BACHELOR PAPER

Term paper submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Engineering at the University of Applied Sciences Technikum Wien - Degree Program Mechatronics/Robotics

Artificial Neural Networks based State Transition Modeling and Place Categorization

By: Andreas Kriegler

Student Number: 1510330012

Supervisor: Wilfried Wöber MSc

Vienna, June 16, 2018

FH University of Applied Sciences
TECHNIKUM WIEN

Declaration

“As author and creator of this work to hand, I confirm with my signature knowledge of the relevant copyright regulations governed by higher education acts (see Urheberrechtsgesetz /Austrian copyright law as amended as well as the Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I hereby declare that I completed the present work independently and that any ideas, whether written by others or by myself, have been fully sourced and referenced. I am aware of any consequences I may face on the part of the degree program director if there should be evidence of missing autonomy and independence or evidence of any intent to fraudulently achieve a pass mark for this work (see Statute on Studies Act Provisions / Examination Regulations of the UAS Technikum Wien as amended).

I further declare that up to this date I have not published the work to hand nor have I presented it to another examination board in the same or similar form. I affirm that the version submitted matches the version in the upload tool.“

Vienna, June 16, 2018 Signature

Abstract

To aid the high-level path-planning decisions of a mobile robot, the robotics system has to know not just where the robot is, but also to identify the type and specifics of that place. This is useful in industrial applications such as surveillance and warehousing robots to restrain them to certain environments, but also for human-robot collaboration and consumer service robots, enabling them to find places such as ”kitchen" and get coffee from it, using a semantic map featuring the different class labels. Customarily, a machine- or deep-learning model such as a convolutional neural network is trained to correctly classify the current location from a video stream. However, a neural network requires fine-tuning for proper generalization and is further limited by the closed-set constraint.

The foundation of this thesis is an extensive state-of-the-art research to identify applications of neural networks, classic machine learning and control algorithms for state transition modeling and solving classification problems. The aim is to improve the generalization of an existing system used at the Queensland University of Technology for place categorization, and to extend the set of classes known to the system to relax the closed-set constraint. The system in its original state was only capable of classification via the neural network, which did not generalize well enough for immediate usage.

The convolutional neural network used for feature extraction of the image stream has been augmented with trained machine learning models such as support vector machines and multi-class logistic regression. The usage of these models has extended the classification to include new place categories that are part of the finite set of classes, has helped to overcome uncertainties from images showing features of multiple classes, and has removed the ambiguities introduced by the Random Forest of One-vs-All classifiers employed in the original system.

The augmented system outperforms the basic neural network by correctly classifying 90% of images (98% for the three-class environment) instead of only 78%, including a place that the neural network was not trained on. In future projects, the images will be captured by mounting the camera on a mobile robot, the classification will be expanded to all 12 of the distinct places observable on campus, the semantic mapping system will be improved and the knowledge acquired will be used to aid high-level path-planning decisions.

Keywords: place categorization, neural networks, machine learning, support vector machines, mobile robotics

Kurzfassung

Um dem Personal in der industriellen Produktion und Fertigung, aber auch dem Konsumenten im Privatleben, die Kollaboration mit einem Robotersystem zu ermöglichen, muss der Roboter nicht nur wissen wo er ist, sondern auch wie man diesen Ort beschreibt. Dazu wird für gewöhnlich ein Maschinelles-Lernen oder Tiefes-Lernen Modell verwendet, wie eine faltendes neuronales Netzwerk, um die Örtlichkeit von einer Videoaufnahme klassifizieren zu können. Ein neuronales Netzwerk benötigt jedoch eine Feinabstimmung, um Generalisierung gewährleisten zu können. Darüber hinaus ist das Netz auf eine limitierte Anzahl von Klassen beschränkt.

Die Grundlagen dieser Arbeit bildet eine umfassende Stand-der-Technik Analyse, in der Anwendungen von neuronalen Netzwerken und klassischen Maschinelles-Lernen sowie Steuerungs- und Regelungstechnikalgorithmen für die Zustandsübergangsmodellierung und zur Lösung von Klassifizierungsproblemen untersucht werden. Das Ziel dieser Arbeit ist die Erweiterung eines bestehenden Systems zur Klassifizierung von Orten, in dem die Generalisierung verbessert und die Anzahl der bekannten Klassen erweitert wird.

Das entwickelte System kann rasch die gesammelten Videodaten analysieren und kategorisieren, um dem Roboter ein semantisches Verständnis zu ermöglichen. Das zur Merkmalextrahierung der Bilder verwendete faltende neuronale Netzwerk wurde mit unterschiedlichen Maschinelles-Lernen Modellen erweitert, wie einer Support Vector Machine, Multiclass Logistic Regression und einem Naive Bayes Filter. Dadurch konnte das Set von Klassen erweitert, Unsicherheiten bei Bildern mit Merkmalen mehrer Klassen beseitigt und Mehrdeutigkeiten des Random-Forests von One-vs-All Modellen des ursprünglichen Systems vermieden werden.

Das erweiterte System überragt die Klassifizierung des neuronalen Netzes, in dem 90% der Bilder (in der 3-Klassen Umgebung sogar 98%) anstatt 78% richtig zugeordnet werden. In zukünftigen Arbeiten soll die Klassifizierung auf alle 12 Klassen des Universitätsgeländes erweitert und das Kartenerstellungssystem verbessert werden. Außerdem soll die semantische Karte verwendet werden um komplexe Pfadplannungsprobleme lösen zu können.

Schlagworte: Standortklassifizierung, Neuronales Netzwerk, Maschinelles-Lernen, Support Vector Machine, Mobile Robotik

Acknowledgements

I would first like to thank my thesis supervisor at the UAS Technikum Wien *Wilfried Wöber* for his guidance during the course of the project and his active support that extends beyond this thesis. Without his in-depth technical knowledge and practical approach this project would have not been possible to be finished.

I would also like to extend my thanks to the degree program director *Corinna Engelhardt-Nowitzki* and the deputy program directors *Horst Orsolits* and *Mohamed Aburaia* for giving me the opportunity to become a part of this project.

Finally I am grateful to my aunts and uncles and my girlfriend for their everlasting support, compassion and love. You have always assured me of my own abilities and have helped me find this path, which has been a great pleasure travelling so far.

Contents

1 Introduction

The field of robotics has shaped industrial and manufacturing processes as well as the daily life of consumers since its inception. With rapid advancements in technology and science, the level of automation in the industry and the autonomy in mobile robotics is ever increasing (Ganek & Corbi, 2003; Siegwart et al., 2011; Brega et al., 2000). Researchers and experts are convinced that robots and artificially intelligent agents will become essential in the future (Tai & Liu, 2016*a*). An *agent* in this context is any system or part of a system that *acts* or behaves. Agents are most prominently defined in (Russell & Norvig, 2003) as computer programs that ”operate autonomously, perceive their environment, persist over a prolonged time period, adapt to change, and create and pursue goals".

As described in (Tai & Liu, 2016*a*), intelligent robotic agents are already used in a variety of real-life applications: warehousing robots, autonomously driven cars, unmanned aerial vehicles, industrial robots, service robots, *etc*. Mobile robots are a new and exciting feat of technology and are of particular interest to the service and healthcare industry, industrial handling and consumer electronics.

Researchers and engineers are nonetheless faced with some non-trivial unsolved challenges: both reliable perception and intelligent controls are fundamental components of a mobile robotics system. A considerable portion of this thesis is therefore dedicated to a state-of-the-art research review, identifying applications utilizing classical machine- and modern deep-learning algorithms for both control and perception tasks.

1.1 From Perception to SLAM and Semantic Mapping

Mobile service robots, that act in complex indoor and outdoor environments alongside humans, need to gain understanding of their surroundings that exceeds basic obstacle-avoidance and autonomous navigation (Sunderhauf et al., 2016). To do this they need to acquire semantic information about the location, by classifying the individual places of the environment they operate in (Galindo et al., 2008). The robot can then answer basic localization and SLAM (simultaneous localization and mapping) questions such as ”Where am I?" (Borenstein et al., 1996), but also ”What is the place I am in like?"(Sunderhauf et al., 2016). By gaining an understanding of the environment this way, the robot is better qualified to make appropriate high-level decisions when interacting with a person by modulating its behavior accordingly.

The underlying problem of properly labeling samples drawn from a data set is known as *classification* in Deep Learning (DL) (Goodfellow et al., 2016) and *place categorization* (Pronobis

et al., 2010; Xiao et al., 2016) when the data refers to parts of the environment. Combining these semantic place labels with the creation of a map is then referred to as *semantic mapping* (Pronobis & Jensfelt, 2012; Hemachandra et al., 2014).

1.2 Popular Methods and Their Shortcomings

Numerous methods following different approaches for the classification and mapping system are used in state-of-the-art place categorization and semantic mapping for mobile robots (Pronobis et al., 2010; Pronobis & Jensfelt, 2012; Ranganathan, 2010; Wu & Rehg, 2011). A convolutional neural network (CNN) is a highly-regarded method of extracting information from an image, most prevalent in computer vision, and is used in (Sunderhauf et al., 2016) for place categorization. In their work, the result of the CNN is used for simple top-1 classification, with no additional fine-tuning to support the generalization. The *closed-set* constraint of the CNN is relaxed, but only by employing a set of One-vs-All (OVA) classifiers. The ambiguity and class imbalance introduced by the OVAs is not taken into account (Bishop, 2016).

1.3 Contributions of the Proposed System

The purpose of this work is to develop a place categorization system employing a combination of a neural network and machine learning (ML) models. This work builds on the previous work by (Sunderhauf et al., 2016) and makes improvements in regards to the generalization and the expendable place categorization. To this end, ML models built on top of the CNN such as multiclass logistic regression (MLR) and support vector machines (SVM) are evaluated and used to enable the system to recognize new places with minimal training, allowing life-long learning for robotic applications that are assumed to encounter unknown environments in their prolonged autonomous operation. This work therefore provides the following contributions:

1. Departing from the classical approach of Bayesian filters, a research review outlines ML- and DL models employed in robotic applications.
2. Deriving a concrete method from this review, the application of a state-of-the-art CNN for classification is shown.
3. To help generalization and to extend the set of classes known to the CNN, three ML models were trained: multiclass logistic regression, support vector machines and naive Bayes filter (NBF).
4. The system is lastly tested and evaluated in a workplace-esque indoor environment on campus and in an industrial setting in the ”digital factory" of the university.

The starting point is Bayesian filters, because mathematical models following Bayesian statistics have given rise to numerous algorithms and filters used in *probabilistic robotics*.

1.4 From Bayesian Statistics to Machine Learning and Deep Learning

Constituting an important stepping-stone in robotics, the basics of probabilistic robotics, state transition modeling and Bayesian filters are outlined first in chapter 2. Following the restrictions of the Bayesian filters, attention then shifts to machine learning theory. The applications of machine learning models are detailed and models used for classification are explained using the support vector machine as example. Focus then shifts once more to the school of thought called deep learning. Numerous architectures of neural networks augmented with various algorithms are detailed for both control and perception tasks in (mobile) robotics.

Having arrived at the work of (Sunderhauf et al., 2016), the developed system is introduced in chapter 3. The implementation of the three ML models augmenting the *Places205* CNN used by Sunderhauf, are then detailed. Chapter 4 discusses the experimental setup and the achieved results. Conclusions are drawn and future work is discussed in chapter 5.

2 Theoretical Background and State-of-the-Art

Following the research of artificial intelligence and its impact on robotics has led to the works of Thrun et al. most notably the idea of probabilistic robotics (Thrun et al., 2005). In this section, the basic ideas of probabilistic robotics will be briefly outlined.

2.1 Probabilistic Mobile Robotics

Robotic systems are part of the physical world, in which they perceive information through sensors and manipulate their surroundings with actuators. In this process there are numerous factors that contribute to a robots *uncertainty* regarding its current state (Thrun et al., 2005):

1. The physical world is by nature unpredictable.
2. The *perception* of a robot is limited because sensors are restricted in the type and amount of information they can get from their surroundings.
3. The uncertainty from actuators stems from the fact that all robot motors, transmissions and other mechanical parts are either affected by control noise or wear-and-tear effects.
4. To ensure computational tractability of programs and control algorithms modeling the real world, various approximations have to be made. This is especially true for highly dynamic motions of the robot.

The effect of these uncertainties on the *belief* of a robot, which is the accumulated knowledge a robot has about its state, and the process of updating this belief as new controls are issued and measurements are taken is visualized in (Thrun et al., 2005) showing the idea of *Markov localization*. It describes how the robot gains confidence in its belief after taking more measurements and how it loses confidence as it moves due to the uncertainties in the movement.

The necessary mathematical models to describe robot movement and perception come from probability theory and follow *Bayes rule* and are outlined in more detail in appendix A.1. Also of note is the concept of *state completeness* as is explained in appendix A.2.

Let X donate a set of possible states x the robot can be in, Z be the set of individual sensor measurements z and U be the set of individual control steps u. A state x_t shall be defined as having taken measurement z_t, and the latest control being u_{t-1}. Following the assumption of state completeness, the state x_t is therefore a sufficient summary of all that transpired *so far*, meaning all controls $u_{1:t-1}$ and all measurements $z_{1:t}$. This means that the transition from one

state x_t to another state x_{t+1} is only influenced by the control taken at that time step u_t (Thrun et al., 2005):

$$p(x_{t+1} \mid x_{0:t}, z_{1:t}, u_{1:t}) = p(x_{t+1} \mid x_t, u_t) \tag{1}$$

The probability distribution $p(x_{t+1}|x_t, u_t)$ is known as *state transition* or *motion model*. A similar shape takes the generation model of the measurements under the assumption of state completeness (Thrun et al., 2005):

$$p(z_{t+1} \mid x_{0:t+1}, z_{1:t}, u_{1:t}) = p(z_{t+1} \mid x_{t+1}) \tag{2}$$

This means that the state x_{t+1} is enough to predict the measurement z_{t+1}. This relation is called *sensor* or *measurement model*. The state transition model and sensor model together describe the entire dynamical stochastic system of the robot and its environment and are brought together in a *dynamic Bayes network* (DBN). These two models are lastly incorporated into the general Bayes filter algorithm (figure 1) which requires the previous belief $bel(x_t)$, the latest control action u_t and the latest measurement z_{t+1} to compute a prior belief $\overline{bel}(x_{t+1})$ over the set of possible states and update it with the measurement model to give the posterior belief $bel(x_{t+1})$.

Algorithm Bayes_filter($bel(x_t), u_t, z_{t+1}$**):**

for all x_t *do*:

$$\overline{bel}(x_{t+1}) = \int p(x_{t+1} \mid x_t, u_t) bel(x_t) dx_t$$

$$bel(x_{t+1}) = \eta p(z_{t+1} \mid x_{t+1}) \overline{bel}(x_{t+1})$$

return $bel(x_{t+1})$

Figure 1: *General algorithm for Bayes filtering.* (Source: edited and taken from (Thrun et al., 2005))

Gaussian filters constitute the earliest tractable implementations of the general Bayesian filter algorithm. *Kalman filters* (KF) are one of the oldest and best documented implementations of a Gaussian filter. All of the filters trade computational tractability with accuracy. While KFs can only deal with linear systems, *Extended Kalman filters* (EKF) relax this linearity constraint by employing methods such as Taylor series expansion to approximate the desired function that gives the probability distribution of a dependent variable over a known input variable. The quality of the prediction of an EKF is still dependent on the local non-linearity of the system as shown in (Thrun et al., 2005).

A popular localization algorithm that uses particle filters instead is *Monte Carlo Localization* (MCL). In MCL, the belief $bel(x_{t+1})$ is represented by a set of M particles $X_{t+1} = \{x^1_{t+1}, x^2_{t+1}, \ldots, x^M_{t+1}\}$ (Thrun et al., 2005); samples are then drawn from the motion model using particles from the present belief. Areas with grouped, higher-valued particles represent states the robot is more likely to be in. MCL is applicable for both local and global localization problems and has become well known, although it struggles in scenarios where the location of the robot abruptly changes due to an external force, since the particles at places other than the most likely pose slowly fade away (Thrun et al., 2005).

2.2 Machine Learning Theory

With these limitations in mind, a modern approach employing neural networks to help determine the state of a robot, was to be found. Neural networks are part of the research field called *deep learning* placed in the more general theory of *machine learning*. Deep learning can be seen as a way to implement a lot of the machine learning models in a computationally tractable manner. Furthermore, the deep learning method that was to be derived (the CNN), had to be improved with some kind of classification model, and machine learning provides a vast amount of models used for regression and classification.

2.2.1 Classical Machine Learning

Machine learning proposes to use statistical techniques to allow a machine, system or algorithm to ”learn" with the usage of large amounts of data. Learning in this context means that the machine progressively improves performance on a specific task (Samuel, 1959). It evolved from the subject of pattern recognition, is closely related to computational statistics and plays a major role in the field of artificial intelligence (AI). Machine learning therefore seeks to construct a model that is able to learn from data and then make predictions on it (Kohavi, 1998).

One major criteria used to classify the type of system is related to the data sets used for teaching. If the desired output of the model for the given input data is known, the system is *supervised* and the approach is called *supervised learning*. If only parts of the desired outputs are known or there is some feedback given to the system in a dynamic environment this is called *semi-supervised* or *reinforcement learning* respectively. If no such labels or feedback is given and the system has to find a structure in the input data by itself it is referred to as *unsupervised learning*. The applications are vast and include all tasks that require classification, regression, clustering, distribution density estimation or dimensionality reduction, the last being used to fight the *Curse of dimensionality*, a term coined by Richard E. Bellman in (Bellman, 2013) originally published in 1957. The curse of dimensionality states, that as the dimensionality of the observed space increases, the volume increases so fast that the data becomes sparse.

Following a polynomial approach, if we have D input variables, then a general polynomial with coefficients up to order 3 would take the form

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{D} w_i x_i + \sum_{i=1}^{D}\sum_{j=1}^{D} w_{ij} x_i x_j + \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{k=1}^{D} w_{ijk} x_i x_j x_k. \tag{3}$$

For a polynomial of order M the growth in the number of coefficients is like D^M, meaning it might be hard to fill all the cells in the higher-dimensional space with data. Multiple approaches to generate the models are being studied such as *decision trees*, *artificial neural networks* (ANN), *support vector machines*, *genetic algorithms*, the aforementioned dynamic bayesian networks and more.

Following the research review given in section 2.3, it was decided to build on the work of (Sunderhauf et al., 2016) and build a system for the purpose of place categorization. While

place categorization via classification is somewhat different than state transition modeling, it can also be used to describe the state of a robot in a context that relates more strongly to the environment. Additionally, their work made no strong demands regarding the hardware to be used; in fact the experimentation detailed in chapter 4 was carried out with a simple webcam. Lastly, they provided existing source code that could easily be expanded upon, making the project feasible in the short time frame available.

To improve the system regarding its generalization and expand the set of 205 classes to new on-campus classes, an additional model used for classification was to be developed. Three different models, all machine learning algorithms, were to be trained, used for predictions and then evaluated: Multinomial logistic regression, a support vector machine and a naive Bayes filter. All three of these are very popular choices for solving classification problems and are also easily implemented in MATLAB with the Statistics and Machine Learning toolbox. An overview of the most prominent method, the support vector machine, shall now be given.

2.2.2 Support Vector Machines

A support vector machine is a supervised learning model that uses algorithms to analyze data so solve classification and regression problems. Given a set of training examples from the *training set*, each marked as belonging to one or the other of two categories (for binary classification), an SVM trains a model that assigns new examples from the *test set* to one category or the other, making it a non-probabilistic binary linear classifier. Given a training data set of n points of the form

$$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n) \tag{4}$$

where y_i be either 1 or -1 (in general, for the binary case), indicating the class to which the point $\boldsymbol{x_i}$ belongs, with $\boldsymbol{x_i}$ being a p-dimensional real vector. The goal is then to separate these points with a $(p-1)$ dimensional hyperplane (in case the datapoints are not linearly separable) such that the Hessian normal distance from the group of points to the "maximum-margin hyperplane" is maximized by minimizing some kind of loss function such as

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max(0, 1 - y_i(\boldsymbol{w}\cdot\boldsymbol{x}_i - b))\right] + \lambda\|\boldsymbol{w}\|^2 \tag{5}$$

where

$$\frac{b}{\|\boldsymbol{w}\|} \tag{6}$$

determines the offset of the hyperplane from the origin along the normal vector $\boldsymbol{w}$, with λ determining the tradeoff between increasing the margin-size and ensuring that $\boldsymbol{x_i}$ lies on the correct side of the margin. For SVMs the loss function is the *hinge-loss*

$$l_{hng}(y) = \max(0, 1 - t \cdot y) \tag{7}$$

where $t = \pm 1$ is the intended output and y is the classifier score of the classifiers's decision function, and not the predicted class label, and is usually given by

$$y = \boldsymbol{w} \cdot \boldsymbol{x} + b \tag{8}$$

for a linear SVM. The hinge-loss is also a *convex* function, so many of the usual convex optimizers can work with it. The hinge-loss in SVMs is also closely related to the logistic regression with the *log-loss*

$$l_{\log}(y) = \ln(1 + e^{-yt}). \tag{9}$$

When looking at the target functions that minimize the expected risk the optimal classifier is given by

$$f^{\star}_{hng}(x) = \begin{cases} 1 & \text{if } p(1 \mid x) > p(-1 \mid x) \\ -1 & \text{otherwise.} \end{cases} \tag{10}$$

For the logistic loss, it is the logit function

$$f^{\star}_{\log}(x) = \ln\left(\frac{p(1 \mid x)}{1 - p(1 \mid x)}\right). \tag{11}$$

Both target functions yield the correct classifier, but the logit-loss actually gives more information than is necessary.

The extension of the binary SVM to a multiclass SVM can be done by reducing the multiclass problem into multiple binary classification problems. Methods for this include the One-versus-All or One-versus-One (OVO) approach or a *directed acyclic graph* (DAG). In (Crammer & Singer, 2001) a multiclass SVM method was proposed that casts the multiclass classification problem into a single optimization problem, rather than decomposing it into multiple binary classification problems.

Finally, kernel methods are a class of algorithms used for pattern analysis that are able to operate in high-dimensional feature space without ever having to compute the coordinates of the data in that space, instead calculating the inner products between all pairs of data, which is referred to as the "kernel trick" (Theodoridis et al., 2008). Kernel methods are used to turn any linear model into a non-linear model by applying the kernel trick to replace the features with a kernel function. *Kernel density estimators* are used for non-parameteric density estimation. Assuming that samples are drawn from an unknown probability density $p(\boldsymbol{x})$ in D-dimensional space, the probability mass associated with a certain region R containing $\boldsymbol{x}$ is given by (Bishop, 2016)

$$P = \int_R p(\boldsymbol{x}) d\boldsymbol{x}. \tag{12}$$

Assuming N observations, the total number K of points falling inside R will be distributed according to the binomial distribution (Bishop, 2016)

$$Bin(K \mid N, P) = \frac{N!}{K!(N-K)!} P^K (1-P)^{1-K}. \tag{13}$$

The desired estimated density at $\boldsymbol{x}$ is lastly given by equation 39.

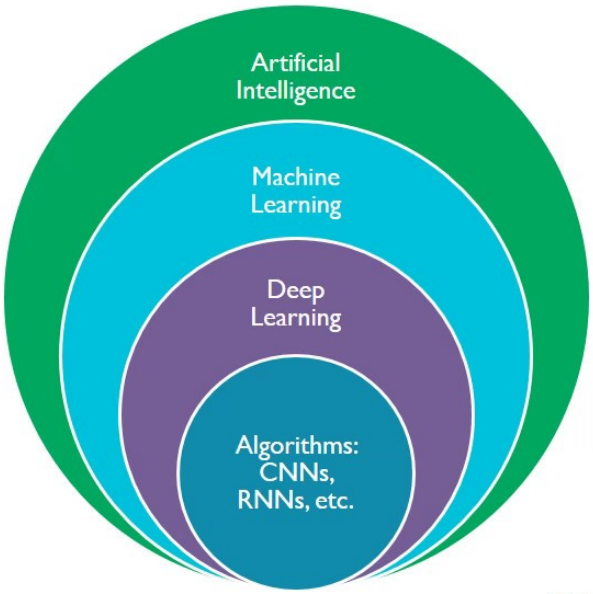

Figure 2: *AI landscape*: An overview of the field of artificial intelligence and its components. (Source: edited and taken from (Davies, 2016))

2.3 Deep Learning

As mentioned previously, this section provides an extensive state-of-the-art review identifying various neural network-based approaches for state transition modeling, robot control- and computer vision tasks, starting with the basic premise of artificial neural networks.

With the computational power of computer components such as the graphics processing unit (GPU) and the central processing unit (CPU) still increasing according to Moore's Law (Moore, 2006) and at the same time the relative cost of such hardware decreasing, an exciting new field has emerged called *deep learning*. Deep learning tries to solve problems of artificial intelligence by enabling the computer to learn from experience in terms of a hierarchy of concepts as explained by (Goodfellow et al., 2016) who then go on to say: "If we draw a graph showing how these concepts are built on top of each other, the graph is deep, with many layers. For this reason, we call this approach to AI deep learning." A general overview of the artificial intelligence landscape including deep learning can be observed in figure 2.

Drawing inspiration from biological neural networks, artificial neural networks or simply *neural networks* (NN) are learning algorithms used to generate the aforementioned input-output models for pattern recognition and statistical structuring of the joint probability distributions between observed variables. Artificial neural networks are expanded in deep learning with a set of multiple hidden layers. The approach of NNs can be called a *heuristic* one. They approximate the stochastic components of the system with some kind of mapping-function while employing a correction method such as reward, error or negative log-likelihood by updating some of their parameters. The basic layout of an ANN is given in figure 3. Every input value is assigned an input neuron $x_{1...k}$ in the input layer forming the *input vector* $\boldsymbol{x}$ where k is the number of different

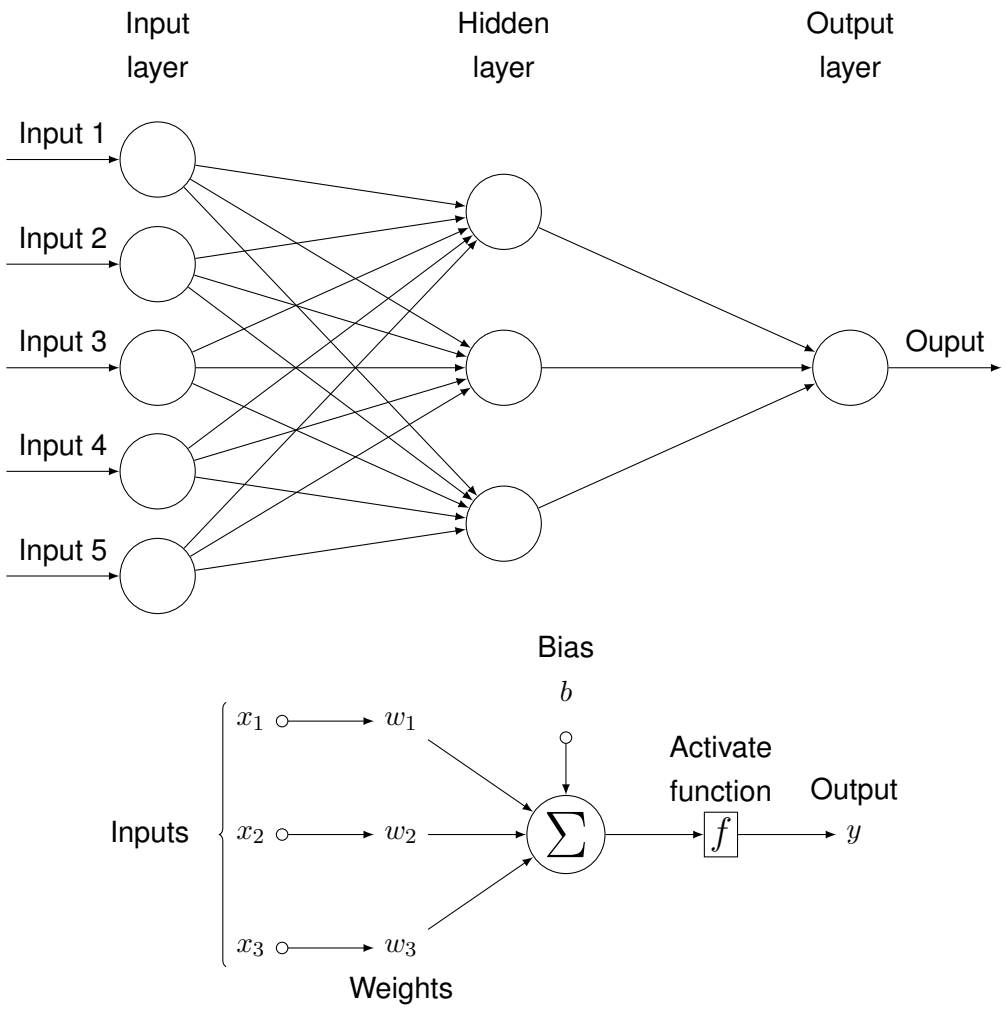

Figure 3: *General layout of an artificial neural network*: On top the basic layout and on the bottom specifics regarding the individual components. (Source: TeX code used from (Medina, 2013))

input variables. The values are duplicated and sent to all the nodes in the hidden layer. Before entering the node each input is multiplied with a weight variable $w_{1...k}$ forming the *weight vector* $\boldsymbol{w}$. The weighted inputs $w_0 \cdot x_0$ through $w_k \cdot x_k$ are then summed up with the corresponding *bias* b and fed into the activation function $f(\cdot)$, usually a *sigmoid* function acting as a smooth threshold that maps the received input to a corresponding output y (Smith et al., 1997):

$$sig(x) = \frac{1}{1 + e^{-x}}. \tag{14}$$

In a mathematical sense the weights influence the ”steepness", meaning the first derivative of the activation function.

$$sig'(x) = s(x)[1 - s(x)] \tag{15}$$

The bias neurons b are placed in an additional layer that has no previous connections and only feeds the neurons in the hidden layer(s). The point of the bias is to shift or *translate* the sigmoid curve along the input axis to allow the function to be more flexible. All bias neurons

output 1.0 and are connected with an extra set of weights. The output of a single sigmoid neuron then follows

$$y = sig(\boldsymbol{w} \cdot \boldsymbol{x} + b) \tag{16}$$

or to put it explicitly:

$$y = \frac{1}{1 + \exp\left(- \sum\limits_{k} w_k x_k - b \right)}. \tag{17}$$

A similar activation function to the sigmoid is *tanh*, a scaled sigmoid function which is bound by $[-1, 1]$ and features much steeper derivatives.

$$tanh(x) = 2sig(2x) - 1 \tag{18}$$

Both of these function face the ”vanishing gradients" problem where the value y changes very little with change in x if x is near the part of the sigmoid or tanh-function where the derivative is almost 0. This leads to the neural network being slow to learn for such inputs (Sharma, 2017).

A *feed forward* neural network (FFNN) was the first and simplest type of ANN devised (Schmidhuber, 2014). The connections between the neurons do not form a cycle, like in a recurrent neural network (RNN) (Zell, 1994). A FFNN in its simplest kind is a *single-layer* perceptron, with only single a layer of output nodes containing the activation function. A *multi-layer* perceptron (MLP) on the other hand, features multiple layers, connected in a feed-forward way in a FFNN.

Looking at the applications of a FFNN for robotics tasks, one such network was compared with a classic EKF for the purpose of state-estimation of a 6-degrees of freedom (DOF) manipulator in (Chouraqui & Benyettou, 2009). The number of neurons in the hidden layer was determined empirically. The experiments showed that the FFNN mean-error was consistently lower than using the EKF, for both pose- and velocity estimation. A Fuzzy adaptive resonance theory neural network (FuzzyART-NN) was used by (Lameski et al., 2009) for position estimation of a mobile robot in an indoor environment. The results were only moderately promising (63% correct labeling).

The different types of more sophisticated NN architectures are now explained on numerous applications in (mobile) robotics. Each individual chapter follows the same layout: First the architecture is outlined, then applications employing this NN are given. We start of with the convolutional neural network, because it has seen large amounts of success recently for both computer-vision tasks, as well as policy optimization problems as part of *reinforcement learning*.

2.3.1 Convolutional-Neural-Networks

A convolutional neural network is a class of deep FFNN that is widely used for analyzing visual imagery by extracting certain features from the images. They use a variation of a MLP designed to require minimal preprocessing. Unlike a regular ANN, the hidden layers in a CNN consist of further convolutional layers, pooling layers, fully connected layers and normalization layers.

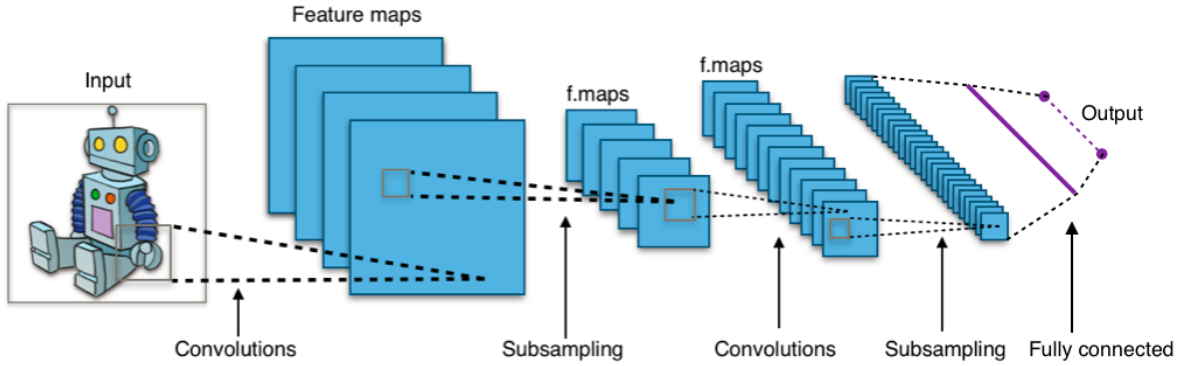

Figure 4: *CNN architecture*: The basic layout and connectivity of the layers in a convolutional neural network. (Source: edited and taken from (Aphex34, 2015))

Mathematically speaking the process in a CNN is a cross-correlation rather than a convolution. The convolutional layer's parameters consist of a set of learnable filters or kernels with a small receptive field that extend thought the full depth of the input volume. During the forward pass, for every filter the dot product between the filter and the input is computed, resulting in a stack of 2-dimensional activation maps that form the full output volume. The basic architecture of a CNN is shown in figure 4.

CNNs and their ability to process large amounts of data in a relatively short time period without significant information loss, made them a natural fit for all applications requiring robust, and fast analysis of visual imagery. This has made them the premier choice for feature extraction and categorization of visual information collected from an autonomous robot (Tai & Liu, 2016*a*). The most prevalent application being the *visual place categorization* (VPC) problem, that refers to the categorization of the semantic category of a place in the environment surrounding the mobile robot (Yang & Wu, 2012).

A CNN was used in (Sunderhauf et al., 2016) for place categorization, in numerous works such as (Khan et al., 2017) for feature extraction and further as a function approximator in *reinforcement learning* that will be discussed in detail in section 2.3.3. The work of (Sunderhauf et al., 2016) constitutes the foundation for this work; the CNN they used was the *Places205* network from the Massachusetts Institute of Technology (MIT). As already mentioned, the CNN required fine-tuning to properly generalize and further an extension of the limited set of 205 classes was desirable. We will now look at the class of *wavelet neural networks*, a simple extension of ANNs that can be easily applied to problems in a two-dimensional cartesian state space, as is common for mobile robots.

2.3.2 Wavelet-Neural-Networks

Wavelet neural networks (WNN) are very similar to classic ANNs, but the standard sigmoid activation function is instead replaced by a function drawn from a wavelet basis. A wavelet is a "small wave" that grows and decays over a finite period, contrary to the harmonic functions.

Unlike the fourier transform (FT) the wavelet transform is able to extinguish between stationary and non-stationary signals. A family of wavelets can be generated by translating and *dilating* the base wavelet function. The output of a single neuron is given by

$$\psi_{\lambda,t}(u) = \psi\Big(\frac{u-t}{\lambda}\Big) \tag{19}$$

where t is the translation parameter and λ is the dilation parameter. These are also learned by the network, whos output can be written as follows with M being the number of neurons in the hidden layer.

$$y(u) = \sum_{i=1}^{M} w_i \psi_{\lambda_i,t_i}(u) + b \tag{20}$$

For practical applications of wavelet neural networks in mobile robotics a discretization of the state space is assumed. They are then easily applied for path-planning problems in Cartesian 2D-space: Each possible state of the mobile robot in space, defined by the variables x and y for the 2 translatory DOF along the axes and θ for the rotation about its own axis along z, is one neuron in the WNN. Assigning values to the excitatory inputs of the target location and inhibitory inputs of obstacles generates a propagating wave through the neural network. The most active neurons are then the different states the mobile robot should travel along to get from start to finish.

A wavelet-NN is used in (Yang & Meng, 2001) for dynamic collision-free trajectory generation. It proposes a self-organized map (SOM) generated by the network. A SOM is a type of ANN trained via unsupervised learning to produce such a low-dimensional, discretized representation of the state space; a map. The model given by equation 20 is incorporated into their main model, a so called *shunting equation*.

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)\left([I_i]^+ + \sum_{j=1}^{k} w_{ij}[x_j]^+\right) - (D + x_i)[I_i]^- \tag{21}$$

Equation 21 characterizes the dynamics of the i-th neuron with k being the number of neighboring neurons of the i-th neuron, A, B and D being nonnegative constants representing the passive decay rate, the upper and lower bounds of the neural activity respectively, $[I_i]^+ + \sum_{j=1}^{k} w_{ij}[x_j]^+$ and $[I_i]^-$ being the excitatory and inhibitory inputs and I_i being an external input to the i-th neuron defined as

$$I_i = \begin{cases} E & \text{if there is a target} \\ -E & \text{if there is an obstacle} \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

where $E \gg B$ is a very large positive constant. Functions $[\cdot]^+$ and $[\cdot]^-$ are defined as $max\{a, 0\}$ and $max\{-a, 0\}$ respectively. The connection weight w_{ij} from the j-th to the i-th neuron is calculated using a monotonically decreasing function on the Euclidean distance

between the position of the two neurons in state space S. Parameters A, B, D and E can be used to adjust the safety of the robot, regulating by how much an obstacle should be avoided. The target location globally attracts the robot while obstacles prohibit the neuronal activity only in a neighborhood of neurons; it therefore cannot get stuck in local minima. The stability and convergence of the model was proven using the Lyapunov stability theory. With this model some simple target chasing was possible. Problematic is that the complexity rises linear with the NN-size $O(n)$ but non-linear with the number of degrees of freedom. (Lebedev et al., 2018) take a very similar approach while employing a discretized hypercube for the configuration space with dimensions according to the DOF of the objects, ie. mobile robot and obstacles.

In (Syed et al., 2014) a more sophisticated approach to wavelet-neural networks is taken. It builds on the premise of the modified pulse coupled neural network (MPCNN). In MPCNNs a neuron i fires only if some neuron j in its neighborhood N_i fires and this sets neuron i as child neuron to fire later. All neurons only fire once and the firing is governed by their internal activity $U(t)$ (Syed et al., 2014)

$$U(t) = \begin{cases} \frac{dU_i(t)}{dt} = F_i + CL_i & \text{for } t > t^{N_i^P} \\ U_i(t) = 0 & \text{for } t \leq t^{N_i^P} \end{cases} \tag{23}$$

where C is a constant, $t^{N_i^P}$ is the parent firing time and $F_i(t)$ and $L_i(t)$ are the feeding and linking fields respectively. Energy of the neuron is constantly compared with the threshold function $\theta_i(t)$.

The proposed guided autowave pulse coupled neural network (GAPCNN) differs from the MPCNN model by providing a different threshold function that also takes the distance λ from the target into account while constraining the directional propagation of the autowave by employing an angle modification in the differential equation of the internal activity $U_i(t)$ (Syed et al., 2014). This is done by modifying the feeding field of the model according to a directional constraint ψ that is calculated using the principle angle of the robot from the target. They combine this model with a vision model and show how it outperforms A* path-planning but no uncertainties of the control and measurement model are taken into account.

While wavelet neural networks are easily applied to two-dimensional problems, attention in recent research has instead shifted away from them towards *Reinforcement learning* (RL) and the idea of *Q-learning*. RL has become fairly well known for solving various control- and path-planning problems, with CNNs being successfully applied as function approximators.

2.3.3 Deep Reinforcement Learning and QLearning

With deep Q-Networks (DQN) (Mnih et al., 2015) being the first to yield stabilization for large-scale reinforcement tasks using CNNs, they have raised interest in research and applications of deep reinforcement learning (DRL) methods (Tai & Liu, 2016*a*). The basic premise of (deep) RL algorithms is the formalization of a robotics task as a *Markov Decision Process* (MDP) thats

part of a Markovian chain as outlined in section 2.1. An MDP is a 5-tuple $\langle X, U, p, R, \gamma \rangle$ with R being the set of all possible rewards and γ being a discount factor in the range of $[0, 1]$. Thus a robot takes an action u_t in state x_t, receives a reward R_{t+1} and transits to the next state x_{t+1} following the transition dynamics $p(x_{t+1} \mid x_t, u_t)$. In robotics MDPs are considered to be *episodic* and the problem of *partial observability* is overcome by either stacking N observations $x_{t-N+1}, x_{t-N+2}, \ldots, x_t$ to represent x_t or by feeding x_t into a RNN to satisfy the Markov property.

Reinforcement learning agents are designed to learn from interactions how to behave to achieve a certain goal (Sutton & Barto, 1998) or to learn how to maximize the *expected discounted return* where the *discounted return* (Tai & Liu, 2016*a*) is defined as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=t}^{T} \gamma^{k-t} R_{k+1} \tag{24}$$

Furthermore the policies π and μ are introduced where $\pi(u \mid x)$ is the stochastic policy with actions drawn from the probability distribution defined by $\pi(u \mid x)$ and $\mu(x)$ being the deterministic policy. The state-value function $V^\pi(x)$ gives the expected return when starting from state x and following the policy π:

$$V^\pi(x) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} R_{k+1} \mid x_t = x \right]. \tag{25}$$

The action-value function $Q^\pi(x, u)$ gives the expected return by taking action u from state x, then following π

$$Q^\pi(x, u) = \mathbb{E}_\pi \left[\sum_{k=t}^{T} \gamma^{k-t} R_{k+1} \mid x_t = x, u_t = u \right] \tag{26}$$

and thus defines the optimal value function $Q^*(x, u)$:

$$Q^*(x, u) = \max_\pi Q^\pi(x, u) \tag{27}$$

and finally the optimal policy $\pi^*(u \mid x)$:

$$\pi^*(u \mid x) = \operatorname*{argmax}_u Q^*(x, u). \tag{28}$$

A fairly simple reinforcement learning neural network (RLNN) was used in (Huang et al., 2005) that combined a classic three-layer back-propagation neural network (BPNN) with an action selection according to the Boltzmann distribution

$$p(u \mid x) = \frac{e^{(\frac{Q(x,u)}{T})}}{\sum_{b \in U} e^{(\frac{Q(x,b)}{T})}} \tag{29}$$

where T is a slowly decreasing parameter that determines the probability of selecting non-greedy actions. They used this RLNN to develop a training algorithm that teaches a mobile robot to avoid obstacles during prolonged exploration times.

Reinforcement learning algorithms can be categorized into *value-based* and *policy-based* approaches, as well as combinations and augmentations of the two called *actor-critic methods*. Details for each of these families of algorithms are given in appendix B.

While reinforcement learning constitutes a very recent research development, the group of algorithms called *model predictive controls* (MPC) have long been used to solve control problems. MPC has its roots in process optimization of industrial sites, refineries and other applications where the state of the process can easily be identified according to process parameters.

2.3.4 Model Predictive Controls

All of the methods discussed so far (with the exception of (Levine & Koltun, 2013)) are *model-free* meaning that the agent is not provided with the underlying transition model, which in the case of robotics is often a complex non-linear dynamics model. The advantages and disadvantages have already been discussed; the usage of MPCs shall be outlined briefly. MPC is an advanced method of process control that seeks to optimize processes while satisfying certain constraints. At a specific time-step t the current state is sampled and a cost minimizing control strategy is computed via Euler-Lagrange equations following

$$\mathcal{L}_x(\cdot) - \frac{d}{dt}\mathcal{L}_{\dot{x}}(\cdot) = 0 \tag{30}$$

until $t + T$ but only the first step of this strategy is implemented until re-sampling and repetition of the calculations. This *prediction horizon* is iteratively shifted forward making MPC a *receding horizon control*, whereas linear-quadratic regulation (LQR) only optimizes in a fixed time window and then uses this single optimal solution for the whole horizon (Wang, 2009). This means that MPC allows real-time optimization although the obtained solution is usually suboptimal since the time window is comparatively small to the whole horizon.

A fairly complete study was made by (Pretorius et al., 2014) that compares the usage of FFNN with physics-based kinematic and dynamic models for the purpose of state prediction. The results show the high computational efficiency and accuracy of ANNs for prediction of a differentially-driven mobile robot in 2D-space.

In (Finn & Levine, 2017) a deep predictive model for visual prediction previously developed in (Finn et al., 2016) was combined with concepts of MPC to teach a 6-DOF robot to push objects to a target location. The MPC algorithm generates a sequence of actions that maximizes the probability of reaching a certain Gaussian distribution

$$p_{\mathcal{M}}(I_{t+1} \mid I_{t-1:t}, x_{t-1:t}, u_t) = \mathcal{N}(\mathcal{M}(I_{t-1:t}, x_{t-1:t}, u_t) \odot I_t, \sigma^2 \boldsymbol{I}) \tag{31}$$

over target pixel locations at the end of the MPC horizon, where $\sigma^2 \boldsymbol{I}$ is the constant diagonal covariance and $\mathcal{M}$ gives the mean. While the results were fairly promising, the assumption that pixels (i.e the robot) always moves slowly limits the practical applicability of the system.

In (Howard et al., 2010) the basic A* regional motion planner was enhanced with a RHMPC that consists of a trajectory follower formulated as a control problem with a motion model that

seeks to minimize a penalty function. The experimental results showed only partial success, with just a $7,2\%$ accuracy increase over the basic A*.

3 Methods and Implementation

Departing from Bayesian filters for the purpose of state transition modeling as part of probabilistic robotics, neural network approaches for state transition modeling were to be found. Having followed this research has led to various kinds of neural networks used for different applications in (mobile) robotics (as was detailed in section 2.3), but very few for transition modeling specifically and even fewer that provided source code which could be used as foundation for this work. The work of (Sunderhauf et al., 2016) provided an interesting contribution to mobile robotics in the form of place categorization and semantic mapping, based on a successful neural network model tested on multiple benchmarks. To ensure this project would be finished on time, it was therefore decided to follow their approach and build on it.

The primary approach of this work thus builds on the premise of using a convolutional neural network for place categorization via image classification. This is somewhat different from using a Bayesian filter for state transition modeling and is closely related to SLAM-tasks: knowledge about the state of the robot is obtained via localization with known correspondences, but the robot also learns about its environment in a semantic context. Combining this knowledge into a semantic map then enables path-planning for the mobile robot on a level that supersedes simple coordinate system approaches.

While many of the discussed approaches for state transition modeling assume local linearity, simple discretization or low dimensionality of the action space, prior knowledge about the environment or a complex dynamical model, using the global information that can be gained from extracting features from a stream of camera images of the surroundings is a simple replacement.

In the following sections the CNN used for place categorization is first outlined, followed by explanations regarding the implementations of the three ML models that have been used to improve and extend the original system.

3.1 Convolutional Neural Network for Place Categorization

This section provides details regarding the architecture of the employed CNN and the used database. The CNN-database combination *Places205* was developed at the MIT for the purpose of place categorization and outperformed the popular *AlexNet* by achieving roughly 8% higher accuracy with top-1 classification on the SUN-397 benchmark (54.3% compared to 46.2%) (Sunderhauf et al., 2016). The *Places205* network was published by (Zhou et al., 2014) and it is still the state-of-the-art neural network for place categorization with the newer version

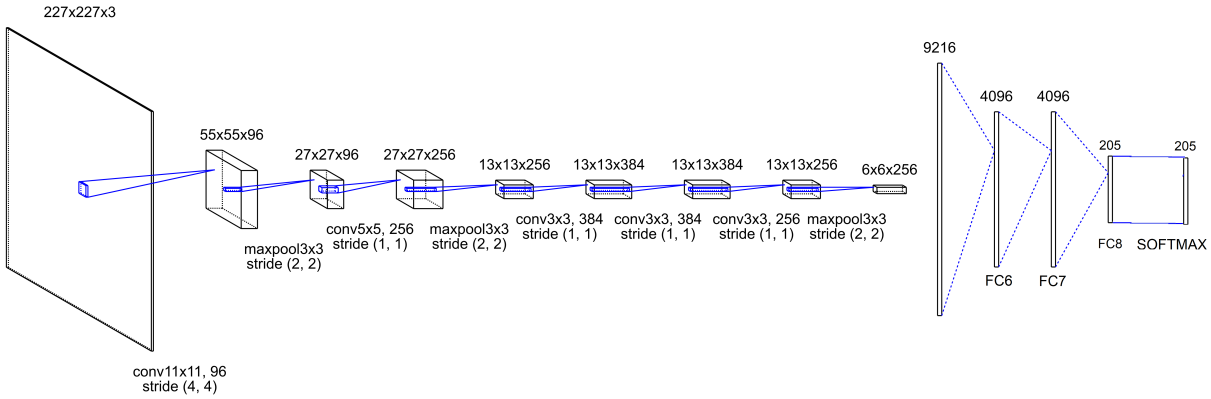

Figure 5: *Architecture of the Places205-CNN*: The ReLU layer after every convolutional layer has been omitted. The stride parameter determines the size of the steps the filter takes during convolution.

Places365 putting up less accurate results thus far. It follows the same architecture as *AlexNet*, with the difference, that the *Places205* network was specifically trained on a database of 205 different place categories for the purpose of place categorization. A sketch of the network architecture can be observed in figure 5. The implementation takes places *Caffe* (Zhou et al., 2014) and the interfacing is done in *ROS* similar to the implementation in (Sunderhauf et al., 2016) with a few key differences.

The layers of note in the CNN are the last *class-independent* layer "FC7", followed by the class-dependent layer "FC8" and lastly a softmax output layer. The output vectors are $\boldsymbol{y}_{\text{FC7}} \in \mathbb{R}^{4096\times 1}$, $\boldsymbol{y}_{\text{FC8}} \in \mathbb{R}^{205\times 1}$ and $\boldsymbol{y}_{\text{Out}} \in \mathbb{R}^{205\times 1}$ respectively. The *cross-correlation* between the individual features or classes is assumed to be low and they are further conditionally independent. In the work of (Sunderhauf et al., 2016) 11 classes were assumed to be encountered on the university premise and the values held by the 11 neurons in the output layer corresponding to those classes were extracted, renormalized and treated as likelihood-estimators. The renormalization was done dividing each individual element with the sum of the elements in the extracted vector:

$$\boldsymbol{y}'_{\text{Out},11} = \frac{y_{\text{Out},11,i}}{\sum_{j=1}^{11} y_{\text{Out},11,j}} \text{ for } i = 1,\ldots,11. \tag{32}$$

While the output of a neural network is never a probability value (NN are optimizations with non-linear, non-probabilistic functions) and instead only a value relating the *activity* of a certain neuron to a given input, *squashing* the output values between 0 and 1 in the softmax-layer allows the usage of probabilistic reasoning (Pearl, 2014). In the softmax-layer a K-dimensional vector $\boldsymbol{z}$ of arbitrary real values is squashed to a K-dimensional vector $\boldsymbol{\sigma}(\boldsymbol{z})$ of real values where each entry is in the range $(0,1)$ and all entries add up to 1:

$$\boldsymbol{\sigma} : \mathbb{R}^K \rightarrow \left\{ \boldsymbol{\sigma} \in \mathbb{R}^K \mid \sigma_i \geq 0, \sum_{i=1}^{K} \sigma_1 = 1 \right\} \tag{33}$$

$$\boldsymbol{\sigma}(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \text{ for } j = 1, \ldots, K. \tag{34}$$

Due to the extraction of the 11 values additional information useful for classification is lost. For example a campus (as one the 11 classes) is assumed to have some similarities with a driveway and perhaps a yard (not being part of the 11 classes), while an office does not.

On the other hand, if the CNN is meant to do the classification, restricting the number of possible classes a sample can belong to, assists the classification. For example if the vector $\boldsymbol{y}_{\text{Out}}$ were to carry the value 0.7 for the class "closet" but 0.1 for the class "office" and 0.05 for "kitchen", the sample would be classified as office (following top-1 classification), since a closet is assumed to not be encountered. This "probability distribution"' over the 11 classes is seen as the final result.

In this work the complete output-vector $\boldsymbol{y}_{\text{Out}} \in \mathbb{R}^{205\times 1}$ is instead continuously extracted from the CNN to create the design-matrix $\boldsymbol{m}_{205} \in \mathbb{R}^{u\times 205}$ and corresponding target-vector $\boldsymbol{t}_{205} \in \mathbb{R}^{u\times 1}$ respectively, where u is the number of analysed frames obtained from the CNN given a specific video as input. The entire data set was split with 75% of the data forming the training set and 25% making up the test set. The data-set and target-vector are then used to train the following models for the purpose of class-prediction.

The three ML models that were trained to classify the places in addition to the CNN will now be highlighted (as explained in section 2.2.1). First off is the naive Bayes filter, since it is also historically one of the first algorithms to see practice for classification purposes (Russell & Norvig, 2003). After explaining some general ideas of naive Bayes filters, the implementation in MATLAB will be detailed.

3.2 Naive Bayes Filter

Naive Bayes filters are probabilistic classifiers that are based on the Bayes' theorem or Bayes' rule outlined in appendix A.1 and assume strong (naive) independence between the features of the observations. Although this assumption is often violated in practice, naive Bayes classifiers usually yield posterior distributions that are robust to biased class density estimates, especially if the posterior is 0.5 (the decision boundary) (Friedman et al., 2001). Since each feature x_i is assumed to be conditionally independent of every other feature x_j for $j \neq i$, given the category C_k

$$p(x_i \mid x_{i+1}, \ldots, x_n, C_k) = p(x_i \mid C_k) \tag{35}$$

with n being the number of features and K being the number of classes, the joint model can be expressed as

$$p(C_k \mid x_1, \ldots, x_n) = p(C_k) \prod_{i=1}^{n} p(x_i \mid C_k), \tag{36}$$

with the extended form of the Bayes' rule, as is used by MATLAB (The MathWorks, 2018*a*), given by:

$$\widehat{p}(C_k \mid x_1, \ldots, x_n) = \frac{\pi(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)}{\sum_{k=1}^{K} \pi(C_k) \prod_{i=1}^{n} p(x_i \mid C_k)}, \text{ for every } k = 1, \ldots, K \tag{37}$$

where $\pi(C_k)$ is the prior probability that a class index is k. The naive Bayes classifier then combines this model with a decision rule. A common rule is to pick the most probable hypothesis; this is known as the *maximum a posteriori* (MAP) decision rule. The classifier then is the function that assigns label $\widehat{y} = k$ for some k according to:

$$\widehat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}} \, \widehat{p}(C_k \mid x_1, \ldots, x_n). \tag{38}$$

The MATLAB function *fitcnb* (The MathWorks, 2018*a*) was used to create the trained multiclass naive Bayes model, and the option *OptimizeHyperParameters* was again used to optimize the hyperparameters. The property *Distribution* is used to specify the estimation used for modeling the probability density function (PDF) of the predictor variable distributions. The following parameters yielded the best results:

1. Distribution: Kernel (see (Bishop, 2016), pg.122)
2. Kernel function type: normal
3. Bandwith: 0.019527

A kernel distribution is a nonparametric representation of the PDF. The kernel density estimator is the estimated PDF of a predictor variable and for any real values of x, the estimator follows (The MathWorks, 2018*b*)

$$\widehat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{39}$$

where $x_1, x_2, \ldots, x_n$ are random samples from an unknown distribution, n is the sample size, $K(\cdot)$ is the kernel smoothing function, and h is the bandwith. The kernel smoothing function, option "normal", uses the Gaussian distribution:

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{(-0.5x^2)} \tag{40}$$

The steps of *fitcnb* are as follows:

1. Estimate the densities of each feature variable of the predictors for each class following equations 39 and 40. This gives a matrix of kernels $\boldsymbol{m}_{kernels} \in \mathbb{R}^{K \times 205}$ with each kernel estimating the density distribution of a specific feature vector for that class.
2. Model the posteriors according to equation 37.
3. Classify an observation according to equation 38.

The second ML model called logistic regression, how it is used for classification and its expansion to the multiclass case will now be hightlighted.

3.3 Multinomial Logistic Regression

Multinomial (or multiclass) logistic regression is an extension of the logistic regression for multiclass problems. It was first developed by David Cox (Cox, 1958) and it is used to estimate the parameters of a logistic model. The log-odds of the probability of an event to occur is a linear combination of independent variables x. For binary classification the categorically distributed dependent variable y can be either 0 or 1. In multiclass logit the dependent variable can take one of multiple discrete outcomes. If the model is parameterized by θ and the (x, y) pairs are drawn uniformly from the underlying distribution then the model can be described with the following equation:

$$\begin{gathered}\lim_{N\to+\infty} N^{-1}\sum_{i=1}^{N}\log p(y_i \mid x_i;\theta) = \\ \sum_{x\in X}\sum_{y\in Y} p(X=x,Y=y)\left(-\log\frac{p(Y=y\mid X=x)}{p(Y=y\mid X=x;\theta)}+\log p(Y=y\mid X=x)\right) = \\ -D_{KL}(Y\|Y_\theta)-H(Y\mid X).\end{gathered} \tag{41}$$

$H(Y \mid X)$ is the conditional entropy and D_{KL} the Kullback-Leibler (KL) divergence. By maximizing the log-likelihood of a model, the KL divergence from the maximal entropy distribution is minimized, thus searching for the model that makes the least number of assumptions in its parameters.

The multinomial logit model was created using the Pattern Recognition and Machine Learning Toolbox from the MATLAB file exchange (Chen, 2018), a package implementing some of the algorithms described in the book (Bishop, 2016). The function *logitMn.m* was used to create and train the model and *logitMnPred.m* was used to make predictions using the test set. LogitMn takes the training set following the design matrix, and the correspoding target vector as outlined in section 3.1. The regularization paramter λ, being the only adjustable parameter, was left at its default value, although it gets updated as the model learns:

1. λ: $1 \cdot 10^{-4}$ (default)

One of the most important steps is the update of the parameter vector $\boldsymbol{w}$ following the *Newton-Raphson* iterative optimization scheme, which uses a local quadratic approximation to the log likelihood function. For minimizing $E(\boldsymbol{w})$ the update takes the form (Fletcher, 2013)

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \mathsf{H}^{-1}\nabla E(\boldsymbol{w}) \tag{42}$$

where H is the Hessian matrix whose elements comprise the second derivatives of $E(\boldsymbol{w})$ with respect to the components of $\boldsymbol{w}$ (Bishop, 2016). The function outputs a trained model structure. This structure is then used in the function logitMnPred together with a matrix comprising the test set for predictions. The output is a vector with predicted labels for every observation from the test set as well as the predict probability for each class. This output vector is of the form $y_{\text{pred}} \in \mathbb{R}^{u\times 1}$ with every entry $\{y_i \mid i \in [0, u]\}$ being a whole number in the range of $[1, k]$ where

k is the number of classes. To calculate the accuracy of the model given the test data, a test target vector $y_{\text{pred-target}}$ was created and compared to y_{pred} according to the following pseudo-code equation:

$$\text{acc-logit} = 100 - 100 \cdot \left(\frac{length(nonzeros(abs(y_{\text{pred},i} - y_{\text{pred-target},i})))}{s} \right) \text{ for every } i = 0, \ldots, s \tag{43}$$

where s is the size of the test set. The input matrix and input vector as well as the output vector holding the predicted class labels take the same shape using the SVM and the naive Bayes filter and the accuracy calculation always follows equation 43 (with additional typecasting).

The third and last of the developed models is the support vector machine. Support vector machines are a very popular method for solving classification problems, that can easily be extended to the multi-class case.

3.4 Support Vector Machines

SVMs try to classify data be separating the data points with a line, or a hyperplane in higher-dimensional space. They can also perform non-linear classification using the kernel trick. Since the underlying methods of support vector machines were already explained in section 2.2.1, some specifics regarding the implementation of the SVM as part of the MATLAB class *ClassificationECOC* (ECOC = error-correcting output code) (Allwein et al., 2001) are now highlighted. The function *templateSVM* was used to create a learner template suitable for ECOC multiclass models; the function has multiple configurable options:

BoxConstraint and KernelScale

BoxConstraint and *KernelScale* (The MathWorks, 2018*c*) are the MATLAB names for the hyperparameters C and γ respectively, since the trained SVM is a soft-margin C-SVM. The C parameter is a regularisation parameter that controls the maximum penalty imposed on margin-violating observations which helps to prevent overfitting. It controls how strict the hyperplane should divide the sets of data points belonging to separate classes and how many examples are accepted to be on the wrong side of this *soft margin*. The parameter γ is only relevant for Gaussian kernel functions; it is the free parameter of the Gaussian radial basis function (RBF):

$$K(x_j, x_k) = \exp(-\gamma \|x_j - x_j\|^2), \gamma > 0. \tag{44}$$

Coding and SaveSupportVectors

The *Coding* property decides the coding scheme, either One versus One or One versus All (The MathWorks, 2018*c*). The OVA strategy trains a single classifier per class (K classifiers in total for K distinct classes), and all samples from that class are positive, while all other samples from all other classes are combined into the ”rest" and treated as negative. This means, that even if the class distribution is balanced in the training set, the binary learners see unbalanced

distributions because the set of negatives is much larger than the set of positives. This problem increases with the number of distinct classes. The OVO scheme instead trains $\frac{K(K-1)}{2}$ binary learners, with one class being positive, one class negative and ignoring the rest. This does not cause a class imbalance problem, since the samples from each individual class are roughly equal. The *SaveSupportVectors* property is lastly used to save the trained support vectors, since the number of vectors carries information regarding the achieved generalization.
Using the *OptimizeParametersOption* has yielded the following optimal (feasible) parameters:

1. Coding: onevsall
2. BoxConstraint: 75.786
3. KernelScale: 0.7278
4. KernelFunction: RBF

4 Results and Discussion

The system was evaluated on 5 of the possible 12 observable places on university campus: the digital factory resembling an industrial setting as the *Places205* class *assembly_line*, a big lecture theater as *auditorium*, a *corridor*, a *kitchen* and an *office*. To show the capabilities of the system to extend the limited set of 205 classes, data of an additional place was gathered not known to the CNN: a door. The videos were captured using the ROS wrapper for Intel®Real Sense™ devices and a BlasterX Senz3D camera, carried around by hand. After storing the videos in a bagfile, playing the bagfile and analysing the stream with the CNN, the training- and test-set and respective target-vectors were created as described in section 3.1. Before training began in MATLAB, the distributions in the data sets were plotted using a Python script (see appendix 3) and can be observed in figure 7 and figure 8 respectively, while figure 6 gives the legend for the two figures. Ideally only 5 peaks would be observed, while the spread for the *door* class was unknown. A few observations, following the analysis of the training- and test-set distributions from figure 7 and 8, can be made:

1. The sets were drawn from the design matrix fairly evenly, with similar distributions.
2. A total of 8 significat peaks, and a few outliers in other classes, can be observed: 5 from the expected classes and 3 further peaks in *closet*, *kitchenette* and *staircase*.
3. The peak in kitchenette is due to the the strong similarities of the kitchen and kitchenette data set in *Places205* and is therefore not problematic.
4. A lot of samples from the lecture theater recording, especially in the training set, share strong similarities with a staircase. This would be a problem for the CNN classification if it was not constrained to the 5 classes, but constraining it helps the robustness in such cases. Inclusion of the class staircase would lead to large misclassifications from the CNN.
5. The peak in closet actually features samples from both the office and the door recordings. This is not problematic for the CNN classification, since the CNN does not know of the class door. This does cause some misclassifications with the trained ML-models though, as will be discussed shortly.
6. The samples from the recording in the digital factory are fairly *noisy*, in the sense that the distribution is spread very evenly.
7. The peak in corridor is very robust, with very few samples showing features of another class.

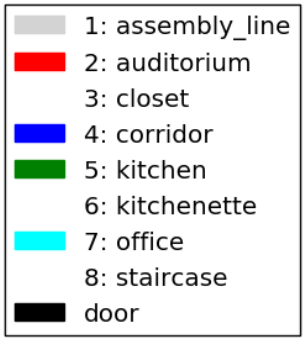

Figure 6: *Legend*: The legend featuring the different label names, colour schemes and peak locations for figures 7 and 8.

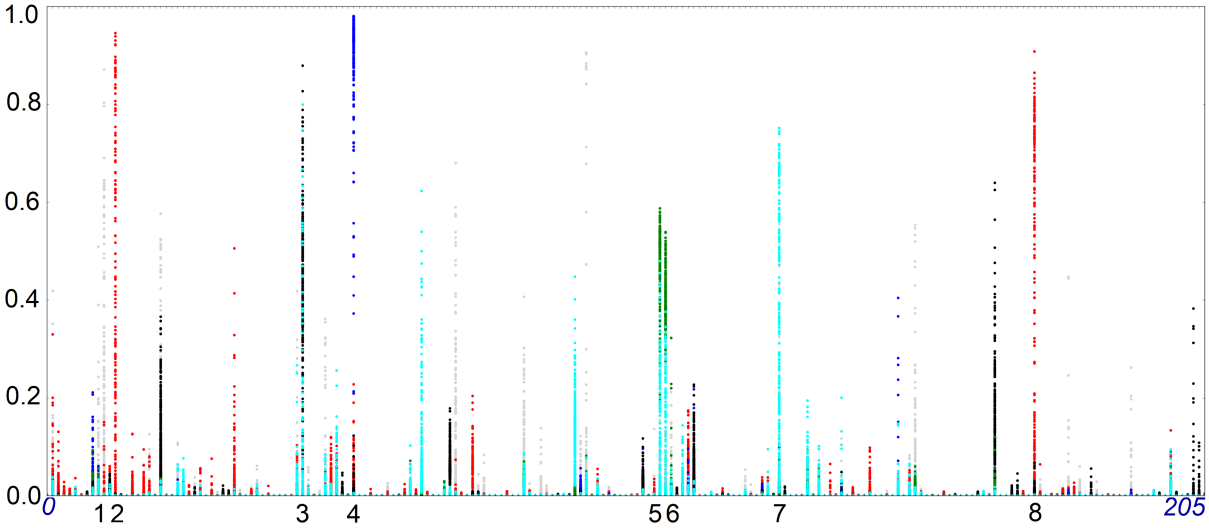

Figure 7: *Training-set spread*: Visualization of the vectors forming the training-set.

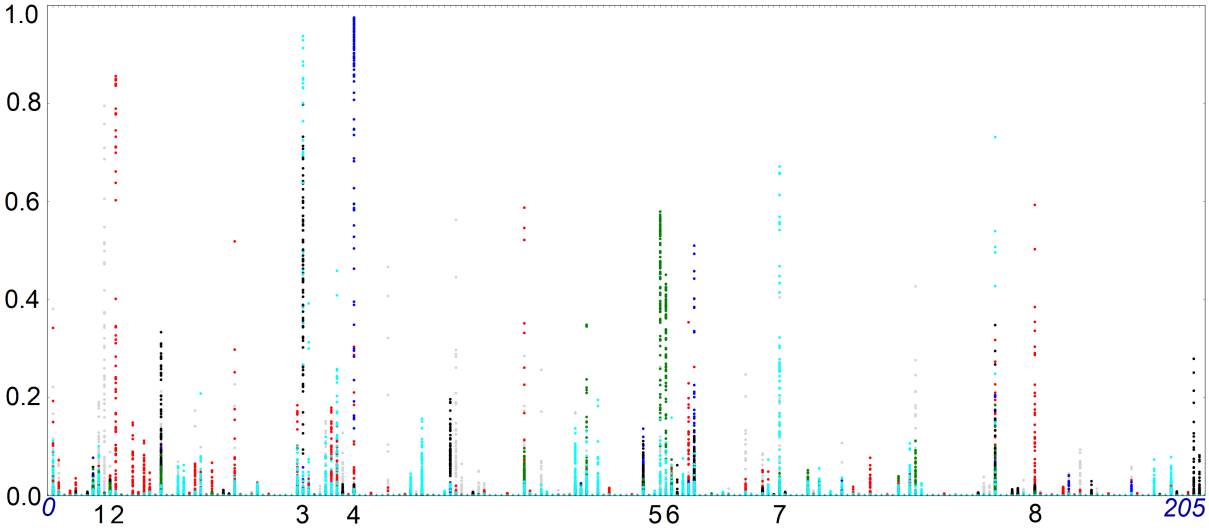

Figure 8: *Test-set spread*: Visualization of the vectors forming the test-set

Table 1: *Accuracies for the different classification models used*: The bottom row gives the average accuracy. Since the number of recorded frames was nearly equal, no weighting had to be done.

Classification results				
Environment	**CNN-5**	**MLR**	**SVM**	**NBF**
digital factory	54%	99%	97%	92%
lecture theater	62%	88%	89%	81%
corridor	100%	93%	93%	95%
door	–	100%	100%	100%
kitchen	100%	89%	87%	83%
office	75%	63%	73%	79%
total average	**78.2%**	**88.7%**	**90.2%**	**88.27%**

In figures 7 and 8 the Y-axis gives the squashed value held by every single neuron in the softmax layer over the X-axis showing the 205 different classes. The numbers 1 through 8 along the X-axis further specify the locations of the peaks and the colours of the points correlate to the place the video was captured in.

As already mentioned in section 3.1, restricting the number of observable classes helps the classification with just the CNN. Nevertheless, the CNN classification is less accurate as all developed learning models, as can be seen in table 1, showing the final evaluation results. It shows the accuracy of the classifications models, that is top-1 accuracy for the CNN, and the accuracy calculations following equation 43 for the ML-models. It should be noted, that the accuracy of the CNN cannot so easily be compared to the accuracies of the ML-models for two main reasons: If an image carries known features correlating to multiple of the 205 classes known to the CNN, it is understandable that the network struggles with classification, while the ML-models are trained with these ”none-correspondences". On the other hand, if an image shows outliers, meaning features of a class not part of the limited five-class set, the CNN is not influenced, while this causes issues with the ML-models. Also, since the door carried similarities to a closet and the office did as well, the models misclassify while the CNN is again not influenced.

The CNN yields robust classification for places with features that are easily distinguishable from the other of the 5 classes. As soon as the spread becomes more leveled or multiple peaks arise, the CNN struggles with classification even in the restricted set of 5 classes, only posting a 54% accuracy in the environment of the digital factory. It manages to post a total classification accuracy of just 78.2%, with the classification relying strongly on samples only featuring one class. All three of the ML-models yield an average classification accuracy of 88 to 90% giving strong results in the digital factory and only really struggling to categorize the office.

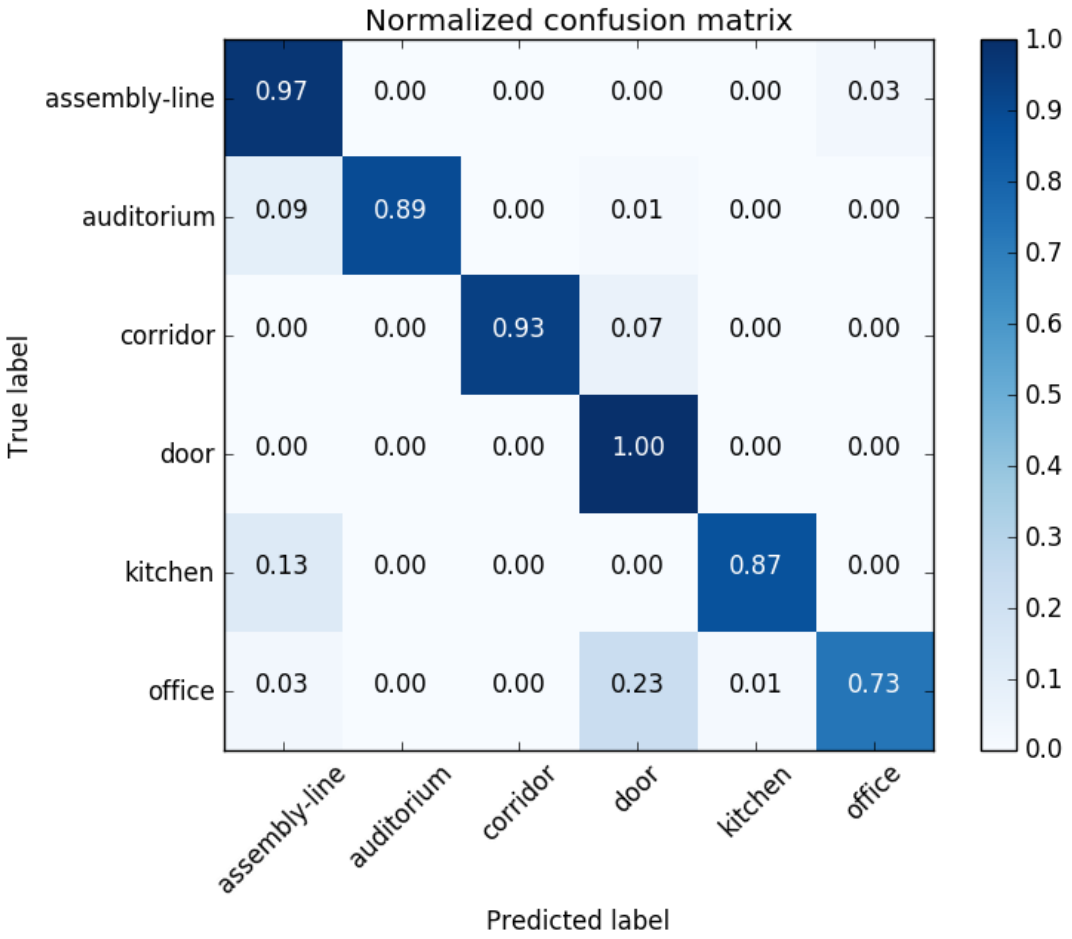

Figure 9: *SVM confusion matrix*: Normalized confusion matrix following the SVM-model

Looking at the detailed results provided in appendix C, or the normalized confusion matrix of the SVM in figure 9, this comes from the fact that a fair number of samples from the office environment were classified as a door, resulting from the similarities of both classes to the closet class. During parts of the recording in the office, the camera was pointed directly at a cabinet and much less at the typical computer desk and chair setup. This largely flat and grey cabinet shared similarities to the later recorded grey door, that didnt have any significant features. Appendix C also shows results of the classification amongst only three classes: auditorium, corridor and kitchen, with the ML-models averaging an accuracy of 98 to 99%, once again showing the problematic impact of the similarities between the door and office data. The confusion matrix in figure 9 shows the correct and false classifications of the most accurate model, the support vector machine. The confusion matrices of the MLR and the NBF can be seen in appendix C.2.

Lastly, figure 10 shows an image captured in the digital factory of the university that was to be classified as ”assembly line". It can be seen that the ML-models confidently predict the correct label, while the CNN struggles since the image shows features of multiple places. This image is part of a video that was created to visualize the predictions using the captured videos

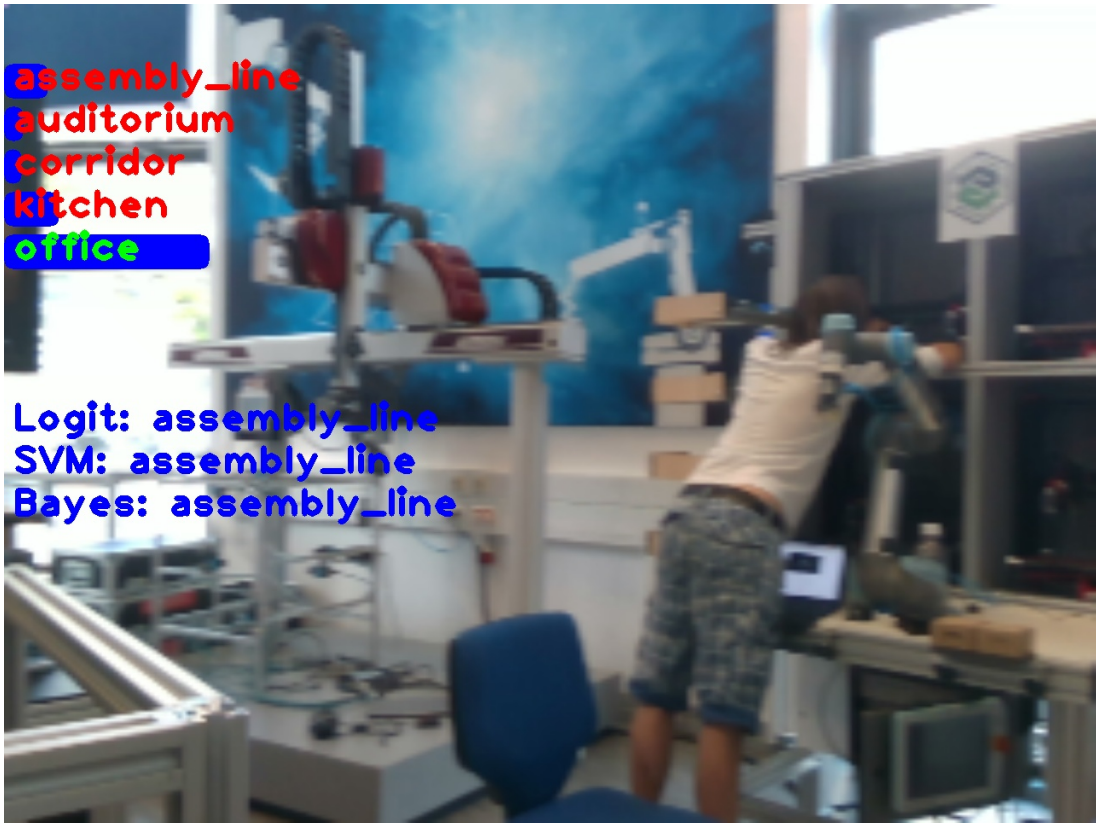

Figure 10: *Frame from the digital factory*: An image captured in the digital factory with the likelihood spread given by the CNN at the top, and the predicted labels of the three ML models at the bottom.

themselves. The predictions sometimes *jump* between various classes, a problem that can be solved by taking the predicted labels as *sensor readings* and applying a Bayes filter, to limit these abrupt place changes, that are physically impossible (temporal coherence).

With the system having correctly labeled 100% of the samples from the door class, it is shown that the classification can easily be extended to environments not part of the 205 places. This is also true for environments without strong peaks in the distribution, such as the digital factory, that the CNN struggles with. It should also be noted, that since the number of distinct classes is low and the data sets are of equal size, the problem of class imbalance due to the OVA scheme is relatively small.

5 Conclusion

This thesis introduced a straight-forward addition to an accurate and inexpensive system used for place categorization. An extensive research identifying the applications of neural networks used for robotics applications, led to finding the system proposed by (Sunderhauf et al., 2016). Having sucessfully used the *Places205* convolutional neural network as the foundation for feature extraction, multiple machine learning models were trained for the purpose of image classification. These include a multinomial logistic regression, a support vector machine and a naive Bayes filter.

The performance of these models was evaluated and compared to the classification using only the neural network. All three models significantly outperformed the CNN, posing an accuracy of roughly 90% in the extended, six-class environment and around 98 to 99% for the three-class scenario, in contrast to the 78% from the CNN. The accuracy drop using the models between the two different data sets can be attributed to suboptimal video recordings.

The system makes no strong hardware or software demands, only using a simple webcam and open source software. The semantic information obtained is an important enabler of more advanced robotics tasks, especially human-robot collaboration, since humans describe places, rooms and goals not with coordinates, but place labels. To this end, the system was successfully tested in an industrial and office-esque environment, proving its usability for manufacturers and the commercial service industry alike.

In future works, the system will be improved and extended upon as follows:

1. The classification will be extended to include all 12 of the on-campus places.
2. The system will be further tested using a mobile robot, gathering both camera and laser scanner data.
3. Using the data from the laser scanner, and the place categorization from the machine learning models, a semantic map can be created.
4. Path-planning decisions can then be made using this semantic map.

Bibliography

Allwein, E. L., Schapire, R. E. & Singer, Y., 2001. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1, pp.113–141. Available at: <https://doi.org/10.1162/15324430152733133> [Accessed 17.05.2018].

Aphex34, 2015. *typical CNN architecture*. [] Available at: <https://commons.wikimedia.org/wiki/File:Typical_cnn.png> [Accessed 17.05.2018].

Bayes, M. & Price, M., 1763. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs. *Philosophical Transactions (1683-1775)*, pp.370–418.

Bellman, R., 2013. *Dynamic programming*: Courier Corporation.

Bishop, M. C., 2016. *PATTERN RECOGNITION AND MACHINE LEARNING.*: Springer-Verlag New York.

Borenstein, J., Everett, H., Feng, L. et al., 1996. Where am i? sensors and methods for mobile robot positioning. *University of Michigan*, 119(120), p.15.

Brega, R., Tomatis, N. & Arras, K. O., 2000. The need for autonomy and real-time in mobile robotics: a case study of xo/2 and pygmalion. In: *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*. 31. October - 05. November, 2000, Takamatsu, Japan: IEEE, pp.1422–1427.

Chen, M., 2018. *Patten Recognition and Machine Learning Toolbox*. [] Available at: <https://de.mathworks.com/matlabcentral/fileexchange/55826-pattern-recognition-and-machine-learning-toolbox> [Accessed 11.06.2018].

Chouraqui, S. & Benyettou, M., 2009. State estimation for mobile robot using neural networks. *Journal of Applied Sciences*, 9(22), pp.3957–3965.

Cox, D. R., 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pp.215–242.

Crammer, K. & Singer, Y., 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of machine learning research*, 2(Dec), pp.265–292.

Davies, J., 2016. *Machine Learning on ARM-powered client devices*. [] Available at: <https://www.youtube.com/watch?v=k4ovpelG9vs> [Accessed 03.05.2018].

Deisenroth, M. P., Neumann, G., Peters, J. et al., 2013. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2), pp.1–142.

Finn, C. & Levine, S., 2017. Deep visual foresight for planning robot motion. In: *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. 29. May - 03. June, 2017, Singapore, Singapore: IEEE, pp.2786–2793.

Finn, C., Goodfellow, I. & Levine, S., 2016. Unsupervised learning for physical interaction through video prediction. In: *Advances in neural information processing systems*. 05. - 10. December, 2016, Barcelona, Spain: NIPS, pp.64–72.

Fletcher, R., 2013. *Practical methods of optimization*: John Wiley & Sons.

Friedman, J., Hastie, T. & Tibshirani, R., 2001. *The elements of statistical learning*. Vol. 1: Springer series in statistics New York.

Fu, M. C., Glover, F. W. & April, J., 2005. Simulation optimization: a review, new developments, and applications. In: *Proceedings of the 37th conference on Winter simulation*. 04. - 07. December, 2005, Orlando, Florida: Winter Simulation Conference, pp.83–95.

Galindo, C., Fernández-Madrigal, J.-A., González, J. & Saffiotti, A., 2008. Robot task planning using semantic maps. *Robot. Auton. Syst.*, 56(11), pp.955–966. Available at: <http://dx.doi.org/10.1016/j.robot.2008.08.007> [Accessed 17.05.2018].

Ganek, A. G. & Corbi, T. A., 2003. The dawning of the autonomic computing era. *IBM systems Journal*, 42(1), pp.5–18.

Goodfellow, I., Bengio, Y. & Courville, A., 2016. *Deep Learning*: MIT Press. http://www.deeplearningbook.org.

Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S. & Lew, M. S., 2016. Deep learning for visual understanding: A review. *Neurocomputing*, 187, pp.27–48.

Hemachandra, S., Walter, M. R., Tellex, S. & Teller, S., 2014. Learning spatial-semantic representations from natural language descriptions and scene classifications. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. 10. - 14. June, 2014, Melbourne, Australia: IEEE, pp.2623–2630.

Howard, T. M., Green, C. J. & Kelly, A., 2010. Receding horizon model-predictive control for mobile robot navigation of intricate paths. In: *Field and Service Robotics*. July, 2010, Cambridge, Massachusetts: Springer, pp.69–78.

Huang, B.-Q., Cao, G.-Y. & Guo, M., 2005. Reinforcement learning neural network to the problem of autonomous mobile robot obstacle avoidance. In: *Machine Learning and Cybernetics, 2005. Proceedings of 2005 International Conference on*. 18. - 21. August, 2005, Guangzhou, China: IEEE, pp.85–89.

Khan, A., Zhang, C., Atanasov, N., Karydis, K., Kumar, V. & Lee, D. D., 2017. Memory augmented control networks. *arXiv preprint arXiv:1709.05706*, .

Kohavi, R., 1998. Glossary of terms. *Machine Learning*, 30, pp.271–274.

Lameski, P., Kulakov, A. & Davcev, D., 2009. Learning and position estimation of a mobile robot in an indoor environment using fuzzyart neural network. In: *Advanced Intelligent Mechatronics, 2009. AIM 2009. IEEE/ASME International Conference on*. 14. - 17. July, 2009, Singapore, Singapore: IEEE, pp.770–774.

Laplace, P. S., 1820. *Théorie analytique des probabilités*: Courcier.

Lebedev, D., Steil, J. & Ritter, H., 2018. A new wave neural network dynamics for planning safe paths of autonomous objects in a dynamically changing world. , .

Lei, T. & Ming, L., 2016. A robot exploration strategy based on q-learning network. In: *Real-time Computing and Robotics (RCAR), IEEE International Conference on*. 6. - 10. June, 2016, Angkor Wat, Cambodia: IEEE, pp.57–62.

Levine, S. & Abbeel, P., 2014. Learning neural network policies with guided policy search under unknown dynamics. In: *Advances in Neural Information Processing Systems*. 08. - 13. December, 2014, Montreal, Canada: NIPS, pp.1071–1079.

Levine, S. & Koltun, V., 2013. Guided policy search. In: *International Conference on Machine Learning*. 16. - 21. June, 2013, Atlanta, Georgia: JLMR, pp.1–9.

Levine, S., Finn, C., Darrell, T. & Abbeel, P., 2016. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1), pp.1334–1373.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. & Wierstra, D., 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, .

Markov, A., 1954. Theory of algorithms [translated by jacques j. schorr-kon and pst staff] imprint moscow, academy of sciences of the ussr, 1954 [jerusalem, israel program for scientific translations, 1961; available from office of technical services, united states department of commerce] added tp in russian translation of works of the mathematical institute, academy of sciences of the ussr, v. 42. *Original title: Teoriya algorifmov.[QA248. M2943 Dartmouth College library. US Dept. of Commerce, Office of Technical Services, number OTS 60-51085]*, .

Medina, G., 2013. *Diagram of an artificial neural network*. [] Available at: <https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network> [Accessed 11.04.2018].

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G. et al., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540), p.529.

Moore, G. E., 2006. Cramming more components onto integrated circuits, reprinted from electronics, volume 38, number 8, april 19, 1965, pp. 114 ff.. *IEEE solid-state circuits society newsletter*, 20(3), pp.33–35.

Pearl, J., 2014. *Probabilistic reasoning in intelligent systems: networks of plausible inference*: Elsevier.

Pretorius, C. J., du Plessis, M. C. & Gonsalves, J. W., 2014. A comparison of neural networks and physics models as motion simulators for simple robotic evolution. In: *Evolutionary Computation (CEC), 2014 IEEE Congress on*. 06. - 11. July, 2014, Beijing, China: IEEE, pp.2793–2800.

Pronobis, A. & Jensfelt, P., 2012. Large-scale semantic mapping and reasoning with heterogeneous modalities. In: *Proceedings of the 2012 IEEE International Conference on Robotics and Automation (ICRA)*. 10. - 11. September, 2012, Saint Paul, MN, USA: ICRA.

Pronobis, A., Mozos, O. M., Caputo, B. & Jensfelt, P., 2010. Multi-modal semantic place classification. *The International Journal of Robotics Research*, 29(2-3), pp.298–320. Available at: <https://doi.org/10.1177/0278364909356483> [Accessed 17.05.2018].

Ranganathan, A., 2010. Pliss: Detecting and labeling places using online change-point detection. In: *in: Proceedings of Robotics: Science and Systems*. 27. - 30. June, 2010, Zaragoza, Spain: MIT Press, Cambridge Massachusetts.

Russell, S. J. & Norvig, P., 2003. *Artificial Intelligence: A Modern Approach*. 2 edition: Pearson Education.

Samuel, A. L., 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), pp.210–229.

Schmidhuber, J., 2014. Deep learning in neural networks: An overview. *CoRR*, abs/1404.7828. Available at: <http://arxiv.org/abs/1404.7828> [Accessed 17.05.2018].

Schulman, J., Moritz, P., Levine, S., Jordan, M. & Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, .

Sharma, A., 2017. *Understanding Activation Functions in Neural Networks*. [] Available at: <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0> [Accessed 12.04.2018].

Siegwart, R., Nourbakhsh, I. R. & Scaramuzza, D., 2011. *Introduction to autonomous mobile robots*: MIT press.

Smith, S. W. et al., 1997. The scientist and engineer's guide to digital signal processing. , .

Sunderhauf, N., Dayoub, F., McMahon, S., Talbot, B., Schulz, R., Corke, P., Wyeth, G., Upcroft, B. & Milford, M., 2016. Place categorization and semantic mapping on a mobile robot. In: *IEEE International Conference on Robotics and Automation (ICRA 2016)*. 16. - 21. May, 2016, Stockholm, Sweden: IEEE.

Sutton, R. S. & Barto, A. G., 1998. *Reinforcement learning: An introduction*. Vol. 1: MIT press Cambridge.

Syed, U. A., Kunwar, F. & Iqbal, M., 2014. Guided autowave pulse coupled neural network (gapcnn) based real time path planning and an obstacle avoidance scheme for mobile robots. *Robotics and autonomous systems*, 62(4), pp.474–486.

Szita, I. & Lörincz, A., 2006. Learning tetris using the noisy cross-entropy method. *Neural computation*, 18(12), pp.2936–2941.

Tai, L. & Liu, M., 2016*a*. Deep-learning in mobile robotics - from perception to control systems: A survey on why and why not. *CoRR*, abs/1612.07139. Available at: <http://arxiv.org/abs/1612.07139> [Accessed 17.05.2018].

Tai, L. & Liu, M., 2016*b*. Towards cognitive exploration through deep reinforcement learning for mobile robots. *arXiv preprint arXiv:1610.01733*, .

The MathWorks, I., 2018*a*. *MATLAB Statistics and Machine Learning Toolbox*. [] Available at: <https://de.mathworks.com/help/stats/fitcnb.html> [Accessed 11.06.2018].

The MathWorks, I., 2018*b*. *MATLAB Statistics and Machine Learning Toolbox*. [] Available at: <https://de.mathworks.com/help/stats/kernel-distribution.html> [Accessed 11.06.2018].

The MathWorks, I., 2018*c*. *MATLAB Statistics and Machine Learning Toolbox*. [] Available at: <https://de.mathworks.com/help/stats/fitcsvm.html> [Accessed 11.06.2018].

Theodoridis, S., Koutroumbas, K. et al., 2008. Pattern recognition. *IEEE Transactions on Neural Networks*, 19(2), p.376.

Thrun, S., Burgard, W. & Fox, D., 2005. *Probabilistic robotics*.

Wang, L., 2009. *Model predictive control system design and implementation using MATLAB®*: Springer Science & Business Media.

Williams, R. J., 1992. *Simple statistical gradient-following algorithms for connectionist reinforcement learning*: Springer.

Wu, J. & Rehg, J. M., 2011. Centrist: A visual descriptor for scene categorization. *IEEE transactions on pattern analysis and machine intelligence*, 33(8), pp.1489–1501.

Xiao, J., Ehinger, K. A., Hays, J., Torralba, A. & Oliva, A., 2016. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, 119(1), pp.3–22.

Yang, H. & Wu, J., 2012. Object templates for visual place categorization. In: *Asian Conference on Computer Vision*. 05. - 09. November, 2012, Daejeon, Korea: Springer, pp.470–483.

Yang, S. X. & Meng, M., 2001. Neural network approaches to dynamic collision-free trajectory generation. *Trans. Sys. Man Cyber. Part B*, 31(3), pp.302–318. Available at: <http://dx.doi.org/10.1109/3477.931512> [Accessed 17.05.2018].

Zell, A., 1994. *Simulation neuronaler netze*. Vol. 1: Addison-Wesley Bonn.

Zhang, F., Leitner, J., Milford, M., Upcroft, B. & Corke, P., 2015. Towards vision-based deep reinforcement learning for robotic motion control. *arXiv preprint arXiv:1511.03791*, .

Zhou, B., Lapedriza, A., Xiao, J., Torralba, A. & Oliva, A., 2014. Learning deep features for scene recognition using places database. In: *Advances in neural information processing systems*. 08. - 13. December, 2014, Montreal, Canada: NIPS, pp.487–495.

List of Figures

List of Tables

List of Code

List of Abbreviations

AI	artificial intelligence
ANN	artificial neural network
BPNN	back propagation neural network
CNN	convolutional neural network
DAG	directed acyclic graph
DBN	dynamic bayes network
DDP	differential dynamic programming
DDPG	deep deterministic policy gradient
DL	deep learning
DOF	degrees of freedom
DPG	deterministic policy gradient
DQN	deep Q-network
DRL	deep reinforcement learning
ECOC	error-correcting output code
EKF	extended kalman filter
FFNN	feed forward neural network
FT	fourier transform
FuzzyART-NN	fuzzy adaptive resonance theory neural network
GAPCNN	guided autowave pulse coupled neural network
GPS	guided policy search
GPU	graphics processing unit
KF	kalman filter

KL	kullback-leibler
LQR	linear-quadratic regulation
MAP	maximum a posteriori
MDP	markov decision process
MIT	massachusetts institue of technology
ML	machine learning
MCL	monte carlo localization
MLR	multinomial logistic regression
MPC	model predictive control
MPCNN	modified pulse coupled neural network
NAF	normalized advantage function
NBF	naive bayes filter
NN	neural network
OVA	one versus all
OVO	one versus one
RBF	radial basis function
RL	reinforcement learning
RLNN	reinforcement learning neural network
RNN	recurrent neural network
SLAM	simultaneous localization and mapping
SPG	stochastic policy gradient
SVM	support vector machine
WNN	wavelet neural network

A — Probabilistic Robotics

A.1 Probability Theory

To express the uncertainty of a robot in a meaningful way and incorporate it into the decision-making process, the mathematical concept of probability theory is proposed by (Thrun et al., 2005). In their own words they say that "... instead of relying on a single "best guess" as to what might be the case, probabilistic algorithms represent information by probability distributions over a whole space of guesses. By doing so, they can represent ambiguity and degree of belief in a mathematically sound way" (Thrun et al., 2005).

All quantities in probabilistic robotics such as sensor measurements, controls and odometry data are modeled as random variables. Probabilistic laws are infered from this data to help model the behavior of the system and make qualified guesses about the state of the robot. The first key concept of probability theory is that of a *joint distribution* and it is given by

$$p(x, y) = p(X = x \wedge Y = y). \tag{45}$$

This describes the probability of the event, that the random variable X takes on the value x and that Y takes on y. If the random variables X and Y are *independent* from one another, equation 45 takes on the form (Thrun et al., 2005)

$$p(x, y) = p(x)p(y) \tag{46}$$

meaning the product of the individual probability densities gives the joint distribution. If the value of one of the variables is known and the probability of the other variable is to be conditioned on that fact then the *conditional* probability distribution can express this in the form of (Thrun et al., 2005)

$$p(x \mid y) = \frac{p(x, y)}{p(y)} \tag{47}$$

or simply

$$p(x \mid y) = \frac{p(x)p(y)}{p(y)} = p(x) \tag{48}$$

for the independent case showing that knowledge of Y is irrelevant if X is to be calculated. In probabilistic robotics (and probability theory in general) *Bayes rule* plays a very important role. It was first formulated by Thomas Bayes in (Bayes & Price, 1763) and then further developed and published in (Laplace, 1820) by Pierre-Simon Laplace which relates the conditional $p(x \mid y)$ to its "inverse" $p(y \mid x)$:

$$p(x \mid y) = \frac{p(y \mid x)p(x)}{p(y)}. \tag{49}$$

A.2 Markov Chains

Another important concept is that of *state completeness* or *Markov chains* named after and outlined by Andrey Markov in (Markov, 1954). The state describes the characteristics of the environment and the robot itself and contains things such as the pose of the robot, joint velocities, location and features of surrounding static or moving objects such as landmarks or humans (Thrun et al., 2005). A state is called complete if it entails all necessary information to predict the future state and no further knowledge of past states, measurements or controls carry any additional information required. This is called the *Markov property*. In other words, all past states $(x_{0:t-1})$ and future states $(x_{t+1:\infty})$ are conditionally independent given the present state x_t.

$$p(x_{t+1:\infty} \mid \cancel{x_{0:t-1}}, x_t) \Rightarrow p(x_{t+1:\infty} \mid x_t) \tag{50}$$

This does not rule out the stochastic evolution of states but instead requires the state x_t to contain all this information about the stochastic variables. The *Markov assumption* is then used to describe a model where the Markov property is assumed to hold. A temporal, discrete-time series of states under this assumption is thus called a Markov chain.

B — Deep Reinforcement Learning

B.1 Value-based Methods

These methods are based on estimating the values of being in a given state and then formulating the control policies given the estimated values (Tai & Liu, 2016*a*) and are based on the recursive *Bellman equations* (Bellman, 2013) with the *Bellman Expectation Equation* given by

$$Q^{\pi}(x,u) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma Q^{\pi}(x_{t+1}, u_{t+1} \mid x_t = x, u_t = u)\right] \tag{51}$$

and the *Bellman Optimality Equation* given by

$$Q^{\pi}(x,u) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma \max_{a'} Q^{\pi}(x_{t+1}, u' \mid x_t = x, u_t = u)\right] \tag{52}$$

The two most popular value-based RL methods (*SARSA* and *Q-learning*) then follow the same recursive backup procedure that starts at the target value y_t and updates the Q-values by a step size α towards it and is given as follows:

$$Q^{\pi}(x_t, u_t) \leftarrow Q^{\pi}(x_t, u_t) + \alpha\delta_t, \tag{53}$$

$$\delta_t = y_t - Q^{\pi}(x_t, u_t). \tag{54}$$

δ_t is termed the *td-error* (temporal difference error) and y_t the *td-target*. The difference between *SARSA* and *Q-learning* lies then in their *td-targets*:

$$y_t^{\textit{SARSA}} = R_{t+1} + \gamma Q^{\pi}(x_{t+1}, u_{t+1}), \tag{55}$$

$$y_t^{\textit{Q-learning}} = R_{t+1} + \gamma \max_{u'} Q^{\pi}(x_{t+1}, u'). \tag{56}$$

SARSA is therefore an *on-policy* method since it updates its Q-value estimates by taking the transitions generated by following the stochastic behavioural policy π into account. *Q-learning* on the other hand is *off-policy*, since it updates its estimations towards a target optimal policy (Tai & Liu, 2016*a*).

A deep Q-Network was proposed and successfully used by (Mnih et al., 2015) to control Atari games using the raw pixel images from the emulator as inputs. The DQN approximates the optimal Q-value function with a deep CNN, whose weights shall be denoted as $\theta^Q : Q(x,u;\theta^Q) \approx Q^*(x,u)$ which modulates the *td-error* and *td-target* into:

$$\delta_t^{DQN} = y_t^{DQN} - Q(x_t, u_t; \theta_t^Q), \tag{57}$$

$$y_t^{DQN} = R_{t+1} + \gamma \max_{u'} Q(x_{t+1}, u'; \theta_t^-). \tag{58}$$

Then an update step is performed using the gradient calculation following the partial derivative with a learning rate of α:

$$\theta_{t+1} \leftarrow \theta_t - \alpha \cdot \left(\frac{\partial (\delta_t^{DQN}(\theta_t^Q))^2}{\partial \theta_t^Q} \right). \tag{59}$$

The two main techniques proposed in the DQN to stabilize learning are *target-network*, with *td-target* being computed using outputs from a *target-network* θ^- which shares the same architecture but only periodically updates the weights by copying θ^Q, and *experience replay* meaning inputs are first stored into a *replay memory* and then random consecutive samples are drawn from it. A Q-learning network was also used by (Lei & Ming, 2016) to the task of robot exploration, teaching a mobile robot to avoid obstacles in an unknown environment. They used a supervised learning model implemented as a CNN to obtain a feature map via depth data. The Q-network takes this feature-map and then follows a value-based approach. They later proposed an alternative approach in (Tai & Liu, 2016*b*) with no preprocessing of the images. A very similar approach was used by (Zhang et al., 2015) for vision-based end-to-end learning for the manipulation of a robotic arm.

B.2 Policy-based Methods

Policy-based methods operate directly on the policies without maintaining value estimations and search for parameters to maximize the policy objective function via either gradient-free (Fu et al., 2005; Szita & Lörincz, 2006) or gradient-based paradigms, with gradient descent methods remaining the most popular choice (Tai & Liu, 2016*a*). Given a policy $\pi_\theta(\cdot)$, policy optimization searches for the best parameters θ that maximizes an objective function $J(\pi_\theta)$

$$J(\pi_\theta) = \mathbb{E}_{\pi_\theta} \left[f_{\pi_\theta}(\cdot) \right] \tag{60}$$

with $f_{\pi_\theta}(\cdot)$ being a *score function* to judge the goodness of a policy with multiple valid choices for the *score function* outlined in (Schulman et al., 2015). The *policy gradient* is then defined as

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta \cdot f_{\pi_\theta}(\cdot) \right] \tag{61}$$

where $\nabla_\theta \log \pi_\theta$ points out the direction in the parameters space to follow that leads to an increase of the probability of good actions being sampled following the policy π_θ. As pointed out by (Deisenroth et al., 2013) directly following the policy gradient might conflict with hard constraints or safety requirements in robotics settings; the search in undesired state space regions should therefore be explicitly discouraged. The stochastic policy gradient (SPG)

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{x,u} \left[\nabla_\theta \log \pi_\theta(u \mid x) \cdot Q^\pi(x, u) \right] \tag{62}$$

is further sample-inefficient in high-dimensional action spaces whereas the deterministic policy gradient (DPG)

$$\nabla_\theta J(\mu_\theta) = \mathbb{E}_x \left[\nabla_\theta \mu_\theta(x) \cdot Q^\mu(x, \mu_\theta(x)) \right] \tag{63}$$

only needs to integrate over the state space.

A fairly sample efficient model-based policy-search algorithm was proposed in (Levine & Koltun, 2013) employing techniques of *Differential Dynamic Programming* (DDP) to relax the constraints regarding the new-sample availability along each individual gradient step. A variant of DDP called *linear-quadratic regulation* (LQR) was used to approximate the Gaussian I-projection over the guiding distribution of the possible action-state pairs following the stochastic policy

$$\pi_G(\boldsymbol{u}_t \mid \boldsymbol{x}_t) = G(\boldsymbol{u}_t; g(\boldsymbol{x}_t), -Q^{-1}_{\boldsymbol{uu}t}). \tag{64}$$

Approximations of the linear dynamics and quadratic rewards under the assumption that the Jacobian and Hessian - matrices exist lead to the optimal policy being a linear Gaussian with the mean function and covariance given by the Q-function. For the case of nonlinear dynamics, $\pi_G(\boldsymbol{u}_t \mid \boldsymbol{x}_t)$ approximates a Gaussian around the nominal trajectory with the feedback usually keeping the samples close to this trajectory (Levine & Koltun, 2013).

While model-free approaches need well parameterized policies, model-based approaches are generally more accurate but require complex non-linear dynamic models that are difficult to design and aren't always computationally tractable. They therefore proposed a hybrid in (Levine & Abbeel, 2014) by using a Gaussian mixture model (GMM) that fits a Gaussian on the taken sample to restrict the trajectory distribution to obtain a piece-wise linear-Gaussian dynamics model. They then combined this with the guided policy search (GPS) from their previous work and transformed it into a GPS Lagrangian with a dual variable to enforce constraint satisfaction and optimize regarding the weights and probability distributions:

$$\mathcal{L}_{\text{traj}}(p(\tau), \eta) = \left[\sum_t E_{p(\boldsymbol{x}_t, \boldsymbol{u}_t)} \Big[l(\boldsymbol{x}_t, \boldsymbol{u}_t) - \eta \log \widehat{p}(\boldsymbol{x}_t, \boldsymbol{u}_t) \Big] \right] - \eta \mathcal{H}(p(\tau)) - \eta\epsilon \tag{65}$$

where E is the expected cost of l and $\mathcal{H}$ gives the differential entropy. Since this alternating optimization leads to high entropy they penalized the entropy directly.

Finally, they applied this work for the task of *visual-servoing* with End-to-End learning in (Levine et al., 2016) where the trajectory-centric RL algorithm generates guiding distributions to supervise the policy-learning algorithm that builds on CNN to train the policy.

B.3 Actor-critic Methods

These algorithms maintain an explicit representation of both the policy (the *actor*) and the value estimates (the *critic*) by replacing the return G_t and the *baseline* $b(x)$, that helps reducing the variance of the estimation, of the standard score function (Williams, 1992)

$$f_{\pi_\theta}(\cdot) = G_t - b_t(x_t) \tag{66}$$

with the unbiased estimate $Q^{\pi_\theta}(x_t, u_t)$ and the baseline function $V^{\pi_\theta}(x_t)$ respectively to form the following *score function*:

$$f_{\pi_\theta}(\cdot) = Q^{\pi_\theta}(x_t, u_t) - V^{\pi_\theta}(x_t) \tag{67}$$

which is also called *advantage function* to estimate the advantage of taking a particular action u in state x (Tai & Liu, 2016*a*):

$$A(x, u) = Q(x, u) - V(x). \tag{68}$$

Other variants of DQN that use *actor-critic* methods or other augmentations of standard *Q*-learning to enable learning in continuous action spaces include the deep deterministic policy gradient (DDPG) proposed in (Lillicrap et al., 2015) or the normalized advantage function (NAF) from (Guo et al., 2016).

C — Detailed Classification Results

C.1 Visualization of the predictions

This chapter includes six images created by the MATLAB function *plot* visualizing the results from table 1 for every trained model, as well as previously achieved results

Figure 11 shows the multinomial logistic regression accuracy results. For every environment, roughly 75 samples made up the test set. MLR in particular struggled with the similarities between the office and the door.

The results from the SVM-classification can be observed in figure 12. The number of support vectors trained for the classes are $36 - 27 - 13 - 19 - 15 - 43$ following the alphabetical order of the classes with the inclusion of door; with the classes assembly-line and office requiring 36 and 43 support vectors respectively. This means that the generalization was difficult for those classes, requiring a higher number of support vectors, to construct the margin.

Figure 13 lastly shows the predictions made by the NBF. While it struggled a bit with both the office and auditorium environments, optimizing the hyperparameters helped boost its performance by quite a bit.

It should also be noted, that the classification was originally done on only three classes: the lecture theater as auditorium, a corridor and a kitchen. With out the strong cross-correlation between the office and door recordings, and their similarity to the closet class, the ML-models gave excellent results, of classifying around 98 to 99% correctly. The predictions for that data set are visualized in figure 14, 15 and 16.

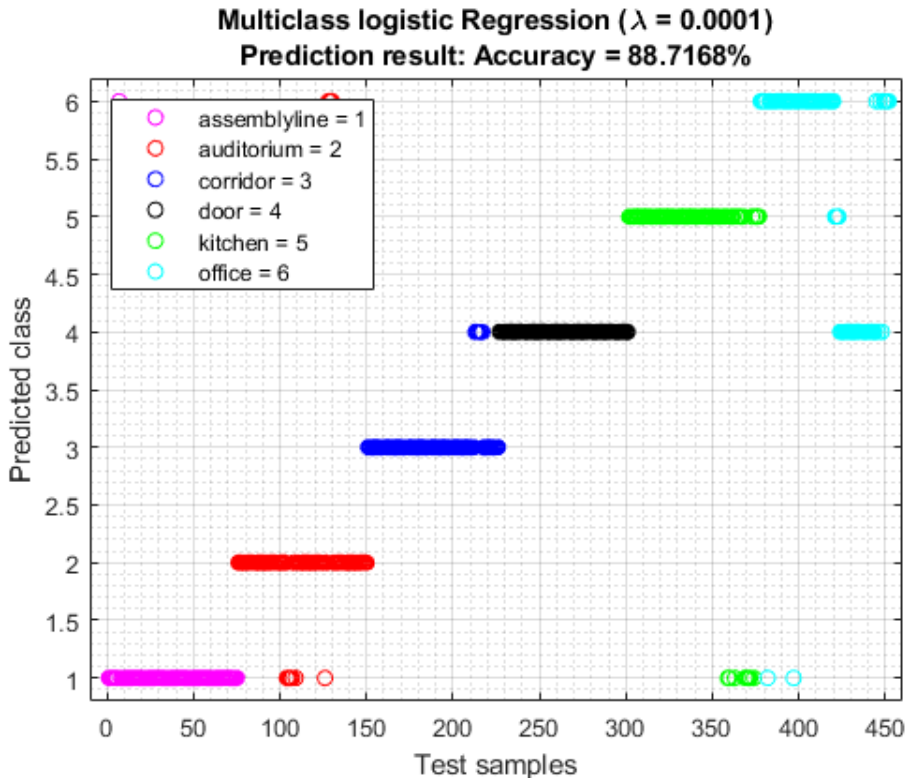

Figure 11: *Multinomial logistic regression results*: Every prediction made by the MLR.

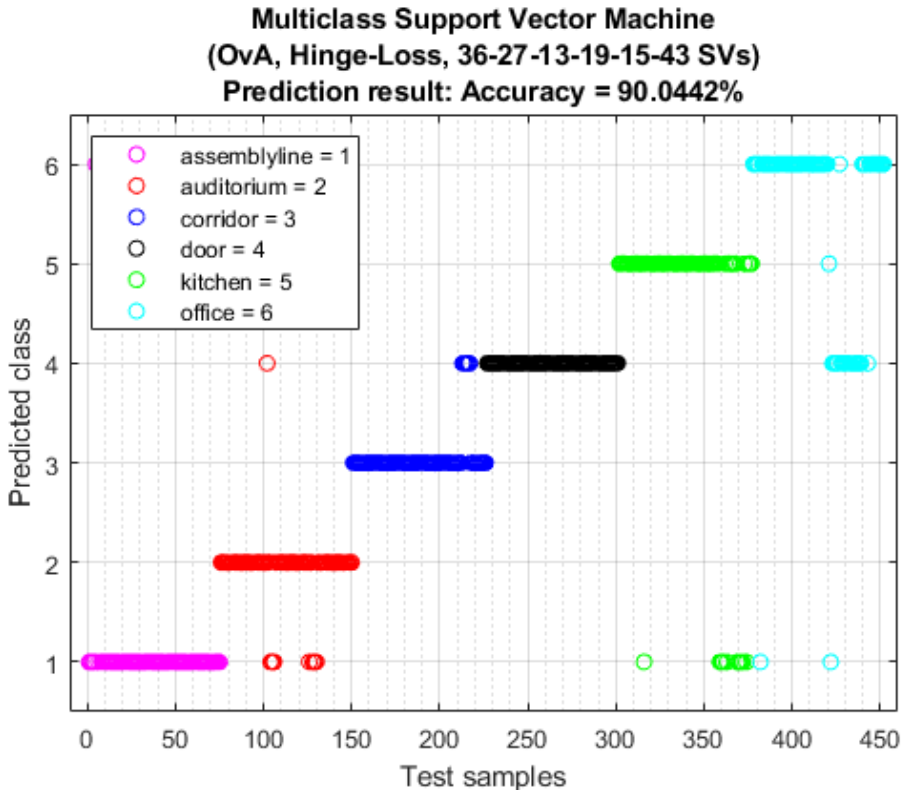

Figure 12: *Support vector machine results*: Predictions made by the support vector machine.

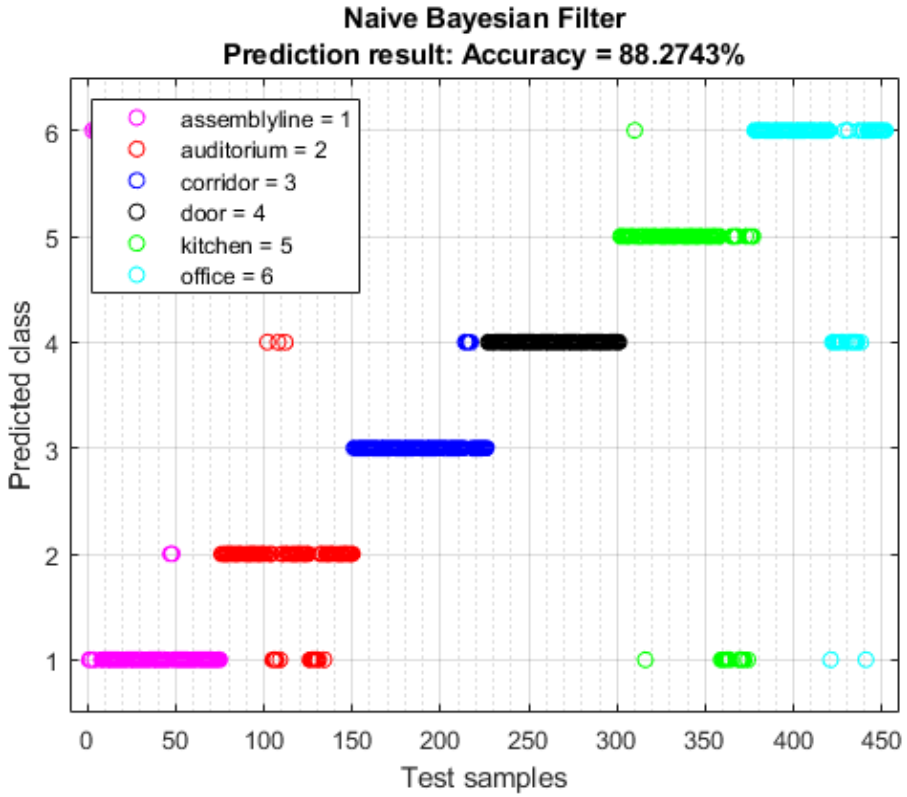

Figure 13: *Naive bayes filter results*: Predictions made by the naive bayes filter.

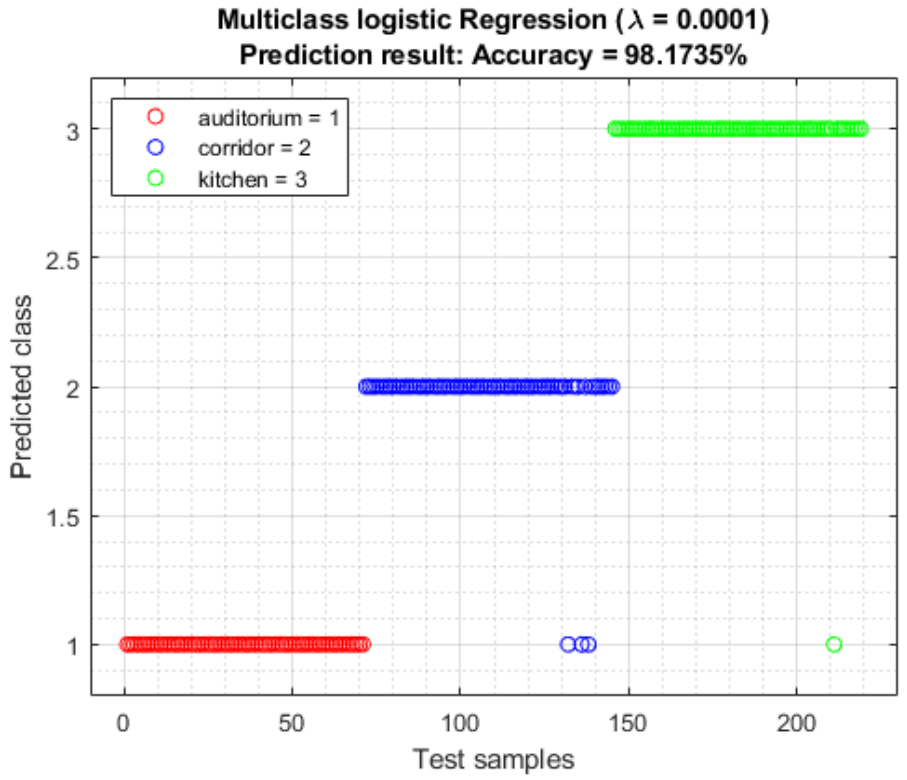

Figure 14: *Multinomial logistic regression results*: Every prediction made by the MLR for the 3-class data set.

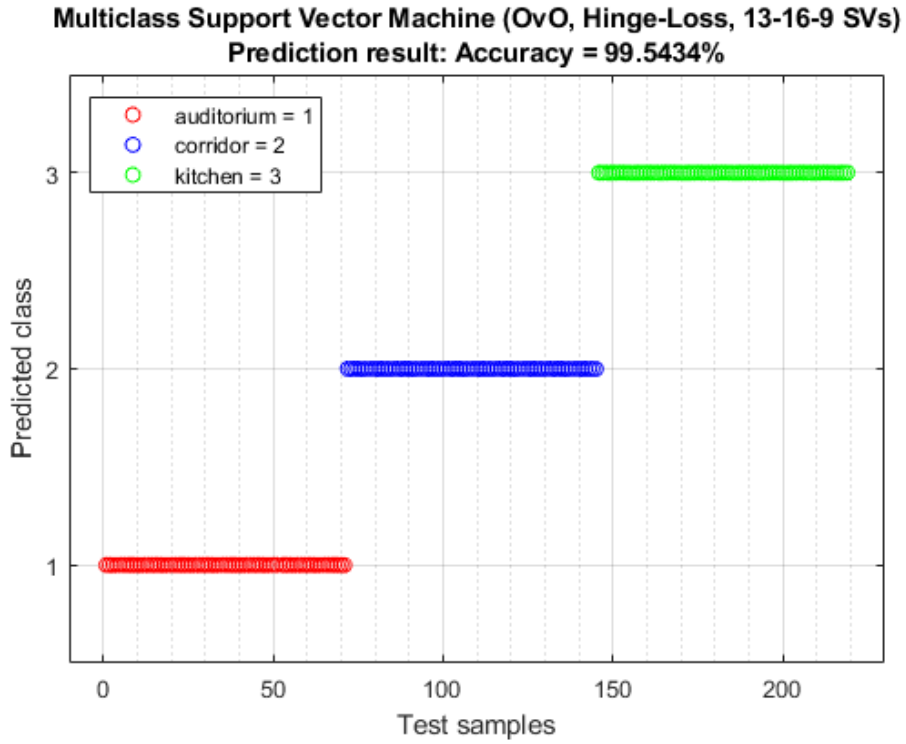

Figure 15: *Support vector machine results*: Predictions made by the support vector machine amongst the limited set.

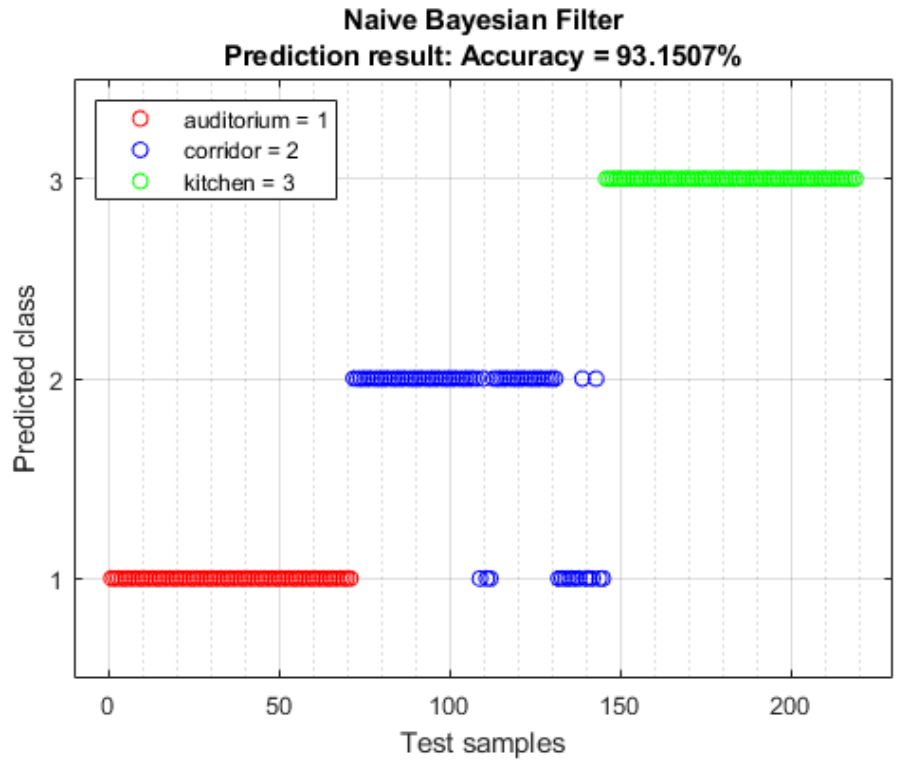

Figure 16: *Naive bayes filter results*: Visualization of the predictions made by the naive bayes lerner for the 3-class set.

C.2 Confusion Matrices

This section provides the two confusion matrices in figure 9 and 18 for the MLR- and the NBF-model, respectively.

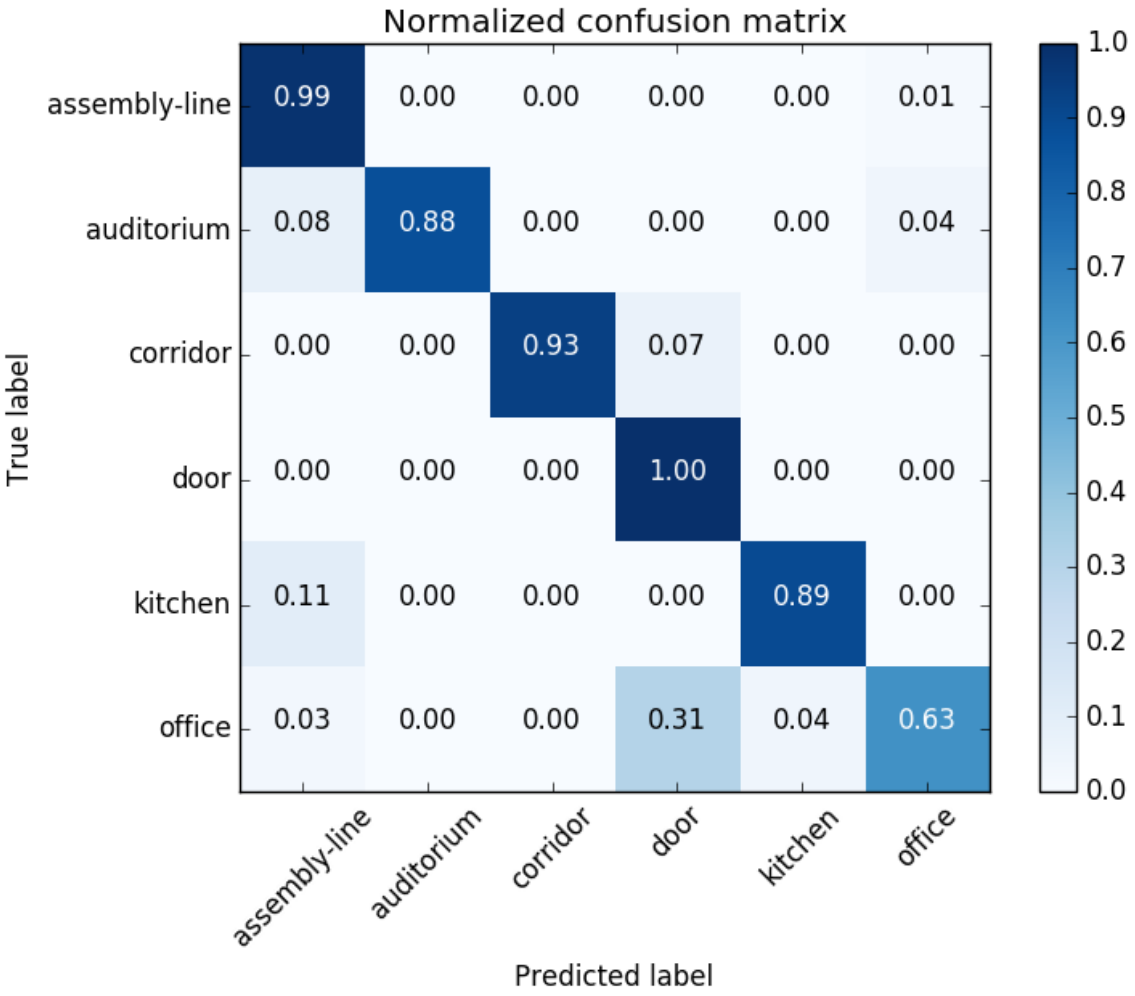

Figure 17: *Logit confusion matrix*: Normalized confusion matrix following the MLR-model

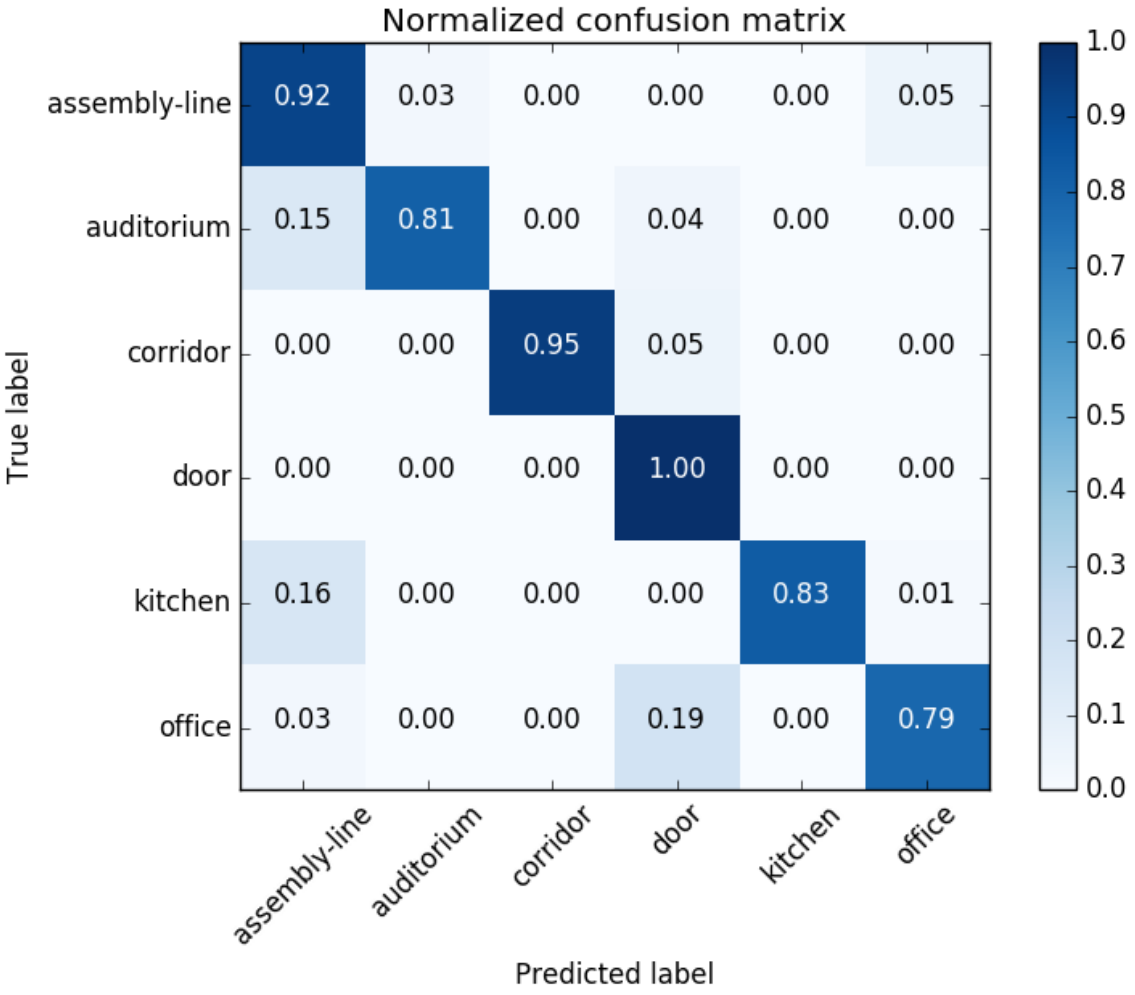

Figure 18: *Naive Bayes confusion matrix*: Normalized confusion matrix following the NBF-model

D — Source Code

D.1 MATLAB source code for the three ML-models

```
clc;
close all;
clear all;
training_set = (dlmread('trainingset.csv', ',', 0, 0)).';
training_target_vector = (dlmread('training_vector.csv', ',', 0, 0)).';
test_set = (dlmread('testset.csv', ',', 0, 0)).';
test_target_vector = (dlmread('test_vector.csv', ',', 0, 0)).';

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  MULTICLASS LOGIT-REG   %
%%%%%%%%%%%%%%%%%%%%%%%%%%%

[model_logit, llh_logit] = logitMn(training_set, training_target_vector, 0.0001);
y_logit = logitMnPred(model_logit, test_set);
figure
plot( 1:75, y_logit(1:75), 'mo',...
        76:150, y_logit(76:150), 'ro', ...
        151:226, y_logit(151:226), 'bo', ...
        227:301, y_logit(227:301), 'ko', ...
        302:377, y_logit(302:377), 'go', ...
        378:452, y_logit(378:452), 'co');
grid on
grid minor
accuracy_logit = 100 - (100*(length(nonzeros(abs(y_logit -
    test_target_vector))))/length(test_set));
title(['Multiclass logistic Regression (\lambda = 0.0001)', newline, 'Prediction
    result: Accuracy = ', num2str(accuracy_logit),'%']);
xlabel('Test samples')
xlim([-10, 460])
ylabel(['Predicted class'])
ylim([0.8, 6.2])
legend('assemblyline = 1', 'auditorium = 2', 'corridor = 3', 'door = 4',
    'kitchen = 5', 'office = 6', 'Location', 'Northwest')

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%     MULTICLASS SVM      %
%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```
%mdl = fitcecoc(training_set.', training_target_vector,
    'OptimizeHyperparameters', 'all', 'HyperparameterOptimizationOptions',
    struct('AcquisitionFunctionName', 'expected-improvement-plus'));
svm_template = templateSVM('Standardize', 0, 'SaveSupportVectors',true,
    'BoxConstraint', 75.786, 'KernelScale', 0.7278, 'KernelFunction', 'gaussian');
training_target_vector = categorical(training_target_vector.');
classOrder = unique(training_target_vector);
rng(1);
responseName = 'Detected Class';
classNames = {num2str(1), num2str(2), num2str(3), num2str(4), num2str(5),
    num2str(6)};
SVM_model = fitcecoc(training_set.', training_target_vector, 'Learners',
    svm_template, 'ClassNames', classOrder, 'Coding', 'onevsall');
y_svm = (predict(SVM_model, test_set.').');
isLoss_SVM = resubLoss(SVM_model);
figure
plot( 1:75, y_svm(1:75), 'mo',...
        76:150, y_svm(76:150), 'ro', ...
        151:226, y_svm(151:226), 'bo', ...
        227:301, y_svm(227:301), 'ko', ...
        302:377, y_svm(302:377), 'go', ...
        378:452, y_svm(378:452), 'co');
grid on
grid minor
accuracy_svm = 100 - (100*(length(nonzeros(abs(double(y_svm) -
    test_target_vector))))/length(test_set));
title(['Multiclass Support Vector Machine', newline, '(OvA, Hinge-Loss,
    36-27-13-19-15-43 SVs)', newline, 'Prediction result: Accuracy = ',
    num2str(accuracy_svm),'%']);
xlabel('Test samples')
xlim([-10, 460])
ylabel(['Predicted class'])
legend('assemblyline = 1', 'auditorium = 2', 'corridor = 3', 'door = 4',
    'kitchen = 5', 'office = 6', 'Location', 'Northwest')


% %%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % NAIVE BAYESIAN FILTER   %
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bayesian_model = fitcnb(training_set.', training_target_vector,
    'OptimizeHyperparameters', 'all', 'HyperparameterOptimizationOptions',
    struct('AcquisitionFunctionName', 'expected-improvement-plus'));
Bayesian_model = fitcnb(training_set.', training_target_vector, 'ClassNames',
    classOrder, 'Distribution', 'kernel', 'Width', 0.019527);
y_bayesian = (predict(Bayesian_model, test_set.').');
isLoss_Bayesian = resubLoss(Bayesian_model);
figure
plot( 1:75, y_bayesian(1:75), 'mo',...
        76:150, y_bayesian(76:150), 'ro', ...
```

```
74          151:226, y_bayesian(151:226), 'bo', ...
75          227:301, y_bayesian(227:301), 'ko', ...
76          302:377, y_bayesian(302:377), 'go', ...
77          378:452, y_bayesian(378:452), 'co');
78  grid on
79  grid minor
80  accuracy_bayesian = 100 - (100*(length(nonzeros(abs(double(y_bayesian) -
        test_target_vector))))/length(test_set));
81  title(['Naive Bayesian Filter', newline, 'Prediction result: Accuracy = ',
        num2str(accuracy_bayesian),'%']);
82  xlabel('Test samples')
83  xlim([-10, 460])
84  ylabel(['Predicted class'])
85  legend('assemblyline = 1', 'auditorium = 2', 'corridor = 3', 'door = 4',
        'kitchen = 5', 'office = 6', 'Location', 'Northwest')
```

Code 1: MATLAB code used for the development of the ML-models. Lines 36 and 67 are used for optimizing the hyperparameters.

D.2 ROS node used for interfacing with caffe

```
#!/usr/bin/env python
'''
Copyright (C) 2016, by
Feras Dayoub (feras.dayoub@gmail.com)

This is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This software package is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Leser General Public License.
If not, see <http://www.gnu.org/licenses/>.
'''
import roslib; roslib.load_manifest('semantic_label_publisher')
import numpy as np
import rospy
import csv
import sys
import cv2
import cPickle
import gzip
import caffe
from sklearn.externals import joblib
from sklearn import svm
import os.path
import time
from read_cat import cats
from sensor_msgs.msg import Image , LaserScan

from cv_bridge import CvBridge, CvBridgeError

from semantic_label_publisher.msg import SemLabel
class SemanticLabel(object):
  def __init__(self,lname,lid,lcolor):
    self.label_name  = lname
    self.label_id    = int(lid)
    self.label_color = [int(lcolor[0]),int(lcolor[1]),int(lcolor[2])]

class SemLabelPub():
  def __init__(self,caffe_root,MODEL_FILE,PRETRAINED,MEAN_FILE,
                        SVM_PICKLE_FILE,SUB_CAT_FILE):
    self.pub       = rospy.Publisher('semantic_label',SemLabel)
```

```
48      self.image_pub = rospy.Publisher("sem_label_image",Image)
49
50      self.image_sub = rospy.Subscriber('/camera/color/image_raw', Image,
            self.image_callback)
51
52      self.bridge = CvBridge()
53
54      self.total_count = 0;
55      self.correct = 0;
56
57      self.net         = caffe.Classifier(MODEL_FILE, PRETRAINED,caffe.TEST)
58      self.transformer = caffe.io.Transformer({'data': (1,3,227,227)})
59
60      self.transformer.set_transpose('data', (2,0,1))
61      self.net.blobs['data'].reshape(1,3,227,227)
62
63      db_mean = np.load(MEAN_FILE)
64      self.transformer.set_mean('data', db_mean.mean(1).mean(1))
65
66      caffe.set_mode_gpu()
67
68      self.msg = SemLabel()
69
70      with open(SUB_CAT_FILE, 'r') as f:
71        txt_data = f.readlines()
72
73      with open(
74        '/home/andreas/catkin_ws/src/ros-semantic-mapper
75        /setup/categoryIndex_places205.csv', 'r' ) as f2:
76        txt_data2 = f2.readlines( )
77
78      self.labels = list()
79      for l in txt_data:
80        if l[0] != '#':
81          label_name    = l.split(" ")[0]
82          label_id      = l.split(" ")[1]
83          label_color   = l.split(" ")[2].split(",")
84          self.labels.append(SemanticLabel(label_name,label_id,label_color))
85
86      self.labels2 = list()
87      for l in txt_data2:
88        label_name_tmp = l.split( "/" )[2]
89        label_name     = label_name_tmp.split( " " )[0]
90        label_id       = l.split( " " )[1]
91        label_color    = [0, 0, 0]
92        self.labels2.append(SemanticLabel(label_name,label_id,label_color))
93
94      open( '/home/andreas/catkin_ws/src/ros-semantic-mapper/
95        results/distribution_205_office.csv', 'ab' ).writelines(
96        ' '.join(str(j.label_name)+',' for j in self.labels2) + '\n' )
```

```
97
98      self.font_size = 3
99      self.font_thickness = 5
100
101   def image_callback(self,data):
102     print "new image received"
103     try:
104       cv_img = self.bridge.imgmsg_to_cv2(data, "bgr8")
105       cv_img = cv2.resize(cv_img,(227,227))
106       print 'image received'
107     except CvBridgeError, e:
108       print e
109
110     im_input = self.transformer.preprocess('data',cv_img)
111     im_input = im_input[np.newaxis]
112     self.net.blobs['data'].reshape(*im_input.shape)
113     self.net.blobs['data'].data[...] = im_input
114     self.net.forward()
115
116     if 'fc7' in self.net.blobs.keys():
117       feature  =  self.net.blobs['fc7'].data
118       prob     =  self.net.blobs['prob'].data
119       result          = list()
120       result2         = list()
121       self.all_lables = list()
122       self.all_lables2= list()
123       for l in self.labels:
124           idx = l.label_id
125           self.all_lables.append(l.label_name)
126           result.append(prob[0,idx])
127       for l in self.labels2:
128           idx = l.label_id
129           self.all_lables.append(l.label_name)
130           result2.append(prob[0,idx])
131       result = np.array(result,np.dtype(float))
132       result = result / np.sum(result)
133       result2 = np.array(result2, np.dtype( float ) )
134
135     outfile2 = open( '/home/andreas/catkin_ws/src/ros-semantic-mapper/
136       results/distribution_205_office.csv', 'ab' )
137     writer2 = csv.writer( outfile2 )
138     writer2.writerow( result2 )
139     outfile2.close( )
140
141     self.msg.header.stamp = data.header.stamp
142     self.msg.header.frame_id = 'base_link'
143     class_idx = np.argmax(result)
144     print class_idx
145     if class_idx == 4:
146       self.correct += 1
```

```
147         outfile = open( '/home/andreas/catkin_ws/src/ros-semantic-mapper/
148             results/office_correctNN.txt', 'ab' )
149         outfile.write('{}'.format(self.correct))
150         outfile.write('\n')
151         outfile.close()
152 
153       class_name = self.all_lables[class_idx]
154       print class_name
155 
156       self.msg.r = [k.label_color[0] for k in self.labels]
157       self.msg.g = [k.label_color[1] for k in self.labels]
158       self.msg.b = [k.label_color[2] for k in self.labels]
159       self.msg.prob = result
160       self.msg.lvl = class_idx
161 
162       self.pub.publish(self.msg)
163       text_x = 10
164       text_y = 100
165       cv_img = cv2.resize(cv_img,(1280,960))
166       font = cv2.FONT_HERSHEY_PLAIN
167       for c in range(len(result)):
168         cv2.rectangle(cv_img,(text_x+0,text_y),(text_x+0+int(300*float(result[c])),
169             text_y - 20), (255,0,0),20)
170         if c == class_idx:
171           cv2.putText(cv_img,self.all_lables[c] + ' ' ,(text_x,text_y), font,
                 self.font_size,(0,255,0),self.font_thickness)
172         else:
173           cv2.putText(cv_img,self.all_lables[c] + ' ' ,(text_x,text_y), font,
                 self.font_size,(0,0,255),self.font_thickness)
174         text_y += 50
175 
176       try:
177         self.image_pub.publish(self.bridge.cv2_to_imgmsg(cv_img, "bgr8"))
178       except CvBridgeError, e:
179         print e
180 
181 def main(args):
182   rospy.init_node('sem_label_pub')
183   caffe_root      = rospy.get_param('~caffe_root', '~/caffe/')
184   MODEL_FILE      = rospy.get_param('~MODEL_FILE_PATH', '~/deploy.prototxt')
185   PRETRAINED      = rospy.get_param('~PRETRAINED_PATH', '~/model.caffemodel')
186   MEAN_FILE       = rospy.get_param('~MEAN_FILE_PATH', '~/mean.npy')
187   SVM_PICKLE_FILE = rospy.get_param('~SVM_PICKLE_FILE_PATH', '~/clf.pkl')
188   SUB_CAT_FILE    = rospy.get_param('~SUB_CAT_FILE','~/sub_cats.txt')
189   try:
190     ne = SemLabelPub(caffe_root,MODEL_FILE,PRETRAINED,MEAN_FILE,
191       SVM_PICKLE_FILE,SUB_CAT_FILE)
192   except rospy.ROSInterruptException: pass
193   rospy.spin()
194 
```

```
195 if __name__ == '__main__':
196   main(sys.argv)
```

Code 2: ROS python node used to interface with caffe, store the anaylsis results in .csv-sheets and publish the top-1 classification on a predefined topic. (Source code edited and taken from (Sunderhauf et al., 2016))

D.3 Python code for data set visualization

```
import csv
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as mpatches
num_classes = 205
labels = [j for j in range(num_classes)]
labels = ['']*len(labels)
labels[9] = '1'
labels[13] = '2'
labels[45] = '3'
labels[54] = '4'
labels[107] = '5'
labels[111] = '6'
labels[129] = '7'
labels[174] = '8'
row_number=0
with open('trainingset.csv','r') as csvfile:
   plots=csv.reader(csvfile, delimiter=',')
   for row in plots:
      row_number+=1
      for i in range(num_classes):
         if row_number <= 227:
                        plt.scatter(i,row[i], color='lightgray')
                elif row_number <= 454:
            plt.scatter(i,row[i], color='red')
         elif row_number  <= 682:
            plt.scatter(i,row[i], color='blue')
         elif row_number  <= 909:
            plt.scatter(i,row[i], color='black')
         elif row_number  <= 1137:
            plt.scatter(i,row[i], color='green')
         else:
            plt.scatter(i,row[i], color='cyan')
              plt.xlim(0, num_classes-1)
                     plt.ylim(0, 1)
                     plt.xticks(np.arange(num_classes), labels)
assembly_patch = mpatches.Patch(color='lightgray', label = '1: assembly_line')
auditorium_patch = mpatches.Patch(color='red', label = '2: auditorium')
closet_patch = mpatches.Patch(color='white', label = '3: closet')
corridor_patch = mpatches.Patch(color='blue', label = '4: corridor')
kitchen_patch = mpatches.Patch(color='green', label = '5: kitchen')
kitchenette_patch = mpatches.Patch(color='white', label = '6: kitchenette')
office_patch = mpatches.Patch(color='cyan', label ='7: office')
staircase_patch = mpatches.Patch(color = 'white', label = '8: staircase')
door_patch = mpatches.Patch(color = 'black', label ='door')

```

```
47 plt.legend(handles=[assembly_patch, auditorium_patch, closet_patch,
       corridor_patch, kitchen_patch, kitchenette_patch, office_patch,
       staircase_patch, door_patch])
48 plt.show()
```

Code 3: Python code used to generate the visualizations of the data set distributions.