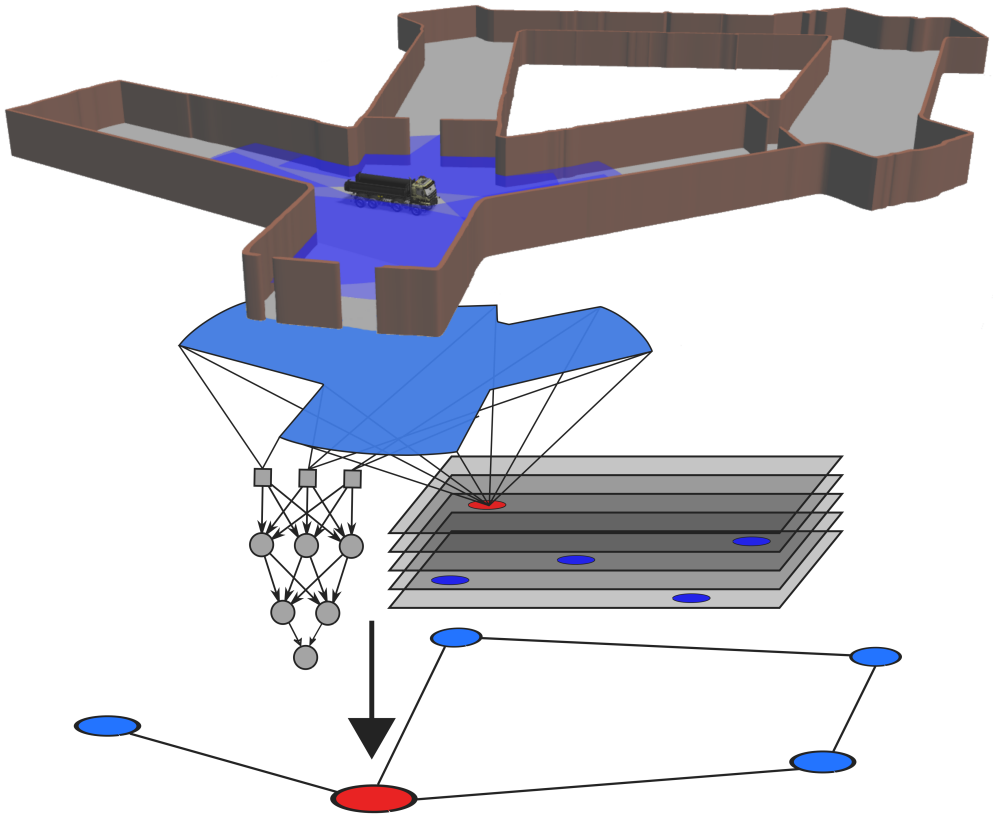
CHALMERS



Hybrid Map for Autonomous Commercial Vehicles

Global localization using topological mapping and machine learning

Master's thesis in Applied Mechanics

GUSTAF JOHANSSON
MATTIAS WASTEBY

Department of Applied Mechanics
CHALMERS UNIVERSITY OF TECHNOLOGY
Göteborg, Sweden 2017

MASTER'S THESIS IN APPLIED MECHANICS

Hybrid Map for Autonomous Commercial Vehicles

Global localization using topological mapping and machine learning

GUSTAF JOHANSSON
MATTIAS WASTEBY

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Göteborg, Sweden 2017

Hybrid Map for Autonomous Commercial Vehicles
Global localization using topological mapping and machine learning
GUSTAF JOHANSSON
MATTIAS WASTEBY

Master's thesis 2017:14
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Conceptual illustration of the complete system presented in this thesis. The characteristics of a two dimensional lidar scan is used to detect and classify nodes within the topology of a hybrid map.

Chalmers Reproservice
Göteborg, Sweden 2017

Hybrid Map for Autonomous Commercial Vehicles
Global localization using topological mapping and machine learning
Master's thesis in Applied Mechanics
GUSTAF JOHANSSON
MATTIAS WASTEBY
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

Abstract

We propose and investigated a novel method for global navigation and localization of an autonomous commercial vehicle within a confined area using a *hybrid map*. The hybrid map is based on a topology using nodes and edges where significant places are adapted as nodes. The hybrid map is able to store different type of machine learning algorithms and its flexible design allows the topology to be easily extended. The hybrid map operates using a node detector algorithm complimented with a node classification algorithm for increased robustness. The machine learning algorithms uses two dimensional lidar data as inputs exclusively. When it comes to the detection of nodes, performance evaluation showed that the *Adam* method are superior to the common gradient descent method when training feed forward neural networks in the considered scenario. In order to classify the nodes, one class support vector machines are preferred.

The performance of the hybrid map system was further on evaluated by implementing it on a Raspberry Pi 3 to prove its simplicity.

In conclusion, our results suggest that the system has potential for implementation in a real vehicle. However, it needs further verification and improvements to ensure a robust system and for it to be useful as a real application.

Keywords: Autonomous vehicles, Machine learning, Hybrid map, Classification, Robot Operating System

Preface

In 2015, a Vinnova (*Verket för innovationssystem*) financed project was initiated with the purpose to increase safety and efficiency in transport systems using autonomous vehicles [1]. The main application area is within the mining industry where autonomous ore trucks would, according to Parreira *et al.*, increase efficiency, reduce the risk of human injuries, and lower fuel consumptions [PM10].

Acknowledgements

This master thesis was carried out at CPAC Systems AB during the spring of 2016. We want to thank CPAC Systems AB for giving us this opportunity and all the resources needed to make this project possible. Especially, we want to show our gratitude to our supervisor Peter Forsberg who has encouraged and motivated us through the whole thesis. We have had many interesting discussions about this topic and his endless passion for technology is hard to not get influenced by.

We also want to thank our examiner, Krister Wolff, who has given us academical feedback throughout the whole thesis time. This type of feedback is most valuable.

[1] `vinnova.se/sv/Resultat/Projekt/Effekta/2009-02186/AUTOMATED-SAFE-AND-EFFICIENT-TRANSPORT-SYSTEM/`

Nomenclature

ANN	**A**rtificial **N**eural **N**etwork
ASGD	**A**veraged **S**tochastic **G**radient **D**escent
BLOB	**B**inary **L**arge **OB**ject
BP	**B**ack-**P**ropagation
BPTT	**B**ack-**P**ropagation **T**hrough **T**ime
ES	**E**xhaustive **S**earch
FFNN	**F**eed **F**orward **N**eural **N**etwork
GD	**G**radient **D**escent
GUI	**G**raphical **U**ser **I**nterface
OSVM	**O**ne-class **S**upport **V**ector **M**achine
RBF	**R**adial **B**asis **F**unction
RNN	**R**ecurrent **N**eural **N**etwork
*RP*3	**R**aspberry **P**i **3**
SGD	**S**tochastic **G**radient **D**escent
SMA	**S**imple **M**oving **A**verage
SQL	**S**tructured **Q**uery **L**anguage
SVM	**S**upport **V**ector **M**achine

Contents

1 Introduction and background

Intelligent vehicle systems has recently been one of the most prominent research areas within the field of robotics. Autonomous vehicles has a rich history with experiments conducted as early as in the 1920s [Sen26]. It began to seem promising already in the 1950s when General Motors and Radio Corporation of America developed automated highway prototypes with radio control for speed and steering [G M14]. The first genuinely autonomous cars emerged in the 1980s with the work of Ernst Dickmanns and his team at Bundeswehr Universität München [Dic07]. They were able to control their own motions in more complex environments such as highways and similar. Advances in sensor and computation technology has aroused an interest in the area and vast progress has been made in the past decade. Today, a majority of the leading car companies and related research organisations have developed numerous prototypes [Vie+15]. In the mid 2000s, DARPA organized the Grand Challenges where teams gathered to compete with their self-driving vehicles [BIS07]. Another major project within autonomous vehicles was initiated in 2009 by Google X which aims to develop highly advanced self driving vehicles using lidars [Cha13].

1.1 Background and context

One of the most difficult tasks in autonomous driving is to generate an accurate estimate of the vehicles position and orientation [Bic+04]. Most autonomous vehicles needs some information regarding its current environment in order to generate appropriate control inputs. This information is often acquired through a variety of sensors mounted on the vehicle that could be of different types. There are several methods solving the positioning and orientation problem in the literature where different methods are more suitable in some circumstances [SNS11]. In some cases, if the environment allows, a camera sensor could be enough in order to maintain the vehicles position. Some autonomous vehicles that travels in urban environments uses information given by its sensors for mapping its environment while simultaneously positioning itself in this map.

Since there are several applications areas in which autonomous driving is being tried out and evaluated, there are just as many solutions to the localization problem [Rao10]. In order to take care of all conceivable scenarios that a vehicle can encounter, more and more sensors are being introduced into the systems. It can be said that the complexity of the system increases with the complexity of the tasks that the vehicle has to be able to take care of [San15]. This is of course also correlated with the computational power because of the considerable amount of data being treated.

Machine learning has become more and more common within autonomous vehicles simply because of the complexity of the problem. The techniques has been available since the late 1950s [Sim13] but it did not start to flourish before the 1990s. With faster computers, machine learning has become feasible solutions to several problems within autonomous vehicles. In April 2016 Nvidia published a preprint of their promising results of DAVE2 [Boj+16], a project that has been going on since October 2015. Their system consists of a neural network that has been trained with less than a 100 hours of random driving data. It is able to map raw pixel data from a camera directly to steering commands of the vehicle. The results was surprising and their method is seen as very powerful in terms of their end-to-end approach. With results as these, it is hard to see a future without machine learning on the road of making a vehicle fully autonomous.

1.2 Purpose

Solving the localization problem is of great interest in the industry today as it opens up a big area of applications in many industrial fields. Many of these applications today involves a human driver who is exposed to dangerous elements such as traffic and non-reliable environments. A completely autonomous vehicle would remove these elements of risk as well as increase efficiency and lower environmental impacts [PM10].

The purpose of this thesis is to investigate the possibility of maintaining the position of an autonomous vehicle within a confined area, such as a mining system, without considering absolute positioning by instead using a relative *hybrid map*. The intention behind this is to minimize data acquisitions and reduce unnecessary computations and further explore the prospects of solving the localization problem in a simplistic fashion.

1.3 Scope

Since the thesis is of an exploratory nature, it is of high importance that the existing work within these areas is evaluated. The final goal, i.e. what is to be accomplished, is defined while how to reach it is to be discovered. The goal is to build a functional concept for global localization and path execution for an autonomous vehicle within a confined area by using a hybrid map. The aim is to base the hybrid map on new underlying ideas and it should be designed in such a way that it contains all information needed in order to let the vehicle navigate and localize itself.

The commencing task consists of studies of relevant literature in combination with experiments in order to define the extent of the thesis work. When the thesis work is characterized, a proposed solution (hypothesis) is presented. Based on the findings of the literature study and experiments, the proposed solution forms a framework which, in theory, solves the identified problems. The remaining part of the thesis work consists of the realization and evaluation of the proposed solution.

1.4 Limitations

This thesis considers the localization part of making a vehicle autonomous. It is assumed that navigation on a local level is available such that no effort will be put into this part. The local navigation system is built up of several algorithms that should be used in different scenarios. These algorithms are referred to as *behaviours* which will be selected depending on the position of the vehicle.

Autonomous vehicles operate in a wide variety of environments such as urban areas, private households and plantations. Developing a system that is able to operate in all these environments is a tough task as the prerequisites are different. This is why a confined environment with the character of a underground mine is considered.

The amount of research in the field of autonomous vehicles is substantial and the amount of research that can be considered in this thesis is limited. Therefore, only machine learning methods are considered as this area of research falls well in line with the purpose of this thesis.

1.5 Thesis outline

This thesis consists of seven chapters. Followed by this introduction, the theory chapter serves the purpose to brief the reader of the different methods and concepts that is used in this thesis work. Main topics described are map theory for autonomous vehicles and various machine learning methods that are commonly used today. This is followed by a methodological chapter of a proposal of what the final system is to result in together with the different methods that are considered. Next is a chapter that explains the platform used to acquire data and to evaluate system performance. This chapter is followed by the implementation of the system. The results from this work is then presented and discussed. Finally, the thesis work is rounded up with a conclusion of the final results as well as future improvements.

2 Theoretical background

In comparison to how a person operates a vehicle, it is equally important to be able to handle the vehicle on the roads as it is to maintain the correct route in order to fully make use of its purpose. This is also the case when dealing with autonomous vehicles. An autonomous vehicle needs to make use of its surroundings in order for it to make appropriate control decisions, while it also needs to exploit this information to locate itself to preserve the right track or route. Information of the vehicles surroundings is received through different sensors mounted on it, as for example a lidar which conveys distances to any objects in the vicinity of the vehicle. The newly arrived sensor information can then be compared to information that has been stored in advance to see if the vehicle is maintaining its course. There are several methods of performing this comparison mentioned in the literature such as map matching, various Bayesian approaches and machine learning techniques [SNS11]. As mentioned in Section 1.4, only machine learning techniques are considered in this thesis work. This chapter provides the information needed to understand these selected machine learning methods, how to evaluate them together with various other utilities necessary to grasp the coming chapters in this thesis.

2.1 Maps

Maps are used by autonomous vehicles as internal representations of their surroundings which can be one solution to the problem that is to successfully plan and move to target locations. Many different kinds of maps are proposed in the literature, all with different advantages and disadvantages [Bus05]. Due to the contrast in convenience of map types, it is common today to create a combination of maps into a so called *hybrid map*. Thus, hybrid maps integrates different kinds of representations in order to draw advantage of each map's strengths.

2.1.1 Metric maps

The most common type of map is the metric map where a distance corresponds to a distance in the real world. A map of this type can be used to for instance return a length of a certain path, the size of a building or an absolute position of a particular object. As a consequence of the high precision of a metric map, it consists of considerable amount of data and it can be utilized when detailed information is of importance.

2.1.2 Topological maps

Kuipers introduced topological maps for autonomous robots in 1978 where nodes represents geographical places and edges represents paths [Kui78]. Kuipers further extended the framework in 1999 with more topological and metric maps [Kui00]. Other map types are also introduced to the framework.

Unlike metric maps, topological maps do not contain metric information such as distance between objects. Instead, topological maps describes how objects are related and connected. This makes it intuitively easy to understand that these kind of maps contain less information than a metric map and are also more easy to interpret. A typical example of a topological map is a subway map which describes how, and in what order different stations are connected.

Topological maps uses different definitions of nodes and edges. Nodes can represent properties of the environment, such as rooms and entrances, which makes them easy for a human to understand. This also facilitates graphical representations of the map.

There are also topological maps where nodes represents internal states of the robot. One such map is proposed by Duckett *et al.*, where nodes are placed when the robot has travelled a certain distance [DMS02]. Edges connects two nodes and holds information about distance and angle between the nodes.

The properties of topological maps make them suitable for global path planning. As the maps consists of nodes and edges, algorithms used in graph theory can be used [Bus05]. Graph search algorithms such as Dijkstra's algorithm can be used for calculating shortest path from a initial node to a destination node [Dij59].

2.1.3 Hybrid maps

As mentioned, hybrid maps are combinations of maps, usually metric and topological maps. One such map can be seen in Figure 2.1. The reason is to utilize their respective advantages. However, several types of hybrid

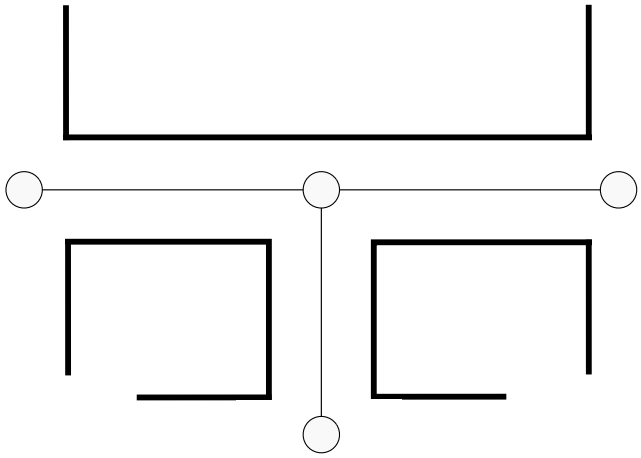


Figure 2.1: *An example of a hybrid map containing a metric map with a topological map that describes the relations between the intersection and corridors.*

maps have been proposed since the late 70s. Buschka attempts to categorize hybrid maps into "parallel maps" and "patchworks maps" [Bus05].

Parallel maps are defined as systems that has at least two maps covering the same area. In many of these maps, a topological map is extracted out of a metric map by using different techniques. This way, the best suited map can be used for a given task. For example, the topologic map could be used for path planning and the metric map used for navigation. Thrun *et al.* uses a topological map and a so called grid-based metric map [TB96]. The topological map is extracted from the grid-based metric map by using Voronoi diagrams such that the metric space has a corresponding node in the topological map.

In patchwork maps, each node in the topological map contains a small metric patch. This way, the topological map covers the whole area and describes how the metric patches are connected. This structure allows the map to cover very large areas with high detail and still be easily scalable. One such type of hybrid map was proposed by Kuipers *et al.* [Kui+04].

2.2 Localization

An autonomous vehicle bases its control decisions on information it receives from various sensors, prior knowledge about the intended route or any other useful information that describes its environment. Usually, the localization problem can be divided into two sub-problems; the *current location* of the vehicle and its *orientation*. The location of the vehicle could correspond to its position in relation to, for example a map of the vehicles surroundings or coordinates in relation to earth. The orientation tells which way the vehicle is facing. Two types of localization is mentioned in this chapter, namely *metric* and *topological* localization.

2.2.1 Metric localization

Estimation of the vehicles location and orientation can be done using motion tracking based on odometry or *dead reckoning*. This means to determine the current position based on the previous position. This is rarely a sufficient solution for most fully autonomous vehicles, due to the fact that the accuracy decreases in time without any further information of the surroundings [Mia+07]. Appropriate sensors available that provides external information of the vehicles' position and orientation would be a global positioning system (GPS) [MKH06]. However, a GPS does not work in all situations as they can have bad reception in some areas.

In order localize the vehicle metrically, both orientation tracking and global positioning requires some sort of information in relation to the vehicles surroundings. Nowadays, it is common to adopt information from a map for the vehicle to use as reference. If the area is known, the map can be constructed in advance which can give high precision results. Some autonomous vehicles are however used in urban environments such that the area is unknown. Thus, it is of interest to generate the map as the vehicle traverses. In order for it do so, it has the recurring problem of knowing its location in the map. A common technique used today is to generate or update a map of the unknown environment while simultaneously keeping track of the vehicle residing within

it. This problem is referred to as SLAM (Simultaneous Localization And Map building) [Dis+01]. Although the SLAM method provides accuracy, it requires computational power which is a limited resource in some applications. Usually the algorithm builds up a metric map of its surroundings which can end up in large data sets depending on the resolution/accuracy of the map as well as how large the area of which the vehicle is operating in is.

2.2.2 Topological localization

Using just a topological map as an instrument to localize, it is not of interest for the vehicle to know its absolute position. By simply having the knowledge of which section or part of the working area a vehicle is residing in could be enough. In such a case, it would be sufficient to use a topological map with nodes and edges corresponding to certain sections of the real world. Thus, it would be enough to know the vehicles position if the it could distinguish in which node or edge it is operating in.

There are many methods of detecting node and edge transitions in the literature. Kosnar *et al.* finds nodes by constantly searching for the number of outgoing edges from the current position [Koš+09]. The edges are found by looking for perpendicular vertices in the lidar data that are larger than a threshold value. If more than two edges are detected, a transition happens.

Choi *et al.* uses a global topological map that is compared to temporary local topological map [Cho+02]. The modified probability method and the Bayesian update rule is then used to compute the most likely position in the global topological map. Another type of topological localization was introduced by Ulrich *et al.* where a panoramic vision system is used to compare reference images, associated with every node, to what the robot currently sees. The similarity is computed by using the Jeffrey divergence [UN00].

2.3 Lidar sensor

Laser scanners or *lidars* are a type of surveying equipment that measures distances by lighting up surfaces using laser light. The technique is similar to the one used by radars but instead of measuring the time delay of radio waves, it measures the delay of light pulses. There are numerous advantages using a lidar instead of, say, a camera. It works in dark areas, the data is usually easier to process, and no calibration is needed.

Lidars are mainly used in a wide range of applications where high resolution is of the essence such as mapping geography or atmospheric physics [Cra07]. By letting light pulses from the lidar hit a rotating mirror, distances in a range of degrees can be measured very efficiently. A visualization can be seen in Figure 2.2 where two objects are blocking the field of view of the lidar.

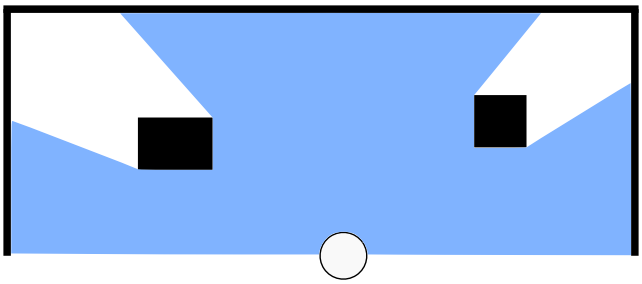


Figure 2.2: *Visible space (in blue) of a Lidar (circle) with 180 degrees vision field.*

This technique is something that is widely used when information of the surroundings rather than a single measurement to a certain object is of interest. Doing this, an array of distances is returned from the lidar where each element corresponds to a certain angle.

One can think of this stream of information as polar coordinates that describes the environment at a specific time instance. These polar coordinates are defined in terms of Cartesian coordinates as

$$\begin{aligned} x_i &= r_i \cdot \cos(\theta_i) \\ y_i &= r_i \cdot \sin(\theta_i) \end{aligned} \tag{2.1}$$

where r_i is radial distance from the origin to any hit object and θ_i is the counterclockwise angle from the horizontal axis. In terms of x and y,

$$\begin{aligned} r_i &= \sqrt{x_i^2 + y_i^2} \\ \theta_i &= \tan^{-1}(\frac{y_i}{x_i}) \end{aligned} \tag{2.2}$$

2.4 Machine learning

Machine learning is a field within computer science that has emerged from studies within pattern recognition and learning theories from artificial intelligence. Arthur Lee Samuel is said to be the pioneer within machine learning and artificial intelligence. His program called Samuels Checkers-playing program became the worlds first self-learning program which demonstrates the fundamental concepts of artificial intelligence [Sam59].

Machine learning is a collection term for several methods that explores the study and construction of algorithms that can learn and make predictions from given data. Machine learning is often used for classification or regression. Classification means to identify class membership, and regression is used for estimating or predicting an answer. An example of how one learning algorithm for classification works is shown in Figure 2.3.

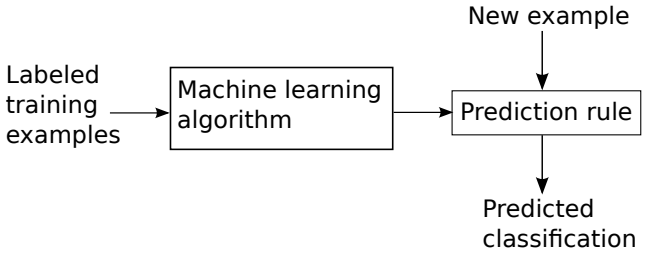


Figure 2.3: *Example of supervised learning algorithm for classification. The training data together with the desired output is given to the algorithm during training.*

The learning that is being done is based on some observed data, such as constructed examples, experiences or given instructions [Dom12]. The learning stage is used to build up a training model that is able to make predictions on other data sets. It can be said that machine learning algorithms learns to do better in the future, based on past attempts or experiences. The soul purpose of these types of algorithms is that they can improve their own learning without any human interaction or assistance in order to perform better.

Traditionally, the way of solving a computational based problem is by identifying the problem and explicitly programming the software such that it solves this specific problem. Instead, the machine learning algorithm builds its own model that solves the problem provided learning data rather than following any strict instructions. Machine learning is often used to solve complex tasks and algorithms that uses large data sets where computational power is a limited resource [CMM83]. In general, it can be said to exist three types of machine learning methods:

- Supervised learning: The algorithm is taught with example inputs with corresponding desired outputs given by a teacher. The goal is to learn a rule that is able to map inputs to specific outputs.

- Unsupervised learning: No labels are given to the algorithm, letting it structure its inputs on its own. This type of learning can be used to find hidden patterns in data or feature data.

- Reinforcement learning: The algorithm is set to act freely and learn from interactions with a dynamic environment with a specific goal (such as traversing forward), without any other tutoring such as its getting closer to this goal.

As mentioned above, these three general methods have advantages and disadvantages depending on the problem it is to solve. This is mainly dependant on of what type of information is available to the algorithm as well as identified by the tutor.

2.4.1 Support Vector Machine

In the field of machine learning, support vector machines (SVMs) are supervised learning models that can be used for labelling (classification) or regression. SVMs has been successfully used within a large variety of fields such as bioinformatics, text and image recognition [BGV92].

Given a set of training data each labelled such that they correspond to a specific category, the SVM builds a model that is able to classify new data as a member of one of these categories. The SVM maps its training data as points in a n-dimensional space where each category is separated as much as possible from the others. New data are then mapped into the same space to belong to one of the classes. An example of a linear SVM performing binary classification is shown in Figure 2.4.

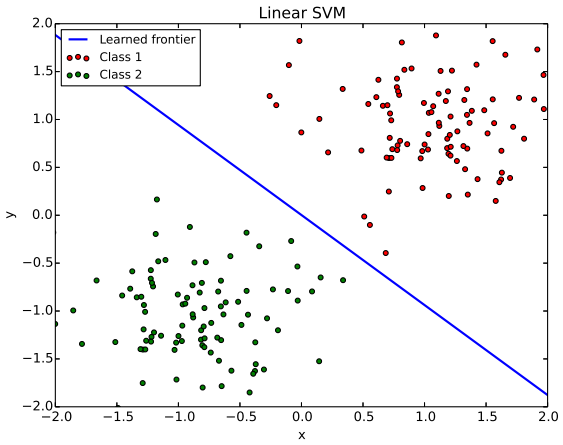


Figure 2.4: *Representation of a linear SVM performing binary classification of 2D data. In this case, all data is separable such that the model reaches convergence.*

In 1999, Schölkopf et. al introduced a modified version of SVM with a new parameter ν which can be used to control the upper bound on the fraction of margin errors and the lower bound of the fraction of support vectors [CL02]. This new parameter is, as opposed to the non-modified SVM, bounded between 0 and 1.

Linear Support Vector Machine

A big advantage of using linear SVMs is that they can be trained using stochastic gradient descent (SGD) which is useful for very large data sets [Men09]. Gradient descent (GD) is a first order optimization algorithm that uses the whole training set in every iteration to find a local minima:

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t) \tag{2.3}$$

where η is the learning rate and $\nabla_w Q(z_i, w_t)$ is the gradient of the empirical risk. Instead, SGD uses only one sample from the data set in every iteration for training the SVM, which makes it faster than GD [Bot12]:

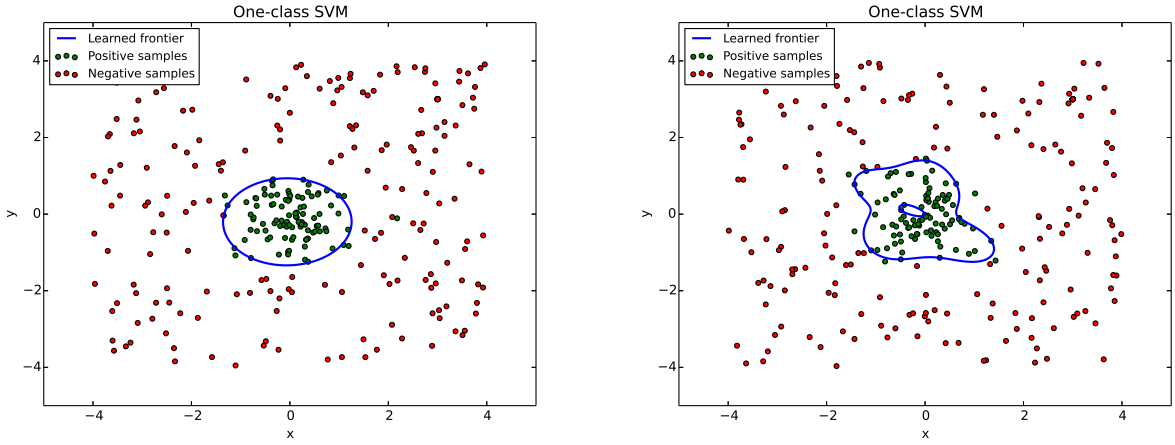
$$w_{t+1} = w_t - \eta \nabla_w Q(z_t, w_t) \tag{2.4}$$

where z_t is a randomly picked sample from the training set. In some cases, the averaged stochastic gradient descent (ASGD) is proven to perform better than SGD [Xu11]. The algorithm first performs the normal SGD update and then computes the average:

$$\bar{w_t} = \frac{1}{t - t_0} \sum_{i=t_0+1}^{t} w_t \tag{2.5}$$

One-class Support Vector Machine

In a scenario when only data from one class is available (i.e. only positive data), supervised training is not possible as no negative data is available. The One-class SVM (OSVM) is a special case of SVM as it is trained unsupervised on only positive data to determine whether a new observation is a outlier or not. This type of SVM were introduced by Schölkopf et. al in 1999 [Sch+01].



(a) $\gamma = 0.1$, $\nu = 0.1$ (b) $\gamma = 1.0$, $\nu = 0.1$

Figure 2.5: *One-class classification of toy 2D data using OSVM with RBF kernel, for two different values of γ. Not all positive samples are classified correctly.*

In linearly non-separable data sets, a kernel can be used to map the data to a higher dimensional space where it linearly separable. A common kernel used in OSVMs is the Gaussian Radial Basis Function (RBF) which only has one hyperparameter, γ, which can, intuitively, be seen as the learning rate:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \, , \; \gamma > 0 \tag{2.6}$$

A large γ means a Gaussian with a small variance so the influence of the sample x_j is smaller compared to when a small γ is used. Two examples of OSVM classification for different γ are illustrated in Figure 2.5.

When designing a OSVM, a common technique to find best combination of γ and ν is to use Exhaustive Search (ES). ES is of brute force type as it iterates through all possible candidates to find the one with the highest amount of correctly classified samples.

2.4.2 Artificial neural network

Artificial neural networks (ANNs) are a collections of mathematical models inspired by the biological neural networks of brains in humans and animals. These networks are used to solve a wide variety of complex tasks without using rule-based programming where the behaviour is decided by the designer. Instead, ANNs are, just like other machine learning methods, systems that learn from data. A comprehensive summary of artificial neural networks were written by S. Haykin [HN04].

ANNs consists of interconnected groups of computational elements called neurons and input elements. All connections have an associated weight consisting of a numerical value. The two basic types of ANNs are feedforward neural networks (FFNNs) and recurrent neural networks (RNNs). The difference between them is that FFNNs only have forward-pointing connections and RNNs have both forward- and backward-pointing connections.

The most common model of an artificial neuron was proposed by McCulloch *et al.* in 1943 and is defined as in Equation (2.7) and vizualised in Figure 2.6 [MP].

$$y = \sigma\left(\sum_{j=0}^{n} w_j x_j\right) \tag{2.7}$$

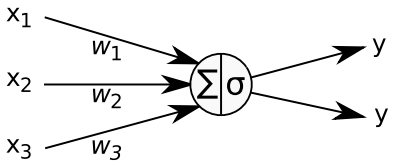


Figure 2.6: *A vizualisation of the McCulloch neuron in Equation* (2.7)

Every connection has a associated weight w which is used to scale every signal x to a neuron. The neuron computes the weighted sum and converts it by using a activation function $\sigma(x)$. The activation function defines the output of the neuron and is, in some cases, used for practical reasons as they often put a upper and lower limit to the neuron output. Several types of activations functions exists but three common functions are the logistic sigmoid function, the hyperbolic tangent function, and the rectified linear unit function:

$$\begin{aligned} \sigma_{ls}(x) &= \frac{1}{1+e^{-x}} \\ \sigma_{ht}(x) &= \frac{2}{1+e^{-2x}} - 1 \\ \sigma_{ReLu}(x) &= \max(0, x) \end{aligned} \tag{2.8}$$

Perceptrons

The perceptron is the simplest type of FFNN as it do not have any hidden neurons, the inputs are directly fed to the output layer via the associated weights. With this structure, the perceptron is only able to perform linear classification. The perceptron was invented by F. Rosenblatt in 1957 [Ros57].

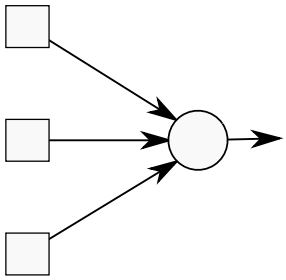


Figure 2.7: *A perceptron with a fully connected input- and output layer.*

Because of the simplicity of the perceptron, it can be trained in a simple and fast way compared to multi-layered ANNs where backpropagation has to be used. However, as the model is a linear classifier, it will never successfully classify all data correct if the data set is not linearly separable. An advantage is that the model is only updated when a false classification has been performed, making it suitable for very large data sets. The weight w_i is updates as

$$\begin{aligned} w_i &= w_i + \Delta w_i \\ \Delta w_i &= \eta(t-o)x_i \end{aligned} \tag{2.9}$$

where x_i is the associated input, η is the learning rate, t is the current training sample's output, and o is the current output of the perceptron.

Hidden layered feedforward neural networks

FFNNs consists of multiple layers with a number of neurons in each layer. Every neuron in a layer is the input to all other neurons in the next layer. One such FFNN can be seen in Figure 2.8. The layers in the middle of the network is called hidden layers, and the last layer is always the output layer.

When designing a FFNN, the amount of hidden layers and the amount of neurons in each layer has to be specified by the designer. A majority of problems can be solved using only one hidden layer and if the data is

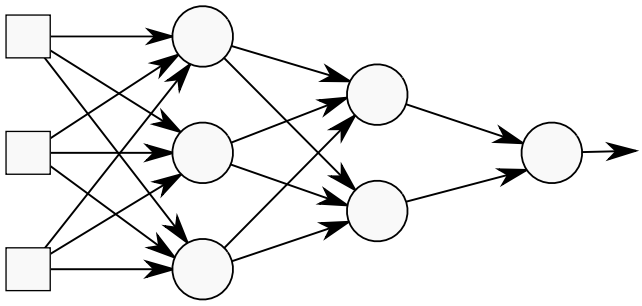


Figure 2.8: *A feedforward neural network (FFNN) with a input layer, two hidden layers, and a output layer. The hidden layers have three and two neurons respectively.*

linearly separable, no hidden layers are needed at all. When choosing the number of neurons in a hidden layer, a common rule-of-thumb is to use 2/3 as many neurons as in the previous layer [Kar12].

A common way to train FFNNs is to use the back-propagation (BP) algorithm. This learning procedure was introduced by Rumelhart *et al.* [RHW88]. In order to use this method every input value needs a desired output value, which is why it is considered a supervised learning method. Another requirement to use BP in the learning process is that the activation function is differentiable [HN04]. Both of the activations functions mentioned above have this property.

First, a forward pass is performed in order to compute a prediction. The error is then calculated by taking the difference between the predicted value and the true value. This error, or cost, is then used to compute weights for all connection in the network, normally using gradient descent described in Section 2.4.1. This step is called the backward pass as the error is passed back into the network. The process is performed iteratively until all training data are predicted correctly or until a stop criterion is fulfilled.

In 2015, Kingma *et al.* introduced a new algorithm, named *Adam*, for gradient-based optimization for use in neural network training [KB14]. This algorithm is similar to stochastic gradient descent as both are stochastic processes but Adam has been shown to be more robust if the learning rate of the SGD is not properly tuned [KB14]. Adam is, just like SGD, preferable over GD when training on very large data sets.

Recurrent neural networks

FFNNs can be used to solve a great deal of problems. However, in these networks the input-output pairs are assumed to be independent. This means that, for any given input, the output signal will be the same. In RNNs, the output is dependent on previous computations, which gives them a dynamic memory. This is useful in a lot of applications where sequential information is fed to the network, such as handwriting recognition by Graves *et al.* and generation of natural language descriptions of images by Karpathy *et al.* [Gra+09] [KF15].

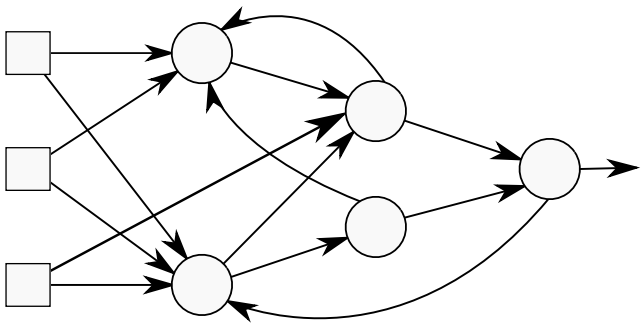


Figure 2.9: *A fully recurrent RNN with five neurons. This network has, opposed to FFNNs, feedback connections.*

Most RNNs have, as opposed to FFNNs, feedback connections where a neuron can receive input signals from any other neuron in the network. One such RNN can be seen in Figure 2.9. A neuron could also be the input to itself. These type of RNNs are called fully recurrent and is the most general form.

Training of RNNs can be performed similar to training FFNNs using the back-propagation algorithm. In this case, the output does not only depend on the current time step but also previous time steps. This is why

Backpropagation Through Time (BPTT) is used. However, training fully recurrent RNNs have known issues, the vanishing and the exploding gradient problems described by Bengio *et al.* in 1994 [BFS93]. The problem manifests itself in that RNNs have problem learning long-term dependencies.

2.5 Performance evaluation

In this section, different methods of evaluation are presented.

2.5.1 Confusion Matrix

A confusion matrix consists of information of the actual and the predicted classifications that has been done by a classification system. The matrix is used as a measurement of how well the classifier is performing. Table 2.1 describes a two classifier system.

Table 2.1: A confusion matrix and a accuracy measure.

	Actual positive	Actual negative
Predicted positive	TP	FP
Predicted negative	FN	TN
Accuracy	ACC	

To explain it further, the entry TN describes how many samples that has been corrtectly predicted as negative. Entry FP describes how many negative samples that has incorrectly been labeled as positive. Entry FN says how many samples that has been labeled as negative by the classifier when these samples actually where positive. And finally, entry TP says how many samples that has been correctly predicted as positive. In order to evaluate the accuracy of the system equation (2.10) is used which is a measure of how often the classifier is correct.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.10}$$

2.5.2 CPU and memory usage

The measure of CPU usage of a specific process is the ratio between CPU time to real time. The CPU time is seen as the duration of which the process has been running and real time is a certain time interval.

$$\text{CPU usage} = \frac{\text{CPU time}}{\text{Real time}} \tag{2.11}$$

This way, the ratio is an averaged CPU measure which might not cover all characteristics of a process if the time interval is large. At times, processes can go from idle to active and vice versa. However, using small enough time intervals the current CPU usage of a process can be monitored.

A common way to evaluate a process's memory usage, is to measure the resident set size (RSS) of the process. RSS is the amount of memory allocated in primary memory by the process. By taking the ratio of the RSS and the total size of RAM, a measure of memory usage is achieved.

$$\text{memory usage} = \frac{\text{RSS}}{\text{Total RAM}} \tag{2.12}$$

3 Proposed hybrid map

This chapter presents and describes the proposed system. The general structure is first described, followed by a more in depth description of the methods to be evaluated and used.

3.1 Overview of the system structure

As mentioned in the introduction, it is of interest to solve the localization problem with the underlying idea that an absolute positioning is not needed. This means, without ruling out any options, that the autonomous system only needs to know its whereabouts in critical locations. The reason behind this is that the vehicle might need to behave differently depending on the vehicle's current environment and situation. A prosed way of doing this is to represent the area of which the vehicle is operating in on a topology, such that critical locations in the actual world represent nodes in a topological map.

The structure of this proposal will consist of three main blocks that together forms a hybrid map. Each block has different tasks which together are able to do node detection, node classification, global positioning, global path planning, and behavior selection. The reason why the map is of a hybrid type is because the vehicle will navigate using topological data combined with temporary local metric information. In Figure 3.1, the relation between the three blocks is shown.

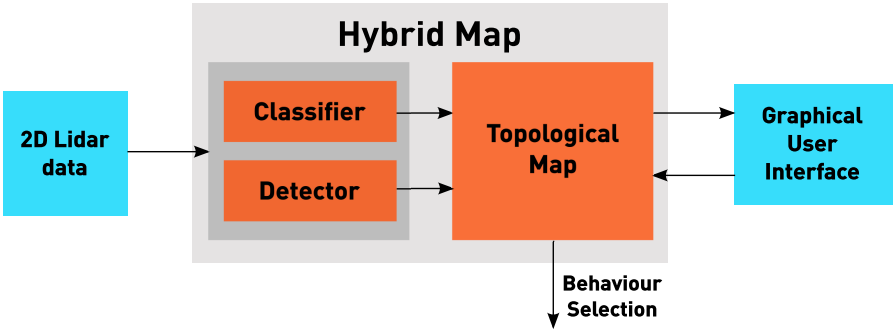


Figure 3.1: *Relationship between the tree main blocks. Lidar data is the input to both the detector and the classifier block. The output of these are then fed to the topological map. The topological map decides the behavior of the vehicle and a graphical user interface is used to represent the position of the vehicle in the topological map.*

The proposed hybrid map is based on a number assumptions:

- The initial position of the vehicle is known.
- The world is known *a priori*. This way, the topological map can be built in advance and sensory data from the world is available.
- Every place is assumed to be distinguishable to the sensors used, i.e. there are no identical places.

The topological map stores the relative position of the vehicle and keeps track of the path in the system. The nodes represents crossings, meeting places, dead ends and other significant places. The edges is seen as the corridors between these significant places. This type of topological map is easy for a human to interpret as all nodes corresponds to physical places. The representation is also suitable for global path planning (e.g. Dijkstra's algorithm [Dij59]) as the length of the edges can be stored in the map. Different behaviors of the vehicle can be selected by the hybrid map depending on criteria, such as the current position and the type of the next node. Using this approach, exact positioning is not possible. On the other hand it is not needed according to this proposal.

By detecting nodes, it is possible to know when an edge or a node transition happened. It is in this way the position in the topological map is updated. The position of the vehicle is described by the topology of the hybrid map as a particular node or edge. The detection of nodes is performed using lidar data exclusively. This way, the system is kept as simple as possible and negative aspects that can arise from using other sensors are avoided. Different methods of node detection is considered and evaluated.

Node classification is needed to make sure that the hybrid map is robust. If a node has been detected, it is possible that the vehicle has drifted away from its course and ended up in a neighbouring node, away from the intended one. This would result in global miss-positioning. By performing classification of the nodes as well, the map does not only know that a node transition has happened, it also knows which nodes it has transitioned into. As the initial position of the vehicle is known, all possible neighbouring places to travel is known. This fact is utilized as the hybrid map will only try to classify the neighbouring nodes from its current node, thus creating synergy in the map and reducing the computational complexity as fewer nodes are considered. Having this structure of a detection block and a classification block, the classification process is simplified as the edges in the map does not have to be considered. The node classifier only considers nodes, while the node detector considers both nodes and edges.

3.2 Machine learning methods

Different machine learning methods for node detection and classification are considered, as mentioned in Section 1.4. An advantage of using these types of methods is that human interaction is kept low which would not be the case if explicit programming methods were used. As the world is assumed to be known in advance, sensory data is available which makes learning based programming methods suitable. Figure 3.2 depicts an example of how a lidar sensor scan can look like represented on a 2D plane when the vehicle is positioned in a crossing.

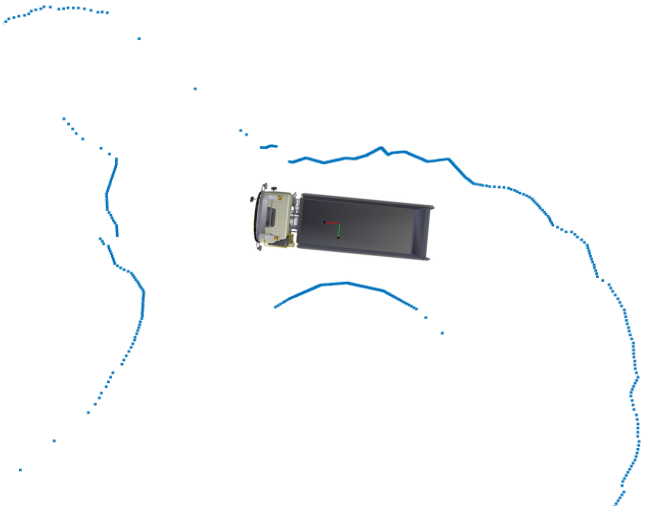


Figure 3.2: *Example of sensory data from lidars when the vehicle is located in a node of type crossing.*

The task is to let the vehicle interpret this data in order to detect that it is a node and to classify which specific node it is using machine learning methods. The main difference between detecting and classifying a node is that the algorithm used in the detection is able to differentiate between node and edge in the system the vehicle operates in. The classification algorithms is able to distinguish nodes from each other in order to tell which node the vehicle currently is in.

The two libraries used for machine learning methods are *PyBrain* library and *Scikit learn* library depending on the learning method. Both libraries are written in Python and are open source.

Recurrent neural networks are decided to not be evaluated, neither as a detector nor as a classifier. The reason is the dependency of previous inputs to the RNN and the character of the problem to be solved by the detector and classifier. It is desirable to at all times be able to, no matter previous inputs, detect and classify the surrounding of the vehicle.

3.2.1 Topological definitions

Let $\mathbf{N}$ be a finite set containing all nodes in the topological map, where the nodes are of type crossing, meeting place, dead end, or room:

$$\mathbf{N} = \{N^c, N^m, N^d, N^r\} \tag{3.1}$$

Further on, $\mathbf{E}$ represents the finite set of edges between all nodes in the topological map.

3.2.2 Definition of data sets

One requirement when using machine learning methods is that the data used in the training process needs to be appropriately configured to the specific method in order for it to be able to build up a feasible model. The data should also be representative of real scenarios. The problem of making the model representative is usually solved by collecting large sets of training data, while configuring it beforehand requires a process of trial and error.

Machine learning methods benefits differently depending on how the data set used for training is composed. As mentioned in the Section 3.1, the area of which the vehicle is to travel can consists of several different places with distinctive features. The data sets used for training are collected in different fashions depending on the intended machine learning method. The factors that can be altered are the field of view of the lidars, what type of data that the set consists of and how it has been labeled.

- D_1: Full field of view. Labeled data. $\forall$ nodes $\in \mathbf{N}$ and $\forall$ edges $\in \mathbf{E}$.
- D_2: Front field of view. Labeled data. $\forall$ nodes $\in \mathbf{N}$ and $\forall$ edges $\in \mathbf{E}$.
- D_3: Full field of view. Data from one node $\in \mathbf{N}$. All scenarios.
- D_4: Full field of view. Data from one node $\in \mathbf{N}$. One specific scenario.

The term *all* or *one specific scenario* can be confusing. If the specific node is of type crossing, there are several ways of entering and exiting this crossing. All scenarios means to collect data from traversing through this crossing in every way, including all routes possible in all directions. The term *One specific scenario* means to collect data from only one specific route through this crossing.

These four data sets will be referred to throughout the report. Sections 4.3 explains how data is collected.

3.2.3 Node detection

The node detection should be able to detect any node type, such as crossings, meeting places, and dead ends. Therefore, the data collected from all these places have varying characters. However, it is assumed that the corridors have, throughout the confined area, a uniform character.

One way to see this problem is to describe it as a binary classification problem. Another way is to describe it as a regression problem where the output variable is the probability of one class compared to another. The nodes in the topological map represents one class $\mathbf{C}_1$, and the edges another $\mathbf{C}_2$ such that:

$$\begin{aligned} \mathbf{C}_1 &= \mathbf{N} \\ \mathbf{C}_2 &= \mathbf{E} \end{aligned} \tag{3.2}$$

There are several methods for these type of problems in the literature. In this thesis, different types of artificial neural networks and support vector machines are considered. Other not considered methods are Decision trees, Bayesian networks and Logistic regression [TSK06] [Ben07] [Cox58] among others.

Linear SVMs are considered as they are trained in a short amount of time and results in very small weight sets compared to ANNs. The evaluated linear SVMs will be trained using the stochastic and averaged stochastic gradient descent in order to find the optimal training method. If the data is not linearly separable, these methods will perform poorly.

Four artificial neural networks with different number of hidden layers and training methods are also considered for node detection. A perceptron trained with SGD, two single layered hidden FFNNs trained with GD and Adam, and a two layered hidden FFNN trained with GD. The FFNN trained with Adam uses ReLu

activation function and the other two uses the hyperbolic tangent sigmoid. The number of neurons in each layer is decided by using the the 2/3 rule-of-thumb mentioned in Section 2.4.2. All these methods are trained for the two different data sets, D_1 and D_2:

$$\begin{aligned} D_1 &: \text{248 inputs, 165 hidden neurons(, 110 hidden neurons), one output} \\ D_2 &: \text{124 inputs, 82 hidden neurons(, 54 hidden neurons), one output} \end{aligned} \tag{3.3}$$

Table 3.1 provides an overview of all combinations of evaluated node detection methods. All methods uses the same class definition, but are evaluated for two different data definitions.

Table 3.1: Overview of the twelve evaluated combinations of node detection. The definitions refers to equation numbers described in this section. The prefixes '1' and '2' in front of 'FFNN' refers to single and dual hidden layers respectively.

Method:	SVM (SGD)		SVM (ASGD)		Perc. (SGD)		1-FFNN (GD)		2-FFNN (GD)		1-FFNN (Adam)	
Data set:	D_1	D_2	D_1	D_2	D_1	D_2	D_1	D_2	D_1	D_2	D_1	D_2
Definition:	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)	(3.2)

3.2.4 Node classification

It is assumed that all corridors has a uniform character such that classification of these are disregarded. The classification of nodes is a very different type of problem, in comparison to the detection of it as every node in the topological map should be distinguishable. One straight-forward way to define this problem is to let every node be one class:

$$\begin{aligned} \mathbf{C}_1 &= \mathbf{N}(1) \\ \mathbf{C}_2 &= \mathbf{N}(2) \\ &\vdots \\ \mathbf{C}_n &= \mathbf{N}(n) \end{aligned} \tag{3.4}$$

This way a single SVM or ANN model could be used for classification of all nodes. However, this would result in a non-modular hybrid map as this classifier would have to be retrained if the confined area were to grow such that the map has to be updated. Because of this reason, this implies a impractical approach and is therefore not considered.

Another, way of defining the classification models is to let every node i have one classifier each where the first class contains node i and the second class contains all other nodes:

$$\begin{aligned} \mathbf{C}_1 &= \mathbf{N}(i) \\ \mathbf{C}_2 &= \mathbf{N} \setminus \mathbf{C}_1 \end{aligned} \tag{3.5}$$

This way the hybrid map can easily be extended as new places are added to the confined area. With this definition, the same machine learning methods as in the node detection are evaluated for the same parameters.

A third option would be if the classifier is able to classify node i solely on positive training data. This would result in a considerably simpler training procedure. Since only positively labeled data is used only one class is defined as:

$$\mathbf{C}_1 = \mathbf{N}(i) \tag{3.6}$$

One class support vector machines are suitable for this type of problem as they are trained with positive data exclusively. Depending on the character of the data, one OSVM could be used for an entire node (data set D_3 or one OSVM for every entrance in the node (data set D_4). The RBF kernel is used in the OSVMs

and initial experiments using exhaustive search showed that using $\nu = 0.1$ and $\gamma = 0.1$ had highest score of classification.

Unlike the node detector, hidden layered neural networks will not be considered as a node classifier. Compared to SVMs, using hidden layered FFNNs results in much larger weight sets. These weight sets should be as small as possible as the map will have at least one weight set per node. The combinations that are evaluated are shown in Table 3.2. All methods except the OSVM uses the class definition described in (3.5).

Table 3.2: Overview of the eight evaluated combinations of node classification. The definitions refers to equation numbers described in this section.

Method:	SVM (SGD)		SVM (ASGD)		Perc. (SGD)		OSVM (Unsupervised)	
Data set:	D3	D4	D3	D4	D3	D4	D3	D4
Definition:	(3.5)	(3.5)	(3.5)	(3.5)	(3.5)	(3.5)	(3.6)	(3.6)

4 Platform for data acquisition and evaluation

In order to evaluate the performance of the software, a simulation environment is built up. Having a simulated environment allows easy testing and evaluation of the proposed system. Also, if the simulation is sufficiently accurate, sensory data can be recorded and used offline to train the machine learning methods.

The platform is written in C++ and Python as so called nodes which are executables in program packages used in Robot Operating System (*ROS*). ROS is an meta-operating system for setting up the software for any kind of robot. It is an open source software that consists of useful libraries and tools with the intent to simplify the development of robotic systems [Qui+09]. There is a substantial amount of third party packages available, many of them widely used within robotics such as linking different frames together or merging sensors. It also has the support of running code across multiple computers. The fact that every function is made up from separate processes makes the system architecture flexible and exchangeable.

The evaluation platform consists of two computers connected via ethernet. The hybrid map itself is implemented on a Raspberry Pi 3 (RP3) running Raspbian Jessie and ROS Indigo. The RP3 has a 1.2 GHz 64-bit quad-core ARMv8 CPU and 1 GB of RAM [16b]. It is presumed that if the system works with the RP3, it is evidently a light-weight system. The other computer is a laptop running Ubuntu 14.04 and ROS Indigo. The scheme showed in Figure 4.1 illustrates the interaction between the two computers and what they are responsible for.

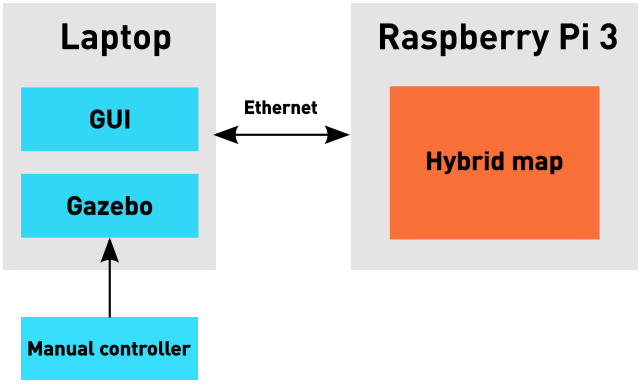


Figure 4.1: *The evaluation platform. The laptop is running the graphical user interface and Gazebo. The simulated vehicle is controlled by a manual controller. The Raspberry Pi 3 is running the hybrid map.*

The laptop runs a so called *Gazebo* simulator tool together with the graphical user interface (GUI). The GUI is used for setting a destination node and for supervising the position of the vehicle. A manual controller is used to interact with ROS in order to control the simulated vehicle in Gazebo. If the system where to be used in a real world application, the RP3 would be integrated with the vehicle. The laptop running Gazebo would of course not be needed.

4.1 World model

The interface *gazebo_ros_pkgs* is used which integrates the open source software *Gazebo* with ROS. Gazebo is chosen as the simulation tool as it comes with a physics engine and graphics with support for three dimensions [KH04].

Images from areas where the vehicle is meant to travel in are available. These images are sections of a mining system or testing environment seen from a birds eye perspective. By seeing these pictures as binary, image tracing is used which essentially performs raster-to-vector conversion. The now vectorized image can be imported into *Google Sketchup* which enables the extension to the third dimension. Sketchup is used since it is a simple and free design tool for 3D models. By utilizing image tracing, any picture can be used to rapidly build up a world that can be used in simulations. This procedure is described in Figure 4.2.

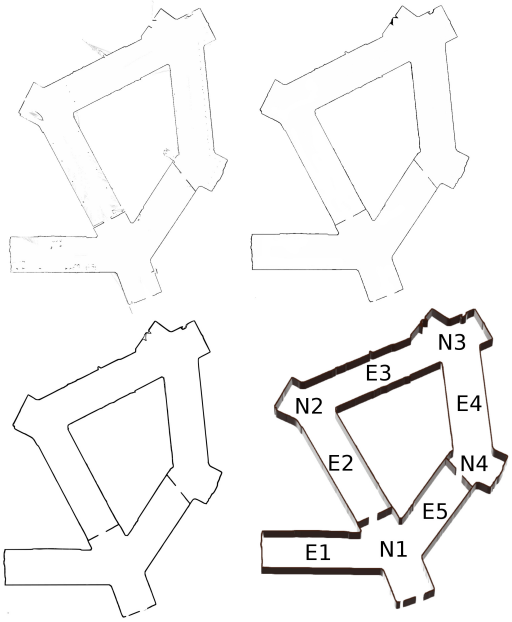


Figure 4.2: *The four different levels in producing a world model from an image. The fourth illustration will be used in the testing process.*

The area seen in Figure 4.2 is a map of *Säve Depå*. It is here training data will be collected and it will serve as testing grounds of the system. The first illustration is a metric map built from sensory data. While this map is an accurate representation of reality, it is too rough to vectorize. The second image is a cleaned up version of the first. In the third illustration, the image has been vectorized such that it describes inner and outer coordinates which forms the thickened lines. The final result comes from extruding the plane into a third dimension using Sketchup, which is presented in the fourth illustration. The nodes and edges are labeled as N and E respectively.

4.2 Vehicle model

In order to gather data from the world model, a model that represents the actual truck is used. This vehicle model is based on the *Volvo FMX* truck which is the one used in the confined area. It can be seen in Figure 4.3.

Figure 4.3: *The model of the Volvo FMX truck used for simulation.*

The most important part of this model is that the sensors are placed in a similar fashion as they are mounted on the real truck. The sensors are themselves models of the *LMS111-SICK* 2D lidar [16a] which are the ones used on the Volvo FMX truck. The sensors have an adjustable field of view, update frequency and number of

readings in each scan.

The vehicle traverses within the world by manual driving using Ackermann steering geometry [BHW04]. It is important that the vehicle behaves the same in the simulation as it does in the real world. The model is designed using real dimensions in order to simulate appropriate dynamics as well as for visualization purposes.

4.3 Data collection

The data used in training and evaluation of the different learning methods is collected by traversing the vehicle model within the built world. The driving is executed in such a way as it would in the real world such as extending the turning angle during a sharp turn and driving slow in more complex areas. While driving, the sensors picks up distances to the surroundings which are stored. The readings from the sensors are received in forms of an array where each slot in the array corresponds to a certain angle in relation to the lidar. The output arrays of all four lidar are concatenated into one named $\mathbf{x}$ for easier management. The data is stored as

$$\mathbf{D} = \{\mathbf{x}_t, \mathbf{x}_{t-1}, \ldots, \mathbf{x}_{t-n}\} \tag{4.1}$$

where every $\mathbf{x}_i$ is timestamped and associated with a label.

An advantage using machine learning techniques is that no regard has to be taken to in which order each lidar output is fed into the training process. If an ANN where to be used, the only requirement is that every output from the lidar represent the same input to the network at every occasion. Each lidar outputs 720 readings evenly spread out over its field of view of 270 degrees. Seeing the field of view of all lasers as one results in a total of 1080 degrees, evidently resulting in angular overlapping. Reducing the number of inputs to the system simplifies the training of the machine learning models. The field of view of the each lidar are therefore reduced in such a way that it is made sure that no critical information is bypassed. This is explained in Figure 4.4.

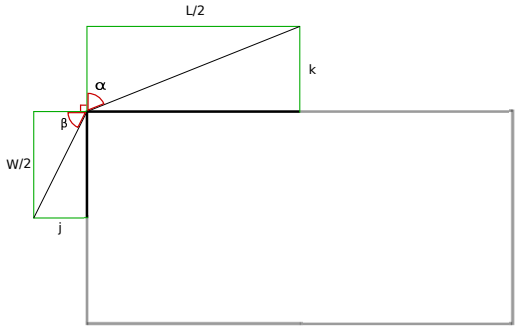


Figure 4.4: *The rectangle represents the vehicle. The field of view for one lidar sensor is described.*

Looking at Figure 4.4, k and j are design variables that are determined based on how close to the vehicle each lidar should receive information. Since the lidars are placed in a symmetric fashion, the critic angles are the most outer ones of each lidars field of view. Thus, the critical distances k and j are evaluated at the middle points of the short and long side of the vehicle. The angular portions α and β are calculated as:

$$\begin{aligned} \alpha &= \tan^{-1}\left(\frac{L/2}{k}\right) \\ \beta &= \tan^{-1}\left(\frac{W/2}{j}\right) \end{aligned} \tag{4.2}$$

Since the two drawn triangles in Figure 4.4 are perpendicular to each other, the reduction of the lidars field of view at the vehicles long and short side ($\hat{\alpha}$ and $\hat{\beta}$ respectively) results in:

$$\begin{aligned} \hat{\alpha} &= (1 - \frac{\alpha}{\pi/2}) \cdot \frac{\pi}{2} \\ \hat{\beta} &= (1 - \frac{\beta}{\pi/2}) \cdot \frac{\pi}{2} \end{aligned} \tag{4.3}$$

where $\frac{\pi}{2}$ represents the original 90 degree portions of the lidar field of view at the vehicles long and short side when $k, j \to 0$. In order to reduce the number of inputs even further, only every n:th value are extracted from the lidar output (seeing the four of them merged into one). This means that the angular increment of which a distance is extracted from can be seen as

$$\frac{n(\frac{3\pi}{2} - (\hat{\alpha} + \hat{\beta}))}{720} \tag{4.4}$$

5 Implementation of the hybrid map

This chapter serves as a description of how the final hybrid map with all its components is designed and implemented. Every part is presented individually in order to describe how they complement each other.

Based on the results from Section 6.1, two candidates of machine learning methods are selected and are used in the final hybrid map. The neural network trained with Adam is used for node detection and OSVMs are used for classification of every entrance of each node. This is mainly based on their ability to make correct classifications according to the performed tests. The data sets that seems the most appropriate in solving this particular problem is D_1 and D_4. Data set D_1 is used in the node detector because of the full field of view. With a 360° view of the surroundings, information of when the vehicle is leaving a node is available. The OSVM performed better classifications with data set D_4 such that this definition is preferable. This means that every node will have as many OSVMs as entrances.

The three blocks, described in Chapter 3.1, which, together with the vehicle behaviors, constitutes the hybrid map are implemented as ROS nodes. These nodes passes information between each other using the framework specified in ROS. A few utility nodes are used as well for various minor tasks. The full system structure is presented in Figure 5.1. All ROS nodes are written in Python.

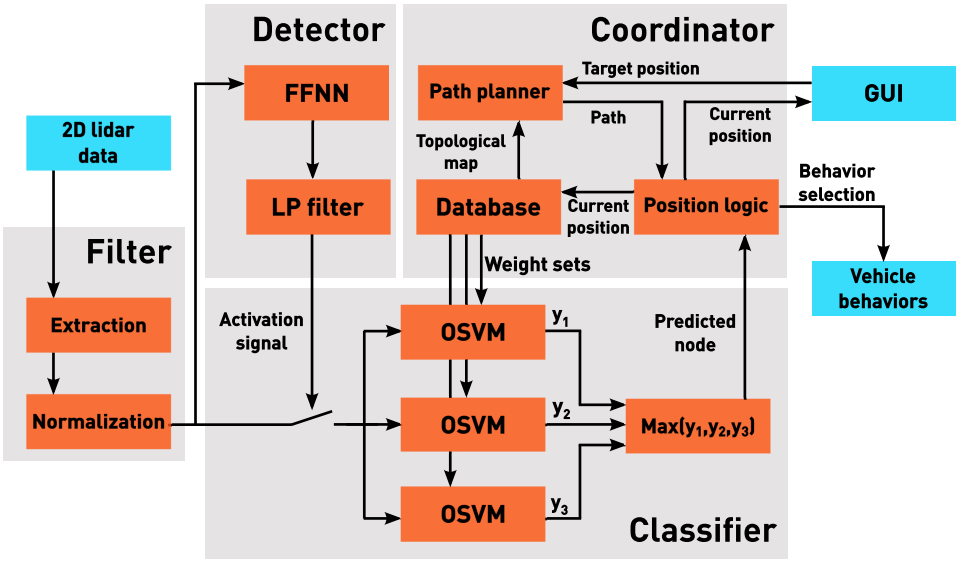


Figure 5.1: *An abstractified illustration of the hybrid map's structure.*

A filter ROS node first performs pre-treatment of the lidar sensor data. The data is first treated as explained in Section 4.3 and thereafter normalized to be in range $[-1, 1]$. The normalization is performed in order to have a standardized input.

The detector ROS node takes one pre-treated lidar scan and activates the neural network with the given data. The output of the detector is a binary activation signal and is constantly fed to the classifier. A fundamental idea behind the implementation is to only have the classification active when the detector indicates the vehicle is located in a node. This allows the classification class definitions presented in Section 3.2.4 but also reduces computational complexity.

The output of the classification is then used to update the position of the vehicle in the topological map. Depending on the position of the vehicle, and additional information, the behavior of the vehicle can be selected. A ROS node, named *Coordinator*, contains the database and handles most of the data communication between the different blocks. It also keeps track of the current position and handles the path planning using Dijkstra's algorithm [Dij59].

5.1 Topological map

The map is stored using *SQLite*, which is a light-weight, serverless, Structured Query Language (SQL) database engine. This database engine was chosen because of these given properties and as it is well documented. The engine is written in C but both python and C++ interfaces are available for integration with ROS.

In the database, relations between places (e.g. nodes and edges) are stored together with unique information about every place. Three tables are used: Edge, Node and Turn. The structure is designed with flexibility in mind. This means that extending the map with additional information is simple such that possible developments of the confined area can easily be added. For example, new nodes and edges can be added under table Node and Edge. In the Edge table, the length of every edge are stored in meter. This information is used by the path planner when computing the shortest route to target destination. The database structure is illustrated with a entity diagram in Figure 5.2.

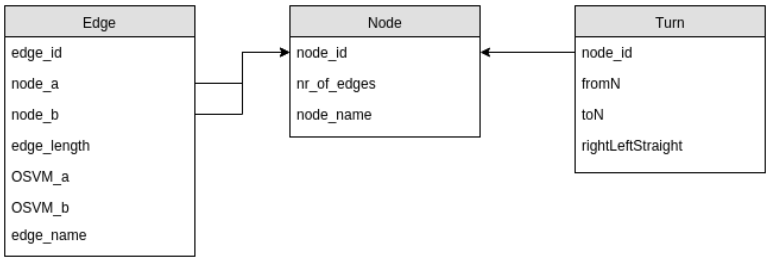


Figure 5.2: *Entity diagram of the SQLite database. The edge table holds information about the edges and relations between nodes. The turn table holds navigation information for the vehicle behaviors.*

The weight sets for node detection and classification are also stored in the database. This way of implementation allows a modular structure where every node has individual weight sets. This structure also allows the topological map to be expanded. These weight sets are stored as binary large objects (BLOBs) in the database, which is a way to store large collection of binary data.

5.2 The coordinator node

As mentioned, a ROS node named *Coordinator* handles most of the data communication and contains the SQL database. The node especially contains logic that computes the next set of possible nodes, depending on current position, and updates the node classifier with new weight sets which are associated with the nodes. The path planner, implemented using Dijkstra's algorithm, is able to compute the shortest path between two nodes [Dij59]. The length of every edge is fetched from the SQL database.

This node also handles the communication with the graphical user interface which allows the user to set a destination node and monitor the current position of the vehicle in the topological map.

5.3 Node detection

The ROS node performing the node detection is a simple implementation. The detector node takes one pre-treated lidar scan and activates the neural network model with the given data. The output, which is a prediction certainty, is then low-pass filtered in order to make the prediction more stable and robust to outliers.

The filter is of simple moving average (SMA) type as it is the mean of the n latest outputs:

$$y_t = \frac{x_t + x_{t-1} + \cdots + x_{t-(n-1)}}{n} \tag{5.1}$$

where x is the output prediction certainty of the neural network. The filtered output is then compared to a threshold value th. Below the threshold, the node outputs a zero for not being in a node. Above or equal, it outputs a one for being in a node:

$$\text{Node detection output } = \begin{cases} 1, & \text{if } y_t \geq th \\ 0, & \text{if } y_t < th \end{cases}$$

5.4 Node classification

The classifier ROS node has two states, activated or deactivated. When activated, classification is performed. The switching between these two states is controlled by the activation signal from the detector node.

When the classifier is deactivated, new weight sets are loaded in to the OSVMs from the database. These weight sets corresponds to the number of possible next nodes. When the detector then indicates a node, the classifier is turned active and classification is performed for as long as the activation signal is high. As soon as the activation signal goes low, the result from the OSVMs are computed and the maximum likely node is outputted as the predicted node. Based on this output, the coordinator ROS node updates the current position in the topological map.

6 Results and discussion

This chapter is divided into two parts. The first section evaluates the different machine learning methods that are considered for use in the proposed hybrid map. The second section evaluates the proposed hybrid map as a whole system. Both its localization ability and computational complexity is evaluated.

6.1 Evaluation of machine learning methods

The machine learning methods are evaluated mainly in terms of their ability to make correct classifications as the robustness of the system is of highest priority. The results will be visualised through confusion matrices, with additional information about the accuracy, as they provide a clear view of the performance.

The methods are also evaluated based on their memory usage and computational complexity as these are important factors in an industrial application. Also, the amount of time required for training is taken into consideration as the system should be able to adapt to the confined area when it expands.

6.1.1 Node detector

The experiment is conducted by traversing in the simulated world starting at E_1 taking one lap counter clockwise before ending up in N_1. Figure 4.2 illustrates the placement of nodes in the simulated world. While traversing, negative samples in edges and positive samples in the nodes are collected. This same particular run has been used for all different methods in the experiment, such that they can be compared fairly. The results from the experiments are displayed as confusion matrices. Table 6.1 shows the results from using data set D_1.

Table 6.1: Confusion matrices for all experiments specified in Table 3.1 with data set D_1. A total of 15664 sample sets were used for training and a total of 1159 test sample sets were used for validation.

	SVM (SGD)		SVM (ASGD)		Perceptron (SGD)	
	Actual node	Actual edge	Actual node	Actual edge	Actual node	Actual edge
Predicted node	548	2	542	8	550	0
Predicted edge	7	602	11	598	94	515
Accuracy	0.992235		0.983607		0.918896	
	1HL-FFNN (GD)		2HL-FFNN (GD)		1HL-FFNN (Adam)	
Predicted node	411	139	399	151	550	0
Predicted edge	53	556	33	576	17	592
Accuracy	0.83434		0.841242		0.985332	

By looking at Table 6.1 it can be seen that the support vector machines has the best performance while the gradient descent trained feed forward neural networks has the worst, in terms of accuracy. The fact that the SVMs has such good results is an indication that the data used in the training process is linearly separable. Seeing that the Perceptron has better results than the hidden layered FFNNs (GD) indicates that it is a simple classification problem. It can also be seen that the training methods used for the neural networks has large impact on the results. Using the Adam training method is favourable as it has by far best performance between the FFNNs. Since the SGD and ASGD training methods are both stochastic, their results vary between training runs which can be seen in Figure 6.1.

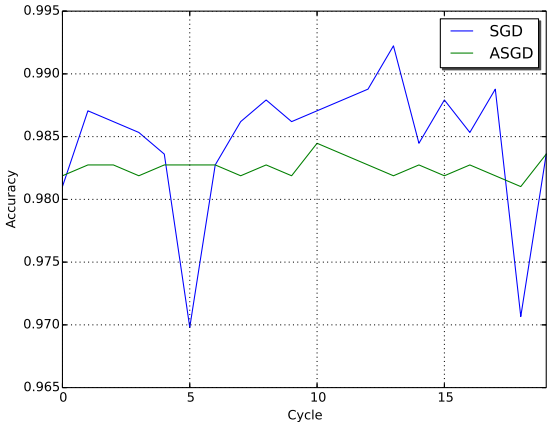
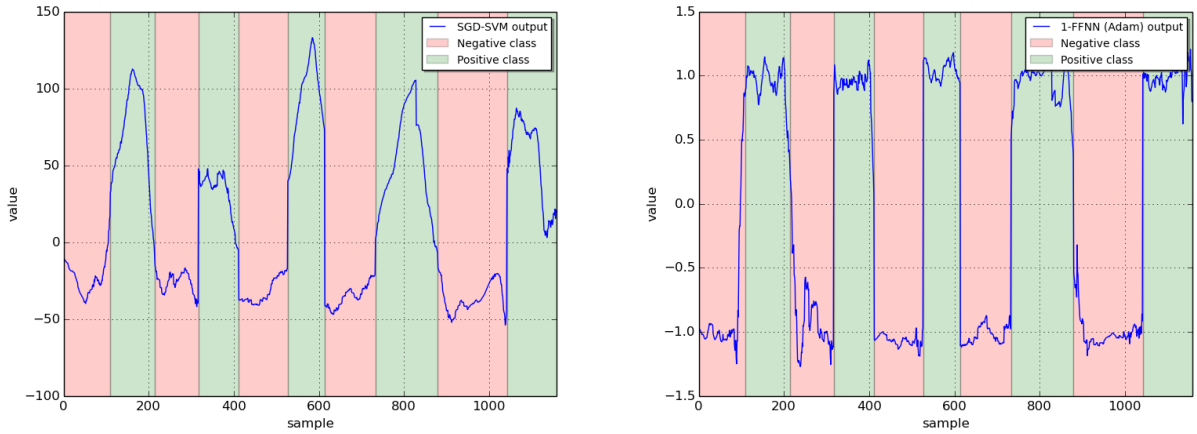


Figure 6.1: *An illustration of the stochasticity in the SGD and ASGD process. The accuracy of the ASGD method is more stable between training runs.*

As mentioned in Section 2.4.1, the ASGD method performs an average in its training. This means that the SGD trained SVM needs to be retrained a few times in order to find a better accuracy while the accuracy of the ASGD is more stable. The overall performance of the SGD and 1HL-FFNN Adam trained models and when they make their respective faulty classifications can be seen in Figure 6.2. The same test data as that has been used to generate Table 6.1 has been used to generate the figures.



(a) *SVM (SGD)*

(b) *1HL-FFNN (Adam)*

Figure 6.2: *Two methods, trained with data set D_1, evaluated with the test scenario. The green and red areas represents the true positive and negative labels respectively.*

Figure 6.2 is meant to compare the differences between two of the methods, the best performing SVM and the best performing FFNN. This is interesting because there are no clear rules in the literature when to use either one. Although, the FFNN is preferable over the linear SVM if the problem is more complex since the FFNN are not limited to linear problems. For the SVM, the y-axis represents a measure of prediction probability, higher positive or negative values means higher certainty. For the FFNN, it strives to classify the input data as one or minus one. In this particular scenario, it is hard to determine which method that has the best performance. The only measure would be to look at the accuracy depicted in 6.1. The figures are interesting in terms of seeing where the respective methods does actually performs a bad classification. As can be seen, no faulty classifications are done in the midst of a node or an edge. Instead, the methods misses in the transitions between the two, which actually is not a big problem. By looking at, for example, gradient descent trained the 2HL-FFNN in Figure 6.3, the reason behind its poor accuracy is clearly depicted.

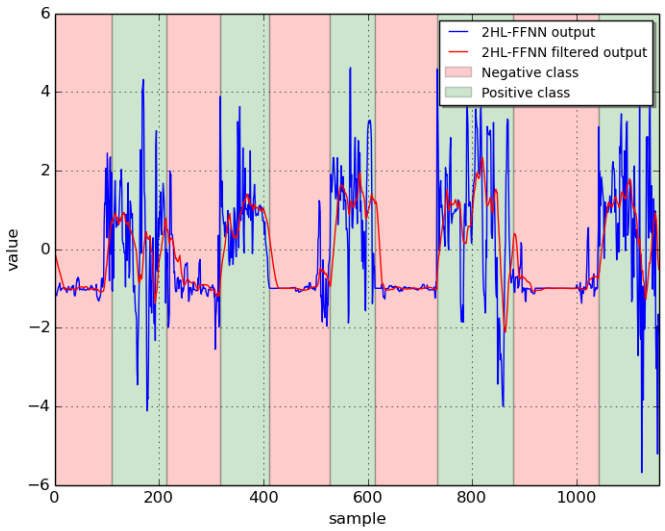


Figure 6.3: *The result of the 2HL-FFNN (GD) method depicted over time. Faulty classifications are performed at positive labeled data.*

It can be seen that the neural network in Figure 6.3 constantly performs poorly when doing classification of positive data. While traversing within the world model, large changes in the sensor scans are being done in nodes in terms of turning while traversing though an edge is very similar each time. This means that the test data differs much while in a node, while it is resembling in an edge. This is clearly shown in the figure. One of the reasons why this particular training provides worse results than the other methods is because no shuffling of the data has been performed while training. This means that the FFNN might have learned the pattern in the training data which does not exactly correspond to the same scenario used for testing the method. This is of course not the desired behavior. Shuffling the data such that the order is more or less completely random is important when working with neural networks [CC98]. This aspect is something that does not affect the support vector machines since it simply is a problem of separating the data. The library used for the Adam method shuffles the data in advance.

Considering the performance of the simple Perceptron, it is unlikely that the bad performance is a result of underfitting. If this where the case, the commonly known solution would be to add more hidden layers or neurons. However, it could be a result of that there is not enough data such that overfitting is one reason behind the bad results. In this case, adding more data or reducing the amount of hidden layers or removing neurons could solve the problem. Another way to avoid overfitting is to use regularization techniques [GJP95].

The fact remains that the best results are achieved from using the SVM (SGD). A likely reason for the superior performance of the linear classifiers could simply derive from the character of the world which the data was recorded in. If the data labelled as positive is distinctly different from the negative labelled data it might be easily separable in a linear fashion. This said, it is yet unknown if the linear classifier would outperform the non-linear FFNN in all types of confined areas.

In order to evaluate if less complex data is beneficial to the different methods, the same tests is performed with data set D_2 which uses 180 degrees field of view in front of the vehicle. The results can be seen in Table 6.2.

Table 6.2: Confusion matrices for all experiments specified in Table 3.1 with data set D_2. A total of 10345 sample sets were used for training and a total of 897 test sample sets were used for validation.

	SVM (SGD)		SVM (ASGD)		Perceptron (SGD)	
	Actual node	Actual edge	Actual node	Actual edge	Actual node	Actual edge
Predicted node	445	8	449	4	419	34
Predicted edge	20	424	26	418	15	429
Accuracy	0.968785		0.966555		0.945373	
	1HL-FFNN (GD)		2HL-FFNN (GD)		1HL-FFNN (Adam)	
Predicted node	356	97	399	54	450	3
Predicted edge	19	425	29	415	14	430
Accuracy	0.87068		0.907469		0.981048	

These results resembles the results in Table 6.1 as the well performing methods has similar results. However, a few slight differences can be observed. The accuracy of the gradient descent trained FFNNs has increased by approximately five percent. They seems to benefit from being trained with less data. This can be the result of several things. Reducing the field of view reduces the number of inputs to the networks. Since the 2/3 thumb of rule (explained in Section 2.4.2) still applies, the size of the hidden layers are also reduced, resulting in overall less complex models. The networks in Table 6.1 and 6.2 are trained using the same amount of cycles, meaning that the models have been treated differently with respect to their complexity. This could be one of the reasons why better results are achieved. Similar results of the networks might have been yielded in Table 6.1 if they would have been trained during more cycles.

The accuracy of the two SVMs has become slightly worse. Less training data should be beneficial to the SVMs as the decision boundary between the two classes is more easily determined. At the same time, less data yields a simpler model that might cover less scenarios in the test data. It is difficult to know exactly why the results differ from using D_1 and D_2. Since the data has been collected manually, it might as well be a results from the human factor. When manually labeling during data collection, it is difficult to determine the exact transitions between the nodes and edges. Inconsistency of this will result in models that gives noisy classifications when transitioning to or from a node. The neural networks can learn to handle badly labeled transitions which is a powerful attribute in terms of, to some extent neglecting the human factor.

Even though similar results are achieved with the two data sets, D_1 is seen as the preferred one. This is because of the fact that information of when a node is left is advantageous. The hybrid map is to provide the navigation system with behaviours that it selects based on its current position and the next upcoming one. Thus, a field of view that covers more ground is more suitable.

6.1.2 Node classifier

The results from the experiment can be seen in Table 6.3 and 6.4. The test scenario consisted of traversing through the one node the models has been trained for and another foreign node.

Table 6.3: Confusion matrices for all experiments specified in Table 3.2 with data set D_3. A total of 2353 sample sets were used for training and a total of 169 test sample sets were used for validation.

	SVM (SGD)		SVM (ASGD)	
	Actual positive	Actual negative	Actual positive	Actual negative
Predicted positive	76	0	76	0
Predicted negative	93	0	93	0
Accuracy	0.449704		0.449704	
	Perceptron (SGD)		OSVM (unsupervised)	
Predicted positive	72	4	61	15
Predicted negative	93	0	51	42
Accuracy	0.426036		0.609467	

Looking at the Table 6.3, it can be seen that the SGD, ASGD, and Perceptron performs horribly. This is a clear indication that the character of the data is not linearly separable such that the line that is to separate negative and positive data is displaced. Since the OSVM is a non-linear classifier, it can be seen to perform a bit better. Although it is not well enough in order to take it into practice. Using data set D_3, the classifiers are

trained such that they should be able to classify a node from all possible entrances and exits. Trying to make this data linearly separable is clearly not possible, while a non-linear classifier seems more appropriate. This theorem is investigated further using data set D_4 which consists of less complex data for one node. The results can be seen in Table 6.4.

Table 6.4: Confusion matrices for all experiments specified in Table 3.2 with data set D_4. A total of 2353 sample sets were used for training and a total of 169 test sample sets were used for validation.

	SVM (SGD)		SVM (ASGD)	
	Actual positive	Actual negative	Actual positive	Actual negative
Predicted positive	68	8	72	4
Predicted negative	76	17	83	10
Accuracy	0.502959		0.449704	
	Perceptron (SGD)		OSVM (unsupervised)	
Predicted positive	75	1	66	10
Predicted negative	90	3	0	93
Accuracy	0.461538		0.940828	

As mentioned, data set D_4 consists of traversing through a node in one specific direction, which means that the data is less complex to the classifiers. Although looking at Table 6.4, the linear classifiers still finds it hard to classify. To them, the difference between the node of which they have been trained for and a foreign node is indistinguishable.

The OSVM benefits from reducing the complexity of the data and it has a satisfying accuracy. Its performance can be seen in Figure 6.4.

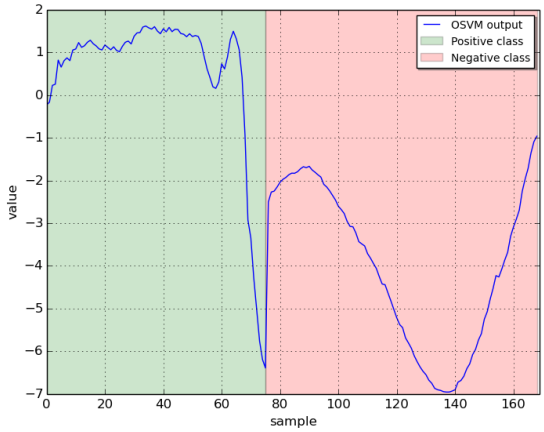


Figure 6.4: *The OSVM, trained with D_4 training data, activated with the same test data used for the confusion matrices above. The y-axis represents a measure of prediction certainty. The green area represents positive data (the node which the OSVM has been trained for) and the red negative data (another node). Note that the OSVM is not trained to recognize the negative data.*

In Figure 6.4 is the one-class SVM tested with the same test data used in the experiments above. Again, this classifier is only trained with data collected from one entrance in one node. The negative data (in red) is collected from another node, which the classifier is able to distinguish as the value is below zero. An interesting observation is the higher certainty of the classifier when the vehicle is in the middle of the node. This is a indicator that the features of a node is the most distinguishable in the center of the node. At an entrance or exit of a node, they are more similar to each other. This can also be observed in the fall of certainty-value between samples 65-75. When the vehicle is leaving the node, the features of the node is harder to classify.

All the methods that have been evaluated has been based on an approximately equal amount of negative and positive data which seemed as an appropriate approach. Since some methods that is evaluated uses a two class classifier, it was interpreted that these models should be built up from as much data from nodes as for edges. In the world models where data is collected, the character of edges are to some extent uniform

while the nodes can be seen as complex polygons that differ much in appearance from node to node. What is meant by this is that in order to make the corresponding systems work better it might be appropriate to train the models using more data from nodes than edges. This is something that has not been evaluated or taken into account when training the different systems but might have influence on the results. It should also be mentioned that the models have been trained with small amounts of data, especially the OSVM classifiers. Getting these results with such a small amount of data is a good indication that they can get a lot better if more effort would be put into the training process.

6.2 Evaluation of the hybrid map

Several aspects are taken into consideration when the hybrid map is evaluated. Primarily is the map's ability to position itself in the topological map. As mention in the introductory chapter 1, the problem should be solved in a simplistic fashion such that heavy computations are avoided. Therefore, other aspects in relation to the CPU and memory usage is to be evaluated.

Two experiments are carried out. The first experiment evaluates the global localization ability of the hybrid map and the second experiment evaluates the CPU and memory usage.

6.2.1 Global localization performance

The hybrids map's ability to do correct localization is evaluated by performing an experiment focusing on the node detection and classification. In the world model, the vehicle is manually driven in a counter clockwise defined path as:

$$\text{E1} \rightarrow \text{N1} \rightarrow \text{E5} \rightarrow \text{N4} \rightarrow \text{E4} \rightarrow \text{N3} \rightarrow \text{E3} \rightarrow \text{N2} \rightarrow \text{E2} \rightarrow \text{N1}$$

The node and edge labels are explained in Figure 4.2. As can be seen, when the vehicle is positioned in Edge 1 (E1) there is only one neighboring node. The weight set for that particular entrance to the node is therefore loaded into the classifier from the database. As soon as the node detector indicates that the vehicle is now located in a node, the classifier is activated and a classification is performed. When the node detector indicates that the vehicle has left the node, the classifier is deactivated and new weight sets for next possible nodes are loaded. This process is repeated for as long as the hybrid map is operating. The results from the experiment can be seen in Figure 6.5.

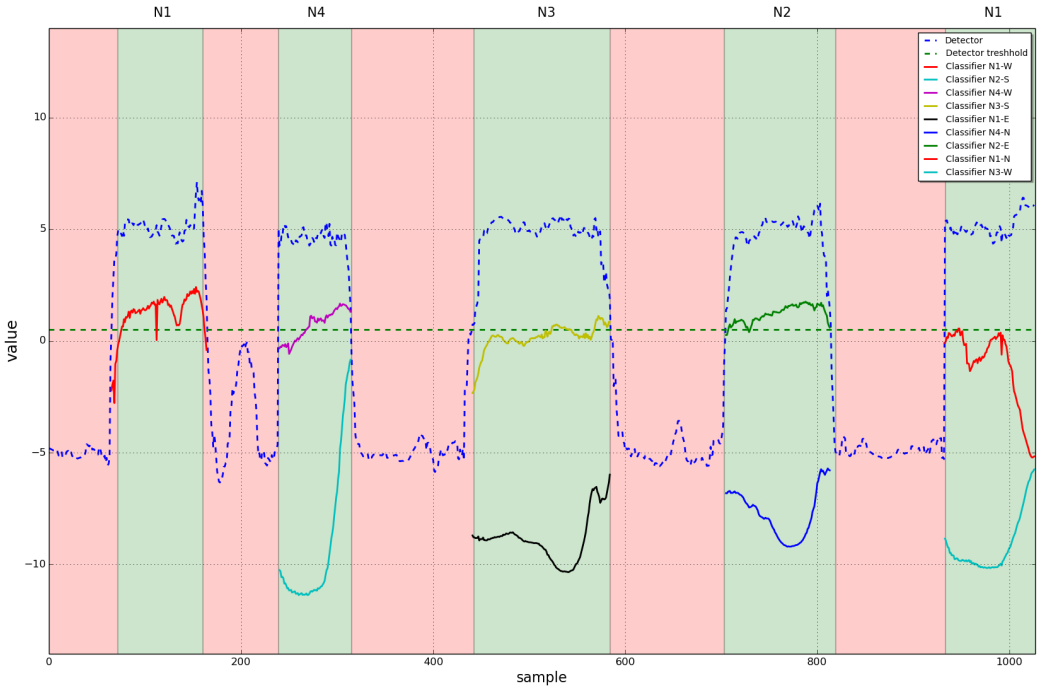


Figure 6.5: *A test of the node classifier and node detector by letting the vehicle travel through the world model in a predefined path. The dashed blue line is the output of the detector and the constant dashed green line is the detector-threshold, using value* $th = 0.1$. *For visualization purposes, the detector output and the threshold is magnified by a factor 5. Whenever the detector indicates a node, the classifier is activated. The classifier classifies the node by running the OSVMs that corresponds to the next possible set of nodes and computes the most likely node. The green and red field are the labels of the test data and can be seen as the ground truth. The text labels above the plot shows what node the green field corresponds to. "N1" is Node 1, as shown in Figure 4.2.*

Looking at Figure 6.5, it can be seen that the detector performs well in terms of node detection. The instances where faulty classifications are occurring is when the vehicle transitions between nodes and edges. As mentioned, this is most likely due to the human factor - when the training and testing data is recorded. This is, however, a minor issue as long as the classifier has enough time to make the correct classification.

Further, the classifier is able to classify all nodes correctly in the experiment. As can be seen in Figure 6.5, the difference in certainty between the OSVMs is, at all times, large such that the classifier has negligible difficulties performing correct classifications.

Even if this experiment indicates good localization performance of the hybrid map, it does not prove the map's ability to perform localization in all types of confined areas. As of now, the map is designed with underground mines in mind where there is a clear difference in character between nodes and edges. The proposed solution would most likely perform worse in large open spaces such as open pit mines.

Comparing this system's global localization ability to other existing similar methods in the literature is not easy. There are no recognized methods for this type of evaluation and the preconditions among the existing localization methods differs with the application area. To our knowledge, this type of approach for solving the topological localization problem is novel.

6.2.2 CPU and memory usage

While running the complete system on the RP3, an experiment to measure the computational complexity is carried out. Each running ROS node is measured separately for average CPU and memory usage. The results can be seen in Figure 6.6. Note that these result is the averaged CPU and memory usage during the whole time interval that these processes were running.

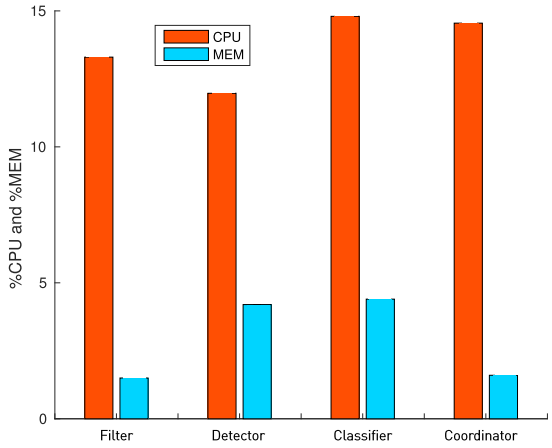


Figure 6.6: *Average CPU and memory usage expressed in percentage for the four ROS nodes shown in Figure 5.1.*

As can be seen, all ROS nodes uses between 12 and 15 percent CPU time each. Summed together they use, in average, 55 percent of the RP3 CPU time. Summed with all background processes not related with the hybrid map, the total CPU usage was at no time above 70 percent.

The memory usage of each ROS node differs. The Detector and Classifier nodes use each, in average, 4,4 percent of the total available memory of 1GB. This is most likely due to the SVMs are contained in these nodes. The other two ROS nodes, Filter and Coordinator, use 1.5 percent of the total memory each. The total memory usage of the hybrid map is 11.7 percent. The total size of the hybrid map on the secondary storage is 2.2 MB.

The RP3 is able to run the system without any complications. With further improvements of the implementation, it is believed that these results would be even better. However, the experiment did not cover the full hybrid map as the vehicle behaviors are excluded as of now. When new nodes and edges are added to the topological map, it is important to note that the CPU and memory usage would stay unaffected as only a subset of all OSVMs are loaded into memory at all times. However, the size of the hybrid map on secondary storage would grow.

7 Conclusions and Future Work

The final solution follows the initial proposal. It consist of a hybrid map that uses the basic features of a topological map which can store behaviour algorithms used for local navigation. These behaviours gets selected based on the location of the vehicle. The hybrid map also contains a neural network weight set used in the detection of nodes and edges as well as support vector machines that classifies nodes. The solution is flexible and modular since more OSVMs can be added to the hybrid map as the confined area grows and the detector is trained in such a way it will detect all types of nodes. It is also shown by running the system on a Raspberry Pi 3 that the solution is of a simple fashion in terms of computation.

It is shown that using a full field of view lidar setup (D_1) gave the best results in terms of accuracy for both the detection of nodes and edges and classification of nodes. The SGD trained SVM turned out to have the highest performance closely followed by the Adam trained neural network. Because of the discussed limitations of an SVM, the neural network is chosen as the preferred method. When it comes to classifying nodes, all methods except the one class SVM had poor performance. The OSVM is also very easy to handle since it only requires positive training data (D_4). It is shown that it is beneficial to have several OSVMs, one for every entrance of a node rather than having one OSVM that covers all entrances.

This thesis work shows that features of a simulated underground mining system is classifiable using 2D lidar data, neural networks and support vector machines. To our knowledge, the research regarding this matter is limited. The work also shows that it is possible to use relative localization using a topological map as opposed to an absolute metric map. This approach is not new in the literature, however, the technique used for node and edge transition is novel.

As of now, no effort has been put into optimization of the code. This has, of course, a negative impact on the CPU and RAM usage performance. With further improvements of the code, an even cheaper computer could have been used. Porting the code to C/C++ would most likely provide a performance improvement.

In order to guarantee a robust localization ability of the system, the system needs to be tested on noisy lidar data. This would probably not be a time consuming evaluation as it is easy to apply Gaussian noise to the lidar data using Gazebo. The system should also be verified in different world models before use in an actual application. It is clear that the world model that has been used for training and evaluation has specific characteristics that does not resemble other application areas.

References

[16a] *LMS111-10100 2D Laser Scanners*. LMS111. SICK Sensor Intelligence. Mar. 2016.

[16b] *Raspberry Pi 3 Model B*. RASPBERRY PI FOUNDATION. Feb. 2016.

[Ben07] I. Ben-Gal. Bayesian networks. *Encyclopedia of statistics in quality and reliability* (2007).

[BFS93] Y. Bengio, P. Frasconi, and P. Simard. "The problem of learning long-term dependencies in recurrent networks". *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 1183–1188.

[BGV92] B. E. Boser, I. M. Guyon, and V. N. Vapnik. "A Training Algorithm for Optimal Margin Classifiers". *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT '92. Pittsburgh, Pennsylvania, USA: ACM, 1992, pp. 144–152. ISBN: 0-89791-497-X. DOI: 10.1145/130385.130401. URL: http://doi.acm.org/10.1145/130385.130401.

[BHW04] D. Bastow, G. Howard, and J. P. Whitehead. *Car suspension and handling*. SAE international Warrendale, 2004.

[Bic+04] A. Bicchi et al. "On the problem of simultaneous localization, map building, and servoing of autonomous vehicles". *Advances in control of articulated and mobile robots*. Springer, 2004, pp. 223–242.

[BIS07] M. Buehler, K. Iagnemma, and S. Singh. *The 2005 DARPA grand challenge: the great robot race*. Vol. 36. Springer Science & Business Media, 2007.

[Boj+16] M. Bojarski et al. End to End Learning for Self-Driving Cars. *arXiv preprint arXiv:1604.07316* (2016).

[Bot12] L. Bottou. "Stochastic gradient descent tricks". *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 421–436.

[Bus05] P. Buschka. An investigation of hybrid maps for mobile robots (2005).

[CC98] C. Campbell and N. Cristianini. *Simple training algorithms for support vector machines*. Tech. rep. Technical Report, Bristol University, http:== lara. enm. bris. ac. uk= cig, 1998.

[Cha13] M. Chafkin. Udacity's Sebastian Thrun, godfather of free online education, changes course. *Fast Company* **14** (2013).

[Cho+02] C.-H. Choi et al. "Topological map building based on thinning and its application to localization". *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Vol. 1. IEEE. 2002, pp. 552–557.

[Cho05] H. M. Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[CL02] C.-C. Chang and C.-b. Lin. Training v-support vector regression: theory and algorithms. *Neural Computation* **14**.8 (2002), 1959–1977.

[CMM83] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. "An overview of machine learning". *Machine learning*. Springer, 1983, pp. 3–23.

[Cox58] D. R. Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)* (1958), 215–242.

[Cra07] A. P. Cracknell. *Introduction to remote sensing*. CRC press, 2007.

[Dic07] E. Dickmanns. *Dynamic Vision for Perception and Control of Motion*. Springer, 2007.

[Dij59] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik* **1**.1 (1959), 269–271.

[Dis+01] M. Dissanayake et al. A solution to the simultaneous localization and map building (SLAM) problem. *Robotics and Automation, IEEE Transactions on* **17**.3 (2001), 229–241.

[DMS02] T. Duckett, S. Marsland, and J. Shapiro. Fast, on-line learning of globally consistent maps. *Autonomous Robots* **12**.3 (2002), 287–300.

[Dom12] P. Domingos. A few useful things to know about machine learning. *Communications of the ACM* **55**.10 (2012), 78–87.

[G M14] S. B. G. Meyer. *Road Vehicle Automation*. Springer, 2014.

[GJP95] F. Girosi, M. B. Jones, and T. Poggio. Regularization theory and neural networks architectures. *Neural computation* **7**.2 (1995), 219–269.

[Gra+09] A. Graves et al. A novel connectionist system for unconstrained handwriting recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **31**.5 (2009), 855–868.

[HN04] S. Haykin and N. Network. A comprehensive foundation. *Neural Networks* **2**.2004 (2004).

[HS97] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation* **9**.8 (1997), 1735–1780.

[Kar12] S. Karsoliya. Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture. *International Journal of Engineering Trends and Technology* **3**.6 (2012), 713–717.

[KB14] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[KF15] A. Karpathy and L. Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 3128–3137.

[KH04] N. Koenig and A. Howard. "Design and use paradigms for gazebo, an open-source multi-robot simulator". *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2004, pp. 2149–2154.

[Koš+09] K. Košnar et al. "LaMa-Large Maps Framework". *Proceedings of Workshop on Field Robotics, Civilian-European Robot Trial*. 2009, pp. 9–16.

[Kui+04] B. Kuipers et al. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. **5** (2004), 4845–4851 Vol.5. ISSN: 1050-4729. DOI: 10.1109/ROBOT.2004.1302485.

[Kui00] B. Kuipers. The spatial semantic hierarchy. *Artificial intelligence* **119**.1 (2000), 191–233.

[Kui78] B. Kuipers. Modeling spatial knowledge. *Cognitive science* **2**.2 (1978), 129–153.

[Men09] A. K. Menon. Large-scale support vector machines: algorithms and theory. *Research Exam, University of California, San Diego* (2009), 1–17.

[Mia+07] M. S. Miah et al. Autonomous Dead-Reckoning Mobile Robot Navigation System With Intelligent Precision Calibration (2007), 1–5. ISSN: 1091-5281.

[Mil10] D. Miljković. Review of novelty detection methods (2010), 593–598.

[MKH06] M. Modsching, R. Kramer, and K. ten Hagen. Field trial on GPS Accuracy in a medium size city: The influence of built-up (2006), 209–218.

[MP] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**.4 (), 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259.

[PM10] J. Parreira and J. Meech. Autonomous vs. manual haulage Trucks–How mine simulation contributes to future haulage system developments. **3** (2010).

[Qin+12] B. Qin et al. Curb-intersection feature based Monte Carlo Localization on urban roads (2012), 2640–2646. ISSN: 1050-4729. DOI: 10.1109/ICRA.2012.6224913.

[Qui+09] M. Quigley et al. "ROS: an open-source Robot Operating System". *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5.

[Rao10] S. Rao. "A COMPOSITE APPROACH TO DEAL WITH LOCALIZATION PROBLEMS IN WIRELESS SENSOR NETWORK". PhD thesis. Blekinge Institute of Technology, 2010.

[Rel12] I. N. Releases. *Intelligent Transportation and Infrastructure Resources*. 2012. URL: http://www.ieee.org/about/news/2012/5september_2_2012.html (visited on 03/08/2016).

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Cognitive modeling* **5**.3 (1988), 1.

[Ros57] F. Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[Sam59] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development* **3**.3 (1959), 210–229.

[San15] D. Sanchez. Collective technologies: autonomous vehicles (2015).

[Sch+01] B. Schölkopf et al. Estimating the support of a high-dimensional distribution. *Neural computation* **13**.7 (2001), 1443–1471.

[Sen26] T. M. Sentinel. *"Phantom Auto, Will tour city*. 1926. URL: news.google.com/newspapers?id=unBQAAAAIBAJ&sjid=QQ8EAAAAIBAJ&pg=7304,3766749&hl=en (visited on 03/08/2016).

[Sim13] P. Simon. *Too Big to Ignore: The Business Case for Big Data*. Wiley, 2013.

[SLK05] K.-S. Shin, T. S. Lee, and H.-j. Kim. An application of support vector machines in bankruptcy prediction model. *Expert Systems with Applications* **28**.1 (2005), 127–135.

[SNS11] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[SSC87] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. **4** (1987), 850–850. DOI: 10.1109/ROBOT.1987.1087846.

[TB96] S. Thrun and A. Bücken. Integrating grid-based and topological maps for mobile robot navigation (1996), 944–951.

[TSK06] P.-N. Tan, M. Steinbach, and V. Kumar. Classification: basic concepts, decision trees and model evaluation. *Introduction to data mining* **1** (2006), 145–205.

[UN00] I. Ulrich and I. Nourbakhsh. "Appearance-based place recognition for topological localization". *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on.* Vol. 2. Ieee. 2000, pp. 1023–1029.

[Vie+15] R. Viereckl et al. Racing Ahead with Autonomous Cars and Digital Innovation. *Auto Tech Review* **4**.12 (2015), 18–23.

[Wah08] M. Wahde. *Biologically inspired optimization methods: an introduction.* WIT press, 2008.

[Xu11] W. Xu. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *arXiv preprint arXiv:1107.2490* (2011).