This represents what we observe, typically on the basis of sensor measurements. A feature vector can consist, for example, of pixel values or range measurements, and in general of any numbers that represent the characteristics of an instance. We will see an example of such a feature vector in Sec. 2.2, where an image region will be represented by a histogram based on differences in pixel intensities. Given a feature vector, the task is to find the class to which the instance belongs. So-called supervised learning algorithms try to identify and represent the relationships between feature vectors and class labels by analyzing a training data set, so that predictions for previously unseen feature vectors can be made. The training data set consists of many pairs of feature vectors and associated class labels. The assignment of labels to feature vectors is usually done by hand. The learning algorithms thus produce an inferred function that takes a feature vector as input, and outputs the predicted class label.

However, in general, these predictions can not be made with absolute certainty, but only with some probability. So, a preferable function is one that indicates a probability value for each class label. We will get to know such a learning algorithm in Sec. 2.3. In the following section, we will formalize this idea and introduce some necessary notation [1].

In order to classify not only small patches of terrain but larger areas, we divide these areas into multiple cells. We refer to this subdivision into individual cells as a *terrain grid*, and index the grid cells with $1,...,M$, where M is the number of cells. Since each cell can take on each terrain label with a certain probability, we model these labels as a set of random variables $\mathcal{Y} = \{Y_1,...,Y_M\}$. A label configuration $\mathbf{y} = \{Y_1 = y_1,...,Y_M = y_M\}$ assigns a class label y_i to each random variable Y_i, for $i \in \{1,...,M\}$. Thereby, a class label is simply the index of the corresponding class in the set of terrain classes. If there are K classes, it is $y_i \in \{1,...,K\}$. The label configuration $\mathbf{y}$ tells us, so to speak, for each terrain cell whether it is asphalt, cobblestones, grass, and so on. The random variables Y are also called output variables, since their predicted assignment is the output of our learned model. The input variables X, on the other hand, are the variables that are fed into the model. They represent the possible values that the feature vectors of each cell may take. While the output variables can only take on discrete values from a small set of class labels, the input variables can in general take on an infinite number of possible values. But in contrast to the output variables, these variables can be observed. By computing feature vectors x_i based on sensor measurements, we can assign a value to each input variable $X_i \in \mathcal{X}$, $\mathcal{X} = \{X_1,...,X_M\}$. Given the feature vector x_i, we then are interested in the probability $p(Y_i = y_i \mid X_i = x_i)$ that the grid cell belongs to the terrain class with label y_i. For better readability, this probability will be denoted hereinafter as $p(y_i \mid x_i)$. A learning algorithm that returns these probability values, and not just a class label, has many advantages, as we shall see in the following chapters. If we nevertheless want to assign a single label y_i^* to the grid cell, we simply choose the one which is most probable [2]:

$$y_i^* = \arg\max_{y_i} p(y_i \mid x_i) \tag{2.1}$$

In the same way that $p(y_i \mid x_i)$ denotes the probability of a single cell label given the corresponding feature vector, $p(\mathbf{y} \mid \mathbf{x})$ denotes the probability of the label configuration $\mathbf{y}$ of the whole grid, given the set $\mathbf{x}$ of all feature vectors computed for this grid. This probability distribution over $\mathcal{X}$ and $\mathcal{Y}$ can be very complex, but if we make the assumption that the labels and features of any pair of grid cells are independent of each other, the probability distribution factorizes into the cell-wise probabilities:

[1] See the symbol table in the appendix for a summary of the notation used.

[2] When two or more labels are equally likely, one of these values is chosen at random.

$$p(\mathbf{y} \mid \mathbf{x}) = \prod_{i=1}^{M} p(y_i \mid x_i) \tag{2.2}$$

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \prod_{i=1}^{M} p(y_i \mid x_i) \tag{2.3}$$

It is clear that this assumption is a strong limitation in the correct description of reality, since labels of two adjacent cells very well do depend on each other. We will see in Sec. 2.4 how we can loosen these restriction and still be able to estimate the most likely label configuration $\mathbf{y}^*$.

2.2 Local Binary Patterns and Local Ternary Patterns

To represent a small image region as a feature vector x we use Local binary patterns (LBP) and Local ternary patterns (LTP). LBP and LTP encode the intensity differences between a pixel and its neighborhood in a binary sequence, and are particularly well suited for textures. LTP is an extension of LBP, and we will therefore describe LBP first.

Local binary patterns

LBP represents an image pixel by the relation to its neighboring pixels. We define this neighborhood to be the eight surrounding pixels (see top left of Fig. 2.1). Hereby we do not use color but grayscale values. Thus, each pixel is described by a single number, normally between 0 and 255. Let c be the grayscale value of the centering pixel. Then, each neighboring pixel with value p is labeled according to a function $s_{\text{LBP}}(c,p)$. If p is greater or equal to c, then $s_{\text{LBP}}(c,p)$ equals 1, otherwise 0:

$$s_{\text{LBP}}(c,p) = \begin{cases} 1, & \text{if } p-c \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

Starting from the right pixel going clockwise, these labels form a binary pattern of length eight. Now, to not only represent single pixels, but a whole image region, first, the LBPs for all pixels in this region are calculated, and then a histogram of the frequencies of occurrences of these patterns is created. Since these patterns are binary strings of length eight, the histogram consists of $2^8 = 256$ bins (see Fig. 2.2). So, the corresponding feature vector x has 256 dimensions.

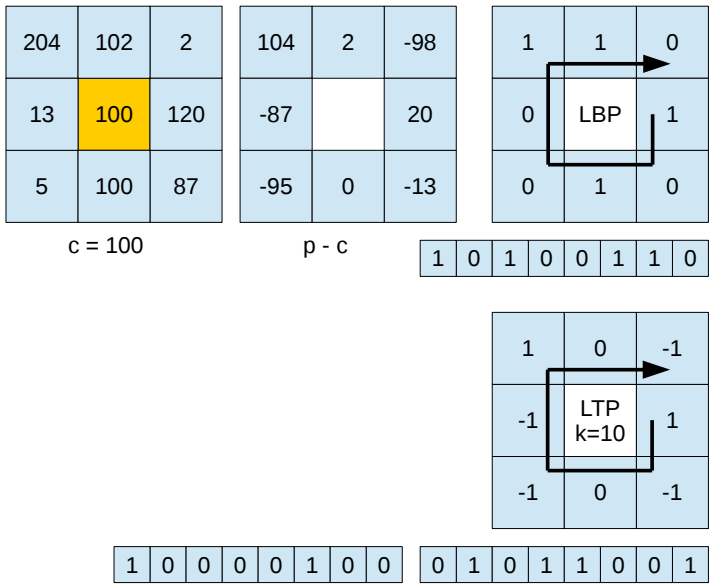


Figure 2.1: Local binary patterns (LBP) and Local ternary patterns (LTP) encode the neighborhood of a pixel in a binary sequence, based on differences in pixel intensities.

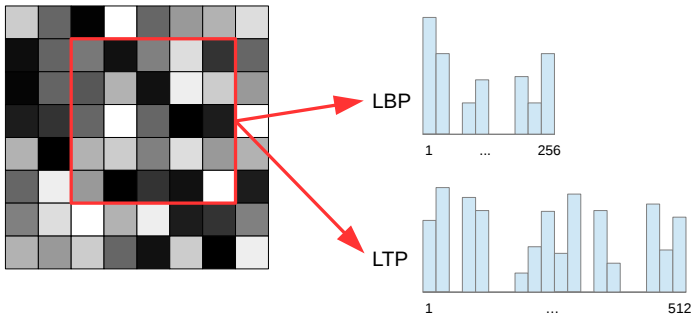


Figure 2.2: An image region (red square) is described by histograms based on the frequency of occurrence of Local binary patterns, or Local ternary patterns respectively, for the individual pixels.

Local ternary patterns

LTP extend LBP by not only checking whether the pixel differences are positive or negative, but by checking whether the differences are greater or smaller than a threshold k:

$$s_{\mathrm{LTP}_k}(c,p) = \begin{cases} 1, & \text{if } p-c > k \\ -1, & \text{if } p-c < k \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

Since each pixel now has three possible labels, the number of possible patterns, and thus the number of histogram bins, would be $3^8 = 6561$. To reduce this large number two binary patterns are created (see Fig. 2.1). In the first pattern, a one is written when the corresponding pixel is marked with one, and otherwise a zero is written. In the second pattern, a one is written when the corresponding pixel is marked with minus one, and otherwise a zero is written. Histograms are calculated for both patterns, which are then concatenated. This results in a 512-dimensional feature vector x. LTP can thus describe textures in more detail than LBP, at the expense of a longer feature vector and a threshold parameter to be set.

2.3 Random Forests

Random forests [Breiman (2001)] is a machine-learning method that uses multiple binary decision trees for classifying an instance, represented by a feature vector, as belonging to one of a set of pre-defined classes. The method can also be used for regression, but we focus on classification, since this is what we use it for in our work. In the construction of trees in the training phase there is randomness involved at various points, hence the name of the method. To classify unseen data, each tree classifies the corresponding feature vector individually, giving a so-called vote for one class. The decision of the forest is then given by the majority of votes.

Training phase

Random forests is a supervised learning method, which means that we need labeled training data. Let $\mathcal{S} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ be the training data set, where x_i is the i-th feature vector and y_i its assigned class label, with $i = 1, ..., N$. The training set is used in the training phase for constructing the decision trees of the forest. The creation of trees is what is referred to as the actual learning process; the knowledge obtained by the available training data is represented by the special structure and parameters of the trees. Hereby, the construction of a decision tree is not deterministic; there are some random elements involved.

First of all, each tree uses a different training set $\mathcal{S}_p$, a so-called bootstrap sample from the training set $\mathcal{S}$, with $p = 1, ..., T$, where T is the number of trees in the forest. Such a bootstrap sample is obtained by (randomly) sampling N training vectors from $\mathcal{S}$ with replacement. This means that the bootstrap sample has the same size as $\mathcal{S}$, but some vectors occur more than once, while others are not included at all. For large training sets

the fraction of unique feature vectors is expected to be $1 - 1/e$ ($\approx 63.2\%$) [Aslam *et al.* (2007)]. The construction of each tree is based on the CART algorithm (**C**lassification **A**nd **R**egression **T**rees) [Breiman *et al.* (1984)]. It starts with the tree consisting only of the root node. Then, the training set $\mathcal{S}_p$ is divided into two disjoint subsets $\mathcal{S}_p^{\text{left}}$ and $\mathcal{S}_p^{\text{right}}$, so that these subsets are as *pure* as possible. In this context, pure, or rather none-pure, is defined by an impurity measure. Intuitively, we want a partitioning of the training data, also called a split, that keeps instances of the same class together, and separates instances of different ones. CART, and also Random forests, use the Gini impurity I_G:

$$0 \leq I_G(\mathcal{S}) = \sum_{i=1}^{K} f_i(1 - f_i) = 1 - \sum_{i=1}^{K} f_i^2 \tag{2.6}$$

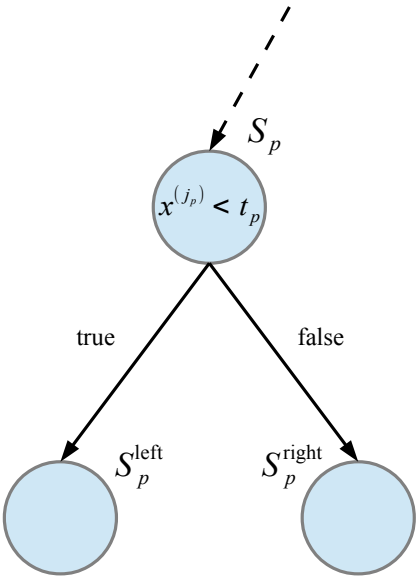
Here, f_i is defined as the fraction of training instances of S that belong to the class indicated by i, that is $y = i$, with a total of K classes. $I_G(\mathcal{S})$ is zero when all training instances belong to the same class; the set S is then as pure as possible. The training sets are recursively divided further until either a specified maximum depth of the tree branch is reached, the number of training samples in the set is less than a specified threshold, the best split is not much better than a random one, or the Gini impurity is zero. The leaf node then contains the class label that occurs the most in the remaining set. In case of a tie, a class label is chosen at random.

The question remains of how the best split at each node is found. Each non-leaf node of the tree contains a decision rule that determines which incoming training instances go to the left and which go to the right. Let $x = (x^{(1)}, \cdots, x^{(D)})$ be a D-dimensional feature vector consisting of D feature values. And let p be the index that identifies all quantities belonging to the p-th node. Then, each decision rule is fully described by two parameters: an index j_p that indicates which feature value $x^{(j_p)}$ is to be considered, and a threshold t_p with which the values are compared (see Fig. 2.3). In order to determine these parameters, each node does not consider all feature values, but a randomly selected subset. The size of this subset is often set to $\sqrt{D}$. All values $x^{(j_p)}$ of the subset are possible thresholds, and the value that yields the minimal Gini index is selected.

Classification phase

Classification of new, unseen data is now straightforward. In the classification phase a new sample x is pushed down each tree so that it ends up in a leaf node with a corresponding class label y. In this way, each tree "votes" for one class. We therefore not only get a label, but a probability $p(y \mid x)$ for each label as the proportion of trees that have voted for this label. The most probable class label then is simply the label y^* with the majority of votes:

$$y^* = \arg\max_y p(y \mid x) \tag{2.9}$$



$$\mathcal{S}_p^{\text{left}} = \{(x,y) \in \mathcal{S}_p : x^{(j_p)} < t_p\} \quad (2.7)$$

$$\mathcal{S}_p^{\text{right}} = \{(x,y) \in \mathcal{S}_p : x^{(j_p)} \geq t_p\} \quad (2.8)$$

Figure 2.3: Constructing a binary decision tree: The decision rule at node p is described by index j_p, which indicates the feature element in question, and threshold t_p.

In case of a tie, a label is selected at random among those in question. As a consequence, we get probability values for all classes, whereas in other models these have yet to be constructed, as for instance with Support vector machines [Cortes and Vapnik (1995)] using Platt's method [Platt (1999)].

The described technique, namely the training of many weak classifiers using different bootstrap samples, and then combining the classification results is also called *Bagging* [Breiman (1996)], which is an acronym for **B**ootstrap **agg**regat**ing**. Each tree learns its training data almost perfectly and therefore has a high variance, which causes overfitting. The creation of many such high-variance classifiers and combining them by voting, however, leads to a strong classifier with low variance.

An implementation of Random forests can be found in OpenCV [Bradski (2000)], where they are called Random trees. [Criminisi *et al.* (2012)] give a good introduction to the topic and present a unified, efficient model of random decision forests. They also present some extensions to the classical Random forests, where, for example, not only the forest provides probabilities for each class label, but also each node.

2.4 Probabilistic Graphical Models

Computing feature vectors like Local ternary patterns and using machine-learning methods like Random forests we can solve the classification problem stated in Sec. 2.1. Hereby, each cell of the terrain grid is classified individually, which leads to a conditional probability distribution $p(\mathbf{y} \mid \mathbf{x})$ that factorizes according to Eq. (2.2). By not considering the spatial dependencies between grid cells we are ignoring important information since terrain usually appears in contiguous areas. Probabilistic graphical models (PGMs) help here. PGMs provide a framework that combines modeling of uncertainty and structure [Koller *et al.* (2007)]. Uncertainty is described using the rules of probability theory, and the inherent structure of the problem is modeled using a graph. There are various types of PGMs, among which Bayesian networks, Hidden Markov models, and Markov random fields are some of the most popular. They are used in a wide variety of fields like natural language processing [Lafferty *et al.* (2001)], computer vision [Li (2009)], or bioinformatics [Durbin *et al.* (1998)]. In general, PGMs represent families of probability distributions over sets of random variables by means of a graph. By assuming certain conditional independence relationships, an otherwise complex distribution can be represented as a product of local functions on subsets of variables. Indeed, the power of the graphical modeling framework lies in the relationship between factorization, conditional independence, and graph structure [Sutton and McCallum (2012)].

We will now describe two graphical models, Markov random fields in Sec. 2.4.1, a generative model that represents a joint probability distribution $p(\mathbf{y},\mathbf{x})$, and Conditional random fields in Sec. 2.4.2, a discriminative model that represents a conditional distribution $p(\mathbf{y} \mid \mathbf{x})$. The theory discussed in the context of Markov random fields also builds the basis for understanding Conditional random fields. We will compare the two approaches in the context of terrain classification, and present a method for maximum-a-posteriori inference in Sec. 2.4.3.

2.4.1 Markov Random Fields

In the literature, there is a variety of — sometimes quite different — ways to approach the topic of Markov random fields (MRFs). Since we are interested in using MRFs in the context of terrain classification, we start with the classification problem in terms of labels $\mathbf{y}$ and observations $\mathbf{x}$. In general, an MRF defines a family of joint probability distributions by means of an undirected graph. In classification, this distribution is $p(\mathbf{y},\mathbf{x})$ and we can relate it to the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ using Bayes' rule.

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{y},\mathbf{x})}{p(\mathbf{x})} \quad \propto_{\mathbf{y}} \quad p(\mathbf{y},\mathbf{x}) = p(\mathbf{x} \mid \mathbf{y})p(\mathbf{y}) \tag{2.10}$$

For a given set of feature vectors $\mathbf{x}$, the conditional probability $p(\mathbf{y} \mid \mathbf{x})$ is proportional

to the joint probability $p(\mathbf{y},\mathbf{x})$ for all $\mathbf{y}$ [3]. This means that when $\mathbf{x}$ is fixed, the label configuration $\mathbf{y}^*$ that maximizes $p(\mathbf{y},\mathbf{x})$ also maximizes $p(\mathbf{y} \mid \mathbf{x})$, assuming that we are using the true probability distributions underlying the data. The joint probability distribution factorizes into two components, the conditional probability distribution $p(\mathbf{x} \mid \mathbf{y})$, and the a-priori probability $p(\mathbf{y})$ of the label configuration.

$$p(\mathbf{x} \mid \mathbf{y}) = \prod_{i=1}^{M} p(x_i \mid y_i) \tag{2.11}$$

$$= \prod_{i=1}^{M} \prod_{d=1}^{D} p(x_i^{[d]} \mid y_i) \tag{2.12}$$

$$= \prod_{i=1}^{M} \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi}\sigma_{y_i}^{[d]}} \exp\left(-\frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} \right) \tag{2.13}$$

$$= \prod_{i=1}^{M} \prod_{d=1}^{D} \exp\left(-\frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} + \log\left(\frac{1}{\sqrt{2\pi\sigma_{y_i}^{[d]2}}} \right) \right) \tag{2.14}$$

$$= \exp\left(-\sum_{i=1}^{M} \sum_{d=1}^{D} \frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} - \log\left(\frac{1}{\sqrt{2\pi\sigma_{y_i}^{[d]2}}} \right) \right) \tag{2.15}$$

$$= \exp\left(-\sum_{i=1}^{M} \sum_{d=1}^{D} \frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} + \log\left(\sqrt{2\pi}\sigma_{y_i}^{[d]} \right) \right) \tag{2.16}$$

The conditional probability distribution $p(\mathbf{x} \mid \mathbf{y})$ models how the labels *generate* the features, and that is why the MRF is called a generative model. This distribution can be quite complex, but we make three simplifying assumptions as in [Häselich *et al.* (2011)]. First, in Eq. (2.11), we assume that the label of a grid cell affects only the feature vector of that very cell. And second, in Eq. (2.12), we assume that each component of the D-dimensional feature vector is independent of the other components given the label of the cell. Then, in Eq. (2.13), the third assumption is that each component of the feature vector is Gaussian distributed, where $\mu_k^{[d]}$ and $\sigma_k^{[d]}$ are the mean and standard deviation of the d-th feature component for class label $k = 1,...,K$. The $K \cdot D$ pairs of mean and standard deviation are learned from training data.

The distribution $p(\mathbf{y})$ is the prior probability distribution of the random field. It describes the probability of a label configuration without considering feature vectors, that is, *prior* to taking into account any measurements. However, it is not obvious how this

[3] See the symbol table in the appendix for the definition of the symbol $\propto_{\mathbf{y}}$.

distribution should look like. In the context of terrain classification, for instance, a label configuration where all grid cells are equal seems more likely than a configuration with a frequent change of labels. We want to model this property of a terrain grid using a PGM. In the following sections, we describe this model and derive the distribution $p(\mathbf{y})$.

The nodes of the graph of the PGM represent the random variables Y for the labels of the grid cells. The edges of the graph represent probabilistic interactions between the variables. Since there is no preferred direction for this interaction, these edges are undirected. Actually, each variable depends on all the others, but to reduce the complexity of the model we set edges only for direct interactions between neighboring nodes.

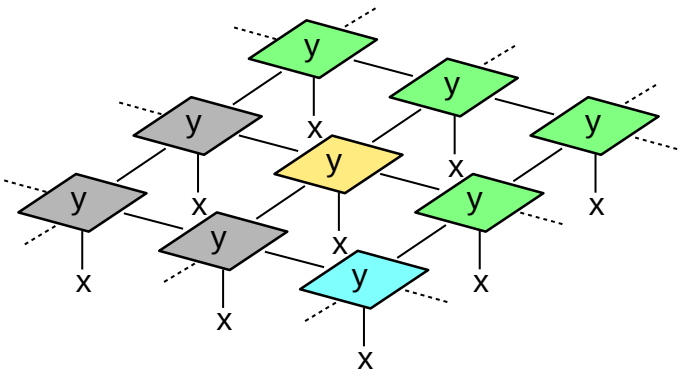


Figure 2.4: The terrain label y of a grid cell depends on the observed features x, but also on the labels of its neighboring cells.

Examples of neighborhoods are the 4-neighborhood consisting of the four direct neighbors as in Fig. 2.4, or the 8-neighborhood consisting of all eight surrounding nodes. In general, the neighborhood $\mathcal{N}_i$ of a node i consists of all nodes $j \neq i$ with a direct edge to node i. The neighborhood system $\mathcal{N}$ is the collection of all neighborhoods of the random field. Using this terminology we can now define a Markov random field $\mathcal{Y}$ consisting of random variables Y as in [Deng and Clausi (2004)].

Definition 1 (Markov random field)**.** *A random field $\mathcal{Y}$ is a Markov random field with respect to the neighborhood system $\mathcal{N}$, if and only if*

1. $p(\mathbf{y}) > 0$ *for all possible configurations* $\mathbf{y}$ *of* $\mathcal{Y}$
2. $p(Y_i = y_i \mid \{Y_j = y_j : j \neq i\}) = p(Y_i = y_i \mid \{Y_j = y_j : j \in \mathcal{N}_i\})$ *for all random variables* $Y_i \in \mathcal{Y}$.

So the probability of a configuration can be arbitrarily small, but not zero. In addition, the value of a variable only depends on its direct neighbors. This is the generalization of the Markov property as it is known, for instance, in Hidden Markov models in the one-dimensional case. It means that the neighborhood $\mathcal{N}_i$ of node i contains all relevant

information for determining y_i, and the knowledge of other values y_k with $k \notin \mathcal{N}_i$ does not affect the certainty about y_i. Y_i and the random variables of the non-adjacent nodes are statistically independent. However, we still do not know how to describe $p(\mathbf{y})$ in terms of a probability distribution. Therefore we need the definition of a Gibbs random field, which describes this distribution in terms of energy functions $E(\mathbf{y})$ and potential functions $V(\mathbf{y})$ over cliques c, and the Hammersley-Clifford theorem.

Definition 2 (Gibbs random field)**.** *A random field $\mathcal{Y}$ is a Gibbs random field with respect to the neighborhood system $\mathcal{N}$, if and only if*

$$p(\mathbf{y}) = \frac{1}{Z}\exp\left(-\beta E(\mathbf{y})\right) \tag{2.17}$$

$$= \frac{1}{Z}\exp\left(-\beta \sum_{c\in\mathcal{C}} V_c(\mathbf{y}_c)\right). \tag{2.18}$$

Theorem 1 (Hammersley-Clifford)**.** *A random field $\mathcal{Y}$ is a Markov random field with respect to the neighborhood system $\mathcal{N}$, if and only if it is a Gibbs random field with respect to $\mathcal{N}$.*

The theorem states the equivalence of Markov random fields and Gibbs random fields, and is also called the fundamental theorem of random fields [Lafferty *et al.* (2001)]. A proof can be found in the original unpublished paper [Hammersley and Clifford (1971)], or in [Besag (1974)]. The theorem allows us to use the Gibbs measure (Eq. (2.17) and (2.18)) as our probability distribution. The function $E(\mathbf{y})$ assigns a real number to a label configuration. It is also called an energy function, since it is interpreted as the energy of the configuration in the context of statistical mechanics. β is a free parameter, and when used in a physical context, is set to the inverse temperature of the system. $Z = \sum_{\mathbf{y}} \exp\left(-\beta E(\mathbf{y})\right)$ is called the partition function and ensures that the total probability for all possible configurations sums to one. The computation of the partition function involves the computation of the probability for every possible label configuration and is thus very time consuming, however, we will see in Sec. 2.4.3 that we do not need to compute it. The energy function is composed of the sum of all potential functions V_c over all cliques $c \in \mathcal{C}$, where $\mathcal{C}$ is the set of all cliques of the corresponding graph. Cliques are subsets of nodes, such that every two nodes in the clique are connected. Using a 4-neighborhood like in Fig. 2.4, the set of cliques $\mathcal{C}$ consists of sets of pairs of connected nodes. When using an 8-neighborhood, the cliques consist of up to four fully connected nodes. However, we only consider pairwise potentials, and set all non-pairwise potentials to zero. It remains to specify the potential function $V_c(\mathbf{y}_c)$ where $\mathbf{y}_c$ denotes the label configuration of the nodes in clique c. Let $\mathcal{C}_2 = \{(i,j) : j \in \mathcal{N}_i\}$ be the set of all pairs of indexes of connected label nodes. We use the pairwise potential in [Deng and Clausi (2004)] for all $c \in \mathcal{C}_2$:

$$V_c(\mathbf{y}_c) = \psi(y_i, y_j) = \begin{cases} -1 & \text{if } y_i = y_j \\ +1 & \text{if } y_i \neq y_j \end{cases} \tag{2.19}$$

For two neighboring cells having the same label the function value is negative, which leads to a higher probability, whereas two different labels decrease the probability. We finally arrive at the prior probability distribution of our MRF:

$$p(\mathbf{y}) = \frac{1}{Z} \exp\left(-\beta \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j) \right) \tag{2.20}$$

Now we can put it all together. Inserting $p(\mathbf{x} \mid \mathbf{y})$ from Eq. (2.16) and $p(\mathbf{y})$ from Eq. (2.20) in $p(\mathbf{y}, \mathbf{x}) = p(\mathbf{x} \mid \mathbf{y}) p(\mathbf{y})$, and setting $\phi(y_i, x_i) := \sum_{d=1}^{D} \frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} + \log\left(\sqrt{2\pi}\sigma_{y_i}^{[d]}\right)$, we get the final form of the joint probability distribution:

$$p(\mathbf{y}, \mathbf{x}) = \exp\left(-\sum_{i=1}^{M} \phi(y_i, x_i) \right) \frac{1}{Z} \exp\left(-\beta \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j) \right) \tag{2.21}$$

$$= \frac{1}{Z} \exp\left(-\sum_{i=1}^{M} \phi(y_i, x_i) - \beta \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j) \right) \tag{2.22}$$

This joint probability distribution over $\mathcal{X}$ and $\mathcal{Y}$ together with the graph shown in Fig. 2.4 defines our MRF. Eq. (2.22) points out the two components of the MRF. On the one hand, a feature-dependent component in the form of a potential function $\phi(y_i, x_i)$ defined over all labels with their associated feature vectors. On the other hand, a context-dependent potential function $\psi(y_i, y_j)$ defined over all pairs of neighboring labels. It is easy to see that our MRF is consistent with Definition 1. The set of random variables is partitioned into a set of label variables $\mathcal{Y}$ and a set of feature variables $\mathcal{X}$. $p(\mathbf{y}, \mathbf{x})$ is always greater than zero since an exponential function is always greater than zero, and the Markov property is satisfied by using the potential functions ϕ and ψ on the pairwise cliques defined by the graph.

However, the MRF model has some shortcomings. E.g., we have to implicitly model the probability distribution $p(\mathbf{x})$ of the features (for all class labels). This distribution can be very complex, and making simplifying assumptions, like modeling it as a Gaussian distribution, can be a too strong restriction. The pairwise factor ψ encourages agreement, but the way in which it does so is inflexible. The probability that two neighboring cells have the same label should be higher when the corresponding features are similar, and vice versa, but ψ in this model is independent of the features $\mathbf{x}$. We will come back to these issues after introducing Conditional random fields in the next section.

2.4.2 Conditional Random Fields

In contrast to MRFs, a Conditional random field (CRF) is a discriminative model, which means that it models the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ directly. As in the last section, we start with Bayes' rule:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{p(\mathbf{x})} \tag{2.23}$$

We can describe both $p(\mathbf{y}, \mathbf{x})$ and $p(\mathbf{x})$ using the same unnormalized joint distribution $\widetilde{p}(\mathbf{y}, \mathbf{x})$:

$$p(\mathbf{y}, \mathbf{x}) = \frac{\widetilde{p}(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}} \int_{\mathbf{x}} \widetilde{p}(\mathbf{y}, \mathbf{x})} \tag{2.24}$$

$$p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x} \mid \mathbf{y}) p(\mathbf{y}) = \sum_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}) = \frac{\sum_{\mathbf{y}} \widetilde{p}(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}} \int_{\mathbf{x}} \widetilde{p}(\mathbf{y}, \mathbf{x})} \tag{2.25}$$

Eq. (2.25) follows from the law of total probability. When we insert Eq. (2.24) and (2.25) into (2.23), the normalization factor that sums over $\mathbf{y}$ and integrates over $\mathbf{x}$ cancels out, and only the normalization factor $Z(\mathbf{x}) = \sum_{\mathbf{y}} \widetilde{p}(\mathbf{y}, \mathbf{x})$ that sums over $\mathbf{y}$ remains:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{\sum_{\mathbf{y}} \widetilde{p}(\mathbf{y}, \mathbf{x})} \widetilde{p}(\mathbf{y}, \mathbf{x}) \tag{2.26}$$

$$= \frac{1}{Z(\mathbf{x})} \widetilde{p}(\mathbf{y}, \mathbf{x}) \tag{2.27}$$

When comparing Eq. (2.24) and (2.26) we see that when $\mathbf{x}$ is given, the normalization factor only sums over $\mathbf{y}$. Again, as with the MRF, we can express the (unnormalized) joint distribution $\widetilde{p}(\mathbf{y}, \mathbf{x})$ as a product of local factors defined by the associated graph, which means that two nodes of the graph are connected by an edge whenever they appear in the scope of the same factor. Since we model the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ directly, there are no potentials that involve only input variables x, but we are otherwise free in choosing the potential functions. We use the CRF model of [Lalonde *et al.* (2010)], where the set of factors is encoded as a log-linear model:

$$p(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(-\lambda \sum_{i=1}^{M} \phi(y_i, x_i) - \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j, x_i, x_j) \right) \tag{2.28}$$

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left(-\lambda \sum_{i=1}^{M} \phi(y_i, x_i) - \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j, x_i, x_j) \right) \tag{2.29}$$

$$\phi(y_i, x_i) = -\log\left(p(y_i \mid x_i)\right) \tag{2.30}$$

$$\psi(y_i, y_j, x_i, x_j) = \mathbf{1}_{\{y_i \neq y_j\}} \exp\left(-\beta (x_i - x_j)^2\right) \tag{2.31}$$

This model can be seen in analogy to Eq. (2.22) defining the MRF model. The partition function $Z(\mathbf{x})$ now additionally depends on the features $\mathbf{x}$, but as with the partition function of the MRF, we do not need to compute it for solving the classification problem, as we will see in the next section. The unary potential $\phi(y_i, x_i)$ models the feature-dependent component and the pairwise potential $\psi(y_i, y_j, x_i, x_j)$ models the context-dependent part. The influence of each of the two components is controlled by the parameter λ. It is now no longer necessary to model the feature distribution $p(\mathbf{x})$ for the unary potential and we get the conditional distribution $p(y_i \mid x_i)$ as output of the Random forest classifier. But just as well, any other classifier can be used that provides these probability values. The potential function $\phi(y_i, x_i)$ in Eq. (2.30) is chosen such that the conditional probability distribution reduces to Eq. (2.2) when settings all pairwise potentials to zero and $\lambda = 1$:

$$p_\phi(\mathbf{y} \mid \mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left(-\sum_{i=1}^{M} \phi(y_i, x_i) \right) \tag{2.32}$$

$$= \frac{1}{Z(\mathbf{x})} \exp\left(\sum_{i=1}^{M} \log\left(p(y_i \mid x_i)\right) \right) \tag{2.33}$$

$$= \frac{1}{Z(\mathbf{x})} \prod_{i=1}^{M} \exp\left(\log\left(p(y_i \mid x_i)\right)\right) \tag{2.34}$$

$$= \prod_{i=1}^{M} p(y_i \mid x_i) \tag{2.35}$$

The partition function cancels out, since $Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{i=1}^{M} p(y_i \mid x_i) = 1$.

Another big difference of the CRF model compared to the MRF is that the pairwise potential $\psi(y_i, y_j, x_i, x_j)$ now also depends on the features, while in Eq. (2.19) the same dissimilarity penalty is imposed regardless of $\mathbf{x}$. The idea behind Eq. (2.31) is that the probability that two neighboring grid cells belong to the same class is high, but if they belong to different classes, their appearance (measured by feature vector x) must also

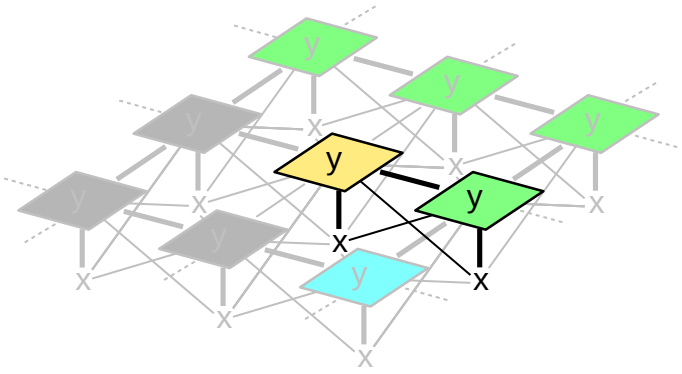


Figure 2.5: In the Conditional random field, the label y of a cell depends not only on the observed features x of its own cell, but also on the features of the neighboring cells. Highlighted are the dependencies between two neighboring cells and their associated features.

differ. The indicator function $\mathbf{1}_{\{y_i \neq y_j\}}$ outputs 1 if the neighboring labels y_i and y_j are different, and zero if they are equal. The potential is the highest if the labels are unequal but the corresponding feature vectors x_i and x_j are the same, and it exponentially decreases with the squared difference of the feature vectors. The parameter β is a so-called contrast-normalization constant as suggested in [Boykov and Jolly (2001)]. In general, the pairwise potential in a CRF can take into account all feature vectors in $\mathbf{x}$, but as with the class labels we limit the scope to the neighboring grid cells. Fig. 2.5 shows these additional dependencies as represented by the additional edges in the corresponding graph structure, highlighted for two neighboring grid cells.

To conclude the sections about MRFs and CRFs we want to emphasize the differences between generative models like the MRF and discriminative models like the CRF in the context of classification. Since the generative and the discriminative model can be converted into each other using Bayes' rule, one might think that it does not really matter which model is used. However, the two models can only then be exactly converted into each other when we use the true distributions underlying the data. But precisely because we do not know the true distributions but have to make assumptions and approximations, the CRF model and the MRF model are different in practice. The CRF only models the conditional probability distribution $p(\mathbf{y} \mid \mathbf{x})$, but this is all that is needed for classification. The conditional distribution *discriminates* directly between the different labels. The MRF, on the other hand, is a full probabilistic model of all variables, and as such it also includes a model of the feature distribution $p(\mathbf{x})$, which may be very difficult to formulate, since the dimensionality of the feature vectors can be very high and there may be complex dependencies between the vector elements. The advantage of the generative model, however, is that it can cope with partially labeled or completely unlabeled training data, and even new samples can be generated by the model, but this is not required in

our classification problem. So, in a nutshell, using the MRF we have to make more assumptions than are necessary to solve the classification problem, whereas in the CRF we only model the conditional distribution and do not model the feature distribution, which may be very complex and is not required for the purpose of classification anyway.

For further informations about CRFs, we refer the interested reader to the excellent introduction in [Sutton and McCallum (2012)], and also in [Nowozin and Lampert (2011)] with a focus on applications in computer vision.

2.4.3 MAP Inference

Now that we have described the terrain grid and the relationships between observed features and class labels through a probabilistic graphical model, we want to solve the classification problem. That is, given the observed features $\mathbf{x}$, we want to find label configurations $\mathbf{y}^*$ of maximum probability. This is also known as MAP (**M**aximum **A** **P**osteriori) inference.

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) \tag{2.36}$$

For a grid with M cells and K class labels there are K^M different possible label configurations $\mathbf{y} \in Y^M$. Even for small grids and few class labels it is not feasible to consider all the exponentially many possible configurations of terrain labels in finding the optimal solution. There exist several heuristics for MAP inference trading off an algorithmic property — like generality, optimality, or determinism — for speed. [Nowozin and Lampert (2011)] present some of the most common methods involving local search, graph cuts [Boykov *et al.* (2001)], or simulated annealing [Kirkpatrick *et al.* (1983); Černý (1985)]. We use a variant of simulated annealing using Gibbs sampling, which yields an approximately optimal solution. It was first introduced by [Geman and Geman (1984)], and was successfully used for image classification [Berthod *et al.* (1996)], image segmentation [Deng and Clausi (2004)], and also for terrain classification [Häselich *et al.* (2011)]. We will see that this method is particularly well suited for our needs. Since simulated annealing works by minimizing energy terms, we first show how to formulate our problem as an energy minimization problem, and then discuss the algorithm in detail.

Defining a problem in terms of energy functions and finding the optimal solution by minimizing these energies is a popular technique, especially in computer vision. The term *energy* is used because some of these methods originate in statistical mechanics. In contrast to the probabilistic interpretation where we are interested in solutions which yield high probabilities, in the energy formulation we are looking for solutions bringing a system in a state of low energy. With the special form of Eq. (2.22) for the MRF and Eq. (2.28) for the CRF the connection between probability and energy can be easily seen. In finding the optimal solutions $\mathbf{y}^*$, maximizing the probability and minimizing the energy is equivalent.

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) \tag{2.37}$$

$$= \arg\max_{\mathbf{y}} \frac{1}{Z(\mathbf{x})} \exp\left(-\sum_F E_F(\mathbf{y}_F, \mathbf{x})\right) \tag{2.38}$$

$$= \arg\min_{\mathbf{y}} \sum_F E_F(\mathbf{y}_F, \mathbf{x}) \tag{2.39}$$

So instead of maximizing the probability $p(\mathbf{y} \mid \mathbf{x})$ of the CRF, our goal is to minimize the sum of the energies E_F over all factors F. The same is true for MRFs, since $\arg\max_{\mathbf{y}} p(\mathbf{y}, \mathbf{x}) = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x})$ (see Eq. (2.10)). Note that we do not need to compute the partition function Z anymore, since it is constant for a given $\mathbf{x}$.

The energy equivalents of the MRF factors (see Eq. (2.16) and Eq. (2.20)) are:

$$E_1^{\text{MRF}}(\mathbf{y}, \mathbf{x}) = \sum_{i=1}^{M} \sum_{d=1}^{D} \frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]2}} + \log\left(\sqrt{2\pi}\sigma_{y_i}^{[d]}\right) \tag{2.40}$$

$$E_2^{\text{MRF}}(\mathbf{y}) = \beta \sum_{(i,j)\in\mathcal{C}_2} \psi(y_i, y_j) \tag{2.41}$$

Here, we denote the energy of the unary potential with E_1, and that of the pairwise potential with E_2. In the same way, the energy equivalents of the CRF factors (see Eq. (2.28), Eq. (2.30), and Eq. (2.31)) are:

$$E_1^{\text{CRF}}(\mathbf{y}, \mathbf{x}) = -\lambda \sum_{i=1}^{M} \log\left(p(y_i \mid x_i)\right) \tag{2.42}$$

$$E_2^{\text{CRF}}(\mathbf{y}, \mathbf{x}) = \sum_{(i,j)\in\mathcal{C}_2} \mathbf{1}_{\{y_i \neq y_j\}} \exp\left(-\beta(x_i - x_j)^2\right) \tag{2.43}$$

Note that the energy term E_2 of the CRF additionally depends on the features $\mathbf{x}$. The parameters λ and β can be learned from training data using the method of cross validation [Stone (1974)].

Now, to bring the terrain grid in a state of low energy, we use a two-stage approach. In the first stage, we minimize the unary energy potentials $E_1(y_i, x_i)$ for every grid cell individually, which corresponds to a cell-wise classification. Then, in the second stage, we iteratively change the label configuration until the total energy of the terrain grid is close to the minimum. So the first stage provides an initialization of the grid for the second stage.

$$E_1^{\text{MRF}}(y_i, x_i) = \sum_{d=1}^{D} \frac{(x_i^{[d]} - \mu_{y_i}^{[d]})^2}{2\sigma_{y_i}^{[d]\,2}} + \log\left(\sqrt{2\pi}\sigma_{y_i}^{[d]}\right) \tag{2.44}$$

$$E_1^{\text{CRF}}(y_i, x_i) = -\lambda \log\left(p(y_i \mid x_i)\right) \tag{2.45}$$

Algorithm 1 Initialization with minimal energy E_1 (Stage I)

```
1: INITIALIZATION(x)
2: Input:
3:     x: observed feature vectors
4: Output:
5:     y: label configuration
6: Algorithm:
7: for i = 1 to M do
8:     y_i ← arg min_y E_1(y, x_i)
9: end for
10: return y
```

In the MRF the initialization corresponds to assigning to each grid cell the label for which the observed feature vector best fits the corresponding mixture of Gaussians learned from the training set. In the CRF the assignment of labels simply corresponds to the output of the Random forest. For both models, the initialization yields a grid for which the energy E_1 is minimal. In order to also incorporate the pairwise energy potentials E_2 we make use of simulated annealing, a heuristic optimization method.

Simulated annealing was independently described by [Kirkpatrick *et al.* (1983)] and [Černý (1985)]. The method is inspired by the process of annealing in metallurgy. There, a material is first heated and then cooled down again. The cooling is done in such a gradual way that the atoms have time to arrange themselves and form crystals. The physical system then reaches a low energy state close to the optimum. In our case, the system consists of the terrain grid with assigned class labels. The optimization algorithm starts with an initialization of the grid, which might be random, but a better alternative is to use a good estimate as provided by Algorithm 1. We then wander the search space in order to find a state of low energy, where each iteration of the algorithm yields another sample from the distribution of label configurations. Thereby, the temperature corresponds to the probability of accepting a worse solution in an iteration step. Also accepting worse solutions is an important ingredient of the algorithm, since otherwise we could get stuck in a local minimum early on. So in the beginning of the algorithm, when the temperature is high, the energy differences do hardly affect the sampling process. Then, when the temperatures decreases, the procedure reduces to a greedy algorithm that only moves

towards better solutions.

The question remains of how to draw (approximate) samples from the distribution in each iteration step. The original formulation of simulated annealing uses the Metropolis-Hastings algorithm [Hastings (1970)], whereas we use the Gibbs sampler as introduced by [Geman and Geman (1984)]. In its basic form, the Gibbs sampler is a special case of the Metropolis-Hastings algorithm. It belongs to the class of Markov chain Monte Carlo (MCMC) methods, which generate approximate samples from a joint probability distribution over many variables. This multivariate distribution may be very complex, and to draw samples from it can be very difficult, if not impossible. MCMC methods help here by constructing a Markov chain with the desired probability distribution as its stationary distribution. Where the Metropolis-Hastings algorithm proposes a sample candidate in each iteration, and accepts or rejects it with a certain probability, the Gibbs sampler accepts all samples. The key idea of the sampler is that, although it might be difficult to sample from the joint distribution, drawing samples from the conditional distribution of a single variable conditioned on the other variables is usually a lot easier.

$$y_i^{(t+1)} \sim p(y_i \mid \{y_j^{(t)} : j \neq i\}, \mathbf{x}^{(t)}) \tag{2.46}$$

$$= p(y_i \mid \{y_j^{(t)}, x_j^{(t)} : j \in \mathcal{N}_i\}) \tag{2.47}$$

So at iteration step $t+1$ each variable y_i is sampled from the one-dimensional distribution where all other variables are fixed. With the neighborhood system $\mathcal{N}$ the variables are then conditioned only on the variables of the neighboring cells. The conditional distribution can be expressed with the help of the corresponding energy potentials. In the simulated annealing scheme, the probability of the state of a (sub-)system with temperature T having energy E is described by a Gibbs distribution:

$$p_T(E) \propto \exp\left(-\frac{E}{T}\right) \tag{2.48}$$

In order to compute the energy term E of a single grid cell i we need the unary potential E_1 (see Eq. (2.40) and Eq. (2.43)) and the corresponding energy term E_2 for a single grid cell, which is the sum of the pairwise potentials of the neighborhood:

$$E_2^{\text{MRF}}(\mathbf{y}, i) = \beta \sum_{j \in \mathcal{N}_i} \psi(y_i, y_j) \tag{2.49}$$

$$E_2^{\text{CRF}}(\mathbf{y}, \mathbf{x}, i) = \sum_{j \in \mathcal{N}_i} \mathbf{1}_{\{y_i \neq y_j\}} \exp\left(-\beta (x_i - x_j)^2\right) \tag{2.50}$$

Then, the total energy of a grid cell is the sum of E_1 and E_2:

$$E^{\text{MRF}}(\mathbf{y}, x_i, i) = E_1^{\text{MRF}}(y_i, x_i) + E_2^{\text{MRF}}(\mathbf{y}, i) \tag{2.51}$$
$$E^{\text{CRF}}(\mathbf{y}, \mathbf{x}, i) = E_1^{\text{CRF}}(y_i, x_i) + E_2^{\text{CRF}}(\mathbf{y}, \mathbf{x}, i) \tag{2.52}$$

In summary, an iteration step of the optimization algorithm consists of changing the label of each grid cell by sampling from the Gibbs distribution in Eq. (2.48) using the energy terms in Eq. (2.51) and (2.52). After each iteration, we can then compute the new total energy $E(\mathbf{y}, \mathbf{x})$ of the whole grid (see Eq. (2.40) to (2.43)):

$$E^{\text{MRF}}(\mathbf{y}, \mathbf{x}) = E_1^{\text{MRF}}(\mathbf{y}, \mathbf{x}) + E_2^{\text{MRF}}(\mathbf{y}) \tag{2.53}$$
$$E^{\text{CRF}}(\mathbf{y}, \mathbf{x}) = E_1^{\text{CRF}}(\mathbf{y}, \mathbf{x}) + E_2^{\text{CRF}}(\mathbf{y}, \mathbf{x}) \tag{2.54}$$

The optimization algorithm in pseudocode is shown in Alg. 2. First, the terrain grid is initialized using Alg. 1. We then compute the initial energy of the grid, and set the temperature to an initial value T_0. After each iteration we compute the new energy of the grid and compare it with the previous one. If the difference is smaller than or equal to a threshold, we stop (see lines 25 to 27), since we then assume that not much will change after that and we have found a state near the optimum. Otherwise, we continue with the next iteration, but we set a limit for the total number of iterations, so that we can bound the maximal computation time. Iterating through the cells sequentially is called a deterministic sweep. An alternative is to iterate through the cells in a random order to reduce the degree of correlation. At the end of each iteration the temperature is decreased by multiplying it with the cooling rate $c < 1$.

In each iteration we apply the aforementioned sample technique to assign new labels to each cell that has an associated feature vector. We therefore need to compute for each grid cell the corresponding energies E_k for any class label k (see lines 16 to 19); and we thus obtain for each cell K energy values, where K is the number of terrain classes. Now, rather than taking the label with the lowest energy, we sample from the Gibbs distribution in line 20. So the label with the lowest energy is taken most likely, but also the other labels can be selected with some probability. Note that the function $\bar{E}(y)$, and thus also the function $\exp(^{-\bar{E}(y)}/_T)$ are only defined for $y \in \{1, \ldots, K\}$. Fig. 2.6 illustrates the effect of different temperature values T on the Gibbs distribution. For high temperatures, the probability of states with similar energy does not change much, and the next sample can result in a lower or higher energy with similar probability. For really high temperatures all label configurations are almost equally likely. On the other hand, for low temperatures the probability mass concentrates at low energy states. Then, it is very unlikely that the next iteration yields a state with an increased amount of energy. This means that the changes in the label configuration can be very large at the beginning of the optimization process, but will get less with the decrease of the temperature. To avoid getting stuck in a local minimum, the temperature decrease has to be slow enough.

Algorithm 2 Gibbs sampling in a simulated annealing scheme (Stage II)

GIBBSSAMPLING(T_0, t, c, N, $\mathbf{x}$)
Input:
  T_0: initial temperature
  t: threshold for energy difference
  c: cooling rate ($c < 1$)
  N: maximum number of iterations
  $\mathbf{x}$: observed feature vectors
Output:
  $\mathbf{y}$: label configuration
Algorithm:
$\mathbf{y} \leftarrow$ INITIALIZATION($\mathbf{x}$)
$E_{\text{old}} \leftarrow E(\mathbf{y}, \mathbf{x})$
$T \leftarrow T_0$
for $j = 1$ to N **do**
  for $i = 1$ to M **do** ▷ For all grid cells
    for $k = 1$ to K **do** ▷ For all classes
      $y_i \leftarrow k$
      $E_k \leftarrow E(\mathbf{y}, \mathbf{x}, i)$ ▷ Compute energy E_k for grid cell i having label k
    end for
    $y \sim \exp\left(-\frac{\bar{E}(y)}{T}\right)$, with $\bar{E}(k) = E_k$ ▷ Sample from Gibbs distribution
    $y_i \leftarrow y$
  end for
  $E \leftarrow E(\mathbf{y}, \mathbf{x})$
  $\Delta E \leftarrow |E - E_{\text{old}}|$
  if $\Delta E \leq t$ **then** ▷ Stop when the energy difference is small enough
    break
  end if
  $E_{\text{old}} \leftarrow E$
  $T \leftarrow c \cdot T$
end for
return $\mathbf{y}$

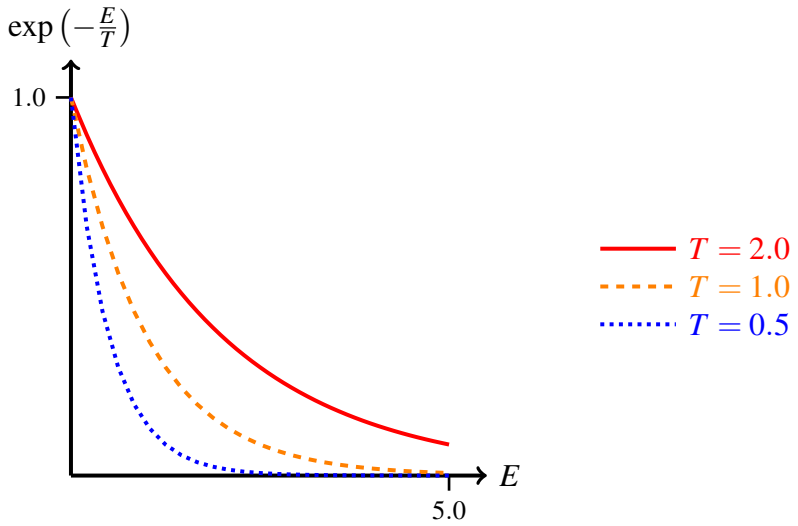


Figure 2.6: Influence of different temperature factors T on the probability of the system being in a state with energy E. The lower the temperature, the more probability mass is concentrated at low energy states.

2.5 Monte Carlo Localization

In the previous sections we presented and discussed several models and algorithms that we use for terrain classification. In the final section of this chapter we turn to another topic: robot localization. We assume that the robot has an internal map of its environment containing information about terrain and obstacles. Then, the problem of robot localization is to estimate the robot's pose $s = (x, y, \theta)$, with position (x, y) and orientation θ, based on odometry and sensor measurements. There are two types of localization problems: global and local localization. In global localization, the robot only knows that it has to be somewhere on the map. When the robot has localized itself, local localization is the task of keeping track of the robot's position as it moves around. The so-called kidnapped-robot problem, where the robot is carried to an arbitrary position without the robot being aware of the change, emphasizes the need for the ability of global localization even when the robot knows its initial position. Instead of getting kidnapped, there is always the possibility that the localization fails completely, or the robot has to reboot its system. We will now discuss Monte Carlo localization (MCL) [Dellaert *et al.* (1999)], which can cope with both types of localization problems.

The various existing localization methods can be categorized by how they represent the estimate of the robot's current state. A Kalman filter [Kalman (1960)] represents the estimated state assuming a Gaussian distribution. So there is only a single hypothesis about where the robot is located, and the estimated position is given by the mean and the covariance matrix. Thus, the Kalman filter can neither solve the global localization problem nor deal with ambiguities. Grid-based localization methods [Burgard *et al.* (1996)]

can represent arbitrarily complex probability densities by using histograms. However, the accuracy of the methods depends on the grid resolution, and the memory requirements grow with the size of the map, and can thus get very high. MCL uses a particle filter for localization, where each particle represents a hypothesis for a potential robot pose. Thus, the probability density function of the robot's state is implicitly represented by a set of samples (or particles) drawn from it. Hence, MCL combines the advantages of the other two methods. It has the ability to represent multi-modal distributions like the grid-based methods, but it is more accurate since it uses a continuous state space like the Kalman filter. Besides, MCL is very easy to implement.

In global localization, when the robot knows nothing about where it is located, the particles are spread randomly across the entire state space; this represents a state of maximum uncertainty. On the other hand, when we know the initial position of the robot, the particles are distributed around this point. Since the particles represent hypothetical *poses* of the robot, each particle not only has a certain position but also an associated orientation. At each time step t, the MCL algorithm updates the robot's belief about its pose, represented by the set of particles $\mathcal{P}_{t-1}$, by considering the current motion u_t and the current measurements z_t, and returns the new set of particles $\mathcal{P}_t$. Thereby, the Markov property is assumed, which means that the future state of the robot only depends on the current state, and not on previous ones.

Alg. 3 shows the MCL algorithm in pseudocode, based on [Thrun *et al.* (2005)]. Herein, a particle is defined as a pair (s, w) of pose s and a weight factor w. The algorithm mainly consists of three steps: a motion update (see function MOTIONUPDATE(u_t, s_{t-1}) in line 12), a sensor update (see function SENSORUPDATE$(z_t, s_t, \mathcal{M})$ in line 13), and a resampling step (see lines 15 to 18). We will describe these steps in the following.

Motion update

When the robot moves at time step t, we also move each particle s_{t-1} based on the control input u_t at time t, where the control input usually is the robot's estimated motion according to its odometry. Thus, when the robot moves forward, so do the particles, whereas the direction of motion may differ, since each particle has its own orientation. Because the odometry information is not perfect, we can predict the movement of the robot only approximately. To represent this, we add Gaussian noise to the movement of the particles. In the context of Bayesian filtering, the motion step is also referred to as prediction step, as we predict the future state of the robot without relying on sensor observations. If we would only use the motion update, the uncertainty regarding the pose of the robot would grow constantly, so that a localization of the robot would become impossible.

Algorithm 3 Monte Carlo localization using N particles

MONTECARLOLOCALIZATION($\mathcal{P}_{t-1}, u_t, z_t, \mathcal{M}$)
Input:
  $\mathcal{P}_{t-1}$: Particles at time $t-1$
  u_t: Control input at time t
  z_t: Measurement at time t
  $\mathcal{M}$: A priori map
Output:
  $\mathcal{P}_t$: Particles at time t
Algorithm:
$\mathcal{P}_t \leftarrow \emptyset$
for $i = 1$ to N **do**
  $s_t^{[i]} \leftarrow$ MOTIONUPDATE$(u_t, s_{t-1}^{[i]})$ ▷ Move particles
  $w_t^{[i]} \leftarrow$ SENSORUPDATE$(z_t, s_t^{[i]}, \mathcal{M})$ ▷ Update particle weights
end for
for $i = 1$ to N **do**
  Sample $j \sim w_t^{[j]}$, with $j \in [1, ..., N]$ ▷ Draw particles with replacement
  $\mathcal{P}_t \leftarrow \mathcal{P}_t + \left(s_t^{[j]}, N^{-1}\right)$ ▷ New distribution of particles
end for
return $\mathcal{P}_t$

Sensor update

In addition to position and orientation, each particle has an associated weight. When the robot senses something at time step t, we update this weight w_t based on the sensor measurement z_t at time t, the position s_t of the particle from the motion update, and the constant map $\mathcal{M}$. We set this weight as the probability that the robot would sense z_t if it had the position and orientation s_t of the corresponding particle. For example, when the robot senses asphalt, then particles that are in front of a patch of grass according to the map have a low probability of being the true robot's pose, and thus would be weighted low.

Resampling

The resampling step is also called correction step, since we correct our prediction based on new observations, which are now represented as particle weights. When there is a total of N particles, we now generate a new set of particles by drawing N particles with replacement, where the probability to draw a specific particle is proportional to its weight. So some particles are drawn more than once, while others are not drawn at all. In general, particles with higher weights are drawn more often than particles with low weights. The new set of particles represents our new belief about the robot's pose. In contrast to the motion step, where the uncertainty increases, the resampling step reduces it by incorporating new information.

Localization consists of a continuous prediction-correction cycle, and the particles should ultimately converge towards the actual pose of the robot. Finally, if we need a single estimate for the robot pose, then to choose the particle with the maximum weight is not a good choice, since this position can jump quite a bit between two time steps. A more robust alternative is to estimate the pose of the robot using the weighted mean of all particles within a certain radius of the particle with the maximum weight.

Chapter 3

Hardware

For all data acquisition and experiments the outdoor robot *Thorin*, which is presented in Sec. 3.1, was used. The main sensors we use in our work are an AVT Marlin F-046C Color Camera and a Nippon Signal FX6 3D LiDAR. We will describe the camera briefly in Sec. 3.2.1. The 3D LiDAR is described in Sec. 3.2.2 in a little more detail, as it is a rather uncommon sensor, and we provide a basic understanding of its functional principles. At the end of the chapter, in Sec. 3.3, we introduce the necessary coordinate systems of the robot and its sensors.

3.1 Robot Platform

The outdoor robot *Thorin* (see Fig. 3.1) is based on a remote-controlled 1:8 model of a monster truck, and was developed and built at our department. It is equipped with a Mini-ITX computer running Ubuntu Linux, featuring a 2.26 GHz Core 2 Duo CPU and a solid-state drive. An additional 32-bit microcontroller provides real-time control. Odometry is provided by encoders inside the wheels, which count the wheel revolutions, and a magnetic compass.

Figure 3.1: Outdoor robot *Thorin* with an AVT Marlin F-046C color camera and a Nippon Signal FX6 3D LiDAR

3.2 Sensors

The robot senses its environment using two sensors: an AVT Marlin F-046C color camera and an FX6 3D LiDAR from Nippon Signal. The camera is only used for terrain classification, and its technical details are described in Sec. 3.2.1. The 3D LiDAR is used for plane detection, obstacle detection, and terrain classification. Technical details and its measurement system are described in Sec. 3.2.2.

3.2.1 Camera

The AVT Marlin F-046C from Allied Vision Technologies GmbH is a fast and compact machine vision camera and frequently used in industrial image processing and product automation. It is equipped with a Sony CCD sensor and a FireWire interface. Resolution and frame rate depend on the used color mode. We operate the camera in color mode YUV422. Then, the picture size is 640×480 pixels, and the frame rate is 36 Hz. Actually, due to Bayer demosaicing, there are only 638×480 pixels available. The camera features both manual and automatic white balance, as well as an auto shutter and auto gain function. We enable all three auto functions, as this is necessary especially in changing lighting conditions in outdoor environments. For more information see the technical manual [AVT (2008)].

Marlin F-046C Color Camera	
Vendor	Allied Vision Tech. GmbH
Resolution	640×480 YUV422 16 bit/pixel
Frame rate	36 Hz YUV422
Interface	IEEE 1394
Pentax TV Lens	
Focal length	4.8 mm
Maximum aperture ratio	1:1.8
Minimum object distance	0.3 m

Figure 3.2: AVT Marlin F-046C Color Camera with a 4.8 mm lens

3.2.2 3D LiDAR

A LiDAR (**Li**ght RA**DAR**) works similar to a RADAR, but uses laser light instead of radio waves or microwaves. We use the LiDAR FX6, which was developed as a prototype

by Nippon Signal. It uses a pulse laser in the near-infrared range and is a time-of-flight sensor, which means that it measures the distance from the sensor to an object by measuring the time that the light needs to travel to the object and back to the sensor. The sensor can measure time intervals as small as $t_{\min} = 30\,\text{picoseconds}$, which corresponds to a lateral resolution of $d_{\min} = \frac{c \cdot t_{\min}}{2} \approx \frac{3{\cdot}10^{8}\,\text{m/s} \cdot 30{\cdot}10^{-12}\,\text{s}}{2} = 0.0045m = 4.5\,\text{mm}$.

FX6 3D LiDAR	
Vendor	The Nippon Signal Co., Ltd.
Resolution	29×59 data points
Frame rate	8 Hz or 16 Hz
Range	0.1 m $\sim$ 16 m
Scan area	50° (hor.) and 60° (vert.)
Laser class	Class 1 (near infrared)
Interface	USB 2.0

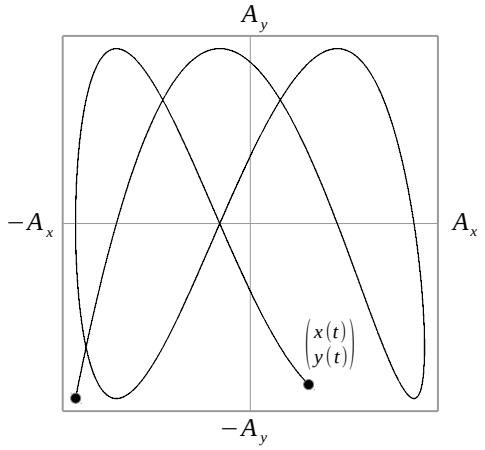
Figure 3.3: The 3D LiDAR FX6 measures distance and intensity values in a two-dimensional grid of 29×59 data points.

Theory of measurement

In a 2D laser scanner, a laser beam is deflected horizontally to get measurements in a scan plane. In the FX6, the laser beam can be deflected using a 2D scanning-mirror system called EcoScan. A moving plate with the mirror on its surface can be tilted horizontally and vertically by changing electric currents. The beam then moves along a so-called Lissajous curve (see Fig. 3.4). Lissajous figures, named after the French physicist Jules Antoine Lissajous (1822-1880), are caused by the superposition of two harmonic oscillations that are perpendicular to each other. More specifically, these figures involve graphs of parametric functions of the following form:

$$t \mapsto \begin{pmatrix} A_x \sin(\omega_1 t + \phi_1) \\ A_y \sin(\omega_2 t + \phi_2) \end{pmatrix}, \quad t \geq 0 \tag{3.1}$$

The curve is bounded by the rectangle $[-A_x, A_x] \times [-A_y, A_y]$, with the amplitudes A_x and A_y. The appearance of the curve mainly depends on the frequency ratio ω_1/ω_2 and the phases ϕ_1 and ϕ_2. The functions are exactly then periodically if the frequency ratio



$$x(t) = A_x \sin\left(2\pi f_x\, t - \frac{\pi}{2}\right) \tag{3.2}$$

$$y(t) = A_y \sin\left(2\pi f_y\, t - \frac{\pi}{2}\right) \tag{3.3}$$

A_x : Maximum scan angle in the x-direction
A_y : Maximum scan angle in the y-direction
f_x : Scan frequency in the x-direction
f_y : Scan frequency in the y-direction

Figure 3.4: The FX6 scans along a Lissajous figure. By measuring at specific time points t a regular grid of measurements is obtained.

is rational; then a closed curve is obtained in finite time. Otherwise, the rectangle will be filled completely by the curve, for $t \to \infty$ [Wikipedia (2014)]. For the FX6, the amplitudes A_x and A_y are determined by the maximum scan angle in the x- and y-direction. With $\omega_1 = 2\pi f_x$ and $\omega_2 = 2\pi f_y$, the frequency ratio results to $\omega_1/\omega_2 = f_x/f_y$, where f_x and f_y are the scan frequencies in the x- and y-direction. This ratio is chosen so that it is rational, and thus the function is periodic. Furthermore, it applies $\phi_1 = \phi_2 = -\pi/2$, and so the curve starts at $t = 0$ at the lower left corner $(x(0), y(0))^T = (-A_x, -A_y)^T$ (see Fig. 3.4). By measuring at specific times a regular grid of measurement points is obtained. With the measured distance and the known scan angle, the 3D coordinates of the measured object point can be determined for each measurement point.

This grid of measurement points is generated at a frequency of 16 Hz. A major disadvantage of the sensor is the low resolution of this grid with only 29×59 data points. A great advantage on the other side is that the sensor is hardly affected by ambient light. The photo sensor with which the returned light is received has an extremely high sensitivity. In order to keep the level of received backscattered light high, and the level of noise produced by ambient light low, both the direction of the laser beam and the field of view of the photo diode are concentrated on each measuring point respectively. The photo diode is also used to determine an intensity value for each measuring point, which indicates the proportion of emitted light which arrives back at the sensor. According to the manufacturer [Nippon (2008)], the sensor can even deal with high sunlight with 100 000 Lux and above. The empirical studies in [Rauscher *et al.* (2014)], as well as our everyday experience with this sensor, confirm the FX6 as a reliable sensor, suitable for the prevailing light conditions outdoors. The successor model FX8 comes with a slightly higher resolution of 53×33 data points when operating at 16 Hz, and with a much higher resolution of 97×61 data points when operating at 4 Hz ([Nippon (2010)]).

3.3 Coordinate Frames

The following figure shows the necessary coordinate systems of the robot and its sensors. Note that all coordinate systems are right-handed systems, as usual in robotics.

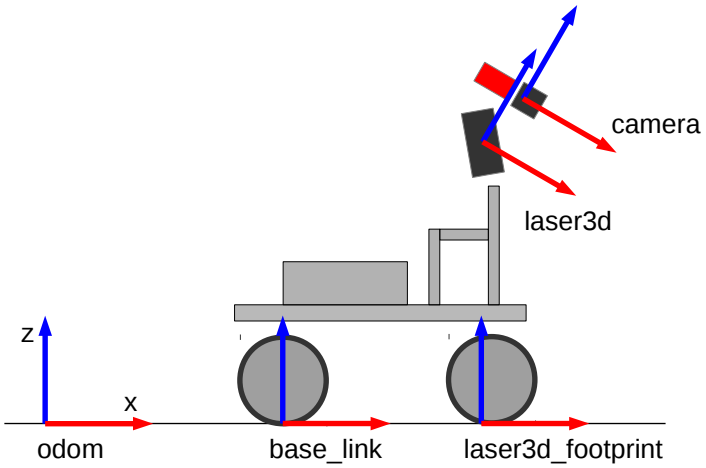


Figure 3.5: The coordinate frames of the robot and its sensors. The red forward-pointing arrows indicate the x-axes, the blue up-pointing arrows indicate the z-axes. The y-axes are pointing away from the viewer and are not shown.

The coordinate system of the robot is referred to as *base_link*. It is located on the ground just between the two rear wheels, with the x-axis pointing forward and the z-axis pointing upward. The *odom* frame is defined by the pose of the robot at the time of the initialization of the system. The origin of *base_link* expressed in coordinates of *odom* indicates the current position of the robot relative to the starting position. We will need this transformation in Chapter 6 when mapping the environment and localizing the robot in a given map. *laser3d* refers to the coordinate system of the 3D LiDAR and *laser3d_footprint* is its projection onto the ground plane. We will see how to determine the pose of the ground plane relative to the LiDAR in Sec. 4.3.2. The transformation between *base_link* and *laser3d_footprint* is assumed to be fixed, and is measured by hand. Finally, we also need to know the intrinsic camera parameters and the transformation between the *laser3d* and the *camera* frame to be able to relate observations made by the two sensors. We therefore use the calibration method described in Sec. 4.5.

Chapter 4

Terrain Classification on Fused 3D LiDAR and Camera Data

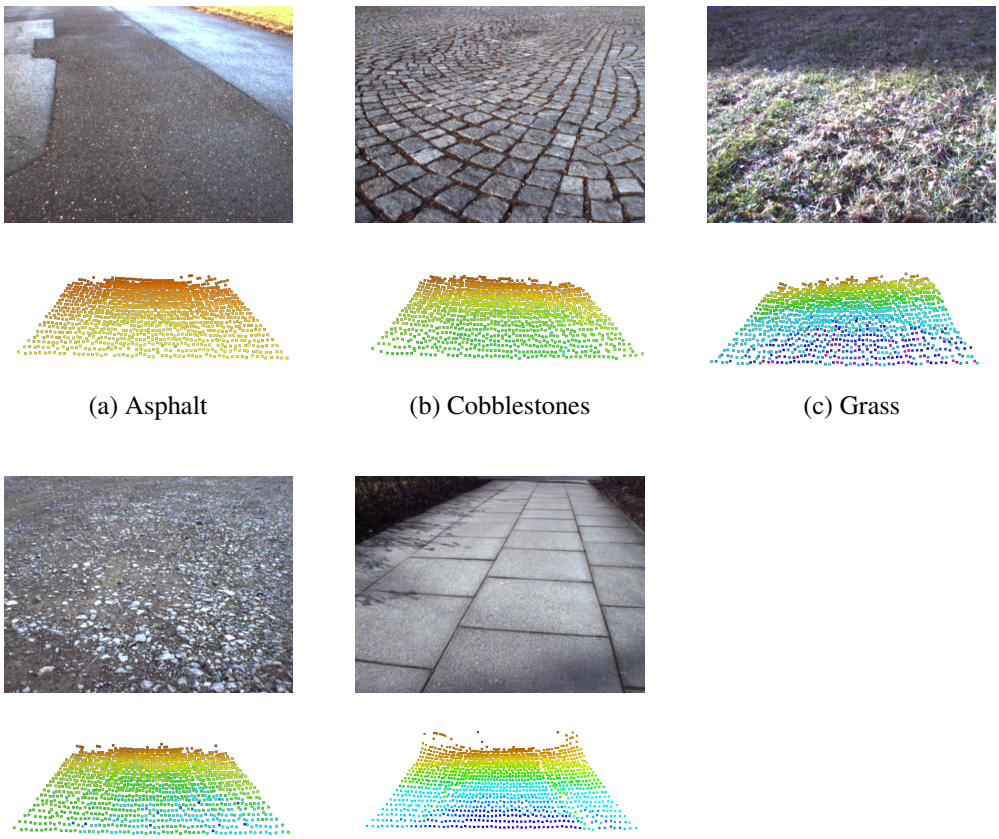
In this chapter, we present our basic terrain classification method using the 3D LiDAR and the color camera described in the previous chapter. The terrain in front of the robot is divided into a grid, and each grid cell is classified individually using the sensor measurements. For grid cells where data of both sensors are present, the classification results are fused accordingly.

The chapter is based on the papers "3D LIDAR- and Camera-Based Terrain Classification Under Different Lighting Conditions" [Laible *et al.* (2012)] and "Terrain Classification With Conditional Random Fields on Fused 3D LIDAR and Camera Data" [Laible *et al.* (2013)] and contains parts that were taken verbatim from these works.

4.1 Introduction

A mobile robot that navigates in outdoor environments is faced with challenges quite different from those occurring in indoor scenarios, such as factories and office buildings. A characteristic feature of such indoor environments is the geometric structure. This structure simplifies autonomous navigation of a mobile robot. The detection of walls and other obstacles is often sufficient for localization and path planning. This means that it is sufficient to detect that there *is* something, without having to classify it. On the other hand, outdoor environments often lack structure. In order to still enable a safe and efficient navigation, a comprehensive semantic perception of the environment is essential. Of particular interest is the surrounding terrain. Outdoor terrain varies and can be uneven or impassable. The robot must be able to decide whether the terrain ahead is passable easily, passable with caution, or whether it is better to avoid this terrain and to plan another path. The detection and correct classification of the surrounding terrain is therefore a fundamental ability of a mobile robot in outdoor navigation.

A major challenge in the field of outdoor robotics is the changing lighting conditions. The texture of the ground may look very different depending on time of day and the currently prevailing weather conditions. Our robot is equipped with a low-resolution



(a) Asphalt (b) Cobblestones (c) Grass

(d) Gravel (e) Tiles

Figure 4.1: Images and 3D scans of the five terrain types under consideration. Scan points are colored according to their intensity values.

3D LiDAR and a color camera, and both sensors are used for terrain classification. Camera-based approaches are well studied and provide good results. A drawback of these approaches, however, is that the quality of the classification is affected by the just mentioned external factors. Laser scanners, on the other hand, are largely illumination-invariant, and 3D LiDARs, which scan an entire area at a high frame rate, provide enough information for terrain analysis. In our experiments we consider five types of terrain that are often encountered: asphalt, cobblestones, grass, gravel, and tiles. Fig. 4.1 shows example images and corresponding 3D scans of the five considered terrain types. There is great variations within the terrain classes. For example, the grass has varying height and density, some spots are covered with moss. The pattern of the cobblestones varies, and gravel and asphalt show different textures at different spots.

This chapter presents a terrain classification method that combines the results of LiDAR- and camera-based classification to cope with the aforementioned challenges. After dis-

cussing related work in Sec. 4.2, we start with LiDAR-based terrain classification in Sec. 4.3. First, by converting the range values of the LiDAR into Cartesian coordinates we get a 3D point cloud wherein a RANSAC-based method finds the ground plane on which the robot drives. This plane is divided into a grid and the grid cells are to be classified into the given terrain classes. We show how to extract features from these 3D scans that are fast to compute and yet are capable of distinguishing different terrain types. We use roughness and intensity histograms for the laser scans. Especially the distribution of the intensity values of the scan points provides characteristic features, since it results from the reflection properties of the different terrain types. The cells in which the height exceeds a threshold are marked as obstacles and are not further classified. Since the LiDAR has a very low resolution, only for cells that are closer than about two meters to the front of the robot enough laser measurements are available to provide a meaningful analysis of the terrain. Finally, Random forests are used for classification.

Sec. 4.4 presents the camera-based terrain classification. We describe two texture-based image descriptors, Local binary patterns and Local ternary patterns, and the interest-point image descriptor TSURF. Local ternary patterns are an extension of Local binary patterns, and both are basically histograms of binary-encoded intensity differences in neighboring pixels. Since only differences are considered, a certain independence from changes in illumination is achieved. TSURF is based on the famous SURF descriptor. Then, in Sec. 4.5, in order to use the data of both sensors in the same coordinate system, we project the grid cells of the ground plane onto the image to get the corresponding pixels for feature extraction and classification. For the projection we need to know the transformation between the LiDAR and the camera coordinate system, which we compute using a calibration method with a checkerboard. In grid cells where data from both sensors are available, the two classification results can then be fused. We consider four very different lighting conditions to test the independence assumption and compare the results of the different methods in Sec. 4.6, and conclude in Sec. 4.7.

4.2 Related Work

Camera-based approaches for terrain classification are well studied. There, the problem is to find efficient and discriminative representations of texture information. This has been done, for instance, in terms of co-occurence matrices, Local binary patterns, and texton-based approaches. We use the methodology of [Khan *et al.* (2011)], where good results were achieved using local image descriptors with a grid-based approach.

There exist several approaches for terrain classification that use range data in addition. In [Rasmussen (2002)] color and texture features are combined with geometric features obtained from laser data for the purpose of road detection. A method for classifying the traversability of terrain is proposed in [Happold *et al.* (2006)]. A stereo camera provides the data to learn geometric features for traversability, and color information is then used to enhance the geometric information. For a quick adaptation to different

lighting conditions color models are learned in an unsupervised fashion. In [Häselich *et al.* (2011)] they use the 3D laser Velodyne HDL-64E S2 in addition to color cameras. The high-resolution data that this laser delivers is used together with color and texture information to classify the terrain in three classes: road, rough and obstacle. They further apply a Markov random field in order to take into account the context-sensitivity of the individual terrain grid cells.

In the just mentioned approaches only the geometric information from range data is used. But most lasers also provide intensity values (also called remission or reflectance values), which indicates the proportion of the emitted light that arrives back at the laser. In [Wurm *et al.* (2009)] these intensity values are used to detect grass-like vegetation. They are able to distinguish between street and grass with an accuracy of over 99%. As an explanation of why this works so well, they state an effect well known from satellite images analysis [Myneni *et al.* (1995)], namely that chlorophyll, a green pigment found in almost all plants, strongly reflects near-IR light, such as that of a laser. Apart from the material the intensity values also depend on the distance and the angle of incidence of the laser beams.

We will show that with these values and the features presented in Sec. 4.3.3 not only grass and non-grass, but several terrain classes can be distinguished from each other. In a previous work where we presented a method for classifying plant species using a 3D LiDAR sensor and supervised learning [Weiss *et al.* (2010)] we also experienced the discriminative power of the intensity values, as the features based on these values were the most important.

4.3 3D LiDAR-Based Terrain Classification

This section describes our method for classifying terrain using only data obtained from the 3D LiDAR. Fig. 4.2 shows the basic steps of the method.

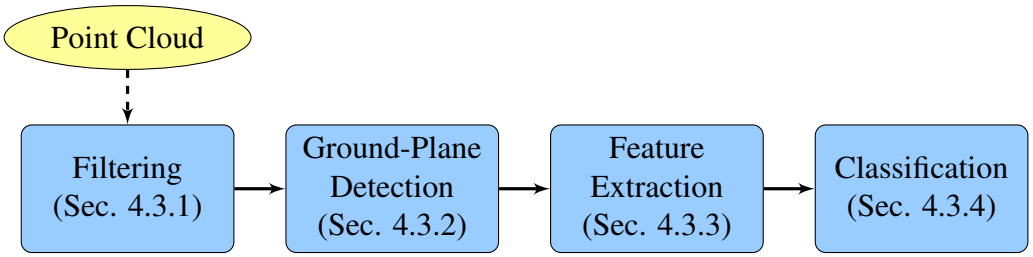


Figure 4.2: Overview of 3D LiDAR-based terrain classification

We get the data in form of a point cloud, where each data point consists of *xyz* coordinates and an intensity value (see Sec. 3.2.2). First, in the filtering step, outliers and noise are removed. In the remaining cloud, the ground plane is detected, which is the plane on

which the robot drives. This plane is then divided into a regular grid. Terrain features are extracted for each cell of this grid, based on which cells can then be classified.

4.3.1 Filtering

Even if the LiDAR as described in Sec. 3.2.2 is hardly affected by sunlight, erroneous measurements and noise are inevitable. Thus, filtering of the point cloud is a crucial preprocessing step. Some invalid measurements are indicated by an intensity value of zero. Such measurements can occur when the laser beam does not encounter any object, the object is too far away, or the surface on which the beam impinges is too shiny. Then, (almost) none of the emitted light returns back at the sensor.

Noise occurs in form of measurement points that usually have no or almost no other points in its vicinity, and do not correspond to any point of the measured scene. So to reduce the noise in the data, every point of the cloud that does not have a certain number m of neighbors within a fixed radius r is removed (see Fig. 4.3). To efficiently search for neighbors in the vicinity of a point, the cloud is organized in a k-d-tree. A k-d-tree is a binary tree, and in our case, every node of the tree corresponds to a point of the cloud. At each node the three-dimensional space is divided by a plane through that point. The space partitioning allows for efficient searches of nearest neighbors or neighboring points within a specified radius in guaranteed $\mathcal{O}(\log n)$ complexity, where n is the number of points of the cloud [Friedman *et al.* (1977)].

Filtering often removes up to 20%–25% of the point cloud, by which the already small number of measurement points is again significantly reduced. That this small number of data points is sufficient to detect the ground plane and obstacles, and to classify terrain, we are going to show in the next sections.

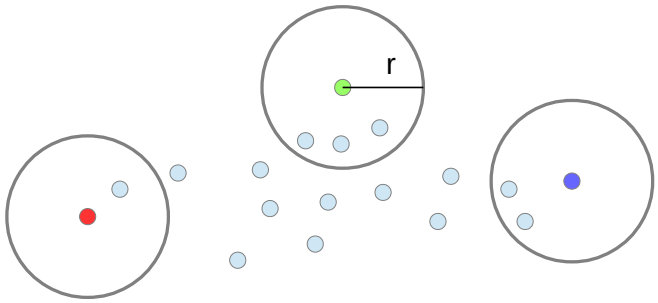


Figure 4.3: Noise removal in point clouds: If $m = 3$, then the green point in the middle would be kept, while the red and blue points would be removed.

4.3.2 Ground-plane detection

After filtering, the typical remaining cloud mainly consists of two types of points: those belonging to the ground plane, and those above (see Fig. 4.4). The ground plane is the plane on which the robot drives; and since the robot is shaking while driving, the pose of this plane relative to the LiDAR's coordinate system changes continuously. The points above the ground plane belong to obstacles by definition, since the robot should not drive there. The points of the ground plane are those of interest for terrain classification.

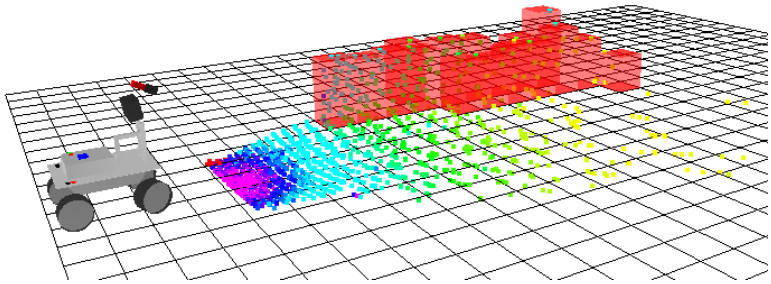


Figure 4.4: In our scenario, the ground plane typically makes up the largest part in the point cloud. The points above the plane belong to obstacles by definition, marked with red boxes.

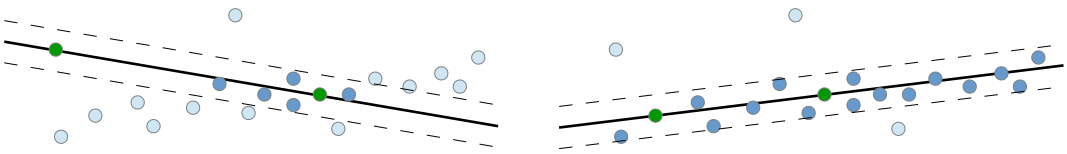
We make the assumption here that the terrain can be described by a flat surface. As the robot drives only on relatively flat terrain in our experiments and the scanning range of the LiDAR is very limited, this assumption is justified. For uneven terrain, a more general model is needed. For example, in [Wellington *et al.* (2006)] the three-dimensional space is discretized into voxels, and each voxel column is modeled as a mixture of hidden semi-Markov models to find the transition between ground and non-ground voxels. In addition, a Markov random field is used for all voxel columns to reflect the assumption that the ground height for different columns varies smoothly with distance. In [McDaniel *et al.* (2010)] a set of range data points are classified as ground or non-ground points in a two-stage approach. Therefore, the three-dimensional space is discretized into columns. In the first stage, the lowest data point of each column is marked as a potential ground point, and in a second stage, non-ground points are discarded by a Support vector machine using geometric features of the data point and the data points of its neighboring columns.

For detecting the ground plane in the point cloud the method of least squares could be used. This method computes the parameters of the plane so that the sum of the squared distances of each point in the cloud to the plane is minimized. This means that each point contributes to the solution; thus the method is very sensitive to outliers. In general, the method of least squares works well only when the data is almost free of outliers. In our

case, all points belonging to obstacles are outliers, and these points can make up a large part of the cloud; and so the method of least squares would give no good results.

RANSAC and LO-RANSAC

A better alternative is to use RANSAC (**RAN**dom **SA**mple **C**onsensus) [Fischler and Bolles (1981)], a method for estimating the parameters of a mathematical model underlying a set of observed data containing outliers. RANSAC is an iterative method, and in each iteration the minimum number of data points required to estimate the model parameters are sampled from the observed data. Fig. 4.5 illustrates this for the case of a line, where two data points are required to estimate the parameters of the model. In the case of a plane, three points have to be sampled. Points that are not farther away from the candidate line (respectively the candidate plane) than a certain threshold belong to the so-called consensus set. If the consensus set consists of sufficiently many points, the model is considered reasonable. Since this model, however, was only calculated from a minimum number of points, and even if these points are actual inliers, it may be a bad estimate of the true model, where the resulting consensus set does not include all other actual inliers. For this reason, in a variant of the method called LO-RANSAC (**L**ocal **O**ptimised RANSAC) [Chum *et al.* (2004)] the model is re-estimated using all points of the inital consensus set, that is, all potential inliers. This time it is appropriate to use the method of least squares, with the assumption that the data points now are free of outliers. In general, this leads to a better estimate with a more representative consensus set. After a specified number of iterations the candidate with the largest consensus set is said to have the highest probability to be close to the real model underlying the data.



(a) Line candidate with small consensus set

(b) Line candidate with large consensus set

Figure 4.5: Line fitting with RANSAC: By sampling two random points (green points), a candidate line is selected. Points that are closer to the line as a certain threshold belong to the so-called consensus set (dark blue points). The candidate with the largest consensus set has the highest probability to be close to the real line underlying the data.

In RANSAC, the model with the largest consensus set is selected, which is the model with the fewest outliers. This is equivalent to selecting the model $\mathcal{M}$ that minimizes the sum E in Eq. (4.1), with the error function $e = e_{\text{RANSAC}}$ in Eq. (4.2).

$$E = \sum_{p \in \mathcal{P}} e\left(\text{dist}(p, \mathcal{M})\right) \tag{4.1}$$

$$e_{\text{RANSAC}}(d) = \begin{cases} 0, & \text{if } d < t \\ 1, & \text{otherwise} \end{cases} \tag{4.2}$$

Here, $dist(p, \mathcal{M})$ is a function that computes a distance measure between a data point $p \in \mathcal{P}$, where $\mathcal{P}$ is the set of all data points, and the model $\mathcal{M}$. In our case, this is the Euclidean distance between a point of the point cloud and the selected plane. Points that are closer to the plane than a threshold t are considered inliers, while the others are considered outliers. So in effect, E simply counts the number of outliers.

MSAC

Increasing the threshold t increases the number of inliers. Since all inliers are weighted equally, this can lead to bad estimates for the model $\mathcal{M}$. Therefore, we use an extension of RANSAC, called MSAC (**M**-Estimator **SA**mple **C**onsensus) [Torr and Zisserman (2000)], which uses the error function $e = e_{\text{MSAC}}$:

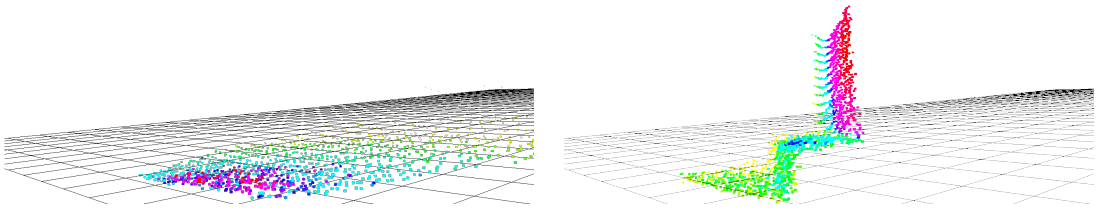
$$e_{\text{MSAC}}(d) = \begin{cases} d, & \text{if } d < t \\ c, & \text{otherwise (with } c > t) \end{cases} \tag{4.3}$$

Here, inliers also contribute to the total error E, precisely with the distance d between the inlier and the plane. Outliers still contribute with a constant factor c to the total error, whereby this factor should be greater than the threshold t. By doing so, inliers that are far away from the model are also punished, which gives more robust estimates without additional computational cost. Thus, not necessarily the candidate with the largest consensus set is selected, but the one with the smallest error E.

Temporal dependence

RANSAC only estimates one model for a given data set. So in the presence of multiple instances of the model in the data usually the dominant one is estimated, but the algorithm might as well fail to find either one. Thus, scenes with multiple planes are problematic, especially when the ground plane is not the largest, e.g. when the robot is driving near a house wall (see Fig. 4.6b).

But the house wall is an extreme case in that it stands 90 degrees to the ground. Such planes can be excluded even if the robot is shaking strongly on rough terrain. It is desirable, however, to have a more general method, which no longer considers each frame individually, but makes the assumption that the plane in the current frame has roughly



(a) (b)

Figure 4.6: Ground-plane detection: The ground plane is detected even when there are larger planes in the scene like in (b), by considering temporal dependencies.

the same pose as in the last ones. A simple yet robust method is to only consider points near a reference plane and to update the parameters of this plane every frame. We use the Hesse normal form to describe the plane: $a \cdot x + b \cdot y + c \cdot z - d = 0$, with the parameters $a, b, c, d \in \mathbb{R}$ and $(x, y, z)^T \in \mathbb{R}^3$.

The plane estimation for each frame consists of three steps:

1. From the set $\mathcal{P}$ of all points consider only those points $\overline{\mathcal{P}} = \{(x,y,z) \ : \ (x,y,z) \in P \ \wedge \ |\overline{a} \cdot x + \overline{b} \cdot y + \overline{c} \cdot z - \overline{d}| \ \leq \ d_{\min}\}$ near the reference plane $R : \overline{a} \cdot x + \overline{b} \cdot y + \overline{c} \cdot z - \overline{d} = 0$, for a fixed threshold $d_{\min}$

2. Estimate the parameters of the plane in $\overline{\mathcal{P}}$: $(a,b,c,d) = \text{MSAC}(\overline{\mathcal{P}})$

3. If $\frac{|\overline{\mathcal{P}}|}{|\mathcal{P}|} \geq m$, for a fixed threshold m, update the reference plane R:
   $\overline{a} := \overline{a} + \frac{(a - \overline{a})}{k}$, $\overline{b}$, $\overline{c}$, $\overline{d}$ resp.
   $k := k + 1$

Hereby, the parameters $d_{\min}, m \in \mathbb{R}$, with $d_{\min} \geq 0$ and $m \in [0,1]$, are user-defined thresholds, the variables $\overline{a}, \overline{b}, \overline{c}, \overline{d} \in \mathbb{R}$ are initialized with zero, and $k \in \mathbb{N}$ with one. In the Hesse normal form, $n = (a,b,c)^T$ represents the unit normal vector of the plane, pointing from the origin of the coordinate system to the plane, and $d \geq 0$ is the distance from the origin to the plane. The distance of any point $p \in \mathbb{R}^3$ to the plane is then simply calculated by the scalar product $n \cdot p$. In step 1. those points $\overline{\mathcal{P}}$ are determined that are not farther away from the reference plane R than the threshold $d_{\min}$. And only these points are used to estimate the plane parameters using MSAC in step 2. Finally, in step 3., the parameters of the reference plane are updated, but only if the proportion of the number of points $|\overline{\mathcal{P}}|$ used for plane estimation to the total number of points $|\mathcal{P}|$ is not smaller than the threshold m. This is to prevent distorting the reference plane by inaccurate plane estimates made with a very small number of points. The update formulas calculate the running average of the plane parameters, although only approximately, but memory and computation time saving [Knuth (1997)].

There are more sophisticated methods for spatio-temporal plane estimation using 3D data, like the one in [Mufti *et al.* (2012)], where RANSAC is extended to four dimensions by incorporating the time dimension, but our simple method has proven to be very robust and reliable in our experiments. For completeness we want to mention that an alternative method for detecting parameterized models like planes is using the Hough transform. In [Dube and Zell (2011)] they efficiently extract planes from depth images based on the Randomized Hough transform. [Borrmann *et al.* (2011)] evaluate different variants of the Hough transform with respect to computational cost and their applicability to detect planes in 3D point clouds.

Terrain grid

With the known parameters of the ground plane the points of the point cloud can now be transformed into the robot coordinate system so that the ground plane is equal to the x–y plane, with the x–axis pointing forward, the y–axis to the left, and the z–axis pointing upwards. Thereafter, the ground plane in front of the robot is divided into a Cartesian grid, which we will refer to from now on as the *terrain grid*. Similarly, we refer to a cell of this grid as a *terrain cell*. Each point of the point cloud belongs to that cell, above, under, or on which it is located, that is, on which its x–y projection falls. We define the height of a cell as the maximum z value of all points of this cell. A pre-classification can then be made by considering all cells with a height exceeding a chosen threshold to contain an obstacle. Among the remaining cells, only those in which there is a minimum number of data points are considered for terrain classification, since otherwise there is just not enough information to draw any meaningful conclusions.

4.3.3 Feature extraction

Having a grid of terrain cells in front of the robot, we now want to compute characteristic features for each cell using the xyz coordinates and the intensity values of the corresponding data points. Due to the low resolution of the LiDAR only very few points per cell are present. Despite this limitation the features must be discriminative enough to distinguish between different terrain types. We use some of the most discriminative features described in [Laible (2009); Weiss *et al.* (2010)], where they were used for the related task of plant-species classification, along with new more terrain-specific features. We group the features in three feature groups: height features, intensity features, and cell features.

Height features

The height of the points above the ground plane not only is suitable for distinguishing between cells with obstacles and free cells, but the height values can also be used to

distinguish between smooth and rough terrain.

1. *Maximum height h_{max}*: Maximum height of all points of a terrain cell regarding the detected ground plane. After the transformation of the point cloud this is simply the maximum z value of all points of the cell
2. *Standard deviation of height h_σ*: Standard deviation of all height values. Again, this is just the standard deviation of all z values

Intensity features

As stated in Sec. 4.2 the intensity values provide good features for terrain classification.

3. – 4. *Minimum and maximum intensities I_{min}, I_{max}*: Minimum and maximum intensity values of all points of a terrain cell
5. *Range of intensity I_r*: Difference between the minimum and maximum intensity value
6. – 8. *Mean, median and standard deviation of intensity I_μ, I_m, I_σ*: Mean, median and standard deviation of the intensity values of all points of a cell

Cell features

The intensity values not only depend on the characteristics of the material which is encountered by the laser beam, but also on the distance of the point of impact, and the angle of incidence. The number of data points of a cell also strongly depends on the distance and angle of the cell in relation to the sensor.

9. *Distance d*: Distance of the cell center to the laser origin
10. *Angle of incidence α*: Angle between the ground plane and the vector from the cell center to the laser origin
11. *Number of points N*: Number of data points belonging to the cell

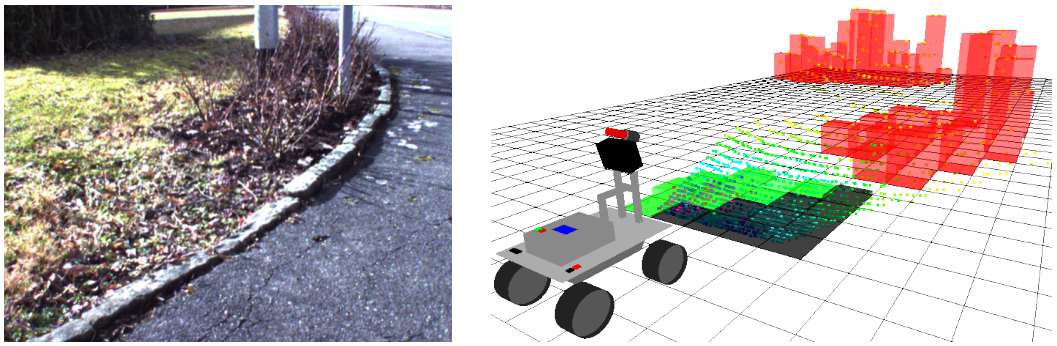
This group of features is different in that it does not describe characteristics of the terrain itself, but characteristics of the terrain cell. But theses values are necessary to interpret the intensity features correctly, and so in order to learn a good representation of the terrain, they need to also be taken into account.

Overall, we thus obtain a feature vector x_{scan} of length eleven:

$$x_{\text{scan}} = \left(h_{\text{max}}, h_\sigma, I_{\text{min}}, I_{\text{max}}, I_{\text{r}}, I_\mu, I_{\text{m}}, I_\sigma, d, \alpha, N\right) \tag{4.4}$$

4.3.4 Classification

Now that we can compute a feature vector x_{scan} for each cell of the terrain grid that has enough scan points in it, we want to use these height and intensity features to predict the correct terrain class labels of these cells. Therefore, we use Random forests, which learn the relationships between features and class labels on the basis of training data. Hereby, the training data set consists of many pairs (x_{scan}, y) of feature vectors with associated class labels y, which have to be assigned manually. The more training data are available and the more diverse the data are, the better is the generalization of the model. Then, the Random forests, as described in Sec. 2.3, assign to each grid cell and to each terrain class the probability $p(y \mid x_{\text{scan}})$ that this cell has class label y given the features x_{scan}. If we want to assign a single label to the cell, we just take the most probable one (see Fig. 4.7).



(a) (b)

Figure 4.7: Result of LiDAR-based terrain classification. Green: grass, black: asphalt, red: obstacles

4.4 Camera-Based Terrain Classification

In addition to the LiDAR-based method we also want to use a camera for terrain classification. Regardless of the classification results cameras have some advantages over LiDARs with respect to terrain classification. A camera has a wider field of view making it possible to plan further ahead when terrain classification is used for planning a safe and efficient path. Also, cameras are usually cheaper and most mobile robots are equipped with cameras anyways. In camera-based terrain classification we classify small image patches. We will see in the next section how we can determine for each terrain grid cell the corresponding pixels in the camera image. Then, for each image patch local features are computed, which characterize the texture information of that patch. We use three local image descriptors: Local binary patterns, Local ternary patterns and TSURF. The former two are texture-based image descriptors and the corresponding feature vectors are

histograms with 256, respectively 512 bins, whereas TSURF is an interest-point image descriptor. These image descriptors have shown their strength in terrain classification on mobile robots [Khan *et al.* (2011)]. While we have presented Local binary patterns and Local ternary patterns in Sec. 2.2, we will now briefly describe TSURF.

TSURF [Khan *et al.* (2011)] is based on the popular interest point detector and descriptor SURF (**S**peeded **U**p **R**obust **F**eatures) [Bay *et al.* (2008)], and the T in the name stands for Terrain. SURF, in turn, is inspired by SIFT (**S**cale **I**nvariant **F**eature **T**ransform) [Lowe (2004)], and while it gives similar results to SIFT it is three times faster according to [Bay *et al.* (2008)]. Interest points are found by using the determinant of the Hessian matrix, which indicates intensity extrema. The elements of the Hessian matrix are Laplacians of Gaussians, and while SIFT approximate these values by differences of Gaussians, SURF uses convolutions with box filters in conjunction with integral images, which is much faster. In order to find repeatable, distinctive, and robust features that are scale invariant, SIFT and SURF look for interest points at different scales of the image. SIFT realizes this so-called scale space through an image pyramid, which is build by iteratively convolving an image with a Gaussian kernel and reducing the size of the image, that is, a Gaussian filter smooths subsequent layers of the pyramid. In SURF, however, the image does not need to be changed at all, only the size of the box filters are varied, which again leads to a significant speed up. Interest points (x, y, σ) are local extrema in this three-dimensional scale space regarding surrounding points in the same scale and in neighboring layers, where σ denotes the scale factor. In addition to scale, a dominant orientation can be determined for each interest point on the basis of Haar wavelet responses in x- and y-directions within a circular neighborhood.

The SURF descriptor describes the distribution of pixel intensities around an interest point. Therefore, a window of size 20σ is considered, with the interest point in the center, and aligned with the dominant orientation. This window is divided into 4×4 subregions, and in each subregion Haar wavelets are computed for 25 uniformly distributed sample points. Haar wavelets are simple filters for finding gradients dx and dy in x- and y-direction. We can then use these 25 wavelet responses to compute a feature vector v for each subregion.

$$v = \left(\sum dx, \sum dy, \sum |dx|, \sum |dy|\right) \tag{4.5}$$

Computing a feature vector for each subregion and concatenating them together yields an overall feature vector x_{TSURF} of size $4 \times 4 \times 4 = 64$.

Put simply, TSURF consists of computing the SURF descriptor at fixed points in the image with a fixed scale factor and an upright orientation. We set the fixed points to be the centers of the terrain grid cells that we want to describe. Since we set the points manually and do not get them as local extrema of the scale space. there is also no corresponding scale factor. This factor is then a parameter to be set, which is then used in the computation of all descriptors. With the assumption that the appearance of the terrain is isotropic, no dominant orientation is determined, which speeds up the computation. It

is important to note that the region in the image that is used for the computation of the descriptor does not depend on the cell size, but is determined only by the scale factor σ. Since the size of this region in pixels is $20\sigma \times 20\sigma$, it can exceed the grid cell significantly, and in extreme cases include the whole image.

Overall, we thus obtain three different types of feature vectors:

$$x_{\text{LBP}} = \left(h_{\text{LBP}}^{[1]}, ..., h_{\text{LBP}}^{[256]}\right) \tag{4.6}$$

$$x_{\text{LTP}} = \left(h_{\text{LTP}}^{[1]}, ..., h_{\text{LTP}}^{[512]}\right) \tag{4.7}$$

$$x_{\text{TSURF}} = \left(v_{(1,1)}, v_{(1,2)}, ..., v_{(4,4)}\right) \tag{4.8}$$

4.5 Fusion of 3D LiDAR and Camera Data

On the one hand, we are now able to classify each cell of the terrain grid in front of the robot using the LiDAR data. On the other side, we can classify patches of camera images. We now want to combine both classification methods, for which we need the rigid transformation between the LiDAR and the camera coordinate system. For the calibration process we use several image-scan pairs of a checkerboard (see Fig. 4.8) taken from different positions and angles. The following sections describe how we first determine the intrinsic parameters of the camera [Bouguet (2008)], and then estimate the transformation between the LiDAR and the camera frame using a two-stage optimization procedure [Unnikrishnan and Hebert (2005)].

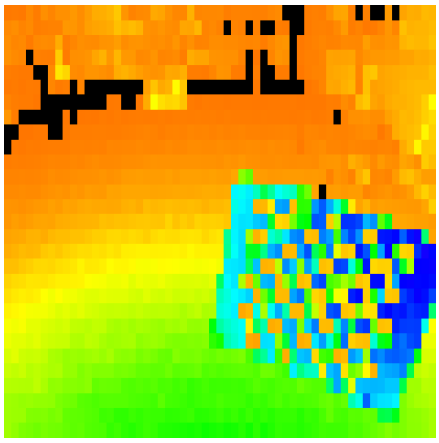
Intrinsic camera calibration

The simplest model to describe the optical properties of a camera is the pinhole camera model. In a pinhole camera the aperture is just a tiny hole and no lenses are used. The parameters of the model are represented by the camera matrix C:

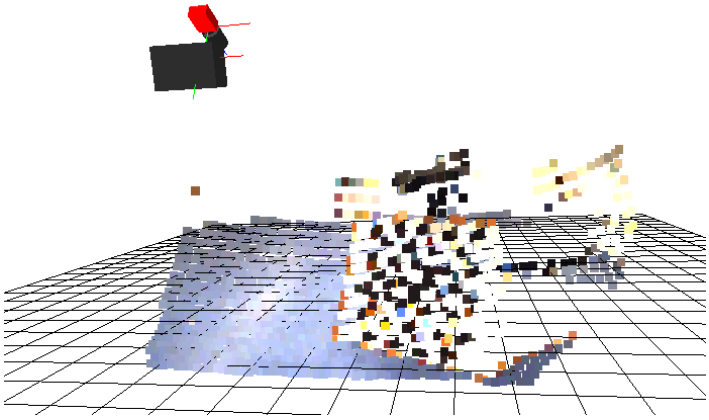
$$C = \begin{pmatrix} f_x & \alpha_c f_x & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{4.9}$$

The camera matrix describes the geometric relation of a point in 3D and its projection on the image plane. It contains the horizontal and vertical focal lengths f_x and f_y, and the principal point (c_x, c_y), which, in the ideal case, is in the center of the image. α_c describes the angle between the x and y sensor axes, and often times is assumed to be 90°. In contrast to the pinhole-camera model, real cameras use lenses to gather sufficient light. These lenses also introduce some distortion effects that cannot be ignored. A more advanced model thus takes into account theses effects with additional distortion

(a) Camera image



(b) Scan image with pixels colored according to the intensity values of the scan points



(c) Colored point cloud

Figure 4.8: Several camera-LiDAR observation pairs of a checkerboard taken from different positions and angles are used to estimate the rigid transformation between the LiDAR and the camera frame. The point cloud can then be colored by reprojecting the points onto the corresponding image.

coefficients D:

$$D = (k_1, k_2, p_1, p_2, k_3) \tag{4.10}$$

D contains the coefficients k_1, k_2, and k_3 of the radial distortion and the coefficients p_1 and p_2 of the tangential distortion. While radial distortion is caused by the curvature of the camera lens and causes straight lines to appear curved in the image, tangential distortion appears when the lens is not aligned perfectly parallel to the imaging plane,

which happens almost inevitably during the manufacturing process.

We use the calibration method of [Bouguet (2008)] to determine the intrinsic camera parameters. There, you need several images of a checkerboard with known dimensions, taken from different angles and positions. By marking the corners between the black and white squares, we obtain equations that must be solved in order to determine the intrinsic parameters. In theory, two images of the checkerboard are enough for a well-posed equation system, but since there is always some image noise, we need several such images in practice. Solving the resulting overdetermined system of equations yields the intrinsic parameters that best fit the data.

With the intrinsic camera parameters we can establish a relationship between the pixel coordinates and the coordinates in the real three-dimensional world. More specifically, we can set up a ray equation for each pixel of the image, and we know that the corresponding object point has to lie somewhere on this ray. Without additional information we can not determine the exact position of the object. For the same reasons we can not determine the size of an object in the real world based on its pixel representation. It can always be the case that the object is, for instance, twice as big and twice as far away as we think, resulting in the same representation of the object in the image. However, since we know the exact dimensions of the checkerboard we use, we can compute the translation and rotation of the checkerboard in relation to the camera for every image used in the calibration process.

Extrinsic LiDAR-camera calibration

To compute the transformation between the LiDAR and the camera, we also need the positions and orientations of the checkerboard as seen by the LiDAR. In the scan of the LiDAR the 3D coordinates of the data points are directly available. We select the points belonging to the board by hand. This is best done in the intensity image since the checkerboard is usually easy to recognize there (see Fig. 4.8b). A least-squares estimator then fits a plane to the selected points. With these image-scan pairs of planes we now want to estimate the rigid transformation between the LiDAR and the camera. Since each pair of observations describes the *same* plane, once from the perspective of the camera, and once from the perspective of the LiDAR, the first thing that comes to mind is to try to find the transformation that minimizes the distance between the two representations. But as they state in [Unnikrishnan and Hebert (2005)], it is not so obvious to find a distance metric for planes, and we instead use their two-stage estimation process to compute the transformation.

In the first stage, the transformation is estimated by considering the translation and the rotation part separately. The optimal translation is the one that translates the planes as seen by the LiDAR such that the differences in distance from the camera origin to each plane representation is minimized; similarly, the optimal rotation is the one that minimizes the angular differences between the plane normals. The second stage is an it-

erative optimization procedure, which takes the result of the first stage as initial estimate. The objective function that is to be minimized here is the sum of the differences of the transformed scan points to the plane as seen by the camera.

With the known transformation between the two sensors we can now determine for each scan point the corresponding pixel coordinate in the image. This could be used, for example, to color the scan points according to the corresponding pixel values (see Fig. 4.8c). We investigated several methods for increasing the resolution of the 3D scans using color information of the images [Hanten (2011)], based on the proposed interpolation methods of [Andreasson *et al.* (2007)]. In [Diebel and Thrun (2005)], they even use a Markov random field to take into account the fact that discontinuities in range and color often occur together. However, this is irrelevant for our method of terrain classification, since we do not gain more information by this kind of interpolation. Instead, we need the intrinsic camera parameters and the transformation between LiDAR and camera to determine for each grid cell of the terrain grid the corresponding pixels in the camera image. Fig. 4.9c shows the projection of the grid onto the image. So, we obtain for each grid cell the relevant image patch and can then compute the corresponding image features. Due to perspective distortion more pixels are available for grid cells nearby than for more distant cells.

Sensor fusion

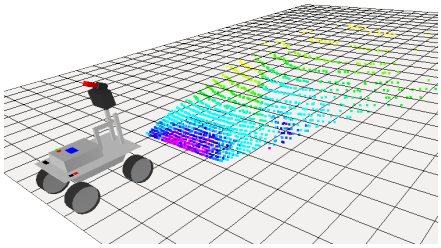
For a terrain grid cell with enough scan measurements we get probabilities $p_{\text{scan}}(y)$ for each terrain class label y. Since we assume that the grid cell belongs exclusively to one of the predefined classes, the probabilities of one cell have to sum up to one, that is, $\sum_y p_{\text{scan}}(y) = 1$. Accordingly, for a cell with an associated image patch that is large enough we get probabilities $p_{\text{image}}(y)$. Again, this probabilities have to sum up to one. Is neither scan nor image information available, we can not favor any class label over another, and all probabilities are equally set to $1/K$, where K is the number of class labels.

For grid cells where data of both sensors are present, we want to combine the probabilities p_{scan} and p_{image} in order to get the best out of each classifier. Two easy ways to do so is by either calculating the weighted sum p_k or by taking the maximum probability $p_{\max}$:

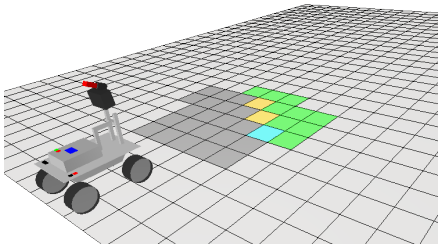
$$p_k = (1-k)\cdot p_{\text{scan}} + k\cdot p_{\text{image}}, \quad \text{with } k \in [0,1] \tag{4.11}$$

$$p_{\max} \propto \max(p_{\text{scan}}, p_{\text{image}}) \tag{4.12}$$

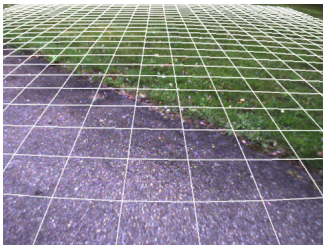
In the weighted sum in Eq. (4.11) we weight the confidence in each sensor with a weighting factor $k \in [0,1]$. In the extreme cases when $k = 0$, we only use the scan data, and when $k = 1$, we only use the image data. The new probability values for a cell also have to sum up to one, and that this is the case is easy to prove:



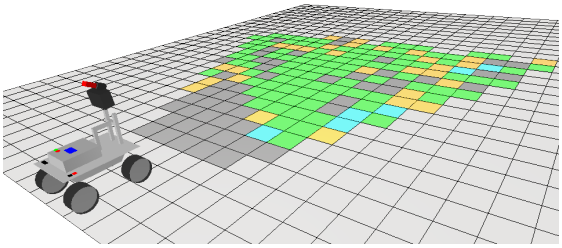
(a) Scan points of the LiDAR



(b) LiDAR-based classification with height and intensity features



(c) Projection of terrain grid onto image for feature extraction



(d) Fused classification result

Figure 4.9: Fusion of LiDAR- and image-based terrain classification. Each cell of the terrain grid is classified based on height and intensity features of the corresponding scan points of the LiDAR. To integrate the image data, the terrain grid is projected onto the image and then, for every projected cell, features based on Local ternary patterns are extracted. (Gray: asphalt, blue: cobblestones, green: grass, yellow: gravel)

$$\sum_y p_k(y) = \sum_y \left((1-k)\cdot p_{\text{scan}}(y) + k\cdot p_{\text{image}}(y)\right) \tag{4.13}$$

$$= (1-k)\cdot \sum_y p_{\text{scan}}(y) + k\cdot \sum_y p_{\text{image}}(y) \tag{4.14}$$

$$= 1-k+k \tag{4.15}$$

$$= 1 \tag{4.16}$$

In Eq. (4.12) we get the new probability value by taking the maximum of p_{scan} and p_{image}, trusting the classifier that is the most confident about the class label. In contrast to the weighted sum, the probabilities here do not sum up to one. In general, it is $\sum_y \max(p_{\text{scan}}(y), p_{\text{image}}(y)) \geq 1$. So to get proper probability values $p_{\max}$, we have to normalize the new values. Fig. 4.9d shows an example of a fused classification result using Eq. (4.12) with $k = 0.5$.

4.6 Experiments and Results

In order to assess the capabilities of the classification methods described in this chapter under varying lighting conditions, we test them in four different settings:

1. *Cloudy morning*: no direct sunlight, soft shadows
2. *Sunny midday*: lots of sun, harsh shadows
3. *Dusk*: rapidly changing illumination
4. *Night*: diffuse light from street lamps

In filtering, we set the search radius $r = 0.15\,\text{m}$ and the minimum number of neighbors $m = 3$ (see Sec. 4.3.1). In ground-plane detection we set the minimal distance $d = 0.1\,\text{m}$ and the maximum number of iterations $m = 100$ (see Sec. 4.3.2).

We start with testing the 3D LiDAR-based approach of Sec. 4.3, without using the camera. We first test the approach for each setting separately to see how well it deal with each in particular. Then, a general model for all scenarios is built by taking the same number of training data from each setting. The method is grid-based and has been tested with different grid resolutions, namely with grid cells of side lengths 20 cm, 35 cm, and 50 cm. The data set used consists of 2 000 samples for each grid resolution, each setting, and each terrain class, for a total of 120 000 samples of terrain patches. We have considered only patches with at least ten scan points, since otherwise there is too little information for a meaningful classification. Then, because of the low resolution of the LiDAR, only for cells that are closer than about two meters to the front of the robot enough measurements are available to classify the cell. Tab. 4.1 shows the classification rates for the different settings and grid resolutions after a five-fold cross-validation using Random forests with 100 trees.

Table 4.1: Classification rates in % for LiDAR-based terrain classification with five terrain classes, for different grid sizes and under different lighting conditions

Grid size \ Setting	Morning	Midday	Dusk	Night	General
20 cm	93.4	94.6	90.0	92.3	90.5
35 cm	92.6	94.4	89.8	91.7	89.9
50 cm	93.1	93.8	90.2	91.1	90.0

Here, the classification rate is the average true positive rate (averaged over all classes), with the true positive rate for a single class being the proportion of instances which were classified as this class among all instances which truly have this class [Hall *et al.* (2009)].

When all classes in the data set have the same number of instances, the classification rate simply is the percentage of correctly classified instances.

The LiDAR-based classification provides consistently good results regardless of external conditions and the chosen grid resolution, with classification rates of up to 90.5% for the general model. It performs best in the midday setting with classification rates of up to 94.6%, and worst in the dusk setting with classification rates of up to 90.2%.

Only considering the two classes of grass and asphalt for classification demonstrates the laser scanner's ability to detect vegetation as mentioned in Sec. 4.2. Then, the classification rates for all settings and all grid resolutions are above 99.9% (see Fig. 4.7b for an example of a classified point cloud).

Table 4.2: Classification rates in % for LiDAR-based terrain classification only considering the terrain classes grass and asphalt

Grid size \ Setting	Morning	Midday	Dusk	Night	General
20 cm	100.00	99.99	100.00	100.00	99.99
35 cm	100.00	100.00	100.00	100.00	99.98
50 cm	100.00	100.00	100.00	99.98	99.99

The choice of grid resolution depends on the specific requirements of the application. A higher resolution increases the computational effort, since then the number of grid cells increases, too, and more features have to be extracted and classified. When the grid cells get too small, too few scan points per cell are available for classification. Taking into account the size and computing power of our robot, 20 cm is an appropriate resolution and we will hereinafter keep it for all our experiments.

In previous work on camera-based terrain classification [Khan *et al.* (2011)] Random forests achieved good results and outperformed the other tested classifiers. In the following experiment we test other classifiers in our LiDAR-based method as well, to see whether Random forests also provides the best results here. For this experiment we use Weka [Hall *et al.* (2009)], a software suite for machine learning written in Java, which provides implementations of many popular machine-learning algorithms. Of these, we use the following four:

- MultilayerPerceptron: Neural net that uses backpropagation to classify instances
- J48: Pruned C4.5 decision tree [Quinlan (1993)]
- Logistic: Multinomial logistic regression model with a ridge estimator, with some modifications to [le Cessie and van Houwelingen (1992)]

- SMO: Implementation of John Platt's sequential minimal optimization algorithm for training a support vector classifier [Platt (1998)]

Table 4.3: Classification rates in % for LiDAR-based terrain classification for different classifiers and different numbers of trees for Random forests

Classifier \ Setting	Morning	Midday	Dusk	Night	General
Random Forests (10 trees)	92.3	94.0	88.8	91.1	89.0
Random Forests (100 trees)	93.4	94.6	90.0	92.3	90.5
Random Forests (200 trees)	93.5	94.7	90.1	92.3	90.7
MultilayerPerceptron	90.5	92.9	87.7	89.4	86.3
J48	89.3	91.6	86.1	87.7	85.0
Logistic	88.0	88.4	86.6	85.4	84.2
SMO	87.4	87.8	85.9	85.0	82.3

In addition to these classifiers, we test Random forests with different numbers of decision trees. The results can be seen in Tab. 4.3. Of the classifiers mentioned above, the neural net performs best with a classification rate of 86.3% for the general setting, followed by the decision tree with 85.0% and the logistic regression model with 84.2%. The support vector machine yields the worst results with only 82.3%. As with the Random forests, the results in all lighting conditions are similar, with the best results at midday and in the morning, and the worst results at dusk and at night. However, Random forests outperform all of these classifiers for all settings. The computation time for Random forests regarding training and classification increases linearly with the number of decision trees. But when more trees are used, there are also more weak classifiers to average over, and the final results get usually better. As can be seen in the result table, the results do not change much whether 100 or 200 trees are used, so using 100 trees is a good trade-off between quality of classification results and computational cost.

Now that we have examined LiDAR-based terrain classification, we turn to the camera-based terrain classification of Sec. 4.4 and the sensor fusion of Sec. 4.5. Therefore, we generated a data set consisting of 125 labeled terrain grids for each of the four settings; and all 500 terrain grids are used for the general setting. Because the cells of a grid are projected onto the camera image in order to get the corresponding image regions, these regions can have very different sizes, and some are very small or elongated. Just as with the LiDAR, no meaningful classification is possible with too little information, and we only consider image regions with at least 200 pixels and a minimum side length of ten pixels, so cells that are closer than about three to four meters are classified using image data. For evaluation using k-fold cross validation, we randomly partition the set

of terrain grids into k subsets. Thus, it can not happen that different cells of the same grid occur in the training set as well as in the test set. This may not make much difference for the image descriptors LBP and LTP, but it can make a difference when using the interest-point descriptor TSURF. Since TSURF, depending on the scale parameter, also considers pixels far outside the respective cell, there could be feature vectors in the training and in the test set that have been partially computed based on the same data, namely the same pixels. This could distort the results of the evaluation; and to prevent this, cells of the same terrain grid are not separated during cross validation. In contrast to the LiDAR-based classification, we use a ten-fold cross validation now.

Table 4.4: Classification rates in % for LiDAR- and camera-based terrain classification

Method \ Setting	Morning	Midday	Dusk	Night	General
LBP	87.7	84.5	66.6	60.2	72.5
LTP	**90.0**	**89.6**	72.5	66.8	**78.5**
TSURF	79.5	81.4	**84.4**	**68.8**	78.1
TSURF (isolated patches)	69.3	72.3	60.5	45.8	59.2
Fusion with LTP (weighted sum)	92.6	**91.7**	**77.5**	**72.6**	**81.4**
Fusion with LTP (maximum value)	**92.7**	90.6	76.6	71.2	80.5

We start with testing the camera-based approach, without using the LiDAR. We tested different values for the threshold parameter $k \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 15, 20, 25, 30\}$ of LTP and different values for the scale parameter $\sigma \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 15, 20, 25, 30, 50\}$ of TSURF. Here, the best results were obtained with $k = 9$ and $\sigma = 30$, and these results are shown in Tab. 4.4. LBP achieves the worst results with a classification rate of 72.5% for the general setting, and LTP outperforms LBP in all settings. LTP achieves better results than TSURF in the morning, midday, and general settings, and worse results in the dusk and night settings. In the camera-based classification the results are more dependent on the lighting conditions and all three descriptors perform worst in the night setting. To compute a feature vector for a grid cell, TSURF considers far more pixels than those belonging to that cell, as mentioned previously. Only considering isolated image patches, results are much worse, as can be seen in the table. TSURF therefore needs large image regions to get good results. Due to the classification results and because LTP is much faster than TSURF, as we shall see shortly, we use LTP as image descriptor for all remaining experiments.

To fuse LiDAR- and camera-based classification results, we test the fusion method using weighted sums (see Eq. (4.11)) with different weighting factors $k \in \{0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}$, and the method using maximum probability (see Eq. (4.12)). Both methods improve the pure camera-based classification. The method using weighted sums with $k = 0.5$ gets the best results with a classification rate of 81.4%

for the general setting.

Table 4.5: Average runtimes per frame and corresponding standard deviations of the main parts of the algorithm

	Average time [ms]	Std. dev. [ms]
Preprocessing		
Filtering	6.3	2.1
Ground-plane detection	0.1	1.4
Feature extraction		
LiDAR	1.5	0.2
LBP	9.3	0.2
LTP	11.3	0.3
TSURF	24.1	1.4
Classification (incl. LiDAR)		
LBP	5.8	0.7
LTP	6.0	0.7
TSURF	6.4	1.4

Finally, we look at the average runtimes of the methods described in this chapter for processing a single frame consisting of a point cloud and an image. For the experiments, a computer having a CPU with 2.8 GHz was used. In the preprocessing, filtering of the point cloud takes a relatively long time with 6.3 ms, but the detection of the ground plane is very fast with only 0.1 ms. In feature extraction, extracting LiDAR features requires by far the least amount of time with 1.5 ms. LTP with 11.3 ms is only slightly slower than LBP with 9.3 ms, and TSURF needs clearly the longest time with 24.1 ms. The classification itself takes about 6 ms, independent of the descriptor used.

4.7 Conclusions

In this chapter, we studied 3D LiDAR- and camera-based terrain classification under different lighting conditions. We consider five terrain classes: asphalt, cobblestones, grass, gravel, and tiles, and four different lighting conditions at different times of day: cloudy morning, sunny midday, dusk, and night. After removing outliers from the point cloud, a RANSAC-based method detects the ground plane in the remaining points. We can then classify a terrain grid in front of the robot. We present easy to compute 3D LiDAR features based on intensity and roughness histograms. The classification results are largely

illumination-invariant and independent of the chosen grid resolution. In the general case, regarding all scenarios, classification rates of up to 90.5% were achieved, using Random forests with 100 decision trees. When only considering the two classes of grass and asphalt, the rates for all settings were above 99.9%. A disadvantage of the 3D LiDAR used in our experiments is the limited range due to its low resolution.

In the camera-based classification the results are more dependent on the lighting conditions, and all tested image descriptors, LBP, LTP, and TSURF, perform worst at night. In terms of classification quality and computation cost, LTP performs best with a classification rate of 78.5% for the general setting and an average runtime of 11.3 ms for extracting all features of a terrain grid. When using both sensors and fusing classification results for grid cells where data of both sensors are present, the final result improves to 81.4%.

Chapter 5

Terrain Classification Considering Spatial Dependencies

In the previous chapter, each cell of the terrain grid was considered and classified independently of the other cells. This ignores the fact that terrain occurs in contiguous areas. By modeling the terrain grid as a Conditional random field we are able to additionally consider spatial dependencies between the cells, which improves classification results substantially.

The chapter is based on the papers "Terrain Classification With Conditional Random Fields on Fused 3D LIDAR and Camera Data" [Laible *et al.* (2013)] and "Building Local Terrain Maps Using SpatioTemporal Classification for Semantic Robot Localization" [Laible and Zell (2014)] and contains parts that were taken verbatim from these works.

5.1 Introduction

Terrain appears in contiguous areas. This is an important information that is ignored when classifying each grid cell individually. There, we get the most likely label $y^* = \arg\max_y p(y \mid x)$ for a cell directly from the Random forest given the extracted features x. It may then happen that the terrain labels assigned to the grid cells change often even in small areas, which rarely occurs in structured outdoor environments in which our robot normally drives. There, rather large coherent areas of the same terrain type occur, with clear boundaries between areas of different terrain. We now want to keep the feature-dependent classification, but also consider the spatial context in which a grid cell appears. To take these spatial dependencies into account, we model the terrain grid using a framework known as Probabilistic graphical models (PGMs). This framework allows us to combine probability theory and graphical modeling. We investigate two kinds of PGMs in the context of terrain classification:

- Markov random fields (MRFs) and
- Conditional random fields (CRFs).

MRFs are generative models and define a family of joint probability distributions $p(\mathbf{y},\mathbf{x})$ represented by means of an undirected graph. CRFs are discriminative models that model the conditional distribution $p(\mathbf{y} \mid \mathbf{x})$ directly.

The remainder of this chapter is organized as follows. In Sec. 5.2 we discuss related work where MRFs and CRFs are used. In Sec. 5.3 we present our new terrain classification method that takes into account spatial dependencies. The theoretical foundations discussed in Sec. 2.4 are important for an understanding of the method. These foundations are repeated here only briefly. Otherwise, we only describe our modifications and extensions in the context of terrain classification. In Sec. 5.4 we present our experiments and results and conclude in Sec. 5.5.

5.2 Related Work

Our work is inspired by [Häselich *et al.* (2011)]. They use data from the high-resolution 3D LiDAR Velodyne HDL-64E S2 in addition to color and texture information from color cameras to classify the terrain into three classes: road, rough and obstacle. To take into account the context-sensitivity of the individual terrain grid cells, they apply an MRF and get a recall ratio of about 90%.

Several works show that CRFs are more appropriate than MRFs for many classification tasks. In [Kumar and Hebert (2003)] the authors compare both models for the task of detecting man-made structures in images, where they also divide the images into grids. The CRF yields higher detection rates with lower false positives. Multi-scale CRFs are used in [He *et al.* (2004)] for the task of image labeling. Their CRFs model local and global structures and yield better results than a MRF, which requires stricter independence assumptions.

A two-stage training is used in [Fulkerson *et al.* (2009)] for identifying and localizing object classes in images. In the first stage, superpixels (small image regions obtained by segmentation) are trained with an SVM, and in the second stage they refine their results by using a CRF. A similar approach is used in [Lalonde *et al.* (2010)] for detecting shadows in images. They first use a decision tree to find potential shadow contours and then optimize the results with a CRF. The model we use for classifying the cells of the terrain grid comes closest to theirs.

5.3 Spatial Terrain Classification

In the previous chapter we presented our method for classifying terrain based on 3D LiDAR and camera data. Therefore, the terrain in front of the robot is divided into a grid of terrain cells, and LiDAR and image features are extracted for each cell. Using these features Random forests then assign a terrain label to each cell. Fig. 4.9d shows a typ-

ical classification result and although quite good, many grid cells are still misclassified. Depending on the particular application this classification quality may not be sufficient. It would be difficult, for instance, to plan a path for a robot that is not capable of driving on rougher terrain like grass, since there is no clear boundary between the asphalt and grass area. Similar problems would be encountered by a robotic lawn mower, which is not allowed to leave the grass area. A human can relatively easy see that the terrain in Fig. 4.9d is in fact divided into two adjacent areas of asphalt and grass. The reason for this is that we do not expect terrain labels assigned to the grid cells to change often in small areas, since this rarely occurs in most outdoor environments, in which our robot normally drives. There, rather large coherent areas of the same terrain type occur, with clear boundaries between areas of different terrain. So, classifying each grid cell individually ignores the context-sensitive nature of the terrain, which means that we ignore important information. We will now describe how to incorporate spatial dependencies in terrain classification in Sec. 5.3.1, where we formulate the classification task as an energy minimization problem. In Sec. 5.3.2, we show how to speed up the spatial classification by defining a new energy term for neighborhood relations.

5.3.1 Spatial Dependencies

On the one hand, we want to keep the feature-dependent classification, but in addition, we now also want to consider the spatial context in which each grid cell appears. To take these spatial dependencies into account, a suitable mathematical model is needed. Such models exist in the form of probabilistic graphical models like Markov random fields (MRF) and Conditional random fields (CRF). MRFs, in essence, define a family of joint probability distributions $p(\mathbf{y}, \mathbf{x})$ represented by means of an undirected graph; CRFs, in contrast, define a family of conditional probability distributions $p(\mathbf{y} \mid \mathbf{x})$. We described these models in Sec. 2.4 extensively and showed how to model the terrain grid as a graph, so we will not go into detail here. But to recap: the nodes of the graph correspond to the random variables used in the classification problem. The set of random variables is split into input and output variables. The input variables correspond to the observed features x, and the output variables correspond to the terrain labels y of the grid cells. The edges of the graph are undirected and represent probabilistic interactions between variables. Of course, there is an edge between the label y of a grid cell and its associated feature vector x, but there are also edges between different labels. In order to limit the complexity of the model we define a neighborhood $\mathcal{N}$ that determines for each grid cell which other cells do have a direct influence on that cell. This neighborhood can consist, for example, of the four direct neighbors as in Fig. 5.1, or as we define it, of the 8-neighborhood, that is, the eight surrounding cells.

The classification process now consists of two stages. In the first stage, features are extracted for each terrain grid cell using LiDAR and camera data. When using the CRF, the grid cells are classified based on this features using Random forests. In the case of the MRF, the mixture of Gaussians which best fits the feature vector determines the label

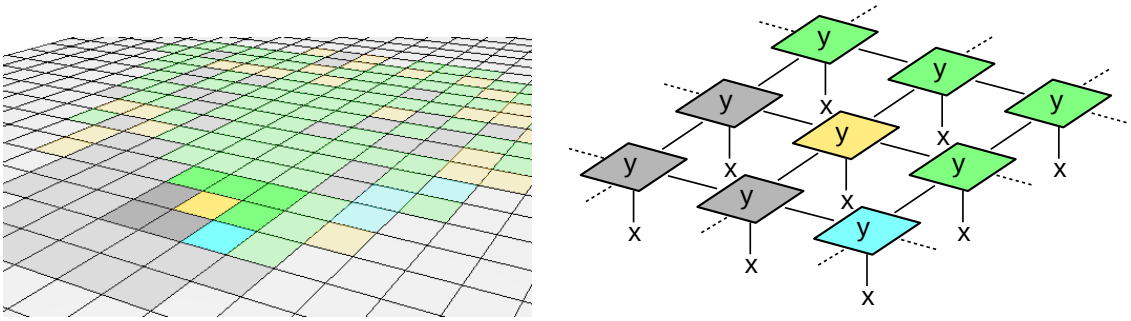


Figure 5.1: The terrain label y of a grid cell depends on the measured features x, but also on the labels of its neighboring grid cells.

of a cell. Both the Random forest models and the mixture of Gaussians are learned from training data. Therefore, the first stage of the CRF-based approach exactly corresponds to the cell-wise classification method described in the previous chapter.

In the second stage, the optimal label configuration $\mathbf{y}^*$, which maximizes the conditional probability $p(\mathbf{y} \mid \mathbf{x})$, is to be found. And now also spatial dependencies between grid cells are taken into account. Since it is not feasible to consider all the exponentially many possible configurations of terrain labels in finding the optimal solution for a grid, we use an approximate inference method, namely a Gibbs sampler in a simulated annealing scheme, introduced in the influential paper of [Geman and Geman (1984)]. In this scheme, the label configuration is changed iteratively until a convergence criterion is reached. These changes can be very large at the beginning, but with the decreasing of a temperature factor will be less. So it is possible to find the global optimum in the presence of many local optima. In general, this optimum, which is the maximum probability $p(\mathbf{y}^* \mid \mathbf{x})$ in our case, is found by bringing a system in a state of minimum energy.

Details about the second stage of our classification approach, including Gibbs sampling and simulated annealing, can be found in Sec. 2.4.3, where we also formulate the classification of the terrain grid as an energy minimization problem. There exist two types of energy potentials: a unary potential E_1 for each grid cell with its associated feature vector, and pairwise potentials E_2 for each pair of neighboring cells. We will recap the neighborhood energy used in the CRF in the next section, along with discussing practical considerations and modifications. In addition, we introduce a new neighborhood-energy formulation that is probabilistically motivated.

5.3.2 Neighborhood Energy

Both MRFs and CRFs have two types of energy potentials, namely unary and pairwise potentials. In contrast to the MRF, the pairwise potentials of the CRF also depend on the feature vectors. This means that for the computation of the pairwise potentials of

the MRF only a priori knowledge is used, whereas the pairwise potentials of the CRF additionally depend on the observed features. We define the neighborhood energy of cell i as the sum of all pairwise energies to neighboring cells, according to the neighborhood $\mathcal{N}_i$. For the CRF, this corresponds to Eq. (2.50):

$$E_2^{\text{CRF}}(\mathbf{y},\mathbf{x},i) = \sum_{j\in\mathcal{N}_i} \mathbf{1}_{\{y_i\neq y_j\}} \exp\left(-\beta(x_i-x_j)^2\right) \tag{5.1}$$

The key idea behind this formulation of E_2^{CRF} is that it is very likely that two adjacent cells belong to the same type of terrain, but if they do not, i.e. $y_i \neq y_j$, their appearance must also differ greatly, with this difference expressed by the squared difference $(x_i-x_j)^2$ of the feature vectors. The cost for computing the energy terms is a crucial factor. The term $\exp(-\beta(x_i-x_j)^2)$ has to be computed for each pair (i,j) of cells for which $y_i \neq y_j$ in at least one label configuration during simulated annealing. The time required for calculating this difference depends on the length of the feature vectors. In fact, it turned out that using LTPs here takes too much computation time, so we used simpler features, namely the average RGB color values of an image patch. Thus, we are using LTPs as global features, which describe the overall appearance of a terrain patch, and we are using the average color to describe the relative change between neighboring patches. On the other hand, it is also questionable whether the difference of two LTP vectors, or other high-dimensional features, reflects the differences in appearance adequately. Therefore, we developed an alternative description of the neighborhood-dependent energy.

We set two requirements for the new energy term $\hat{E}_2^{\text{CRF}}$. First, the differences in appearance are not to be described by the difference of two feature vectors, but also by probabilities. And second, the distinction should not simply be made between neighbors of the same terrain type and those of different terrain types. Instead, the actual types should be considered, since different types are adjacent to each other with a different probability.

We meet these requirements by defining the neighborhood energy $\hat{E}_2^{\text{CRF}}$ as the energy equivalent of the average probability that a neighboring cell is of its assigned type y_j, given the observed features x_j of that cell and the label y_i of the centered cell (see Fig. 5.2), with a weighting factor β:

$$\hat{E}_2^{\text{CRF}}(\mathbf{y},\mathbf{x},i) = -\beta\,|\mathcal{N}_i|^{-1} \sum_{j\in\mathcal{N}_i} \log(p(y_j \mid x_j, y_i)) \tag{5.2}$$

Using Bayes' theorem and the observation that x_j and y_i are conditionally independent (see Fig. 5.2), it follows that:

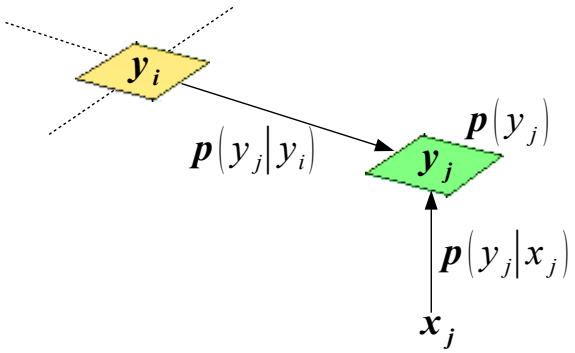


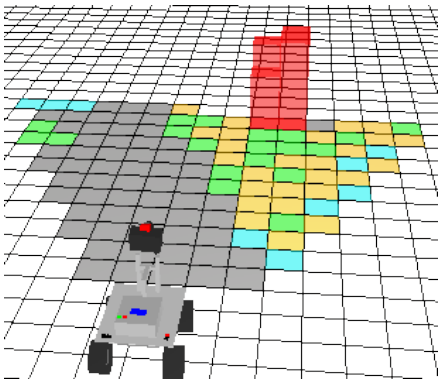
Figure 5.2: Both the feature vector x_j and the label y_i of the adjacent grid cell have an influence on the probability of the label y_j.

$$\begin{aligned} p(y_j \mid x_j, y_i) &= \frac{p(x_j \mid y_j, y_i)p(y_j \mid y_i)}{p(x_j \mid y_i)} \\ &= \frac{p(x_j \mid y_j)p(y_j \mid y_i)}{p(x_j)} \\ &= \frac{p(y_j \mid x_j)p(x_j)p(y_j \mid y_i)}{p(y_j)p(x_j)} \\ &= \frac{p(y_j \mid x_j)p(y_j \mid y_i)}{p(y_j)} \end{aligned} \tag{5.3}$$

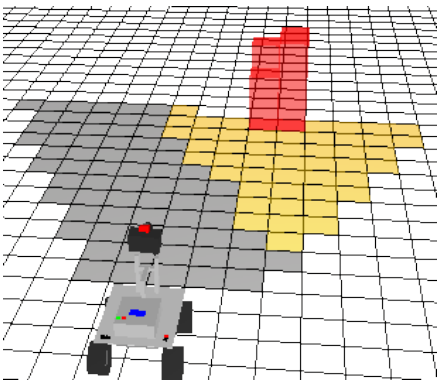
Note that we do not need x_i in the computation of the probability of y_j. Since we set y_i to a fixed value, y_j and x_i are conditionally independent, that is, $p(y_j \mid x_i, x_j, y_i) = p(y_j \mid x_j, y_i)$. The features x_i are important for determining the label y_i, but since we know y_i already, x_i gives no additional information. In the energy term E_2^{CRF} in Eq. (5.1) the feature vector of a cell is compared to the feature vectors of its neighboring cells in order to compare their visual appearance. The best way to compute the visual differences depends on the particular type of features (LBP, LTP, TSURF, ...); and a simple vector difference may not be an appropriate representation. Although the feature vector x_i does not occur in the energy term $\bar{E}_2^{\text{CRF}}$, all features of the surrounding cells are taken into account. In contrast to the MRF, where no features but only labels are considered in the pairwise energy terms, $p(y_j \mid x_j)$ in Eq. (5.3) serves as a weighting factor that weights the probability according to the labels with the probability according to the features. Since we represent the influence of the features using probability values, the formulation is independent of the actual type of features, and the corresponding length of the feature vectors does not affect computation time. The probabilities $p(y_j \mid x_j)$ are provided by the Random forests, and $p(y_j \mid y_i)$ and $p(y_j)$ can be set as required or be learned from

training data, for instance by simply counting how often terrain patches with label y_j are adjacent to patches with label y_i. On our campus, for example, asphalt and grass are often adjacent to each other, while asphalt and gravel are not. So the probability $p(\text{“grass”} \mid \text{“asphalt”})$ is high, and the probability $p(\text{“gravel”} \mid \text{“asphalt”})$ is almost zero. As Fig. 5.3 illustrates, having a bad initial classification based on Random forests, using the energy term E_2^{CRF} of Eq. (5.1) can make things even worse, whereas the energy term $\hat{E}_2^{\text{CRF}}$ of Eq. (5.2) improves the result significantly.

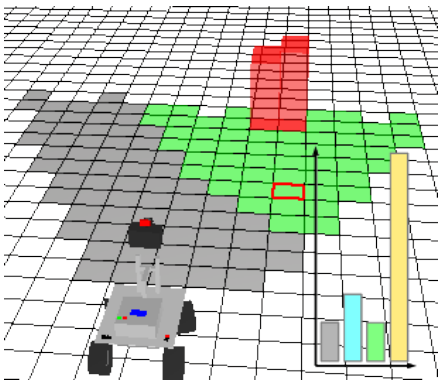
(a) Camera image of two adjacent areas of different terrain



(b) Result after classifying each grid cell individually



(c) Context-sensitive classification went wrong



(d) Good classification result using $\hat{E}_2^{\text{CRF}}$ (see Eq. (5.2))

Figure 5.3: The classification of the terrain in the camera image (a) can give poor results when the grid cells are classified only individually (b). Since most cells were incorrectly classified as gravel, the results get even worse when considering spatial context (c). However, when integrating the observation that asphalt and gravel areas are rarely adjacent in the specific environment in which the robot operates, the classification result gets very satisfactory (d). (Gray: asphalt, blue: cobblestones, green: grass, yellow: gravel, red: obstacles)

The histogram in Fig. 5.3d shows the probability distribution of the four terrain classes for the red-bordered cell. This is an example that illustrates that a cell that would be wrongly labeled as gravel when only considering the measured features, is now correctly assigned to the class "grass" by also considering the spatial context of that cell, and the characteristics of the specific environment. The new energy term can therefore use additional information about the environment to make better predictions. As we will see in the next section, it is also much faster to compute without loss of quality.

5.4 Experiments and Results

In the experiments of the previous chapter, all cells of a terrain grid were of the same type. Since we are now interested in classification that takes into account spatial relationships between cells, the terrain grids of this data set were taken to a large extent at transitions between two (or sometimes three) types of terrain (see Fig. 5.4). We consider a total of four types of terrain, which make up most of the ground of our campus, where the experiments were conducted, namely asphalt, cobblestones, grass, and gravel. In contrast to the previous chapter, we no longer consider tiles, since there are too few terrain transitions involving tiles for a meaningful analysis. For the experiments in this section, we use two different data sets. We first introduce these data sets and then investigate context-sensitive terrain classification with MRFs and CRFs. Finally, we look at the runtimes of the main parts of the classification algorithm.

To evaluate the different classification methods we need a set of frames (scans and corresponding images) with ground-truth data for the terrain grid. We therefore hand-labeled the images, which build the basis for the labeling of the terrain grid. This labeling was refined using the scan points of the LiDAR. The ground in front of the robot is divided into a grid with a cell size of $20\,\text{cm} \times 20\,\text{cm}$, which is a high enough resolution for subsequent tasks such as path planning. This is the same for both data sets, but there are also differences:

- **Data set I** (from [Laible *et al.* (2013)]):

  The data set consists of 135 hand-labeled terrain grids. For the LiDAR-based classification only cells with at least ten scan points are considered. Therefore, because of the low resolution of the FX6, only cells that are closer than about two meters to the front of the robot can be classified with this sensor. The projection of the terrain grid determines the size of the individual image patches. Only patches with at least 200 pixels are considered, so cells that are closer than about three to four meters are classified with the image data.

- **Data set II** (from [Laible and Zell (2014)]):

  The LiDAR was re-adjusted for this data set and it is now tilted a little further to the front. It is mounted on the robot at a height of approximately 50 cm and at an angle of about 25° to the horizontal, with the camera mounted on top of it. Again, we only classify grid cells with at least ten laser measurement points in it, or with at least 200 pixels and, in addition, a minimum side length of ten pixels for the corresponding image patches. These limitations lead to a lower detection range, the LiDAR can detect terrain in a range of about 1.5 m in front of the robot, whereas the camera has a detection range of about 3 m; but the classification provides better results, which pays off in the end in temporal classification in the next chapter. This data set consists of 200 terrain grids.

In the first experiments, we test our MRF and CRF models with data set I. All classification rates are determined using 10-fold cross-validation. When using the MRF, the first stage of our two-stage classification approach consists of assigning to each cell of the terrain grid that label that best fits the learned mean and standard deviation of the underlying Gaussian model. It turns out that this yields very poor results in our case. Only regarding the LiDAR data, the classification rate is 49.5%. Considering spatial dependencies in the second stage only moderately improves the classification result to 54.9%. For the image data the results using the Gaussian model are even worse with 33.8%. With such a bad feature-dependent classification it makes no sense to try to improve the classification by considering the neighborhood of the cells. The poor results show that the assumption of a Gaussian distribution for the features used in our work does not hold. Especially the elements of the LTP feature vector, which represent histogram bins, do not appear to be Gaussian distributed.

Table 5.1: Classification results for data set I

Classification method	Classification rate in %
Image-based	80.4
Fusion of LiDAR and image data	81.5
Conditional random field	94.2

Classification rates after 10-fold cross-validation for the image-based classification with local ternary patterns, the fused classification, and the classification with a Conditional random field.

Tab. 5.1 shows the classification results for our standard CRF model described in Sec. 2.4.2. Using only the LiDAR data and the Random forest classifier we get a classification rate of 93.1%. This result seems very good, but it also has to be considered that

only a small area in front of the robot can be classified hereby. Therefore, it can not be directly compared with the other results. Using Random forest with the LTP features of the image (where a threshold value of 2 gave the best results) yields a rate of 80.4%. By fusing both sensor data this can be improved to 81.5%, whereby each sensor was equally weighted. Again, the reason that this improvement is so low is due to the small area which is covered by the laser.

The parameters $\lambda = 0.5$ and $\beta = 2.0$ of the CRF were determined using cross-validation. With this setting the CRF achieves a classification rate of 94.2%, which is a huge improvement. A few typical classification results can be seen in Fig. 5.4, where transitions between terrain types are shown, as these are the most interesting cases.

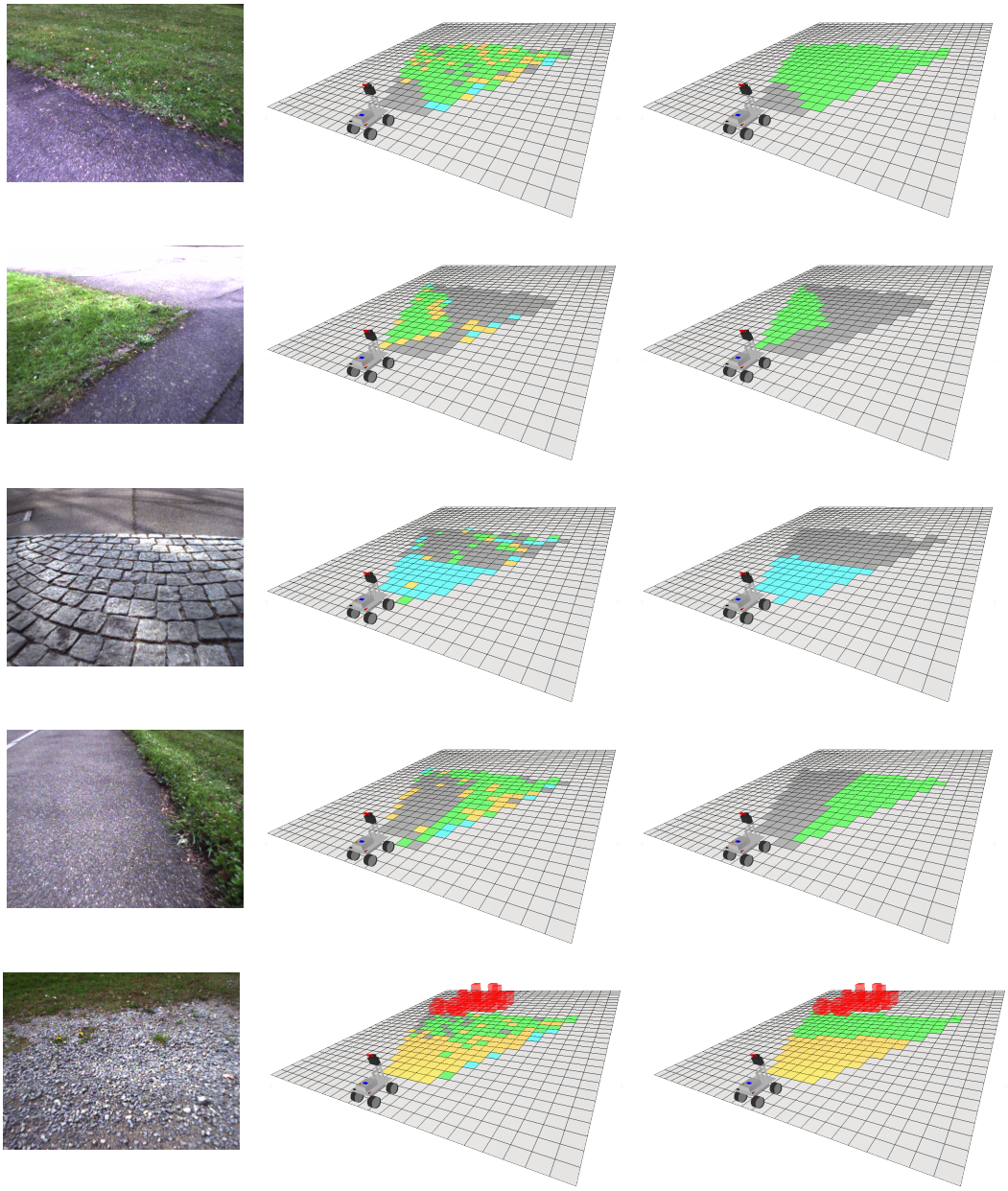
Table 5.2: Classification results for data set II

Classification method	Classification rate in %
Cell-wise classification	82.0
CRF with E_2^{CRF} (Eq. (5.1))	96.8
CRF with $\hat{E}_2^{\text{CRF}}$ (Eq. (5.2))	96.8

Classification rates using 10-fold cross-validation for the cell-wise classification, and the spatial classification with the old and new energy terms E_2^{CRF} and $\hat{E}_2^{\text{CRF}}$, respectively.

In the next experiment, we test the influence of the different neighborhood-energy terms of Sec. 5.3.2 with data set II, and the results are shown in Tab. 5.2. Classifying each grid cell individually using Random forests only yields a classification rate of 82.0%. For our dataset we get the same results, on average, regardless of whether we are using the old or the new energy term, namely 96.8%. However, in some environments as we have seen in Sec. 5.3.2, using the new energy term $\hat{E}_2^{\text{CRF}}$ along with the additional information about the terrain can improve results significantly. Moreover, the computation is also much faster as shown in Tab. 5.3.

Since we want to use terrain classification on the robot in real-time, we are interested in the runtimes of the algorithms. Tab. 5.3 shows the average runtimes of the main parts of the spatial terrain classification method for data set II, using a CPU with 3.20 GHz. The spatial classification consists of the LiDAR- and image-feature extraction, the initial cell-wise classification, and the simulated annealing used for finding the optimal label configuration considering spatial dependencies. The features of the LiDAR data are very fast and simple to compute and the total computation takes only 0.4 ms. The computation of the LTPs takes 9.5 ms and the initialization of the CRF takes 11.2 ms on average. The Gibbs sampler in the simulated annealing scheme takes 8.9 ms when



(a) Images of transitions between two terrain classes

(b) Fusion of LiDAR- and image-based classification

(c) Final classification with a Conditional random field

Figure 5.4: Terrain classification with Conditional random fields on fused LiDAR and image data. The left column shows images of terrain transitions captured by the robot. The middle column shows the results of the classification with Random forests, which contain many wrongly classified cells. In the right column the results of the classification using a CRF are shown, which are significantly better. This image is best viewed in color. (Gray: asphalt, blue: cobblestones, green: grass, yellow: gravel, red: cells with high elevations (> 0.15 m))

the neighborhood-energy term E_2^{CRF} (Eq. (5.1)) is used. The standard deviation is high, because the annealing scheme stops when the error falls below threshold $t = 0.05$, and this takes different amounts of time. Since the feature-dependent classification on its own already gives good results, and the annealing process is initialized with this classification, we start with a low initial temperature $T_0 = 2$ ($c = 0.98$, $N = 100$), which accelerates convergence. The use of the new energy term $\hat{E}_2^{\text{CRF}}$ (Eq. (5.2)) accelerates the process considerably. Simulated annealing then only takes 1.7 ms, and the whole spatial classification takes 22.8 ms on average.

Table 5.3: Average runtimes of the main parts of the spatial terrain classification method

	Average time [ms]	Std. dev. [ms]
LiDAR-feature extraction	0.4	0.1
Image-feature extraction	9.5	0.4
Cell-wise classification	11.2	1.5
Simulated annealing with E_2^{CRF} (Eq. (5.1))	8.9	6.7
Simulated annealing with $\hat{E}_2^{\text{CRF}}$ (Eq. (5.2))	1.7	1.0
Entire spatial classification with $\hat{E}_2^{\text{CRF}}$	**22.8**	**3.0**

5.5 Conclusions

In this chapter, we extended our terrain classification method by taking into account spatial dependencies between the cells of the terrain grid. We therefore modeled the grid as a Markov random field (MRF) and a Conditional random field (CRF). To find an (approximately) optimal label configuration we have to formulate the classification as an energy minimization problem, and we can then use a Gibbs sampler in a simulated annealing scheme. In a classification setting with four terrain classes the classification with the MRF gave very bad results. The reason that the MRF does not work for our problem lies in the type of features that we use. Since we not only differentiate between passable and non-passable terrain but also classify the type of terrain, we need a model that is suited for more complex dependencies between the features. For simple features that can be modeled as Gaussians, the MRF can provide great results as shown in [Häselich *et al.* (2011)] in the context of terrain classification. The CRF, however, gave very good results and we could significantly improve the cell-wise classification with a classification rate of 81.5% to a classification rate of 94.2% for the context-sensitive classification.

We also presented a new way to formulate the neighborhood-dependent energies in the CRF. We tested both energy formulations, the previous and the new one, in another

data set. In both cases, the classification rate could be improved to 96.8%, compared to the cell-wise classification with a classification rate of 82.0%. Although both terms provide the same results in our experiments, the new one is still preferable because it is much faster to compute, and these energies have to be computed very often in simulated annealing. The annealing then only takes 1.7 ms on average, whereas it takes 8.9 ms with the previous formulation. The whole spatial classification then only takes 22.8 ms on average.

Chapter 6

Building Terrain Maps for Semantic Robot Localization

In this chapter, we now also consider temporal dependencies as the robot moves. This not only further improves results, but makes it possible to build terrain maps of the environment. We describe how to efficiently integrate the classification results of each time step into the map in a probabilistic manner. By also detecting obstacles with the LiDAR, the robot can build combined terrain and elevation maps. We show that these maps can be used for semantic robot localization.

The chapter is based on the paper "Building Local Terrain Maps Using Spatio-Temporal Classification for Semantic Robot Localization" [Laible and Zell (2014)] and contains parts that were taken verbatim from that work.

6.1 Introduction

In the previous two chapters, we presented our method for classifying terrain using LiDAR and camera data. With this method, we can classify a terrain grid in front of the robot. By additionally taking into account spatial dependencies between grid cells, we can improve the classification results significantly. However, *temporal* dependencies have not been considered so far. In each time step, the grid is classified independently of previous classification results. But since the robot moves just a little further between two time steps, the same terrain is classified multiple times. And using results from previous time steps would give additional information for further improving results. We show in Sec. 6.3 how to build a whole terrain map of the environment by updating this map at each time step with the current classification result while considering temporal dependencies. In addition to terrain, the LiDAR provides us with information about obstacles in front of the robot. We describe in Sec. 6.4 how to incorporate this information into our map to build combined terrain and elevation maps.

One of the most frequently mentioned applications for terrain classification in robotics is the recognition and avoidance of impassable terrain. But apart from this, the knowledge about the surrounding terrain is also valuable for robot localization, as an addition

or alternative when GPS (Global Positioning System) is too unreliable or not available at all, e.g. near buildings, under trees, or in general when there is no clear line of sight to the satellites. The accuracy requirements of such a localization depend on whether the robot is to follow, for example, a field boundary or a road, or whether it should travel long distances over open field. The terrain maps together with the robot's odometry do not provide a precise positioning of the robot, but a semantic localization, where, for example, transitions between different types of terrain are recognized. We discuss the localization framework in Sec. 6.5. It is easy to incorporate other types of information into this framework, like to what kind of object a detected obstacle belongs. It makes a big difference for localization if the robot finds itself next to a car or next to a rose bush, for instance. Therefore, we investigate object detection and classification based on 3D scan points in Sec. 6.6. Finally, we discuss experiments in Sec. 6.7 and conclude in Sec. 6.8.

Before we start with temporal classification, we first discuss related work on terrain mapping and semantic localization.

6.2 Related Work

There are several related works about terrain classification and mapping that consider spatial or temporal dependencies. In Wolf *et al.* (2005) a 2D LiDAR is used for terrain mapping. The terrain is classified in navigable and non-navigable regions using hidden Markov models. Small classification errors in the map are removed with a Markov random field. In Komma and Zell (2010) vibration data acquired by an inertial measurement unit is used to segment different types of terrain with an unsupervised learning approach. The clustering is based on a Markov random field to consider temporal dependencies between consecutive measurements. Häselich *et al.* (2013) also use Markov random fields, for considering spatial dependencies between cells of a terrain grid. They use a high-resolution 3D LiDAR and several color cameras to classify the terrain into the classes road, rough and obstacle. In addition to taking into account spatial dependencies, they use the robot's estimated egomotion to fuse previous classification results with current sensor data. A 2D LiDAR is used in Wurm *et al.* (2013) to distinguish between vegetation and asphalt by looking at the intensity values of laser measurement points. They show that it is also possible to distinguish between a dark street and light tiles. Results from consecutive measurements are combined probabilistically.

Semantic perception and classification of the environment for mapping and localization is used, for example, in Weiss *et al.* (2010) in the domain of agricultural robotics. They present an alternative to GPS-based navigation for a robot driving through a maize field. The robot can localize itself by detecting states like being in the middle of a row, at a row gap, or at the end of the row. In Nüchter and Hertzberg (2008) semantic maps are generated by labeling coarse scene features like walls and floors, and more complex objects detected by a classifier. Such maps that contain annotations of known objects in

addition to spatial information can be very beneficial for localization tasks.

6.3 Temporal Classification

This section presents our method for spatio-temporal terrain classification. As the robot moves, we constantly update a terrain map with the current classification results. In this way, we are not only able to exploit temporal dependencies, but we are also building whole terrain maps of the environment. When building these maps, we only use odometry to estimate the robot's position and we do not use techniques like scan matching and loop closure detection to reduce or eliminate the inevitable drift in odometry. So our maps are only locally consistent, but this is quite sufficient for many applications.

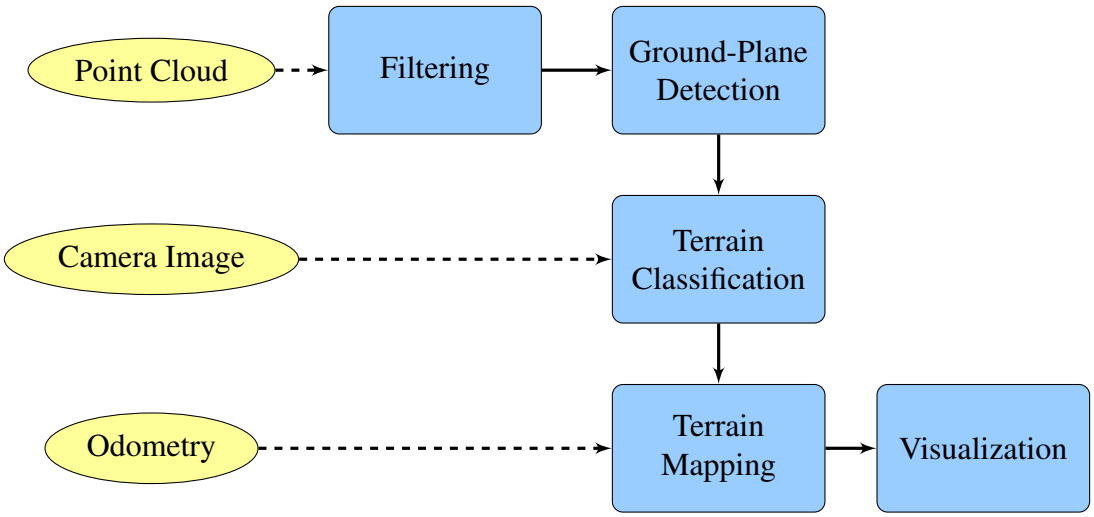


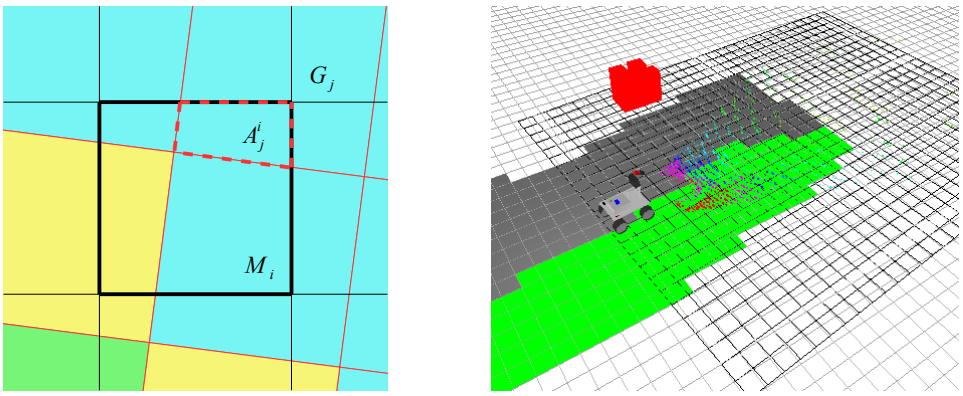
Figure 6.1: Overview of spatio-temporal terrain classification

In the following, we will differentiate between the terrain grid $\mathcal{G}$ in front of the robot and the terrain map $\mathcal{M}$ (see Fig. 6.2b). Since the map is to be built as the robot moves, and since in the beginning we do not know how large the map will be, we let the map grow dynamically. Starting with a single cell (or a specified minimum number of cells), each time the robot reaches a border of the map, the size of the map is doubled in this direction. Internally, the map is stored as a one-dimensional array, where an array element corresponds to a cell of the map. Every time the map grows, the elements of the array have to be reordered. Letting the map grow in powers of two is a good trade-off between keeping the map size small and reducing the number of array reorderings, and is inspired by the vector class of the C++ standard library [ISO (2012)], which automatically handles memory allocations in this manner. If the building of the map is completed, the size of the map can still be reduced accordingly afterwards.

We start in Sec. 6.3.1 by describing how to enter the terrain probabilities of the terrain grid into the map. Since the map cells already contain probabilities themselves, the values have to be combined accordingly, as shown in Sec. 6.3.2. As it will turn out, the results of the spatial classification cannot be integrated into the map without further ado. Sec. 6.3.3 presents our method to solve this problem.

6.3.1 Projection of Grid Cells Onto the Terrain Map

At each time step, terrain classification consists of assigning to each cell i of a terrain grid $\mathcal{G}$ in front of the robot probability values $p(Y_i = y)$ for all terrain labels y. And just like the grid cells, each cell of the terrain map $\mathcal{M}$ contains probability values for all labels, too. Thus, updating the terrain map with the current classification result means updating the terrain probabilities of the relevant map cells. Fig. 6.2b shows an example of how $\mathcal{G}$ and $\mathcal{M}$ might overlap.



(a) (b)

Figure 6.2: (a) Projection of grid $\mathcal{G}$ (red lines) onto map $\mathcal{M}$ (black lines). A_j^i outlines the sectional area between grid cell G_j and map cell M_i. (b) As the robot moves, the current classification results (represented by local terrain grid $\mathcal{G}$) are used to constantly update a terrain map $\mathcal{M}$ of the environment.

To compute the probabilities for map cell M_i that correspond to the current measurement, the probabilities of all grid cells G_j that overlap with M_i need to be considered (see Fig. 6.2a). The influence of each grid cell G_j is proportional to the sectional area A_j^i of M_i and G_j. So for all terrain classes y the probability $p(Y_i^M = y)$ that map cell i has label y is calculated as:

$$p(Y_i^M = y) = \eta \sum_j \frac{A_j^i}{A_i} p(Y_j^G = y) \tag{6.1}$$

$$\eta = \left(\sum_y \sum_j \frac{A_j^i}{A_i} p(Y_j^G = y) \right)^{-1} \tag{6.2}$$

Here, A_i is the area of the map cell M_i, which is usually constant throughout the map. To compute the sectional area A_j^i efficiently we use the Sutherland-Hodgman algorithm [Sutherland and Hodgman (1974)], originally invented for fast polygon clipping in the field of computer graphics. Fig. 6.3 illustrates the individual steps of the algorithm for clipping a projected grid cell (blue polygon) against a map cell (black square).

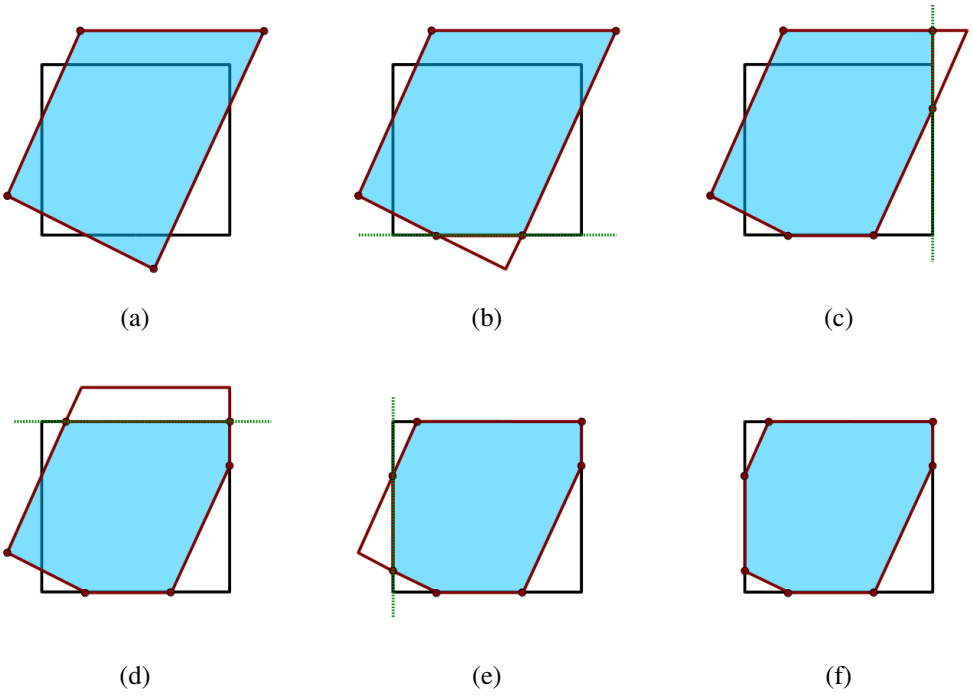


Figure 6.3: Sutherland-Hodgman algorithm for clipping polygons. At each iteration of the algorithm the blue polygon is clipped against an extended edge of the square.

In general, the algorithm works for clipping arbitrary, convex or concave polygons by arbitrary, convex polygons, and in our case, the polygons are always convex and quadrilateral. At each iteration, an edge of the clip polygon is extended in both directions. We start with the bottom edge in Fig. 6.3b. The path of the blue polygon is traversed and every time we cross the extended bottom edge of the clip polygon, we add the intersection to the list of vertices. On the other side, vertices that are below the edge are removed.

This is repeated for all edges of the black square. As we can see in Fig. 6.3c and 6.3d, it is possible that in one iteration step a new vertex is added that lies outside the black square, but this vertex is removed again in the next step. At the end of the algorithm, the vertices $(a_1,b_1),(a_2,b_2),\ldots,(a_n,b_n)$ of the sectional area remain. The actual surface area A^i_j can then be calculated as follows [Beyer (1984)]:

$$A^i_j = \frac{1}{2}\sum_{k=1}^{n}(a_k b_{k+1} - a_{k+1}b_k) \tag{6.3}$$

6.3.2 Temporal Updating of Terrain Probabilities

We just showed how to associate the probabilities of the grid cells with the cells of the map. In contrast to the grid cells whose values are based solely on the actual measurement at time step t, the values in a map cell represent the terrain probabilities given all previous measurements $\mathbf{x}_{1:t-1}$. To update the map with the current measurement $\mathbf{x}_t$, we use the same update formula as in [Wurm *et al.* (2013)] where they map two terrain classes, vegetation and non-vegetation, and which was originally used for Occupancy grid maps [Moravec (1988)].

$$\widetilde{p}(y \mid \mathbf{x}_{1:t}) = \left(1 + \frac{1-p(y \mid \mathbf{x}_t)}{p(y \mid \mathbf{x}_t)}\frac{1-p(y \mid \mathbf{x}_{1:t-1})}{p(y \mid \mathbf{x}_{1:t-1})}\frac{p(y)}{1-p(y)}\right)^{-1} \tag{6.4}$$

$$p(y \mid \mathbf{x}_{1:t}) = \eta\ \widetilde{p}(y \mid \mathbf{x}_{1:t}) \tag{6.5}$$

$$\eta = \left(\sum_{y=1}^{K}\widetilde{p}(y \mid \mathbf{x}_{1:t})\right)^{-1} \tag{6.6}$$

Hereby, the probability values $p(y \mid \mathbf{x}_t)$ are stored in the grid cells, and the probability values $p(y \mid \mathbf{x}_{1:t-1})$ are stored in the map cells. Since we not only have two classes, such as occupied and unoccupied, or vegetation and non-vegetation, we have to normalize the probabilities by dividing by $\sum_{y=1}^{K}\widetilde{p}(y \mid \mathbf{x}_{1:t})$, where K is the number of different terrain classes. The derivation of the update formula uses Bayes' rule and the Markov assumption that the current observation is independent of previous observations given the actual terrain label, and can be found in [Wurm *et al.* (2013)].

6.3.3 Recomputation of Terrain Probabilities

Until now, when we update the terrain map with the update formula of the last section, we loose the results of the spatial classification, and the probability values stored in the map cells are a result of a purely temporal classification.

The reason for this is that the temporal updating of the map is solely based on probabilities, which means in particular that the actual label configuration $\mathbf{y}$ is not considered.

On the other hand, only labels and no probabilities are changed in the second stage of the spatial-classification process. In order to take into account the result of the MAP inference method, namely the optimal configuration $\mathbf{y}^*$ (see Eq. (2.36)), we need to incorporate this configuration in the terrain probabilities of the grid cells. The idea now is similar to that for the definition of the neighborhood energy in Eq. (5.2), namely to involve the neighborhood of a cell for computing new terrain probabilities (see Fig. 6.4).

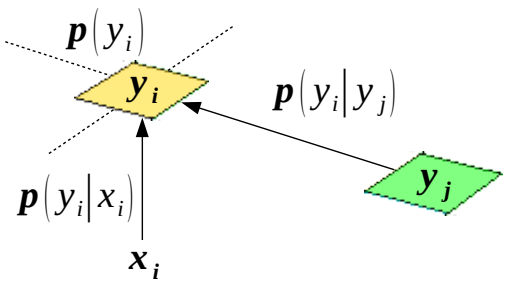


Figure 6.4: The updated terrain probabilities $p(y_i \mid x_i, \{y_j : j \in \mathcal{N}_i\})$ stored in the map not only depend on the features x_i, but also on the labels $\{y_j : j \in \mathcal{N}_i\}$ of the neighboring cells.

Using analogous considerations as in the derivation of Eq. (5.3), and considering the whole neighborhood $\mathcal{N}_i$, it follows that:

$$\begin{aligned} p(y_i \mid x_i, \{y_j : j \in \mathcal{N}_i\}) &= \frac{p(x_i \mid y_i, \{y_j\})p(y_i \mid \{y_j\})}{p(x_i \mid \{y_j\})} \\ &= \frac{p(x_i \mid y_i)p(y_i \mid \{y_j\})}{p(x_i)} \\ &= \frac{p(y_i \mid x_i)p(x_i)p(y_i \mid \{y_j\})}{p(y_i)p(x_i)} \\ &= \frac{p(y_i \mid x_i)p(y_i \mid \{y_j\})}{p(y_i)} \\ &= \frac{p(y_i \mid x_i)}{p(y_i)} \prod_{j \in \mathcal{N}_i} p(y_i \mid y_j) \end{aligned} \tag{6.7}$$

(For better readability, we abbreviated $\{y_j : j \in \mathcal{N}_i\}$ with $\{y_j\}$ in the intermediate steps.)

The recomputation of the terrain probabilities makes in fact a big difference in the

final classification results as can be seen in Fig. 6.6d. After this final step, our method of terrain classification now considers both, spatial and temporal dependencies.

6.4 Elevation Mapping

In addition to the probabilities of the terrain classes each cell of our map also stores a height value h and the corresponding variance σ_h^2. We use a Kalman filter-based approach for elevation mapping as described in [Kleiner and Dornhege (2007)] where the height of a cell is estimated given all previous height observations.

After detecting the ground-plane in the point cloud of the 3D LiDAR and transforming the cloud so that the ground plane equals the xy plane with the z-axis pointing upwards, the current height observation of a cell is simply the maximal z-value z_{max} of the scan points belonging to that cell. The estimated variance σ_z^2 of this observed height is computed by the plane-detection module and is updated at each time step. With z_{max} and σ_z^2 we can now update h and σ_h^2 by applying a Kalman filter. Since the laser scanner is tilted it can measure very different heights for the same cell when driving towards vertical obstacles like walls. To account for this, no Kalman update is performed when $|h - z_{max}| > \sigma_h$, but h is set to $\max\{h, z_{max}\}$ instead. With this exception rule the estimated height h can grow by leaps and bounds, but not shrink in the same manner. The latter is, however, desirable in case of dynamic obstacles or incorrectly measured observations. We change the rule in [Kleiner and Dornhege (2007)] thus slightly and update yet again if the observed height z is below a given threshold, that is, almost zero.

6.5 Semantic Localization

With the spatio-temporal classification method described in the previous sections and chapters the robot is now able to build maps of the environment, containing terrain and obstacles. Detecting and classifying obstacles and terrain by itself is an important ability of a mobile outdoor robot for planning safe and efficient driving maneuvers. Moreover, as we will show now, such maps can also be used for robot localization. Our goal here is not a precise localization of a robot working in a small area, but a semantic localization when traversing long distances. "The semantic localization problem in robotics consists," according to [Martínez-Gómez *et al.* (2016)] "in determining the place where a robot is located by means of semantic categories." In our case, these semantic categories are the different terrain classes; or different object classes as we will see in Sec. 6.6. In many ways, semantic localization corresponds more to how a human would localize itself, as opposed to a precise metric localization. A transition from a lawn area to an asphalt road, for example, often is a more important information than high-precision coordinates.

In order to utilize the results of terrain classification for localization, we use Monte Carlo localization [Dellaert *et al.* (1999)], since this method has proven to be very ef-

fective for our purposes. Monte Carlo localization uses a particle filter for estimating the pose of the robot, that is, both the position and the orientation; and each particle is a candidate for the actual pose. In our case the motion prediction for the particles is based solely on the odometry. Measurements are used to set importance weights for the particles, with these weights being proportional to the resampling probability. These two steps, movement and measurement, let the particles converge to the true pose of the robot. In our case the measurements are, on the one hand, the grid cells with assigned terrain labels, and on the other side, the height values of the obstacles. We discuss the general method of Monte Carlo localization in Sec. 2.5, so we will not go into detail here, but instead show how to compute importance weights $w_t, w_h \in [0,1]$ from terrain and height measurements, respectively. Since often several thousand particles are used, the calculation of these weights must be very fast.

6.5.1 Particle Weights

Let I_t be the set of indices of those grid cells to which a terrain label was assigned, with that label being denoted as y_j for $j \in I_t$. Let further be $i(j)$ the corresponding map cell index obtained by projecting the grid cell center onto the map, and $p_{i(j)}$ the distribution of terrain labels of that cell. Then, the measurement weight $w_t \in [0,1]$ for the terrain labels is defined as:

$$w_t = |I_t|^{-1} \sum_{j \in I_t} p_{i(j)}(y_j) \tag{6.8}$$

We call a cell occupied if its height value h exceeds a threshold τ. As a measure of whether the observation is consistent with the map, we define for every grid cell with index $j \in I_h$, where I_h is the set of indices of those grid cells that are occupied:

$$H_j = \begin{cases} 1 & \text{if } h_{i(j)} > \tau \\ 0 & \text{else} \end{cases} \tag{6.9}$$

We then set the weight $w_h \in [0,1]$ as follows:

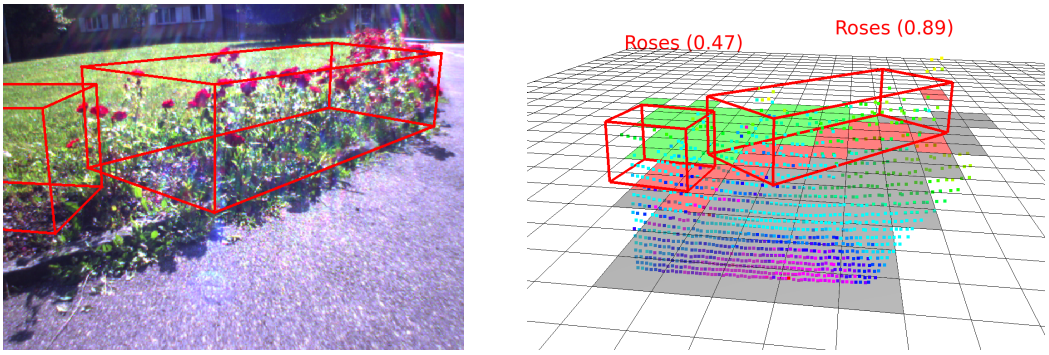
$$w_h = |I_h|^{-1} \sum_{j \in I_h} H_j \tag{6.10}$$

The final importance weight $w = \kappa w_t + (1-\kappa) w_h$ of a particle is a combination of w_t and w_h, with $\kappa \in [0,1]$.

6.6 3D LiDAR-Based Object Detection and Classification

In the previous section we have seen how to use information about terrain and obstacles to help the robot localize. Therefore, we differentiated between different types of terrain,

and between grid cells containing obstacles, and those without. Instead of treating all obstacles the same, it would be preferable to also classify them according to a set of predefined classes, just like with the terrain. For example, whether a robot is located next to a lamppost or next to a rose bush is a valuable information. In this section we present our approach for 3D LiDAR-based object detection and classification. We first cluster the scan points to detect objects in the point cloud, and then extract features for each object in order to classify them.



(a) (b)

Figure 6.5: Object detection and classification based on 3D LiDAR data. We first cluster the scan points to detect objects in the point cloud, and then extract features for each object in order to classify them according to a set of predefined classes.

To detect objects in a point cloud we only consider scan points that are above the ground plane. These points are grouped into clusters using a method referred to as Euclidean cluster extraction [Rusu (2009)]. First, a k-d tree representation of the scan points is created (see Sec. 4.3.1), so that we can efficiently find the neighbors of a point that are within a threshold radius r. A cluster $\mathcal{C}$ with respect to r is then defined as follows: The cluster consists either of a single point, or for each point $p \in \mathcal{C}$ there exists another point $q \in \mathcal{C}, q \neq p$, so that the Euclidean distance between p and q is smaller than r. To find the maximal clusters of the point cloud, we start with an arbitrary point and add it to an empty cluster. Every neighboring point that lies within radius r will also be added to the cluster. The first point is then marked as processed. We now iterate recursively through all unprocessed cluster points, adding scan points that lie within radius r and that are not yet processed, until no unprocessed cluster point is left. Then, the resulting cluster is maximal. We continue with another unprocessed scan point and a new cluster, until all points of the cloud are processed. Clusters where the number of points is too low are discarded. We then determine a bounding box for each cluster. Since we assume that most objects are perpendicular to the ground, we assume the bounding box to be upright as well. The orientation in the xy plane is determined by computing the oriented bounding

box of the two-dimensional projection of the points onto the *xy* plane, using their Eigen vectors. The height of the bounding box results from the highest cluster point. We will need this bounding boxes later, for feature extraction and for integrating the classification results into our map. Since we know the transformation between the LiDAR and the camera (see Sec. 4.5), we can also project the bounding boxes onto the image plane (see Fig. 6.5a). This could be used in image-based object classification, for example.

Each cluster contains all the scan points belonging to an object, and we want to determine for each object a feature vector that describes it. Therefore, we adapt the features computed for the terrain grid cells, with some modifications. We group the features in two feature groups: geometric features and intensity features.

Geometric features

1. – 3. *Dimensions x, y, z*: Dimensions of the bounding box of the cluster in x, y, and z direction

4. *Density* ρ: Density of the points in the cluster, that is, the number of points N divided by the volume of the bounding box: $\rho = \frac{N}{x \cdot y \cdot z}$

Intensity features

5. – 6. *Minimum and maximum intensities* I_{min}, I_{max}: Minimum and maximum intensity values of all points of a cluster

7. *Range of intensity* I_r: Difference between the minimum and maximum intensity value

8. – 10. *Mean, median and standard deviation of intensity* I_μ, I_m, I_σ: Mean, median and standard deviation of the intensity values of all points of a cluster

11., ... *Intensity histogram* $h^{[1]}, ..., h^{[B]}$: Histogram with B bins of the intensity values of all points of a cluster

Overall, we thus obtain a feature vector x_{object} of length $10 + B$:

$$x_{\text{object}} = \left(x, y, z, \rho, I_{\text{min}}, I_{\text{max}}, I_{\text{r}}, I_\mu, I_{\text{m}}, I_\sigma, h^{[1]}, ..., h^{[B]}\right) \tag{6.11}$$

After training a model, like Random forests, we can classify objects using these features. In addition to terrain probabilities, all grid and map cells now also store probability values for each object class. To integrate the classification results into the map,

we project the bounding box of each cluster onto the map, and each map cell whose center lies within the projected bounding box is updated with the current classification result. We can use the same update rule here as for the terrain (see Eq. (6.4) - (6.6)). And finally, when we want to use the results of object classification for localization, as mentioned above, we can now also use the same particle weights here as for the terrain (see Eq. (6.8)).

6.7 Experiments and Results

In this section, we present our experiments on temporal terrain classification, semantic robot localizaton, and 3D LiDAR-based object classification.

Temporal terrain classification

For evaluation of the temporal-classification method five terrain maps like the one shown in Fig. 6.6c were labeled by hand for having ground truth. Since the ground-truth data generated for such large maps is less precise than for the individual terrain grids, the classification rates are only approximate values, but with a tendency towards under-estimation. Tab. 6.1 shows the results of temporal classification, and for comparison, the results of cell-wise and spatial classification from the previous chapter. Only using temporal classification, that is, without considering spatial dependencies, a classification rate of 92.2% is achieved. This is a better result then classifying cells individually, but it is worse than the spatial classification results. And finally, using spatio-temporal classification with recomputed probabilities as described in Sec. 6.3.3 yields an almost perfect classification with 98.4%. An example of what a difference the recomputation of terrain probabilities makes can be seen in Fig. 6.6d. Here, the information of the left map cannot be used by the robot to drive along the grass border.

Table 6.1: Results of the various terrain-classification methods

Classification method	Classification rate in %
Cell-wise classification	82.0
Spatial classification	96.8
Temporal classification w/o recomp.	92.2
Temporal classification w/ recomp. (Eq. (6.7))	98.4

Classification rates using 10-fold cross-validation for cell-wise classification, spatial classification, and temporal classification without and with recomputation of terrain probabilities

Table 6.2: Average runtimes of the various terrain-classification methods and robot localization using a particle filter

	Average time [ms]	Std. dev. [ms]
Spatial classification	22.8	3.0
Temporal classification	1.0	0.2
Elevation mapping	0.1	0.4
Spatio-temporal classification	**23.9**	**3.6**
Localization w/ 2000 particles	29.9	0.4

Tab. 6.2 shows the average runtimes of the various terrain-classification methods (using a CPU with 3.20 GHz). Temporal classification, which consists of projecting grid cells onto the terrain map, and recomputation and temporal updating of terrain probabilities, is very fast and only takes 1.0 ms on average. Together with spatial classification and elevation mapping, we get an average runtime for spatio-temporal classification of 23.9 ms, which corresponds to about 41.8 Hz. This shows that our classification method is real-time capable.

Semantic robot localization

To show that these terrain and elevation maps can also be used for robot localization, we recorded five log files of our robot driving along the test track seen in Fig. 6.6a, with an approximate length of 120 m. Since we are not interested in a precise but a semantic localization, we define a successful localization as one where the robot can localize itself from the beginning (at spot ①) to the end (at spot ③) of the track, and where it knows at any time in which exact terrain segment of the track it is located; otherwise, the whole localization test is regarded as not successful. For this purpose, from each of the log files a map was built, while localization was tested with the other four. The whole procedure was repeated five times for a total of 80 localization tests. Using a particle filter with 2000 particles and $\kappa = 0.5$, of these 80 tests 66 were successful, which is 82.5%. The localization uncertainty grows as the robot drives over open field (see Fig. 6.6c), but decreases abruptly as a different terrain type or a wall is seen.

3D LiDAR-based object classification

For the experiments on our object-classification method, we consider six classes of objects that occur outdoors: bench, birch, boxwood, car, roses, and stone (see Fig. 6.8). In order to generate a ground-truth data set, approximately 500 samples per class were

labeled by hand. We tested different numbers of bins $B = 2, ..., 30$ for the intensity histogram, and various classifiers. For descriptions and references of the tested classifiers see Sec. 4.6. Tab. 6.1 shows the results after 10-fold cross-validation, with the number B that gave the best results. Again, random forests outperform the other tested classifiers, with a classification rate of 93.6% (if 100 trees are used).

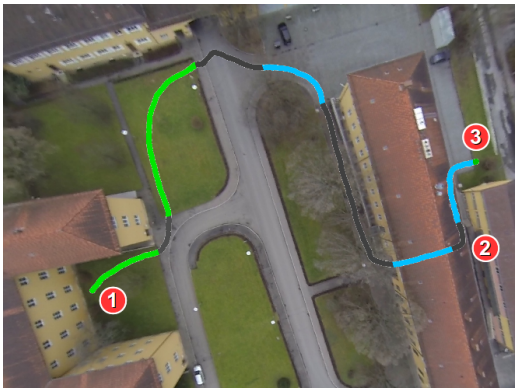
Table 6.3: Classification rates for 3D LiDAR-based object classification

Classifier	Number of bins B	Classification rate in %
RandomForest (10 trees)	8	91.6
RandomForest (100 trees)	**14**	**93.6**
RandomForest (200 trees)	13	93.8
MultilayerPerceptron	27	87.9
J48	18	85.5
Logistic	25	84.0
SMO	25	81.0

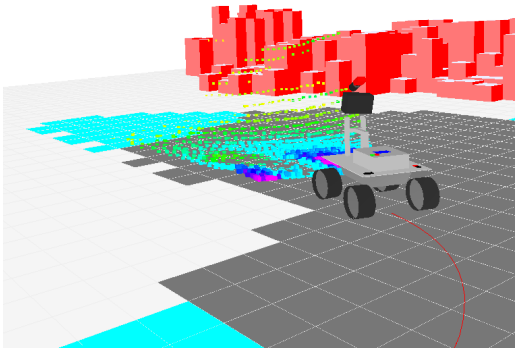
Table 6.4: Average runtimes of 3D LiDAR-based object classification

	Average time [ms]	Std. dev. [ms]
Clustering (per point cloud)	2.4	3.5
Feature extraction (per object)	0.1	0.3
Classification (per object) (random forest w/ 100 trees)	0.1	0.2

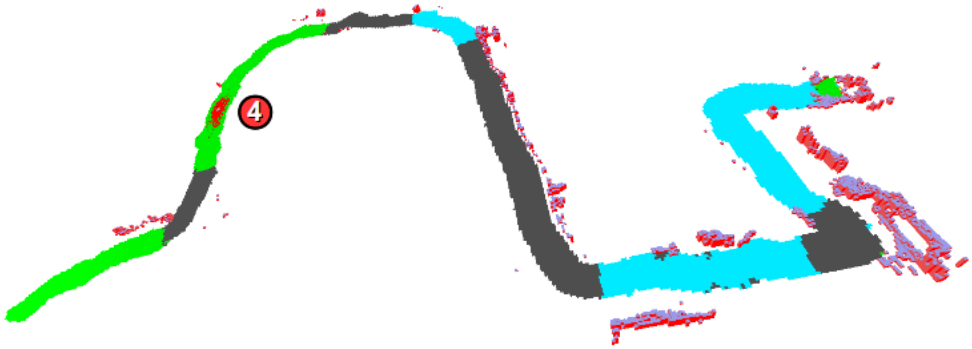
Tab. 6.4 shows the average runtimes of the method (using a CPU with 2.80 GHz). For clustering we set the radius $r = 0.2$ m. Clustering then takes 2.4 ms per point cloud, with a high standard deviation since the runtime depends on how many scan points lie above the ground plane. Feature extraction and classification is very fast with a mean runtime of 0.1 ms each per detected object.



(a) Aerial view of the test track (only used for illustrative purposes). On the right side, the route leads through a covered passage.

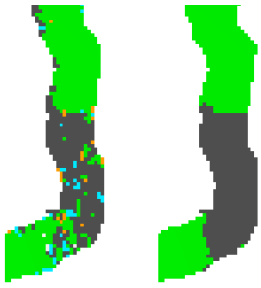


(b) The robot building a map as it drives past spot ②



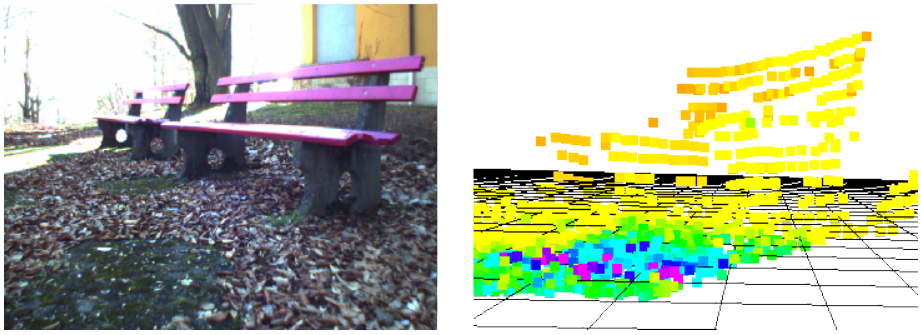
(c) The final terrain and elevation map. At spot ④ the particles of the particle filter are seen as red dots.

(d) Left: Mapping without recomputation of terrain probabilities, right: with recomputation as described in Sec. 6.3.3

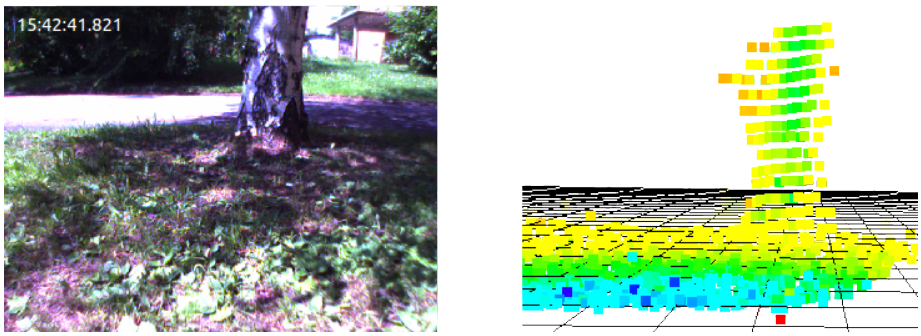


(e) Another example of the impact of considering spatial dependencies in terrain classification

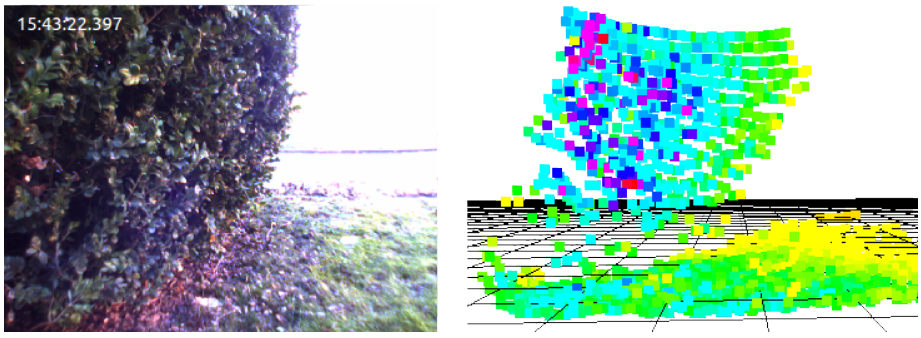
Figure 6.6: An aerial view of the test track with manually drawn route is shown in (a). (b) shows a 3D view of the mapping process with visible laser measurement points. The final terrain and elevation map can be seen in (c), and (d) shows what a difference the consideration of spatial dependencies can make. (Gray: asphalt, blue: cobblestones, green: grass, yellow: gravel, red:obstacles)



(a) Bench

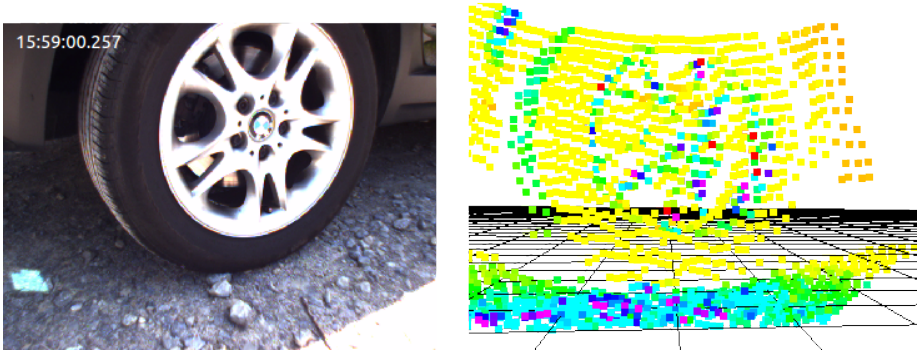


(b) Birch

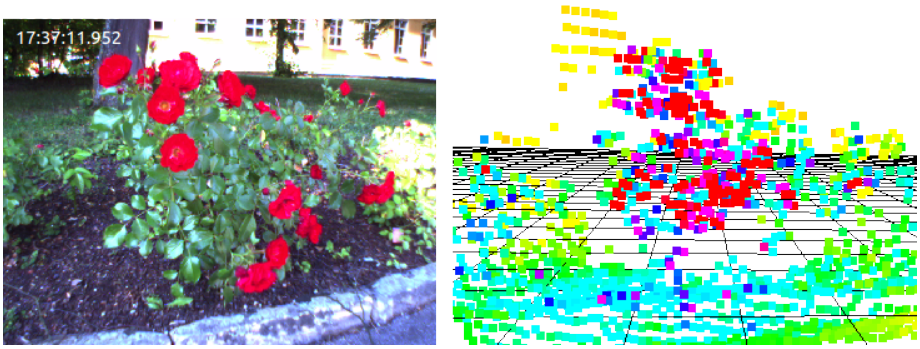


(c) Boxwood

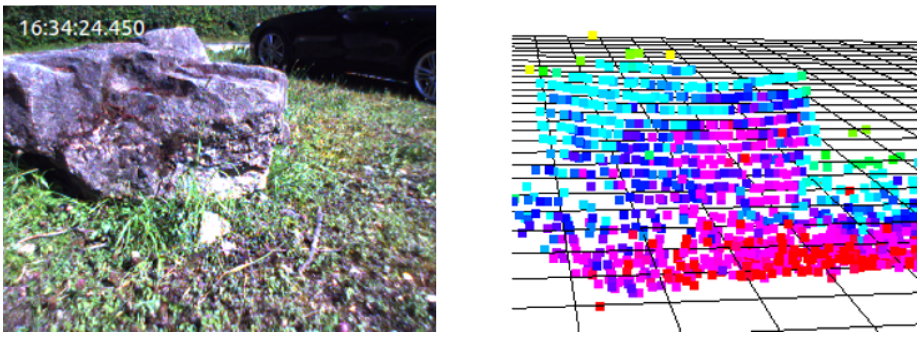
Figure 6.7: Images and 3D scans of three object classes under consideration



(a) Car



(b) Roses



(c) Stone

Figure 6.8: Images and 3D scans of another three object classes under consideration

6.8 Conclusions

We presented a method for building local terrain and elevation maps with spatio-temporal terrain classification. The surrounding terrain of a driving robot could be classified with a classification rate of 98.4% using four terrain classes. (Only considering spatial dependencies gave a classification rate of 96.8%, and only considering temporal dependencies gave 92.2%.) We exploit the fact that terrain occurs in contiguous areas and therefore has a high spatial and temporal coherence. Considering this dependencies in a probabilistic manner provides very high classification rates. The presented method is hereby not limited to only terrain classification, but can be used for all grid-based classification problems. We also showed how the results of spatial classification can be used for temporal classification by updating the terrain probabilities accordingly and by integrating the terrain grids into the map efficiently. The whole classification and mapping process runs at 41.8 Hz and is thus real-time capable.

We also showed that the terrain maps can be used for semantic robot localization using a particle filter, where the robot localizes itself based on terrain and obstacles. Other information can be easily integrated into our existing framework as well. We therefore presented a 3D LiDAR-based object-classification method and got a classification rate of 93.6 % in our experiments considering six object classes.

Chapter 7

Conclusions

This conclusion offers a brief review of the most important points and significant results. Since all scientific work is only an intermediate step, this section also includes possible directions for future work.

Chapter 4 began with the classification of terrain using only a 3D LiDAR and a consideration of the five terrain classes: asphalt, cobblestones, grass, gravel, and tiles. A robust method was presented for detecting the ground plane in each 3D scan by exploiting temporal coherences between consecutive frames. The terrain in front of the robot can then be divided into a terrain grid. Despite the fact that the resolution of the LiDAR is very low, and therefore only a few scan points per grid cell are available, we were able to extract characteristic feature vectors for each cell. It was possible to create 3D features that are based on height and intensity variations and that are very fast and easy to compute. In that case, the intensity values provide particularly good features, since each of the various types of terrain reflects the incident infrared beams of the LiDAR in a different characteristic manner. The feature extraction for one grid only takes 1.5 ms on average. Since for robots that drive outdoors changing lighting conditions represent a formidable challenge, this approach was tested at different times of the day, from bright sunlight at midday to only diffuse lights from street lamps at night. The LiDAR-based approach provides consistently good results independent of the conditions, with a best classification rate of 94.6% at midday, and the worst result at dusk, with 90.0%, using a terrain grid with a resolution of 20 cm. A general model trained for use at all conditions achieves a classification rate of 90.5%. For classification, we use Random forests since they outperform all other tested classifiers. Interestingly, when only considering the two classes of asphalt and grass the classification results are consistently above 99.9% for all conditions. The chlorophyll in grass reflects near-IR light like that of the LiDAR much more than asphalt, so these two classes can be distinguished almost perfectly.

Due to the low resolution of the 3D LiDAR only grid cells under two meters in front of the robot can be classified. Cameras have a wider field of view, and this approach incorporates the texture information of the camera images in order to classify more distant cells. To this end, we used LTP, a texture descriptor based on differences in pixel intensities. For grid cells where data of both sensors are present, the results are fused

accordingly. We can then classify grid cells up to four meters in front of the robot. In contrast to the LiDAR-based approach, the results of camera-based classification, and thus also the fused results, depend to a great deal on the prevailing light conditions. In the experiments the best fused results occur in the morning with 92.6% and the worst in the night, with 72.7%, and a classification rate of 81.4% for the general model.

In Chapter 5, the cell-wise classification is extended by taking into account spatial dependencies between grid cells using Conditional Random Fields (CRFs). We show that they are better suited for our task than Markov Random Fields. This new classification method consists of two stages. In the first stage, the cells of the terrain grid are classified individually using the extracted LiDAR and camera features and the learned Random forests, as in Chapter 4. In the second stage, the classification result is improved. This is achieved by considering the feature-dependent results, on the one hand, but additionally favoring contiguous areas of terrain, on the other. An approximate optimal solution can be found in formulating the classification task as an energy-minimization problem and using Gibbs sampling in a simulated-annealing scheme. To test the method we use a data set with the four terrain classes: asphalt, cobblestones, grass, and gravel, (since these are the terrain classes mainly encountered by our robot), and with many terrain grids with two or more terrain classes. These experiments show that taking spatial dependencies into account indeed leads to a huge improvement, with a classification rate of 81.5% after the first stage, and a rate of 94.2% after the second stage of our approach. In addition, we have developed a new energy term for describing spatial dependencies between neighboring cells, which is very fast to compute. Tested on a new data set, both, the previous as well as the new energy term, can improve cell-wise classification with a classification rate of 82.0% to 96.8% in the second stage. However, with the new energy term, the computation of the second stage is more than five times faster.

After incorporating spatial dependencies, temporal dependencies are additionally considered in Chapter 6. This not only improves results further, but we are now able to build whole terrain maps of the environment by using the classification result of the current time step to constantly update the map as the robot drives. This chapter shows how to efficiently incorporate the terrain probabilities of the terrain grid into the map, how to combine them with results from previous time steps, and how to integrate the label configuration obtained from simulated annealing by recomputing the terrain probabilities afterwards. In experiments using a new data set, spatial classification yields a classification rate of 96.8%, only using temporal classification (without recomputation of terrain probabilities) yields 92.2%, and finally spatio-temporal classification yields the best result with 98.4%.

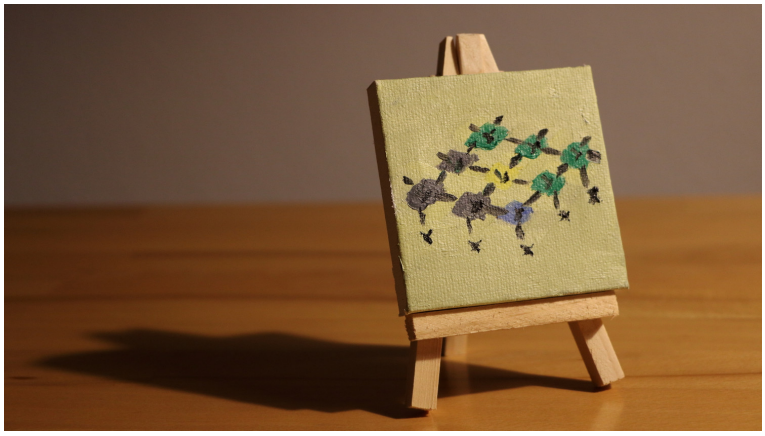
In addition to terrain, this method also uses information about detected obstacles in our maps to build combined terrain and elevation maps. Thereby, each map cell contains a height value with associated variance in addition. Cells with a height value that exceeds a certain threshold are considered as obstacles. Spatio-temporal classification and eleva-

tion mapping take 23.9 ms on average, which corresponds to about 41.8 Hz. This shows that our classification method is real-time capable. We can use these maps to localize the robot using Monte Carlo localization. Therefore, we compute particle weights that involve measurements about terrain and obstacles. Using a particle filter, it is simple to incorporate additional information like GPS or visual odometry.

This thesis demonstrates how considering spatial and temporal dependencies improves results significantly, with cell-wise classification as the basis of the method. Despite the fact that we get nearly perfect results in these test scenarios, each model and each method has its limitations, and we discuss some potential directions for future work.

In recent work [Otte *et al.* (2015)] we deal with improving cell-wise classification. There, we use local ternary patterns in combination with recurrent neural networks instead of Random forests. Another way to improve the classification results is to use a more complex model of the terrain grid. Since this study only considers probabilistic interactions between neighboring terrain cells, frequent cases — like an asphalt road with grass on both sides — are not covered. Therefore, long-distant interactions had to be considered, which would lead to a more complex model of the terrain grid and a much higher computational effort. A possible approach to make this model tractable is to not consider all interactions between grid cells in finding the optimal label configuration, but rather only consider a few by using sampling techniques.

The approaches mentioned above would further improve results, but they do not change our general approach: namely that we use a model of the terrain, regarding appearance and spatial dependencies, that is learned once and then never touched at all. Although this a-priori knowledge can be very powerful, and the learned models can cope with terrain that looks very different from the training data — in terms of lighting, texture, or even leaves on the ground — this approach has its limitations. There are problems when the appearance of the terrain differs too much from the learned model, because of seasonal changes, for instance, like snow or foliage, or because of mowed grass. Even if the learned model could cover all conceivable cases, the quality of this general model would suffer with respect to a more specialized model. A solution to this problem could be to use a semi-supervised learning method with an adaptive model that starts with a-priori knowledge, but constantly updates its knowledge as new information is received. To incorporate this into our method, the Random forests would need to be updated with features of terrain patches that belong to a certain terrain class with a high probability, but that look different than the other patches of this class. Hereby, the terrain map helps, since as well as we can draw conclusions about the position of the robot based on terrain, we can also draw conclusions about the terrain in front of the robot when we know the robot's position. So when the appearance of the terrain gradually changes, the model can adapt.



Symbols

$\mathcal{S}$	Sets are denoted by calligraphic letters
N	The cardinality of a set is denoted by capital letters
x	D-dimensional feature vector of a terrain grid cell
$x^{[i]}$	i-th component of feature vector x
y	Class label of a terrain grid cell
x_i	Feature vector of i-th cell
y_i	Class label of i-th cell
$\mathbf{x}$	Feature configuration of entire terrain grid
$\mathbf{y}$	Label configuration of entire terrain grid
X, Y	Random variables
$\mathcal{X}, \mathcal{Y}$	Sets of random variables
$X = x$	X takes value x
$p(Y = y \mid X = x)$	Probability of Y taking value y, given that X takes value x
$p(y \mid x)$	Shorthand notation for $p(Y = y \mid X = x)$
$\widetilde{p}(y)$	Unnormalized probability value, which can be normalized by dividing by $\sum_y \widetilde{p}(y)$
$F(y,x) \propto_y G(y,x)$	$\equiv \forall x \exists k \forall y \; F(y,x) = kG(y,x)$ $F(y,x)$ is proportional to $G(y,x)$ for all y, when x is fixed.
$\arg\max_y F(y)$	$= \{y : \forall y' \; F(y') \leq F(y)\}$ Set of values of y that maximize $F(y)$
$\mathbf{1}_{\{a \neq b\}}$	Indicator function defined as: $\mathbf{1}_{\{a \neq b\}} = \begin{cases} 1, & \text{if } a \neq b \\ 0, & \text{if } a = b \end{cases}$
$y \sim p(y)$	Sample y from the distribution $p(y)$

Bibliography

Andreasson, H., Triebel, R., and Lilienthal, A. (2007). Non-iterative Vision-based Interpolation of 3D Laser Scans. *Autonomous Robots and Agents*, **76**, 83–90.

Aslam, J. A., Popa, R. A., and Rivest, R. L. (2007). On Estimating the Size and Confidence of a Statistical Audit. In *Proceedings of the USENIX Workshop on Accurate Electronic Voting Technology*, EVT'07, page 8, Berkeley, CA, USA. USENIX Association.

AVT (2008). *Technical Manual For CCD models with serial numbers: xx/yy-6zzzzzzz and all CMOS models*. Allied Vision Technologies GmbH, Taschenweg 2a D-07646 Stadtroda / Germany, v2.4.0 edition.

Bay, H., Ess, A., Tuytelaars, T., and van Gool, L. (2008). Speeded-up Robust Features (SURF). *Computer Vision and Image Understanding (CVIU)*, **110**(3), 346–359.

Berthod, M., Kato, Z., Yu, S., and Zerubia, J. (1996). Bayesian Image Classification Using Markov Random Fields. *Image and Vision Computing*, **14**, 285–295.

Besag, J. (1974). Spatial interaction and the statistical analysis of lattice systems. *Journal of the Royal Statistical Society. Series B*, **36**, 192–236.

Beyer, W. H. (1984). *C.R.C. Standard Mathematical Tables*. CRC Press.

Borrmann, D., Elseberg, J., Lingemann, K., and Nüchter, A. (2011). The 3D Hough Transform for Plane Detection in Point Clouds: A Review and a New Accumulator Design. *3D Res.*, **2**(2), 32:1–32:13.

Bouguet, J. Y. (2008). Camera calibration toolbox for Matlab.

Boykov, Y. and Jolly, M.-P. (2001). Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images. In *ICCV*, pages 105–112.

Boykov, Y., Veksler, O., and Zabih, R. (2001). Fast Approximate Energy Minimization via Graph Cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, **23**(11), 1222–1239.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (2001). Random Forests. *Machine Learning*, **45**(1), 5–32.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and Regression Trees*. Wadsworth.

Burgard, W., Fox, D., Hennig, D., and Schmidt, T. (1996). Estimating the Absolute Position of a Mobile Robot Using Position Probability Grids. In *AAAI/IAAI, Vol. 2*, pages 896–901.

Chum, O., Matas, J., and Obdržálek, Š. (2004). Enhancing RANSAC by Generalized Model Optimization. In K.-S. Hong and Z. Zhang, editors, *Proc. of the Asian Conference on Computer Vision (ACCV)*, volume 2, pages 812–817, Seoul, Korea South. Asian Federation of Computer Vision Societies.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, **20**(3), 273–297.

Criminisi, A., Shotton, J., and Konukoglu, E. (2012). Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends® in Computer Graphics and Vision*, **7**(2–3), 81–227.

Dellaert, F., Fox, D., Burgard, W., and Thrun, S. (1999). Monte Carlo Localization for Mobile Robots. In *IEEE International Conference on Robotics and Automation (ICRA99)*.

Deng, H. and Clausi, D. A. (2004). Unsupervised Image Segmentation Using A Simple MRF Model with A New Implementation Scheme. In *ICPR (2)*, pages 691–694.

Diebel, J. and Thrun, S. (2005). An Application of Markov Random Fields to Range Sensing. In *Proceedings of Conference on Neural Information Processing Systems (NIPS)*, Cambridge, MA. MIT Press.

Dube, D. and Zell, A. (2011). Real-time plane extraction from depth images with the Randomized Hough Transform. In *Workshop on Challenges and Opportunities in Robot Perception (ICCV 2011)*.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. J. (1998). *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press.

Fischler, M. A. and Bolles, R. C. (1981). Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, **24**(6), 381–395.

Friedman, J. H., Bentley, J. L., and Finkel, R. A. (1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.*, **3**(3), 209–226.

Fulkerson, B., Vedaldi, A., and Soatto, S. (2009). Class Segmentation and Object Localization with Superpixel Neighborhoods. In *Proceedings of the International Conference on Computer Vision (ICCV)*.

Geman, S. and Geman, D. (1984). Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**(6), 721–741.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, **11**(1), 10–18.

Hammersley, J. and Clifford, P. (1971). Markov fields on finite graphs and lattices. Unpublished.

Hanten, R. (2011). *Fusion von 3D-Laserscanner-Daten mit 2D-Bilddaten*. Bachelor's thesis, University of Tuebingen.

Happold, M., Ollis, M., and Johnson, N. (2006). Enhancing Supervised Terrain Classification with Predictive Unsupervised Learning. In *Robotics: Science and Systems*.

Häselich, M., Arends, M., Lang, D., and Paulus, D. (2011). Terrain Classification with Markov Random Fields on fused Camera and 3D Laser Range Data. In *Proceedings of the 5th European Conference on Mobile Robotics (ECMR)*, pages 153–158.

Häselich, M., Arends, M., Wojke, N., Neuhaus, F., and Paulus, D. (2013). Probabilistic terrain classification in unstructured environments. *Robotics and Autonomous Systems*, **61**(10), 1051–1059.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, **57**(1), 97–109.

He, X., Zemel, R. S., and Carreira-Perpiñán, M. Á. (2004). Multiscale Conditional Random Fields for Image Labeling. In *CVPR (2)*, pages 695–702.

ISO (2012). *ISO/IEC 14882:2011 Information technology — Programming languages — C++*. International Organization for Standardization, Geneva, Switzerland.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, **82**(Series D), 35–45.

Khan, Y. N., Komma, P., and Zell, A. (2011). High Resolution Visual Terrain Classification for Outdoor Robots. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1014–1021, Barcelona, Spain.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *SCIENCE*, **220**(4598), 671–680.

Kleiner, A. and Dornhege, C. (2007). Real-time localization and elevation mapping within urban search and rescue scenarios. *Journal of Field Robotics*, **24**(8-9), 723–745.

Knuth, D. E. (1997). *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3 edition.

Koller, D., Friedman, N., Getoor, L., and Taskar, B. (2007). Graphical Models in a Nutshell. In L. Getoor and B. Taskar, editors, *An Introduction to Statistical Relational Learning*. MIT Press.

Komma, P. and Zell, A. (2010). Markov Random Field-based Clustering of Vibration Data. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)*, pages 1902–1908, Taipei, Taiwan.

Kumar, S. and Hebert, M. (2003). Discriminative Fields for Modeling Spatial Dependencies in Natural Images. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, pages 1531–1538. MIT Press.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Laible, S. (2009). *Klassifikation von Pflanzen anhand 3D-Laserscanner-Daten mittels maschineller Lernverfahren*. Diplomarbeit, University of Tuebingen, Sand 1, 72076 Tübingen.

Laible, S. and Zell, A. (2014). Building Local Terrain Maps Using Spatio–Temporal Classification for Semantic Robot Localization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, Chicago, Illinois, USA.

Laible, S., Khan, Y. N., Bohlmann, K., and Zell, A. (2012). 3D LIDAR- and Camera-Based Terrain Classification Under Different Lighting Conditions. In *Autonomous Mobile Systems 2012*, Informatik aktuell, pages 21–29. Springer Berlin Heidelberg.

Laible, S., Khan, Y. N., and Zell, A. (2013). Terrain Classification With Conditional Random Fields on Fused 3D LIDAR and Camera Data. In *European Conference on Mobile Robots (ECMR 2013)*, Barcelona, Catalonia, Spain.

Lalonde, J.-F., Efros, A. A., and Narasimhan, S. G. (2010). Detecting ground shadows in outdoor consumer photographs. In *European Conference on Computer Vision (ECCV 2010)*, pages 322–335. Springer.

le Cessie, S. and van Houwelingen, J. (1992). Ridge Estimators in Logistic Regression. *Applied Statistics*, **41**(1), 191–201.

Li, S. Z. (2009). *Markov Random Field Modeling in Image Analysis*. Springer Publishing Company, Incorporated, 3rd edition.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, **60(2)**, 91–110.

Martínez-Gómez, J., Morell, V., Cazorla, M., and García-Varea, I. (2016). Semantic localization in the PCL library. *Robotics and Autonomous Systems*, **75**, 641–648.

McDaniel, M. W., Nishihata, T., Brooks, C. A., and Iagnemma, K. (2010). Ground plane identification using LIDAR in forested environments. In *ICRA*, pages 3831–3836. IEEE.

Moravec, H. (1988). Sensor Fusion in Certainty Grids for Mobile Robots. *AI Mag.*, **9**(2), 61–74.

Mufti, F., Mahony, R. E., and Heinzmann, J. (2012). Robust estimation of planar surfaces using spatio-temporal RANSAC for applications in autonomous vehicle navigation. *Robotics and Autonomous Systems*, **60**(1), 16–28.

Myneni, R. B., Hall, F. G., Sellers, P. J., and Marshak, A. L. (1995). The interpretation of spectral vegetation indexes. *Geoscience and Remote Sensing, IEEE Transactions on*, **33(2)**, 481–486+.

Nippon (2008). *Laser Ranging Image / Sensor model FX6*. Micro Electro Mechanical Systems Promotion Dept. Visionary Business Center The Nippon Signal Cp., LTD.

Nippon (2010). *FX8 Range-Finding Sensor*. MEMS Promotion Department, Visionary Business Center, Nippon Signal Co., Ltd.

Nowozin, S. and Lampert, C. H. (2011). Structured Learning and Prediction in Computer Vision. *Foundations and Trends in Computer Graphics and Vision*, **6**(3–4), 185–365.

Nüchter, A. and Hertzberg, J. (2008). Towards semantic maps for mobile robots. *Robotics and Autonomous Systems*, **56**(11), 915–926.

Otte, S., Laible, S., Hanten, R., Liwicki, M., and Zell, A. (2015). Robust Visual Terrain Classification with Recurrent Neural Networks. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning (ESANN 2015)*.

Platt, J. (1998). Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press.

Platt, J. (1999). Probabilistic outputs for support vector machines and comparison to regularize likelihood methods. In A. Smola, P. Bartlett, B. Schoelkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 61–74.

Quinlan, R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA.

Rasmussen, C. (2002). Combining Laser Range, Color, and Texture Cues for Autonomous Road Following. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 4320–4325.

Rauscher, G., Dube, D., and Zell, A. (2014). A Comparison of 3D Sensors for Wheeled Mobile Robots. In *2014 International Conference on Intelligent Autonomous Systems (IAS-13)*, Padova, Italy.

Rusu, R. B. (2009). *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. Ph.D. thesis, Technische Universität München.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 111–147.

Sutherland, I. E. and Hodgman, G. W. (1974). Reentrant Polygon Clipping. *Commun. ACM*, **17**(1), 32–42.

Sutton, C. A. and McCallum, A. (2012). An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, **4**(4), 267–373.

Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Torr, P. H. S. and Zisserman, A. (2000). MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. *Computer Vision and Image Understanding*, **78**, 2000.

Unnikrishnan, R. and Hebert, M. (2005). Fast Extrinsic Calibration of a Laser Rangefinder to a Camera. Technical Report CMU-RI-TR-05-09, Robotics Institute, Pittsburgh, PA.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, **45**(1), 41–51.

Weiss, U., Biber, P., Laible, S., Bohlmann, K., and Zell, A. (2010). Plant Species Classification Using a 3D LIDAR Sensor and Machine Learning. In *Machine Learning and Applications (ICMLA), 2010 Ninth International Conference on*, pages 339–345.

Wellington, C., Courville, A., and Stentz, A. T. (2006). A Generative Model of Terrain for Autonomous Navigation in Vegetation. *The International Journal of Robotics Research*, **25**(12), 1287–1304.

Wikipedia (2014). Lissajous-Figur — Wikipedia, Die freie Enzyklopädie. [Online; Stand 23. November 2014].

Wolf, D. F., Sukhatme, G. S., Fox, D., and Burgard, W. (2005). Autonomous Terrain Mapping and Classification Using Hidden Markov Models. In *International Conference on Robotics and Automation*, pages 2038–2043.

Wurm, K. M., Stachniss, C., Kümmerle, R., and Burgard, W. (2009). Improving Robot Navigation in Structured Outdoor Environments by Identifying Vegetation from Laser Data. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, St. Louis, MO, USA.

Wurm, K. M., Kretzschmar, H., Kümmerle, R., Stachniss, C., and Burgard, W. (2013). Identifying Vegetation from Laser Data in Structured Outdoor Environments. *Robotics and Autonomous Systems*.