Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master Thesis

Deep Learning Based Mutual Robot Detection and Relative Position Estimation

Robin Dietrich

Course of Study:	Informatik
Examiner:	Ph.D. Daniel Hennes
Supervisor:	Dipl.-Ing. Stefan Dörr
Commenced:	December 15, 2017
Completed:	July 27, 2018

Abstract

Self-localization for mobile robots in dynamic environments is a challenging task, especially when relying solely on odometry and 2D LIDAR data. When operating in fleets, mutual detection and the exchange of position information can support each robots localization. Detecting and classifying different robot types in a heterogeneous fleet, however, is non-trivial if only 2D LIDAR data is used. Object shapes detected with this sensor in the real-world can vary heavily due to irregularities of the environment.

In this thesis a novel approach for mutual robot detection and relative position estimation based on a combination of convolutional and ConvLSTM layers is therefore presented in order to solve this issue. The algorithm learns an end-to-end classification of robot shapes using 2D LIDAR information transformed into a grid-map. Subsequently multiple hypotheses for each robot type are extracted out of the heatmap output of the network using a hierarchical clustering algorithm combined with a centroid calculation for each hypothesis. These position hypotheses can then be used in an overall multi-robot localization in order to increase the localization of the robots. Due to the similarities that many robot shapes in 2D LIDAR data share with static objects in the environment (rectangles, circles), an additional pre-processing of the data is performed. Three different end-to-end approaches for semi- and fully-dynamic object detection are introduced. The first one is using stacked laser-scans with a Convolutional Network in order to detect spatio-temporal features of moving objects. The second one transforms the sensor data into a 2D grid-map and accumulates multiple consecutive maps to create a map with spatio-temporal features for moving objects. This map is then used as an input for a ConvLSTM network. Both approaches only detect semi-dynamic objects, since the spatio-temporal features in both cases are only visible for robots moving at the currently viewed time-span. Another simple approach is presented for extracting fully-dynamic objects by calculating the difference between the static grid-map provided by the robots Localization & Mapping algorithm and the grid-map calculated from the laser-scan.

We conduct an in-depth evaluation in order to determine the networks' ability to generalize to different environments and cope with uncertainty in the input data. Furthermore the benefit of using simulation data for pre-training real-world models is evaluated together with the improvement of pre-processing the data by the dynamic object detection networks. The results of our evaluation show, that the classification network is able to achieve a precision of 94% on real-world data with a position estimation error of $13cm$. Pre-processing on the other hand does not improve the networks performance overall, although the two 2D approaches for dynamic object detection are able to identify most of the moving robots correctly. Using 1D scan data on the other hand does not lead to any promising results for dynamic object detection. In order to determine the benefit of the mutual robot detection and position estimation, the system will be integrated into a multi-robot localization subsequent to this thesis.

Kurzfassung

Die selbstständige Lokalisierung mobiler Roboter in dynamischen Umgebungen ist eine anspruchsvolle Aufgabe, speziell wenn lediglich Odometry und 2D LIDAR Daten verfügbar sind. In einer Flotte mobiler Roboter kann jedoch die gegenseitige Detektion sowie der Austausch von Positionsinformationen die Lokalisierung der einzelnen Roboter verbessern. Mobile Roboter in einer heterogenen Flotte zu erkennen und zu klassifizieren, ist jedoch nicht trivial, wenn nur ein 2D LIDAR Sensor verwendet wird. Die Form eines Objektes, welches von diesem Sensor erkannt wird, kann in der realen Welt aufgrund von Unregelmäßigkeiten in der Umgebung große Variationen aufweisen.

In dieser Thesis wird daher ein neuartiger Ansatz zur gegenseitigen Detektion von Robotern und relativer Positionsbestimmung basierend auf einer Kombination von convolutional und ConvLSTM Layern vorgestellt, um dieses Problem zu lösen. Der Algorithmus lernt eine Ende-zu-Ende Klassifizierung von Roboterformen unter der Verwendung von 2D LIDAR Daten, welche in eine Grid-Map transformiert werden. Anschließend werden mehrere Hypothesen für jeden Robotertyp aus dem Heatmap Output des Netzwerks mit Hilfe eines hierarchischen Clustering Algorithmus mit Zentroid Kalkulation extrahiert. Diese Hypothesen können anschließend in einer Multi-Roboter Lokalisierung verwendet werden. Aufgrund der Ähnlichkeit vieler Roboterformen mit typischen statischen Objekten (Rechteck, Kreis) werden die Daten zusätzlich noch vorverarbeitet. Drei verschiedene Ende-zu-Ende Ansätze zur semi- sowie voll-dynamischen Objekterkennung werden vorgestellt. Der erste Ansatz verwendet gestapelte 1D Laser-Scans mit einem Convolutional Network, um räumlich-zeitliche Feature von sich bewegenden Objekten zu erkennen. Der zweite transformiert die Sensordaten in eine 2D Grid-Map und akkumuliert mehrere aufeinanderfolgende Karten, um so zeitlich-räumliche Feature zu erzeugen. Diese akkumulierte Karte wird dann von einem ConvLSTM Netzwerk verarbeitet. Beide Ansätze sind lediglich fähig semi-dynamische Objekte zu erkennen, welche sich innerhalb des betrachteten Zeitrahmens bewegen. Ein weiterer Ansatz extrahiert hingegen alle dynamischen Objekte, indem es mit Hilfe eines Convolutional Networks die Differenz der statischen Karte des robotereigenen Langzeit-SLAMs und der dynamischen Karte der Sensordaten berechnet.

Wir führen eine tiefgehende Evaluierung durch, um die Fähigkeit der Netzwerke festzustellen, sich an verschiedene Umgebungen anzupassen und mit Unsicherheit in den Inputdaten umzugehen. Des Weiteren wird der Vorteil von mit Simulationsdaten vortrainierten Modellen, welche anschließend auf Realdaten trainiert werden zusammen mit der Verbesserung durch ein Vorverarbeiten der Daten evaluiert. Die Resultate der Evaluierung zeigen, dass das Klassifikations-Netzwerk eine Präzision von 94% auf Realdaten erreicht, mit einem durchschnittlichen Fehler in der Positionsbestimmung von $13cm$. Die Vorverarbeitung der Daten führt im Allgemeinen nicht zu einer Verbesserung der Leistung des Netwerks, obwohl die Netze zur dynamischen Objekterkennung die meisten dynamischen Objekte korrekt identifizieren können (80%). Die Verwendung von 1D Sensordaten führt wiederum zu keinen vielversprechenden Resultaten in der dynamischen Objekterkennung. Um die Verbesserung der gegenseitigen Robotererkennung und Positionsbestimmung zu bestimmen, wird das vorgestellte System im Anschluss an diese Arbeit in eine Multi-Roboter Lokalisierung eingebunden.

Contents

List of Figures

List of Tables

List of Abbreviations

AdaGrad adaptive gradient algorithm. 26

Adam adaptive moment estimation. 26

APC application center. 9

CLS classification. 9

CLS-RAY ray-tracing classification. 9

CNN convolutional neural network. 18

ConvLSTM Convolutional LSTM. 7

CTD continued. 9

DIF difference. 11

DYN-OBJ dynamic objects. 66

DYN-STACK stacked dynamic object detection. 11

DYN-MAP-LSTM map-based dynamic object detection. 11

DYN-LSTM dynamic object detection LSTM. 11

EKF extended kalman filter. 11

GRT ground-truth. 11

ICP iterative closest point. 35

LAB robotics laboratory. 9

LSTM long short-term memory. 7

LTS Long-Term SLAM. 16

MLP multilayer perceptron. 21

MR machine-room. 66

NEW newly trained. 11

REA real-world. 11

ReLu rectified linear activation function. 7

RMSProp root mean square propagation. 26

RNN recurrent neural network. 7

ROS Robot Operating System. 63

SGD stochastic gradient descent. 26

SIM simulation. 11

SLAM simultaneous localization and mapping. 16

STC static. 11

STC-ROB static robots. 66

STC-OBJ static objects. 66

SVM support vector machine. 35

WH warehouse. 66

1 Introduction

1.1 Motivation

Perceiving, modeling, and tracking an autonomous vehicle's environment is crucial for it to enable autonomous reasoning and navigation. Personal or home robots need to be aware of people or pets running around, cars must be aware of bicycles or pedestrians, and industrial mobile robots must be aware of workers and other robots or machines. Environmental perception allows autonomous vehicles to avoid collisions and plan an optimal path to a goal.

Many environmental perception algorithms focus on object detection and tracking in either indoor [AAA05], [LE01], [MTW02] or outdoor [MNM+13], [WPN15], [TL09] environments. They follow a common process including object detection, association (between consecutive scans), classification, and tracking [ODWP16]. Few of them will classify these objects, since for many use cases it is not important. In general it is of interest to know the position/orientation of an object, as well as its velocity and acceleration. There is a special case set in the world of industrial robotics. Numerous factories, e.g. the ones from automobile manufacturers, use many autonomous mobile robots for transportation tasks. Consider a scenario as shown in figure 1.1, where one robot (A, red) is driving through a hallway which is feature-poor [1], therefore its localization becomes progressively uncertain. However, another robot (B, blue) is located in a feature-rich room in front of robot A, hence is well-localized. If robot B could detect and classify robot A, they could exchange information to increase the accuracy of robot A's position estimation. Beyond this scenario the classification and tracking of dynamic objects also facilitates better local path planning and obstacle avoidance for the robots.

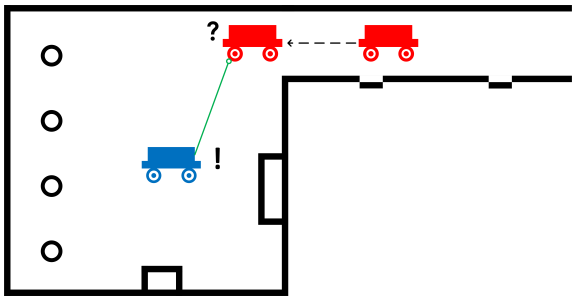


Figure 1.1: The problem, motivating this thesis. Robot A (red) is detected by robot B (blue), receives relative position information and improves its localization.

[1] Mobile robots use features, such as corners, circles, etc. to localize themselves within an environment. The more features, the better the localization. An empty room does not contain any features and therefore poses a challenge for the robots localization.

Most approaches found in the literature on multi-robot localization by incorporating mutual robot detection measurements don't focus on the detection part of the system [RB02], [MPS05], [FOS09]. They just assume that there is a robot detection module based on some exteroceptive[2] sensor which returns position estimates for each nearby robot. Since only few approaches actually deal with the robot detection, classification and position estimation itself [FBKT00], we present a novel mutual robot detection and relative position estimation approach for the introduced scenario.

1.2 Terminology

In the context of environmental perception for autonomous vehicles, some terms appear which describe similar actions, e.g. detection and classification. We therefore define the words as they will be used throughout this thesis:

Detection The process of detecting a (dynamic) object independent from its shape, and separating its boundary from the static environment.

Classification Determining one out of a fixed set of classes an object belongs to, e.g. robot/human/machine.

Tracking The (complete) dynamic state of the object[3] is tracked, possibly enabling a prediction of future states and/or positions of it.

Beyond these common terms, a few special ones used within this thesis are introduced in the following:

Ego vehicle/robot The vehicle/robot carrying the sensor used for data collection or object classification.

Ego motion The motion of the vehicle/robot carrying the sensor used for data collection or object classification.

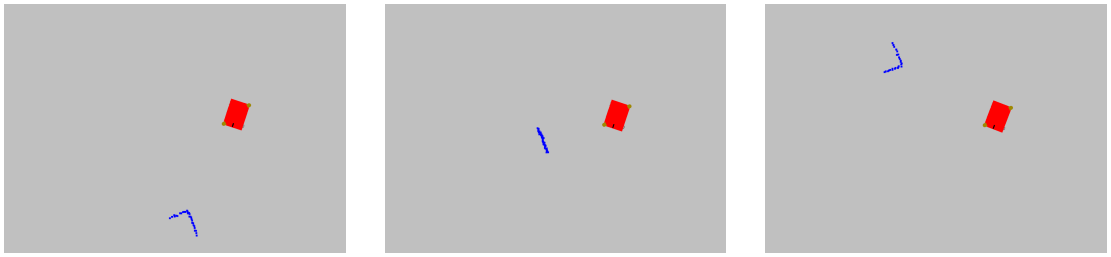
1.3 Problem Definition

The learning algorithm for robot detection and relative position estimation, presented in this thesis, will be integrated into a larger system for collaborative multi-robot localization called *cooperative Long-Term SLAM (LTS)*[4]. The purpose of this system is to enhance the localization in a multi-robot scenario, something that is commonly occurring in industrial applications. Each robot runs a local LTS and sends information of updated grid-map cells to a server. The server then distributes this information back to the robots depending on their location. When a robot e.g. enters a new room,

[2] Exteroceptive sensors are those measuring the environment around a robot (including LIDAR, SONAR, etc.). Proprioceptive sensors on the other hand measure the internal state of the robotic system, e.g. the battery level or wheel position.

[3] The dynamic state of an object in environmental perception commonly includes the objects position, orientation, velocity and acceleration.

[4] simultaneous localization and mapping (SLAM) defines the process of an autonomous system constructing a map of the environment while localizing itself in this probabilistic representation of its surroundings [TBF05].



(a) (b) (c)

Figure 1.2: A laser scans sequence (a-c) of a robot (red) capturing another robot (blue) using a 2D LIDAR scanner.

it receives map information regarding this room from the server. This information might include updated cells another robot has recently detected as being occupied. Therefore the server always maintains a global map of the environment including all static objects that have been placed there temporarily.

This approach improves the localization of the robots by providing more complete maps, including the latest information about the environment. Thus far, there is no module for mutual robot detection included in the system. Since we commonly operate a heterogeneous[5] fleet of robots it is necessary to not only detect other moving objects but also to classify them. We therefore intend to solve the problem of mutual robot detection, classification and relative positions estimation using only 2D LIDAR data.

Although 3D LIDAR scanners and high quality cameras have become more easily available and cheaper, the most frequently used sensor in mobile robotics is still a 2D LIDAR scanner. On one hand, this is due to safety requirements that only 2D LIDAR scanners[6] are able to fulfill. On the other hand, these sensors are extremely reliable and compared to newer, ”better” alternatives, cost efficient. However, they do not produce as detailed information about the environment as e.g. a 3D LIDAR, RGB-D or even monocular vision sensor. These would simplify the task of object detection and classification with the downside of producing much more data per time-step than a 2D LIDAR. The output of the latter can therefore be processed using less computational power. Due to these reasons 2D LIDAR sensors were used as the only information source for the algorithms presented in this thesis.

If detected in a 2D LIDAR scan, many robot shapes are ambiguous with regards to their orientation. A visualization of this issue is shown in figure 1.2. The series of consecutive laser scans shows one robot moving past another. If only a single scan is provided at each time step, there is no way of determining the robots exact orientation. In the second scan 1.2b for example, only the longer side of the robot is visible. Therefore it could be driving up or downwards. Due to these difficulties, there will be no orientation estimation of the target robot, since it wouldn't be possible to make a high quality assumption about its orientation. Another constraint of this system will be, that in

[5] A fleet of robots is defined as being heterogeneous in case it is composed of more than one type of robot. A fleet with only robots of the same kind is thus referred to as being homogeneous.

[6] There exist some 2D LIDAR scanners with a safety certificate acknowledging the fact, that they have an integrated emergency break mechanism which is a requirement for some robots in Germany.

contrast to many previous object detection approaches, the sensor will be mounted on a moving platform, i.e. a mobile robot. Hence, the algorithm needs to be able to deal with the ego motion of the sensor.

1.4 State-of-the-art

As elucidated in the section 1.1, there are not many approaches that explicitly solve the problem of mutual robot detection, classification and position estimation. There is on the other hand a wide range of approaches that tackle the problem of dynamic obstacle detection and classification, only a few of which are using 2D LIDAR sensors. Most of them are based on classic [XPC+05], [ZSKS06] or learning based [QCS+16], [PN06] algorithms. The challenge for using classical approaches is that a model for each class needs to be specified, which can be difficult due to the fact, that an objects shape can vary heavily between different measurements. Learning based approaches on the other hand rely in general on hand-selected features.

Within recent years, however, deep learning has produced impressive results in the field of image classification [HZRS15], [SWY+15], object detection [SEZ+13], [RHGS15] and semantic segmentation [BKC15], [SLD17]. For object detection and semantic segmentation, convolutional neural networks (CNNs) are used to classify not only the image as a whole but also per pixel. This pixel-wise classification leads to a probability distribution over all possible classes for each pixel in the input. Although the most popular input data for these kind of networks are RGB images, a few attempts exist to apply these algorithms to different problems. A number of approaches for RNN based detection, classification and tracking was developed by the Mobile Robotics Group at the University of Oxford [OP16], [ODWP16] and [DRO+16]. They all build upon each other while solving different problems. The first approach is able to detect and track objects in a simple simulated environment, even occluded ones. The second one applies a similar network to object detection, classification and tracking at an intersection. The network, however, learns a representation of the environment in a static memory and therefore doesn't generalize well to other environments. The most recent approach is the first to use their technique on a moving ego vehicle. They successfully detect and track dynamic objects but don't classify them.

These recent successes in using deep neural networks for classification and tracking of dynamic objects based on 2D LIDAR data are promising, yet, they do not provide a complete solution to the problem introduced in the previous section. None of them solves the problem of object classification from a moving ego vehicle using deep learning. Besides the partial successes in this field, the advantages of deep networks over traditional approaches are another reason for their application to this problem. In contrast to traditional approaches, a deep neural network can learn features to classify the shape of an object even when it is not exactly the same for each detection. The only requirement is a large amount of data for training the network, which can be provided by automated simulations.

1.5 Objective

The goal of this thesis is to develop a deep learning based classification of different object types (robots) as well as a relative position estimation to them, using solely 2D LIDAR data. It is not necessary to track the whole dynamic state of the object in order to e.g. predict its future trajectory, as done by many other approaches [AA08], [TL09]. Although some recent approaches used deep learning for object tracking in 2D LIDAR scans [OP16], [ODWP16], [DRO+16], to the best of our knowledge there is no approach using a deep learning based algorithm for this purpose. Especially not in a scenario where the sensor is mounted on a moving platform (robot). The output of the algorithm includes a list of robots within a certain range around the ego robot. This list contains information about each robots type, its relative position as well as an uncertainty measurement for each of them.

Throughout this thesis, there are two main questions we attempt to answer. First, how powerful is a 2D LIDAR scanner, especially in combination with deep learning? This specifically aims to answer, whether it is possible at all to detect and classify moving objects in a 2D LIDAR scan. As stated in the previous section, there are not many approaches using deep networks on 2D LIDAR data, especially ones that try to solve a problem with similar constraints. The second major issue we attempt in this thesis is the data representation of 2D LIDAR. Since this sensor commonly produces an output in polar coordinates, which is difficult to interpret, especially for humans, it is important to figure out how the sensor data needs to be represented to allow a neural network to learn useful features.

1.6 Approach

Detecting and classifying robots in 2D LIDAR scans is a difficult problem, especially because there are many other shapes in the environment that have similarities with common robot shapes (circle or rectangle). The idea of the author is therefore to divide this overall problem into two subproblems. One algorithm initially identifies all dynamic objects, a second one classifies those objects and estimates a relative position to them.

The first contribution of this thesis is the implementation and evaluation of three different deep learning algorithms for distinguishing moving (dynamic) from non-moving (static) objects in the sensor input. The first method proposed for extracting dynamic objects from a laser scan is using stacked, raw input data, which is processed using a CNN. This approach leverages spatio-temporal information included in the stacked sensor data. The network is only able to detect objects that have been moving during the time-span of the stacked data. Further it is not able to work on a moving ego vehicle. A second approach is therefore introduced, which creates a 2D occupancy grid-map from the scan data. After transforming all data points into one coordinate frame, the data from multiple consecutive scans (maps) is accumulated over a short time interval. The ego motion of the sensor is now captured implicitly in the sum of the scans. This allows the neural network to work with data gathered from a moving ego vehicle. A network using ConvLSTM layers is implemented for the detection of dynamic objects that were moving during the time-span of the accumulated data. The last network implemented leverages the benefits of the LTS that the algorithm will be integrated into (see section 1.3). Since the LTS always keeps an updated map of its static environment, all objects that are not part of this map are most likely dynamic. A simple network is therefore used

to calculate the difference between a static grid-map and a dynamic grid-map generated using the latest sensor data. Given that the static map is sufficiently good, this approach is able to detect dynamic objects even when they are not moving for a certain amount of time.

For the actual classification and relative position estimation of the robots, a combination of convolutional and ConvLSTM layers are used. The input data, however, is not accumulated over time. Only the scan data grid-map of the current time-step is used as an input to the network, since this is what the previously introduced dynamic object detection networks output as well. The prediction of the classifier is a heatmap of possible robot locations for each type. Using a clustering algorithm together with a centroid calculation of each cluster, multiple robot position hypotheses are generated.

All networks are trained in a supervises manner using simulated as well as real-world data. The evaluation is conducted in both kinds of environments as well and features investigations with regards to the networks architecture, localization uncertainty and the object distance.

1.7 Overview

The remainder of this thesis is organized as follows. Chapter 2 explains the principles of convolutional and recurrent neural networks, as well as specific functions and layers used later on. This includes different layer types together with optimization functions and parameters. Chapter 3 describes the state-of-the-art in (dynamic) object detection, classification and tracking using different types of sensors. Furthermore this chapter also elaborates on approaches using deep learning techniques for object detection, classification and tracking in images. In Chapter 4, the first part of the authors approach will be explained - extraction of scan points belonging to dynamic objects. This chapter includes information on the data representation, network architecture and training of the network. Thereafter, the data, architecture and functions used for the classification and position estimation network will be explained in Chapter 5. Furthermore the position extraction method will be illustrated in this part of the thesis. Chapter 6 will give an in depth evaluation of the presented networks with regards to the network architecture, localization uncertainty, (un-)known environments and the performance in simulation compared to real-world scenarios. Finally the author concludes the work and gives a brief outlook of future work that could be done in this field in Chapter 7.

2 Background

2.1 Convolutional Neural Networks

The idea of the convolutional neural network was first introduced by LeCun [CC89] in 1989 and then applied to handwritten digit recognition thereafter [CJB+89]. It was developed to handle grid-like data, such as images or time-series data, since this structured data is not supported by traditional neural networks. Due to several improvements over fully connected networks as well as the dramatic increase in computational power (GPU), CNNs are now able to process image data in significantly less time than during their invention. Within the past decade they have achieved state-of-the-art performance in several disciplines, such as image classification [KSH12], [SZ14], object detection [SEZ+13], [RHGS15] or image segmentation [BKC15], [SLD17].

One of the core enhancements introduced in CNNs is the weight sharing. Opposed to multilayer perceptrons, CNNs are not fully connected, but share weights using the **convolutional** operation. Besides a significant reduction in computational time during a forward pass through the system, this further enables a CNN to detect features independently from their position within the input data. This is a core property of CNNs, since it allows them to e.g. classify an image as being a dog, no matter where the dog is in the image. Previous, traditional approaches with fully-connected layers were not capable to do this.

In the following, this section further describes the general idea and functionality behind convolutional neural networks. In addition to that, relevant parameters, functions and layers commonly used for CNNs will be described, e.g. max-pooling, loss-functions and optimization functions. Only the ones used throughout this thesis will be explained in detail, though.

2.1.1 Convolutional Principles

The core of CNNs is the convolution which, in its most general form, is an operation of two functions of real-valued arguments [GBC16]. The purpose of this function is to generate a weighted average over input values to the functions that are close to the desired one. Especially for the purpose of CNNs, the convolution is mostly used over more than one axis at a time. Images, for example, provide a two-dimensional **input** I to the function. The **kernel** K, which is also referred to as the weight matrix in this case, must be two-dimensional as well. The output generated by the convolution is often declared as the **feature map** (S). The following equation shows the most widely used version of the convolution, implemented in most neural network libraries like this:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n). \tag{2.1}$$

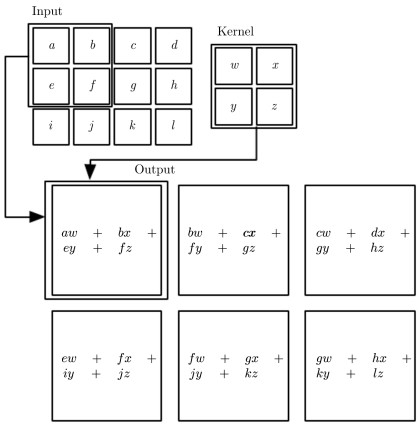


Figure 2.1: An example of convolution on a 2D input with a 2D kernel using Equation 2.1 [GBC16].

This function is actually called **cross-correlation** but is called convolution by many machine learning libraries. An example of this equation applied to 2-D convolution can be seen in Figure 2.1. The output for e.g. input cell *a* is given here by calculating a weighted sum of its surrounding values, where the size and weights are defined by the kernel. This forms the output of a layer in a CNN, similar to the equations used for an multilayer perceptron (MLP) [GBC16]. Compared to the classic neural network approach not all input dimensions have their own weights. This leads to some important properties of CNNs, that distinguish them from the classic networks - **sparse interactions**, **parameter sharing** and **equivariant representations**.

In traditional neural networks all neurons in two subsequent layers maintain a connection, i.e. a weight, between each other. This means if there are $1,000$ input dimensions followed by a hidden layer with $5,000$ neurons, a number of $5,000,000$ weights need to be learned just for the connection of these two layers. This renders classic neural network architectures impractical for image processing, since they often have input dimensions of tens or hundreds of thousands of pixels. The convolution reduces the number of learnable weights dramatically. In general small kernels with weights of size $3x3$ or $5x5$ are applied to the input, reducing the number of weights significantly and allowing the network to have **sparse interactions**[1] between connected layers. These kernels, however, still allow the network to detect meaningful features such as edges, patterns or body parts, while reducing the memory requirements as well as the computational power needed for training and predictions significantly. The Figures 2.2a and 2.2b visualize the sparsity of a convolutional network and the increasing *receptive field*[2] of neurons in deep layers, respectively.

The second idea leveraged in CNNs, **parameter sharing**, reduces the number of parameters to learn further by reusing weights across functions. As stated in the previous paragraph, traditional neural networks connect each all neurons of two subsequent layers with unique weights for each connection. While sparse interactions reduce the connections to neighboring neurons, parameter sharing further reduces the number of weights by applying the same weights at each location of the input. Figure 2.3 demonstrates this behavior on a simple example. Instead of learning three

[1] Sometimes *sparse interactions* are also referred to as *sparse connectivity* or *sparse weights*.

[2] The receptive field of a neuron is composed of the neurons from previous layers it shares a direct or indirect (through intermediate layers) connection with.

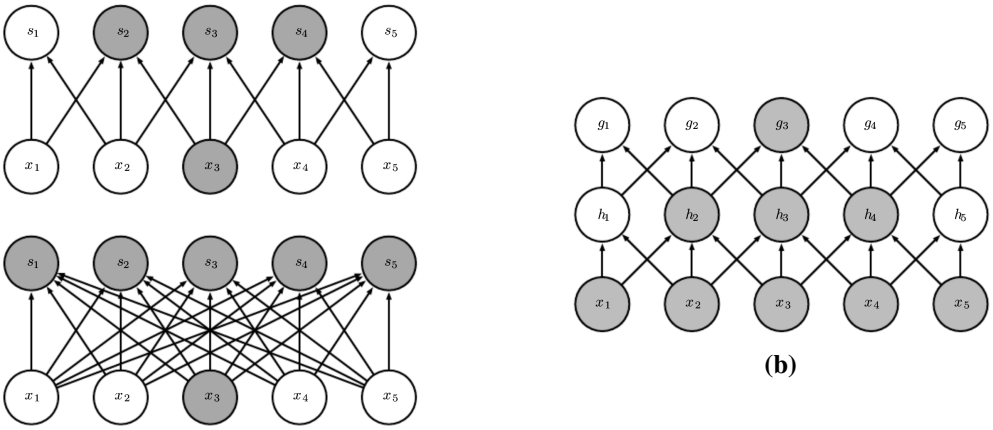


Figure 2.2: An example of sparse connectivity (a), where one input unit is highlighted, along with the output units it is connected to. The upper image shows a convolutional network with sparse connections, the input neuron is not connected to all output neurons. The bottom image visualizes the same scenario for a fully-connected network, which maintains connections from one input neuron to all output neurons. The receptive field of deeper layers in convolutional networks (b) is increased with the depth of the network [GBC16].

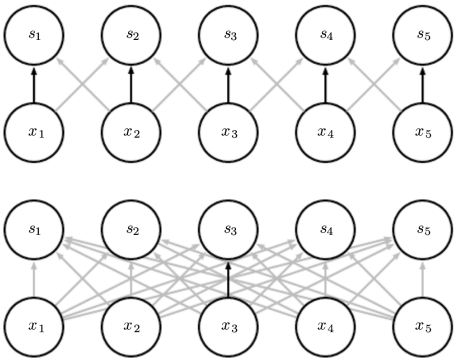


Figure 2.3: A visualization of the parameter sharing used in convolutional networks [GBC16]. Black arrows indicate a shared weight, used more than once throughout the model. The network at the top uses a kernel of size 3 and shares weights across each input dimension. The marked center parameter is used at all input locations for the connection to the respective neuron in the next layer. In contrast to that, the fully-connected layers in the bottom image don't share any parameters. The weight matrix used in this model therefore maintains a parameter for each connection.

separate weights for each input neuron, in this example only three weights in total have to be learned. This, again, reduces the number of learnable parameters in the network. This time, however, not the number of connections. Hence the computational time needed for a forward pass through the network is not diminished. On the other hand, the computational cost for training the network as well as the memory requirements of it are reduced again.

The idea of parameter sharing leads itself to the third property of convolutional networks - **equivariance** to translation. According to Goodfellow, Bengio, and Courville [GBC16], a function is defined as equivariant, if in case the input changes, the output changes in the same way. To explain this in the sense of convolutional networks, let's say we have an input image that we want to shift x pixels to the right but we also want to detect edges in it. It does not matter whether we first apply the convolution to the original input and then shift the output or the other way round. In general this means that if a function $f(x)$ is said to be equivariant to $g(x)$ then $f(g(x)) = g(f(x))$. This property leads to an interesting behavior in convolutional networks, since it doesn't matter where in the images features are. Due to the parameter sharing the features get detected wherever they are. That makes it easy, e.g. for classification tasks to identify objects independently from their location in the input image.

The core idea behind convolutional networks as well as the properties described in this part have demonstrated the superiority of CNNs over traditional approaches like the MLP, especially when handling grid-like data. Therefore it is mainly used for data like audio sequences, color images or videos. In the following, the typical structure of a CNN is explained together with common layers and learning methods.

2.1.2 Architecture & Layers

In its most general form, a typical CNN consists of a convolutional layer followed by a nonlinear activation function and a pooling stage. The functionality of each of these layers as well as possible parameters will be described in the following. Everything will be described using the example of 2D convolution, since this will be also used throughout this thesis.

As described in the previous part of this section, the first layer applies several convolutional kernels to the input, producing a set of linear activations. The parameters of this layer are summarized in Table 2.1. One can specify how many filters the layer should have, their size, whether to use a stride and which padding to use. Depending on the task, input and desired output data, these parameters certainly vary heavily among different architectures. Some of them, however, tend to be the same in the majority of the use cases. The number of filters commonly ranges between $16 - 64$ and their size is most often $(3, 3)$. In some cases the size is larger $((5, 5)/(7, 7)$. This increases the computation time for learning the weights, as well as a forward-pass through the network, but it also allows for larger feature detection. The stride defines the distance between consecutive applications of the kernel and is commonly chosen to be $(1, 1)$. In a 2D input, this means that after applying a kernel at the first pixel of the input, we shift it by one and continue. A higher stride reduces the computational complexity as well as the size of the input data and the overlap of the kernels receptive fields. The input data can further be padded, i.e. expanded by e.g. zeros surrounding the

Parameter	Filters	Kernel size	Stride	Padding
Type	int	(int, int)	(int, int)	$\in \{none, zeros\}$
Common	16-64	(3, 3)	(1, 1)	zeros

Table 2.1: Important parameters for convolutional layers together with common value choices.

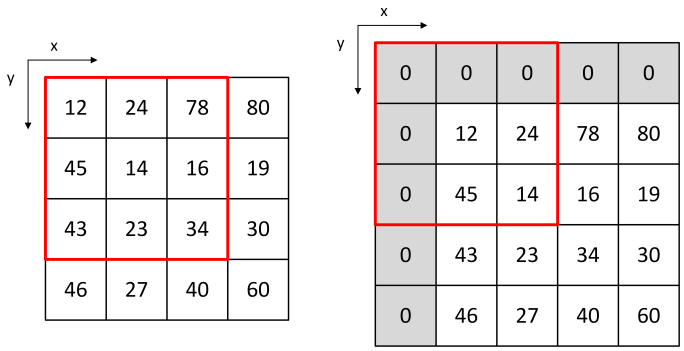


Figure 2.4: A sample piece of an array to which a $3x3$ convolution is applied. The left image shows the result without zero-padding, the right one with. Without zero-padding the first convolution can't be applied to the first column/row.

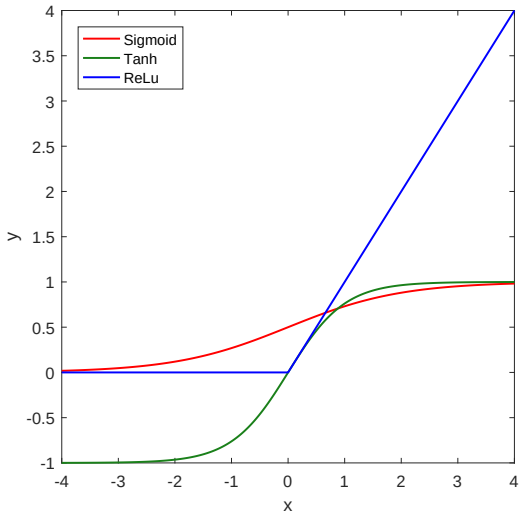


Figure 2.5: A comparison of three of the most common activation functions for neural networks: sigmoid, tanh and ReLu.

actual input. In the majority of the cases the data is padded with zeros around the input to keep its dimensions. Without the padding, the first and last column/row would be dropped, since the filter can't be applied there. An example, demonstrating the impact of padding is shown in Figure 2.4.

After the convolutional layer, a standard nonlinear activation follows. Due to the fact, that this layer produces the actual output of the convolutions, it is also named the **detector stage**. In classic neural networks, this was generally either the *tanh* or the *sigmoid* function [KO11]. In CNNs the default recommendation is to use the *ReLu* activation function [JKRL09]. The differences between these activation functions are visualized in Figure 2.5. The image highlights, that the ReLu, compared to sigmoid and tanh, constantly rises and its derivative doesn't vanish. Beyond the fact that ReLu has a (piece-wise) constant gradient of 1, it also leads to a sparse output matrix, since all values < 0 result in a zero output. These are both important properties for neural networks because they keep the gradients from vanishing during the learning phase and reduce computational cost, respectively.

The final layer in a typical convolutional stack is the **pooling function**. Its purpose is to reduce the output size of the previous layer by statistically summarizing it. The most common operation for this is called **max pooling** [ZC88], and it returns the maximum output of a rectangular neighborhood. Although there do exist more operations, like taking the average or a distance-based weighted average with respect to the central pixel, almost all convolutional network approaches use max-pooling. In most cases the size of the pooling window is set to $(2, 2)$, since higher pooling rates drop too much information. The max pooling reduces the output size, since it sums multiple values into a single one. This further leads to an invariance to local translations. An invariance to small translations can be an important property if we care more about whether a feature is present rather than its exact position. Many use cases of CNNs target the pure classification of an image or a sequence. In these cases it is not important where exactly a feature is located within the image, but the detection of its presence is of importance.

For those CNN architectures that have fully-connected layers after the convolutions, the fact that pooling reduces the size is of great importance. It is therefore possible to specify a stride for the pooling layer as well. The stride can further reduce the size but also drops more information.

2.1.3 Learning & Optimization

Training a deep network like a CNN is similar to the training of a traditional neural network with gradient descent. They are trained using an iterative, gradient-based optimizer which reduces the cost function to a low value. Using backpropagation of the gradients together with e.g. stochastic gradient descent (SGD) we can learn suitable weights for the network. Beyond SGD there exist many more optimizers, most of them deal with the problem of learning rate adaption over time. Among the most important optimizers are the root mean square propagation (RMSProp) [Hin12], the adaptive gradient algorithm (AdaGrad) [DHS11] and the adaptive moment estimation (Adam) [KB14]. In general these three approaches are similar, as they all try to improve the networks learning by maintaining one learning rate per parameter instead of a single global one [Rud16]. AdaGrad adapts the learning rate for each parameter by performing larger updates for infrequent and smaller updates for frequent parameters. It suffers, however, from radically diminishing learning rates. With RMSProp, an enhanced version is therefore introduced, which improves the learning rates by adapting them based on the average of recent squared gradients. Adam combines both approaches and provides an optimizer with state-of-the-art performance in most cases. In addition to keeping an exponentially decaying average of past squared gradients, as RMSProp does, Adam also maintains a corresponding value for all gradients. All three optimizers work much better on sparse gradients, due to their adaptive learning rate based on the gradients. Adam, however, outperforms the other optimizers in most cases and is therefore the most commonly used optimizer for deep learning applications (to the best of our knowledge). For more information on the differences between the respective optimizers, we refer to a a great review of gradient decent optimizers in [Rud16].

In section 2.1.2 three activation functions commonly used for intermediate layers of neural networks were already introduced - sigmoid, tanh and ReLu. The final output activation of the network is usually chosen based on the task of it, classification, or regression. This thesis will only use neural networks for classification purposes, thus the activation functions and losses discussed in this section will be suitable for classification.

In case the network is supposed to do a binary (two-class) classification, commonly the activation function chosen for the final layer is the **sigmoid** function. As shown in figure 2.5, the sigmoid function returns a value between 0 and 1, ideal for binary classification. If the purpose of the network is multi-class classification, the sigmoid function is not necessarily the right option to choose. The reason for this is, that the output for one class is here independent of the output from another one. This means that, theoretically, there could be an output of 1 for a given data point x for each existing class. In most cases this is not a desired behavior, since generally a data sample either belongs to class A or class B but not both. For binary classification the sigmoid worked fine with the same behavior, since we just output a single number defining whether the sample belongs to the class or not. For multi-class classification the **softmax** activation function enables the desired classification result. It is defined as:

$$softmax(z)_i = \frac{exp(z_i}{sum_j exp(z_j)}, \tag{2.2}$$

where z is the output of the last layer (if applied to neural networks). This function returns the probability of output z belonging to class i. As the function sums the output for all classes, the probability for one class is no longer independent from the others. Since the softmax activation calculates a probability, the sum over the activations of all classes is naturally 1.

Similarly to the activation functions, there also exists a suitable loss function for both, binary and multi-class classification. The general (multi-class) form of the equation is called **cross-entropy**[3] and defined as follows:

$$H_{categorical}(y, \hat{y}) = \sum_i y_i log \frac{1}{\hat{y}_i} \tag{2.3}$$

$$= -\sum_i y_i log \hat{y}_i \tag{2.4}$$

with y and $\hat{y}$ being the desired output and the calculated prediction, respectively. The sum goes over all classes i in this case. However, in case of binary classification, there are just two classes and a single output. The equation can therefore be transformed into:

$$H_{binary}(y, \hat{y}) = -(\hat{y}_i log y_i + (1 - \hat{y}_i) log(1 - y_i)). \tag{2.5}$$

This function is also called **binary cross-entropy**, especially in the various machine learning libraries that currently exist.

As in traditional neural networks, deep networks also cope with issues like underfitting and overfitting. Generalization of the model is still one of the most crucial challenges in deep learning. To counter this, several effective methods for regularization of the network have been developed, some of which will be presented briefly in this section, since they will be used in the network presented in this thesis.

[3]The cross-entropy is sometimes also denoted as *negative log-likelihood* or *categorical cross-entropy*.

CNNs are commonly trained using mini-batches. Therefore a first method for regularizing the network is **batch normalization**. The outputs of the respective layers are here normalized throughout each dimension. This reduces the activation of the layers, since the values are now within a range between 0 and 1. Therefore the weights don't get shifted too much from one learning epoch to another. A second method for achieving a similar result is the **L1** or **L2-regularization**. This form of regularizing the learning of the network adds another term to the loss function including a sum over the networks weights. Since during the learning phase, the loss function is minimized, the sum (squared sum for L2) over all weights is minimized as well, preventing them from exploding.

Another popular regularization method for deep networks like CNNs is the so called **dropout**. As the name suggests, with this method some random outputs, of the layer before the dropout is applied, are dropped and not forwarded to the next layer. A dropout rate specifies how many outputs are dropped (percentage). This method for preventing a network from overfitting has proven to be effective in recent years [SHK+14] and has thus gained a huge popularity.

2.1.4 Autoencoders

There exist many different architecture types for CNNs, one of which is referred to as *autoencoders*. The networks using this type of architecture maintain an *encoder-decoder* structure. At first the input data x is transformed into a hidden state h by an encoder function $h = f(x)$. Afterwards, the function $r = g(h)$ reconstructs the output, commonly with the same size as the input. The default structure of such an autoencoder is shown in figure 2.6a, together with a common architecture using max-pooling and up-sampling layers in figure 2.6b. The objective of an autoencoder is not simply to learn an exact mapping from the input data to the output $g(f(x) = x$. The goal is rather, to learn an approximate representation of the input, while also learning important features of the data. Further, the encoder commonly reduces the dimensionality of the input data, which makes it less computational expensive to learn features between the encoder and the decoder [GBC16].

Autoencoders were first introduced by Ballard [Bal87] for pre-training artificial neural networks. As of today they have been used for a wide range of applications, e.g. natural language processing [DZMS14], image manipulation [LAS17] or semantic segmentation [BKC15]. For further information on autoencoders we refer the reader to [GBC16], which includes an in depth description of the various types of autoencoders.

2.2 Recurrent Neural Networks

The recurrent neural network was first introduced by Rumelhart et al. in 1986 [RHW86] for learning coherences in sequential data. Opposing to the previously introduced CNNs, which are made to handle grid-like data X, these networks are built for dealing with sequences of values $x^{(1)}, ..., x^{(\tau)}$ [GBC16]. There are many different application domains for RNNs, especially in fields like natural language processing and semantic image or video labeling. Naturally a wide variety of different approaches and architectures exists for different combinations of input and output data shapes. In the following a brief overview will be given on what RNNs are exactly and how they process the sequential data. Further the different types of architectures and application cases will be elaborated on. Finally two enhancements of the traditional RNN will be presented for improved processing of grid-like sequential input data.

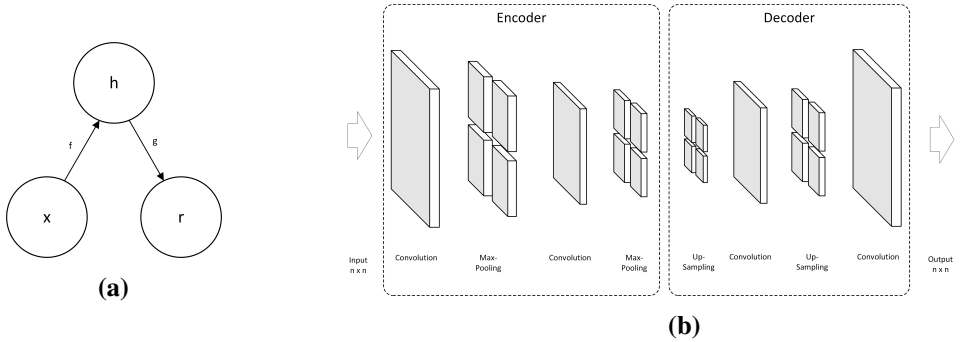


Figure 2.6: The general structure of an autoencoder (a) [GBC16], where x is the input, h the hidden state and r the output of the network. The hidden state is computed using the function $h = f(x)$ and the output by using $r = g(h)$. An example architecture of an autoencoder including two stages of max-pooling/up-sampling (b).

For more information on RNNs, we refer the reader to the textbooks of Goodfellow, Bengio, and Courville [GBC16] and Graves [Gra12].

2.2.1 RNN as Graphical Models

The inner structure of an RNN can be visualized by a graphical model to demonstrate the dataflow in the network. Figure 2.7 shows a network which learns an association from a sequence of inputs x to a sequence of outputs o with the same length (many-to-many). The unfolded graph of the network (left) illustrates the recursive nature of this network type. Opposed to traditional networks, this network maintains a loop from the hidden layer to itself. The meaning of this is illustrated in the right part of the image. It shows the unrolled graph for three time steps ($\{x_{t-1}, x_t, x_{t+1}\}$). With the help of this unrolling, it is easier to understand what a RNN actually does. In this version, a connection between hidden states s_{t-1} and s_t is maintained. Hence the hidden state of the previous time step is always incorporated into the calculation of the current output at time t.

The figure further contains three weight matrices - U, V and W. As in traditional networks, the matrices U and V are applied against the input x and the hidden layer output s, respectively. The weight matrix W, however, is different from traditional networks, since it is part of the recursion. This leads us to a core property of RNNs, which was already introduced for CNNs in the previous section - **parameter sharing**. The right part of Figure 2.7 clearly outlines, that the weight matrix W is the same one in each iteration ($s_{t-1}, s_t, \ldots$). This is possible due to the weight sharing between different time steps. If separate parameters for each time step were used, the model wouldn't generalize to sequence lengths not included in the training data. In addition to that, no statistical strength could be shared across different sequence lengths or different positions in time.

The forward pass through a recurrent network is, after examination of the structure, straightforward. First the initial hidden state h^0 has to be specified by initializing it e.g. randomly. Depending on the length of the sequence to process τ, the following equations are then repeatedly applied:

$$a^t = b + Wh^{t-1} + Ux^t h^t = act_h(a^t) o^t = c + Vh^t \hat{y}^t = act_{\hat{y}}(o^t) \tag{2.6}$$

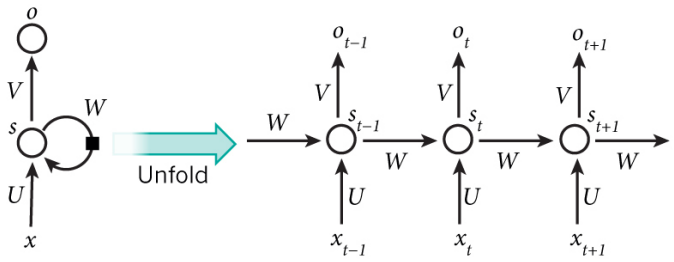


Figure 2.7: An example of a simple RNN, mapping a sequence of data x to a sequence of outputs o the left part of the figure demonstrates the recurrence in the network, whilst on the right the network is unrolled for three time steps [18c].

with the previously described weight matrices U, V and W together with the bias vectors b **and** c. The activation function act_h of the hidden state is here not specified but is usually either tanh or ReLu. The one for the final predicted output $act_{\hat{y}}$ is defined according to the output type, e.g. softmax for multi-label or sigmoid for single label classification.

The total loss function is then given by the sum of all separate losses $L^{(t)}$ from the respective time steps. Let's say we want to calculate the overall loss with the negative log-likelihood of $y^{(t)}$ given $x^{(1)}, ..., x^{(t)}$, the corresponding equation would look as follows:

$$\begin{aligned} L(\{x^{(1)}, ..., x^{(\tau)}\}, \{y^{(1)}, ..., y^{(\tau)}\}) & \\ &= \sum_t L^{(t)} \\ &= -\sum_t logp_{model}(y^{(t)} | \{x^{(1)}, ..., x^{(t)}\}) \end{aligned} \tag{2.7}$$

The computation of the gradient is expensive for RNNs, since it requires a forward propagation pass iteratively from left to right through the unrolled graph in figure 2.7. This further means, that the computation cannot be parallelized as e.g. the convolutions in a CNN. Hence RNNs are expensive to train with a runtime of $O(\tau)$ but also provide a powerful neural network architecture for processing sequential data of arbitrary length.

2.2.2 Design Patterns & Architectures

There exist many different architectures and types of RNNs, an overview is shown in Figure 2.8. The image demonstrates the input/output data combinations that are commonly used for these networks. The first graph shows a traditional network, without sequences (e.g. MLP or CNN). The second type shown is learning a sequence of outputs for a single input, this can be used for e.g. captioning an image. Basically the opposite is visualized in the next one, where an association from a sequence to a single output is learned. Application cases for this are the classification of sentences or the classification of a sequence of sensor inputs. Finally there are two different versions that map a sequence of inputs to a sequence of outputs. The first one is an asynchronous mapping for e.g. translation of a sentence from one language into another, where the network is first processing the

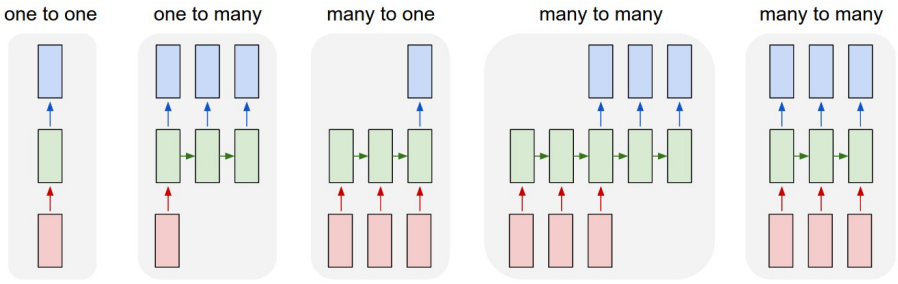


Figure 2.8: An overview over the different kinds of RNN data association types. The red boxes define the input to the network and the blue boxes its output. The green boxes represent the internal state of the RNN. [18b].

input sentence completely until the output is generated. As an alternative, the network can also be constructed, so that it learns a synchronized association. The result of this is a sequence of outputs, where each output belongs to the respective input in the input sequence.

There do exist more variants of RNNs, like bidirectional or recursive networks, for the purpose of this thesis, however, they are not of interest. For more information on additional models, we point the reader to [GBC16] and [Gra12], respectively.

2.2.3 Long Short Term Memory

One of the most crucial parts of RNNs is the learning of long-term dependencies [Hoc91], [BSF94]. Due to the repeated application of the weight matrix W during training, problems arise in the long run. Gradients that are propagated over many stages tend to either vanish or explode, both render learning impractical in most cases. This can be reduced in some way by choosing an appropriately small learning rate or one of the optimizers introduced in section 2.1.3, however, this does not solve the problem adequately.

The most effective method for tackling this issue, as of today, are **gated RNNs**. In addition to traditional RNNs, these network types utilize gates to control the flow of information between different nodes in the network. There exist two main types, the **long short-term memory** and the **gated recurrent unit**. Both are similar in performance and since this thesis solely uses the long short-term memory, gated recurrent units will not be elucidated in this chapter.

The LSTM introduces three new gates to the architecture of a standard RNN. LSTMs networks have *LSTM cells* which replace the usual hidden units of ordinary RNNs. These cells introduce an additional recurrence within the cell itself (self-loop). The diagram of an LSTM cell is shown in figure 2.9.

Due to the newly introduced gates, the equations for the forward pass through the network change slightly from the ones presented in section 2.2.1. The input data is now used not only as a pure input for the hidden layer, but also as an input for calculating the activation of all gates in the network. Each gate outputs a value between 0 and 1 computed by the non-linear activation function sigmoid. This value specifies how much information can pass through the respective gate. The first one in the line is the **external input gate** unit $g_i^{(t)}$, which determines how much of the input data is incorporated into the hidden state calculus:

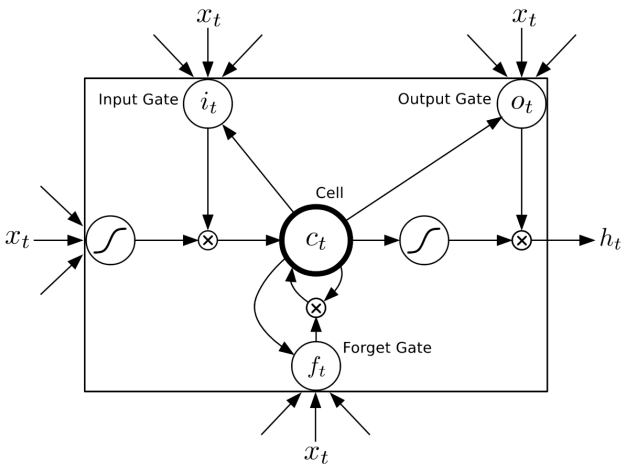


Figure 2.9: The sketch of a LSTM memory cell including an input, output and forget gate [18a].

$$g_i^{(t)} = \sigma(b_i^g + sum_j U_{i,j}^g x_j^{(t)} + sum_j W_{i,j}^g h_j^{(t-1)}), \tag{2.8}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + sum_j U_{i,j} x_j^{(t)} + sum_j W_{i,j} h_j^{(t-1)}), \tag{2.9}$$

with U^g**,** W^g and b^g being the input gate g specific weight matrices for input/recurrent data and the biases, respectively. The current input vector is denoted by $x^{(t)}$ and the respective hidden layer vector by $h^{(t)}$. Similarly the standard weights (**U, W**) and biases (**b**) are use for the calculation of the hidden state $s_i^{(t)}$. In addition to that the **forget gate** unit specifies here, how much information of the previous hidden state is includes in this time steps calculation. This is done similarly to the calculation of the external input gate unit $g_i^{(t)}$ but again with own weights and biases:

$$f_i^{(t)} = \sigma(b_i^f + sum_j U_{i,j}^f x_j^{(t)} + sum_j W_{i,j}^f h_j^{(t-1)}). \tag{2.10}$$

Finally the calculation of the hidden units output

$$h_i^{(t)} = act_h(s_i^{(t)}) q_i^{(t)} \tag{2.11}$$

can be influenced by the **output gate** unit $q_i^{(t)}$, regulating the influence of the hidden state $s_i^{(t)}$ on the output of the LSTM cell:

$$q_i^{(t)} = \sigma(b_i^q + sum_j U_{i,j}^q x_j^{(t)} + sum_j W_{i,j}^q h_j^{(t-1)}). \tag{2.12}$$

In addition to this formulation of the equations, one could choose to incorporate the hidden state $s_i^{(t)}$ into the calculations of the respective gate activations as well. This would introduce another term added to the sum over all inputs and hidden outputs, respectively.

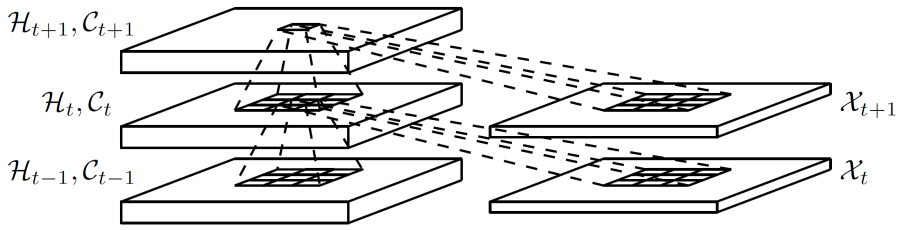


Figure 2.10: The inner structure of a ConvLSTM [SCW+15].

The use of LSTMs on sequential data with long-term dependencies in several recent approaches [DAG+15], [TSM15] have shown, that they are able to cope well with these challenges. Even in these challenging sequence processing tasks, the enhanced RNN produces state-of-the-art learning results. These networks, however, are designed for processing sequential data. Similarly to traditional neural networks like the MLP, they are not optimized for working with grid-like data as CNNs. The following subsection will therefore discuss another extension of the LSTM for dealing with those drawbacks.

2.2.4 Convolutional LSTM

To deal with the drawbacks of traditional LSTMs in learning spatio-temporal features for input-to-state and state-to-state transitions, an approach using convolutional operations instead of full connections is proposed by SHI et al. [SCW+15]. The authors introduce the ConvLSTM, a new architecture using 3D tensors[4] for the inputs $x_1, ..., x_\tau$, the hidden states $h_1, ..., h_\tau$, the cell output $s_1, ..., s_\tau$, as well as the output of the gates g_t, f_t, q_t, to enable the encoding of spatial information within the network. The new state-to-state and input-to-state transitions are visualized in figure 2.10. With the operator substitution, the equations for the ConvLSTM naturally differ slightly from the standard RNN equations as well:

$$g_i^{(t)} = \sigma(b_i^g + sum_j U_{i,j}^g * x_j^{(t)} + sum_j W_{i,j}^g * h_j^{(t-1)} + sum_j W_{i,j}^g s_j^{(t-1)}), \tag{2.13}$$

$$f_i^{(t)} = \sigma(b_i^f + sum_j U_{i,j}^f * x_j^{(t)} + sum_j W_{i,j}^f * h_j^{(t-1)} + sum_j W_{i,j}^f s_j^{(t-1)}), \tag{2.14}$$

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma(b_i + sum_j U_{i,j} * x_j^{(t)} + sum_j W_{i,j} * h_j^{(t-1)}), \tag{2.15}$$

$$q_i^{(t)} = \sigma(b_i^q + sum_j U_{i,j}^q * x_j^{(t)} + sum_j W_{i,j}^q * h_j^{(t-1)} + sum_j W_{i,j}^q s_j^{(t-1)}), \tag{2.16}$$

$$h_i^{(t)} = act_h(s_i^{(t)}) q_i^{(t)}, \tag{2.17}$$

where '$*$' denotes the convolutional operation. Beyond the newly introduced operator, this network further incorporates the output of the hidden state $s_i^{(t-1)}$ into the computation of the gate units activation. This is an addition to the equations presented in section 2.2.3.

[4]The last two dimensions of these 3D tensors are rows and columns, respectively.

This novel architecture can be used for e.g. tracking of a moving object over time. The network is capable of identifying and memorizing spatio-temporal features. By increasing the kernel size used by the transitions, a larger space in the spatial data is considered and therefore faster movements of the objects can be tracked. The authors further utilize padding, which was introduced in section 2.1.2, to maintain the same number of columns and rows within the whole network.

3 Related Work

In the following related approaches in shape independent object detection and tracking using classical approaches and different sensors will be elucidated first. Most of them rely on probabilistic filters, such as the Kalman or Particle Filter, sometimes in combination with the iterative closest point (ICP). Although the goal of this thesis is to classify objects based on their shape, there are many similarities between the two purposes and it is the foundation for object classification. Furthermore, most approaches in the literature deal with shape independent object tracking since in many domains the exact type of an object is often not important, and shapeless approaches generalize better to unknown objects. The subsequent review of related dynamic object classification methods includes classical learning (e.g. support vector machine (SVM)) as well as non-learning (e.g. ICP) approaches. In this chapter we present algorithms for different sensors and environments. Afterwards there will be an overview over recent advances in the field of deep learning based object classification and semantic segmentation in images. The presented techniques will be primarily based on CNNs and similar, further developed networks. The chapter is concluded with a summary of recent methods for object tracking using RNNs on grid-like data.

3.1 Shape Independent Object Detection and Tracking

There exists a vast variety of approaches in the literature dealing with shape independent object detection and tracking. They can be distinguished into indoor and outdoor approaches. The indoor ones often deal with the issue of detecting and following people. The methods developed for outdoor environments mostly detect and track vehicles, bicycles and sometimes pedestrians in traffic scenes like intersections. A further distinction can be made regarding the placement of the sensors as well as the sensor types themselves. Sensors can be located statically in the environment, e.g. along a road, or they can also mounted on a moving vehicle or robot. The information of a dynamic sensor is significantly harder to process due to the ego-movement of the equipped vehicle. Moreover, if a 2D LIDAR sensor is mounted on e.g. a car, it is subject to minimal height (pitch/roll) variations over time, which leads to differences between subsequent scans because objects have different structures on different heights (this especially applies for outdoor applications). To account for these problems, a lot of methods dealing with outdoor object detection deploy multiple sensors on their test vehicles and fuse their output. Common sensors include 3D Depth or LIDAR (360° LIDAR) and stereo cameras. One of the downsides that 3D pointcloud data has, is the vast amount of data which is produced at a high frequency by the sensor. It is challenging to process this sensors data in real-time. Cameras on the other hand provide detailed information for object detection, but they are also highly dependent on the local lighting conditions and can fail fatally. They further do not allow for accurate distance measurements.

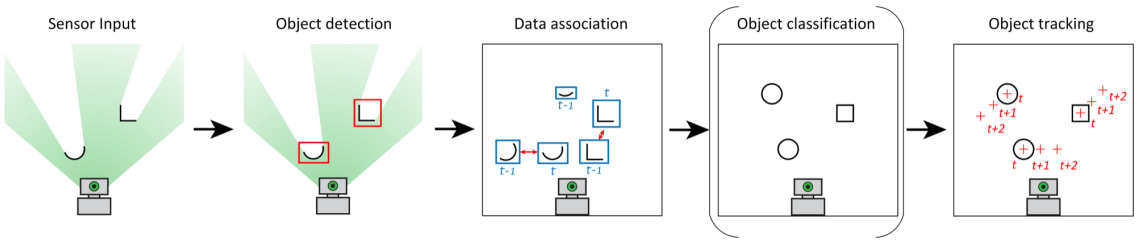


Figure 3.1: The pipeline shared by most object detection and tracking approaches composed of 4 (5) stages [ODWP16].

Despite these differences, there is a common pipeline, shared by most approaches independent from the type of sensor input. Figure 3.1 visualizes a diagram of this pipeline with four stages and an additional fifth classification stage which is only done rarely. Most approaches only segment and cluster the sensor data into objects, associate these objects between consecutive scans and track their entire dynamic state including their pose and velocity for the prediction of future states. For the purpose of this thesis, the most important stages of this pipeline are the object detection and association. The following literature review will therefore focus on these two topics. Many of the related approaches introduced in this section still build on 2D LIDARs due to their reliability in nearly all conditions and its highly accurate distance measurements. However, for a more complete understanding, some 3D LIDAR and data fusion examples will be presented as well. For an extended overview of object detection and tracking techniques that goes beyond the references mentioned in the following, we refer to an extensive summary presented in Mertz et al. [MNM+13].

3.1.1 2D Data

The most basic setup for dynamic object detection is composed of a static sensor placed in an indoor environment. Several approaches exist for people tracking using statically placed LIDAR sensors. The approaches presented in Zhao and Shibasaki [ZS05] and Fod, Howard, and Mataric [FHM02] both use a Kalman Filter for people tracking based on data received from multiple LIDARS. In Zhao and Shibasaki [ZS05] a background image of the environment is initially generated and updated over time (e.g. every 30 min). Moving objects are then identified by comparing the value measured at a laser-beam position at time t with the one saved in the background image. If the difference exceeds a defined threshold, the beam is classified as belonging to a moving object. All these labeled measurements are then clustered (similar to equation 3.1) and tracked by the filter. Fod, Howard, and Mataric [FHM02] use a similar approach. They first extract foreground from background readings, analogous to the method described before. Afterwards they further classify the measurements into blobs and objects, with objects being composed of one or multiple blobs. This subsequently serves as input for the tracking filter.

When the sensor is placed onto a moving platform the problem gets significantly more difficult to solve. A common information resource for autonomous vehicles in this case is an occupancy grid-map. It stores a discretized representation of the environment in a grid-like structure and is maintained by most autonomous vehicles, especially in indoor environments. One approach, combining particle filters with sample based joint probabilistic data association (SBJPDA) filters is presented in Almeida, Almeida, and Araujo [AAA05] and Almeida and Araujo [AA08]. Initially, moving objects are identified by building multiple consecutive occupancy grid-maps. The grid-map

calculated for time-step t is combined with the one at $t + 1$. Cells containing a different value after the update are defined as belonging to a moving object. Afterwards this information is used to estimate the state of an object with a particle filter. The data association between consecutive time-steps is achieved with the SBJPDA filter, by assigning features detected from sensor data to filters. This method allows for the tracking of multiple moving objects without the specification of an explicit shape in real-time. However, maintaining an occupancy grid-map is cost-intensive and error-prone since the change of a probabilistic cell value is the only indication for moving object extraction.

Another occupancy grid related method for moving object detection and tracking is described in Lindstrom and Eklundh [LE01]. Although a grid-map is not used explicitly, the authors mimic the use of one by identifying objects solely based on the assumption, that they occur in a previously free space. To achieve this, the scan is transformed into a polygon defining free space around the robot at time t. If at time $t + 1$ scan points lie within this polygon, they presumably belong to a moving object. This approach is evaluated in a simple living room. The authors claim that it works well in simple rooms like living rooms or offices, because they are not as complex and dynamic as a factory or outdoor environment.

In Montemerlo, Thrun, and Whittaker [MTW02] a conditional particle filter for both, people tracking and pose estimation of the ego robot is presented. The setup is simplified in the sense that there are no other moving objects in a known (pre-mapped) environment. Therefore all scan points not belonging to static (mapped) objects are simply identified as belonging to people. Beyond that, the number of people visible in the scans is always known, which is a challenging task to figure out in real-world applications. Nevertheless the authors prove that there system using multiple filters, one for each person and one for the robot, that delivers reliable tracking and localization results even under global uncertainties.

Many of the assumptions made by the introduced approaches fail when applied to an outdoor environment. Mostly because there is no static background modeled by a grid-map. The background is constantly changing if the ego vehicle is moving. The algorithms for object detection and association therefore need to adapt to these constraints. A common way of classifying objects or associating data between scans in outdoor scenarios is the ICP. This algorithm was originally introduced for the registration of 3-dimensional (3D) shapes [BM92]. Today it is widely used for matching two multidimensional sets of points, e.g. for scan point association in consecutive scans. In Thuy and Leon [TL09] it is combined with a particle filter for shape independent object tracking in outdoor environments. First new objects are identified by calculating the *distance-dependent spacing* ΔD between two consecutive points of the measurement using the following equation:

$$\Delta D(r_{i+1}, r_i) = \sqrt{r_i^2 + r_{i+1}^2 - 2 r_i r_{i+1} cos(\Delta\alpha)}, \tag{3.1}$$

with $\Delta\alpha$ being the constant scanning angle between two LIDAR beams. Neighboring points with ΔD below some threshold d_{th} are grouped together as one object. This is a common way to extract objects out of a 2D LIDAR scan. The objects shape is stored at time t and then matched with the new measurement at time $t + 1$ using ICP. The Particle Filter finally assures a non-linear accurate tracking and prediction of the object state over time.

Wang, Posner, and Newman [WPN15] describe an approach using a unified Bayesian framework to jointly estimate the sensor pose, a local static background and the dynamic state of an object. The source scanner is mounted on a vehicle driving on a road. Moving objects are tracked in this method by specifying boundary points for the static background as well as each object based on raw laser measurements. New dynamic objects are detected by utilizing constrained initialization [Wil01]. For all data points that cannot be associated with an existing dynamic object or the background, a new object as well as a new dynamic track is created and marked as *tentative*. As soon as the object has been tracked for a certain amount of time it is marked as *mature*. Thereafter, the system checks whether it belongs to an existing dynamic track or the background, and in case one applies, merges the respective entities. If there is no overlap, the new track is declared *established* and appended to the list of existing dynamic tracks. This is all done in the Bayesian filtering framework, which subsequently tracks the dynamic objects over time.

The approach presented in Mertz et al. [MNM+13] detects and tracks moving objects by extracting geometric features using multiple 2D or 3D LIDARS in an outdoor environment. After the sensor data is segmented into multiple clusters based on their distance to each other (see equation 3.1), the algorithm extracts lines and corners from the clusters. These features are then used for tracking the objects with a Kalman Filter. The association between consecutive scans is performed by first predicting the outline of all objects at time t and then testing which objects identified in the measurements at $t + 1$ belong to already existing ones. If two objects are overlapping with at least one scan point they are merged. Static objects are not distinguished from dynamic ones in this approach and are therefore tracked as well. This increases the computational power.

3.1.2 3D Data, Camera-Images and Fusion

The use of 3D (LIDAR/depth) data and stereo camera images has been used increasingly in recent publications, since these types of sensors as well as the computational power to process the huge amount of data produced by them have become available more easily. Moreover these sensors or the fusion with 2D LIDAR provides data with much more detail, which is especially important for outdoor applications. It allows for more accurate detection and tracking of dynamic objects. Nevertheless, this thesis primarily focuses on 2D LIDAR data, therefore the section will only give a brief overview of the algorithms available for these sensors.

As a result of the vast amount of different approaches existing in the literature, an extensive evaluation of 3D LIDAR tracking methods is presented in Morton, Douillard, and Underwood [MDU11]. The authors conclude, that classic ICP approaches can be outperformed by simply tracking the centroids of the observations. Nevertheless it is still used extensively in current research. Asvadi et al. [APPN16] use ICP for the detection of the road. The discrimination between static and moving obstacles is done by a discriminative analysis of a 3D voxel grid followed by a Log-Likelihood Ratio (LLR) for computing binary masks for stationary and moving voxels. The detection and association of moving obstacles in consecutive scans is realized in Dewan et al. [DCTB16] using only motion cues obtained by RANSAC [FB81]. This method estimates motion models for the ego vehicle as well as the objects, which are used to distinguish between static and moving objects. Afterwards a Bayesian approach is deployed to detect all points that follow a motion model and track them.

The deployment of a camera increases the amount of information available in the sensor data, however, as stated previously (1) visual sensor have the drawback of being highly dependent on light conditions. Nevertheless they are often used, since detecting and classifying objects in imagery is relatively accurate. Most classic approaches use (Extended) Kalman Filters for tracking the objects [ELSG09], [BD06] and spatial-temporal features together with simple graphical mathematical operations (e.g. morphology) to detect and separate objects from the background. However, even though objects can be detected easily in images, their position in the surrounding environment can't be estimated accurately.

Therefore, another common approach is to combine both previously described sensors, a 3D LIDAR scanner and visual data from a camera, to allow for better visual detection (camera) and state estimation (LIDAR) of the objects. To reduce the computational cost, some approaches only classify objects in images based on regions of interest (ROI) defined by the LIDAR [CA16]. Others use spatial-temporal features, like approaches with 2D LIDAR [MSRD06].

The fusion of various different sensors is becoming increasingly popular, especially in the field of automated driving, since each sensor type has drawbacks and blind spots which a network of multiple different sensors overcomes. For the DARPA Urban Challenge 2007[1] one team developed an obstacle detection and tracking algorithm by fusing the information of 13 different environmental sensors, including RADAR, 2D and 3D LIDAR. Moving objects are identifies by tracking the movement of all identified shapes and specifying two flags - *(not) moving* and *(not) observed moving*. Using an Extended Kalman Filter, the state of all objects is tracked including dynamics.

3.2 Object Classification in Dynamic Scenes

Compared to shape independent object detection, there exist fewer algorithms for object classification in dynamic scenes, i.e. around an autonomous vehicle or robot, based on 2D LIDAR data. This is at least somehow due to the fact that, in many use cases, the class of the specific object is not very important. In these cases it is sufficient for the autonomous vehicle to know the dynamic objects pose, velocity, acceleration and boundary. These values carry an implicit description of the objects class, since a pedestrian for example does not walk as fast as a car drives. In the following some approaches based on 2D and 3D data are presented. Optical approaches are often combined with LIDAR data for a better relative pose estimation of the object. Some of these will be presented in this section, however, the majority of optical object classification is introduced in the section thereafter, since those are deep learning-based approaches.

3.2.1 2D Data

The utilization of spatio-temporal (spatial-temporal) features for improved dynamic object recognition and tracking is presented in Qin et al. [QCS+16]. The authors first perform a segmentation on accumulated sensor information acquired from a 2D LIDAR. Then a classification using a Support Vector Machine (SVM) is deployed to identify vehicles in the traffic scene. The paper proves

[1] On November 3, 2007 DARPA organized the Urban Challenge, an autonomous vehicle race in an urban environment including three different missions and accumulating over 60 miles in under 6 hours.

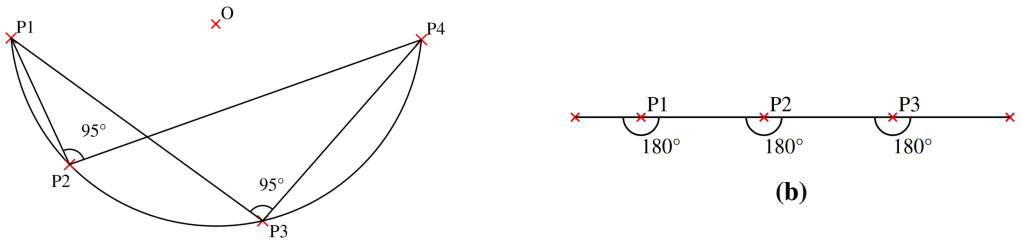


Figure 3.2: The angular differences of an arc (a) and a line (b) for each three points on the objects shape [XPC+05].

that spatio-temporal features improve the classification rate, especially in cluttered environments. Nonetheless, the classification is binary (car, no car) which is simple compared to multi-class classification (car, bicycle, pedestrian, ...).

The detection and classification of geometric primitives (lines, arc/circles) as well as legs is discussed in Xavier et al. [XPC+05]. At first, the system applies a standard clustering method based on the distance of points in consecutive scans (see equation 3.1). To classify the obtained clusters thereafter, different algorithms were used based on the various shape properties of each class. In the approach, a recursive line fitting algorithm for the extraction of lines from laser scan data is applied as well as a new method called inscribed angle variance (IAV) for the identification of arcs and circles. This method compares the angular differences of triangles spanned by each three points in a cluster. The difference between an arc and a line is shown in Figure 3.2a and 3.2b, respectively. Legs are then detected by using the data obtained by the IAV algorithm. If the first and last point in a cluster fall within the range of expected leg diameters ($0.1 - 0.25m$) the cluster is classified as legs. Although this method works well for the introduced geometric shapes with some miss-classifications (legs are sometimes classified as circles), the geometric shape of the object always has to be modeled by hand which can be difficult for more complex shapes.

A more advanced, multi-label classification and tracking approach using a statically placed 2D LIDAR scanner at an intersection is introduced in Zhao et al. [ZSKS06]. The system classifies the data into 0-axis objects (e.g. pedestrians), 1-axis objects (e.g. bicycles) and 2-axis objects (e.g. cars, trucks, buses). The model of an object is not specified upfront but instead picked dynamically from a laser measurement. Using a Markov Chain the state of the object (shape, kinematic properties) is estimated by leveraging the information which has been collected over time. The authors prove that their approach works well (95% success rate) at the given intersection. Nevertheless the system only distinguishes three groups of classes and no different objects, e.g. it does not distinguish between exact shapes but between shape types. Furthermore it is a static approach which gets significantly more difficult if the scanner is placed on a moving vehicle.

A similar problem is addressed in Premebida and Nunes [PN06] using Gaussian Mixture Models (GMM) for classification of the objects. The LIDAR data is first segmented using a linear KF-based method [BA04]. The segmented data is then used to extract features for the GMM classifier. The object classes are modeled by a weighted combination of Gaussian probability density functions (PDF) whose parameters are estimated using the expectation-maximization (EM) learning algorithm. To actually identify the class for current observations, a maximum a posteriori (MAP) decision rule is used. The authors distinguish three different classes in their implementation - tree trunks/posts,

people and vehicles. They do not present any in depth or long term evaluation of their approach, neither do they mention whether or not the vehicle carrying the sensor is moving during the tests. They show that within the one tested scene their approach classifies the seven objects in the environment correctly.

3.2.2 3D Data, Camera-Images and Fusion

Using 3D LIDARs, vision or a combination of multiple sensors allows for better classification of a wide range of objects due to the more detailed information available with these sensors. However, the time to process this large amount of information by these sensors often poses a problem for real-time requirements.

The approach presented in Douillard et al. [DUV+14] processes 3D point cloud data to extract the ground surface together with segmented individual objects. The identification of the voxels belonging to the ground surface is done by checking predefined criteria. All voxels that haven't been marked during this first step will then be segmented according to 3D adjacency. The object classification is then performed with a feature-less ICP approach, which aligns the 3D shapes with predefined templates of each class and tries to find the one with the smallest error. The authors compare this method with feature-based classification and find that their approach leads similar accuracy (92.1%) while being 30% faster. Nevertheless, for the proposed, feature-less method, templates for each object need to be specified upfront to enable classification.

The combination of 2D LIDAR data with visual information from a camera provides accurate distance measurements (LIDAR) as well as comprehensive environmental information (vision). As part of a multi-robot localization system presented in Fox et al. [FBKT00], a mutual robot detection combining visual with proximity information from a laser range-finder is introduced. Markers with different colors are used for each robot to enable its detection with local color histograms and decision trees. The detection is therefore completely independent from the robots shape.

Premebida and Nunes [PN06] present their multi-modal system based on similar sensors for detecting, tracking and classifying objects in outdoor environments. The part of the work that is based on laser range data is almost the same as presented in the previous section. After segmenting the laser range data, the result is used for feature extraction for the LIDAR-based classifiers and as additional input for the vision-based classification. The segments extracted from LIDAR data specify regions of interests for the vision-based classifier. The previously described AdaBoost learning algorithm is used for selecting a small set of features from the camera images and train the object classifiers with it. The classifiers for the LIDAR data are a GMM and a majority voting scheme [MBN04].

Since the representation of an object in a laser scan changes dramatically over time, the extracted features change as well. The authors therefore introduce a second classification approach using a majority voting scheme, which considers all hypotheses over time, until a high classification confidence is reached. Each feature is here defined as a voter actor, with a weight depending on its importance for the characterization of the object. The third classifier implemented by the authors is based on vision data and utilizes AdaBoost for the selection of a small set of features as well as the classification of each object classifier. It is not possible to detect the object category with one of the object classifiers alone, however, they all react to some simple features related to an object.

AdaBoost then learns one strong classifier from a lot of weak ones [VJ01]. The classifiers can be used independently or combined. The authors show in their evaluation on pedestrians and vehicles, that the combined classifier produces the best results for all classes.

3.3 Deep Learning Based Object Classification and Tracking

The previous section introduced approaches for detecting and tracking dynamic objects in indoor/outdoor environments as well as object classification in those scenes. Most of them were either based on classic or machine learning algorithms. In the following an overview of recent advances in image segmentation and object classification using deep neural networks will be given. This mostly includes vision data from cameras. A few implementations of video based object detection and tracking using RNNs will be outlined together with a small number of approaches using grid-maps as the input to the network, similar to the approach presented throughout this thesis.

3.3.1 Object Classification and Segmentation

Since their first application in handwritten digit recognition by Cun et al. [CJB+89], CNNs have become much more complex and successful. This is mostly due to the rapid progress in GPU development, which allows for deeper and more complex networks. Early approaches based on CNNs focused on the classification of a whole image based on the dominant object within [KSH12], [SZ14], [HZRS15], [SWY+15]. Recently, however, region-based object recognition, image segmentation, and pixel-wise classification have become popular.

To detect an object within an image, many algorithms slide a detector window over the image to find objects within, as done in the Overfeat system [SEZ+13]. In this approach objects are detected, classified and localized with bounding boxes surrounding them. A similar method, which can be further used for semantic segmentation, is introduced in Girshick et al. [GDDM13] and improved versions in Girshick [Gir15] and Ren et al. [RHGS15]. At first, category-independent region proposals are calculated, which are then used to extract features for those regions with a CNN. Finally a set of class-specific linear SVMs is utilized for the classification of the detected objects.

Although these architectures allow for a more precise object detection and localization, they still can't produce a classification result for all input data points (pixels). Some approaches leverage the sliding-window method employed in the Overfeat system to pixel-wise classify parts of the image [PSJV15], [PC15], [TMT+16]. The architectures used in these systems, however, do not allow to label pixels on the image as a whole. The final, fully-connected nature of most CNNs makes it hard to build such an architecture. They are responsible for the association of features to a classification decision but they also eliminate spatial information from the convolutions. These fully-connected layers can be transformed into convolutional layers to generate more detailed, pixel-wise labels for images. This makes the output space equal to the input space.

This idea was first introduced by Long, Shelhamer, and Darrell [LSD15] and carried on in Shelhamer, Long, and Darrell [SLD17]. The authors use e.g. the VGG 16-layer net [SZ14], discard the final classification layer and transform all fully connected into convolutional layers. Afterwards, a 1x1

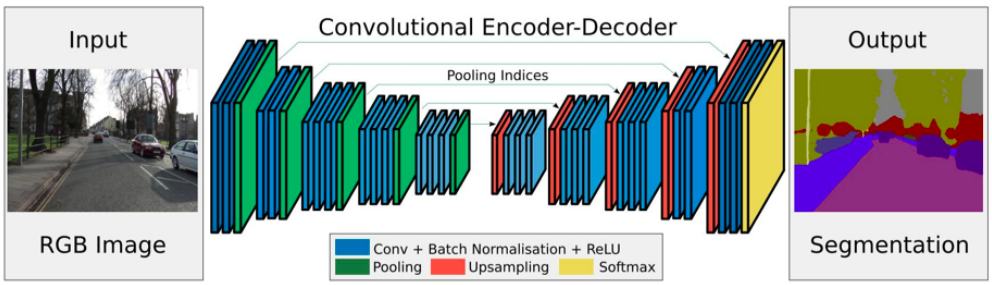


Figure 3.3: The Encoder-Decoder architecture of the fully-convolutional SegNet introduced by Badrinarayanan, Kendall, and Cipolla [BKC15].

convolution with the number of classes as channel dimension gets appended after the convolutional layers. This layer outputs a coarse segmentation of the input image which is then upsampled using bilinear interpolation to a pixel-wise labeling output.

A different method for replacing the fully connected layers with convolutional ones is an architecture based on autoencoders, introduced in section 2.1.4. The *SegNet* developed by Badrinarayanan, Kendall, and Cipolla [BKC15] implements such an architecture as seen in Figure 3.3. The first part (encoder) of the network is identical to the one used by Simonyan et al. in the previous approach, however, the part including the replacement layers for the fully connected layers as well as the upsampling (decoder) is implemented in a different way. Instead of upsampling to the input shape in one step, the authors here use a step-wise approach with multiple upsampling steps, as in autoencoders. The novelty here is that the decoder uses pooling indices from the max-pooling step in the encoder to perform non-linear upsampling in the decoder. This allows to label an image pixel-wise into different categories of objects.

These methods for object localization and classification, pixel-wise labeling or segmentation have demonstrated their ability on large RGB datasets. However, few approaches exist that apply these techniques to input data like occupancy-grid maps or other representations of 2D LIDAR data, which is commonly used for mobile robots. One related approach is presented in Brunner et al. [BRWW17], where the authors teach a machine to navigate through a maze with deep reinforcement learning. The only input data is a static map of the environment and an image of the current surrounding environment. The static map is an occupancy-grid map, like the one used for the first approach in this thesis. This map is processed by two convolutional layers followed by a ReLu activation to output a 3-channel reward map. Another system using deep neural networks for pixel-wise classification of objects in grid-maps is presented in Ondruska et al. [ODWP16]. The focus of this approach is on the tracking and prediction of object positions. Hence it will be described in more detail within the following section covering object detection and tracking algorithms using deep neural networks.

3.3.2 Object Detection and Tracking

The majority of the object tracking approaches based on deep learning, process sequential image data to detect and track people or objects. Many systems leverage the idea of fully-convolutional networks. Bertinetto et al. [BVH+16] propose a fully-convolutional system for visual tracking

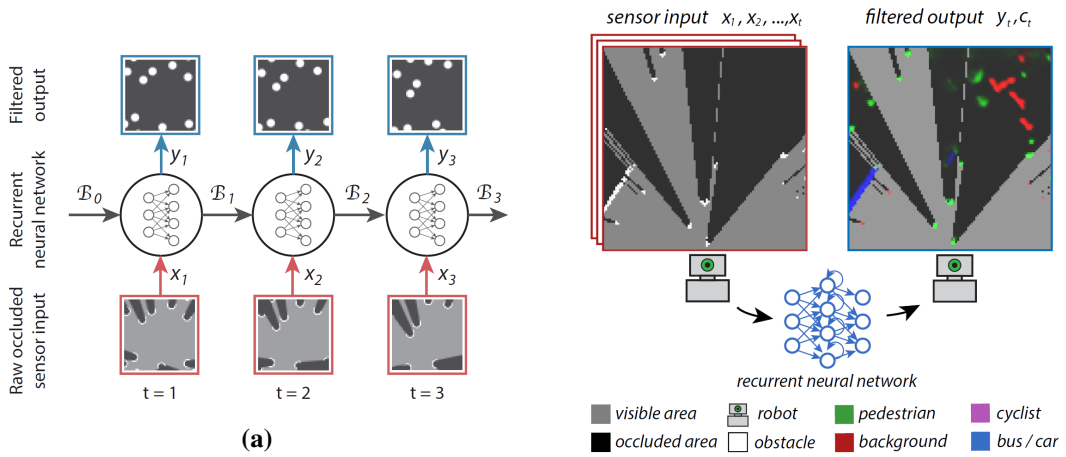


Figure 3.4: The filtering process used in Ondruska and Posner [OP16] to track (occluded) dynamic objects (a). The input/output data of the recurrent approach for dynamic object detection, classification and tracking presented in Ondruska et al. [ODWP16] (b).

based on the VGG network. An architecture with a fully-convolutional siamese network for object tracking in videos is presented in Bertinetto [Ber18]. Some initial architectures based on RNNs exist for shape independent object tracking [GGZC15], [KMM15]. Beyond this there exists one approach for legged robot detection based on visual data using a deep LSTM network [FB17].

The most closely related approach to this thesis is a novel *Deep Tracking* method for moving objects using 2D LIDAR data and RNNs with convolutional operations [OP16]. The first version of this system detects and tracks dynamic objects (circles/rectangles) from a static sensor in an otherwise empty environment as shown in figure 3.4a. In general the setup is simple compared to real-world scenarios. The sensor is static, all data is generated using a simulation and the objects are all of the same shape moving at constant speeds. This makes it straightforward to predict their motion even in occluded spaces. However, using only two recurrent layers with convolutional operations in addition to an input and output convolutional layer, the authors achieve a reliable tracking of objects even in occluded areas.

An improved version, trained and tested on real-world data from an intersection is presented in Ondruska et al. [ODWP16]. The algorithm is not only capable of tracking dynamic objects but also classifying them as visualized in figure 3.4b. The network is extended by another recurrent layer but stays otherwise the same. A 75 minute log from a stationary sensor is used for learning and evaluation of the network. Although the results prove that the algorithm works in the given setting, the authors don't provide any tests in different scenarios, i.e. moving sensor or different environment. The reason for that is, that the network maintains a *static* memory, which itself is an implicit representation of this specific intersection. This means that the network learns where static objects or pedestrians commonly occur within the image and saves those places in its internal memory. The network therefore doesn't learn to classify the objects but rather regions where an object type commonly occurs. Laser scan points that lie within a region trained as pedestrian region are classified as pedestrians. This is a suitable approach for a static sensor setup but these assumptions don't hold when the ego vehicle is moving.

Another enhancement of this approach is introduced in Dequaire et al. [DRO+16]. The data gathering for training and evaluation is done again within an urban environment. The sensor, however, is now mounted on a moving vehicle. To take the ego motion into account, a *Spatial Transformer* module transforms the output of the hidden (recurrent) layers at time *t-1* into the new frame at time *t* using additional odometry information. The experimental evaluation shows, that the system does work a little bit better with the spatial transformer but also tracks most objects correctly without it. Compared to the prior version of the system, this implementation does not support classification of objects. The reason for that might be the static memory mentioned in the previous paragraph.

Despite the fact, that the Deep Tracking system covers a vast range of problems in dynamic object detection/classification/tracking, it does not provide a sufficient solution to the problem approached in this thesis. The different versions of the algorithm are able to track and predict the trajectory of objects from a static or moving position, but they are not capable of classifying an object based on its shape; especially not when the sensor is mounted on a moving vehicle. The approaches presented in the following two chapters are therefore combining different architectures and ideas introduced in this chapter in order to form an algorithm that is able to detect and classify dynamic objects as well as estimate a relative position of them.

4 Dynamic Object Detection

4.1 Problem Definition

For the main task of this thesis, robot classification, a pre-processing of the data is performed to eliminate scan points that don't belong to a robot. One way of doing this is to label the input data into static and dynamic objects. However, when looking at the surrounding environment of a robot for a certain timespan Δt, a dynamic object might have not moved during this timespan but during a previous one. These objects will be referred to in the following as *fully-dynamic objects*. For the remainder of this thesis a fully-dynamic object will be defined as an object, that is moving during Δt or has been moving during any previous time span. In contrast, a *semi-dynamic object* denotes an object, which is observed as a moving object only during the current time span Δt. During previous time spans the semi-dynamic object either wasn't visible in the scan or was not moving. An object which had been detected as a moving object in the past but is detected as being static during Δt is defined as a *semi-static object*. Accordingly, objects that were never detected as moving ones are denoted as *fully-static objects*. Since there will be no explicit tracking of the objects, an object is redefined every single time it is detected by the algorithm, after not detecting it for at least one time step.

This chapter presents multiple approaches for semi-dynamic object detection. Their common goal is to identify all scan points belonging to an object which has been moving during the current timespan Δt. There is no need to include information about previous states of the scan points or objects. The problem to solve is therefore $P(z_{t,i} = semi - dyn | z_{t-\Delta t:t})$, the probability of the point i at time t being dynamic or not based on the sensor measurements of the past Δt time steps, $z_{t-\Delta t:t}$. The resulting output could be additionally tracked over time to detect fully-dynamic objects. This, however, is not the goal of these approaches, since it requires significantly more computational power while hardly increasing the performance of the overall system, i.e. the benefit for the following classification and position estimation. The reason for that is contingent on the fact, that a robot is generally moving, especially when it tries to increase its localization.

The LTS system, where the algorithms presented in this thesis will be integrated into already maintains some information necessary to identify fully-dynamic objects. Usually the state of the detected object needs to be stored in addition to the past Δt sensor measurements used for the detection of semi-dynamic objects. But since the LTS is continuously mapping its environment, it captures fully- and semi-static objects within its map. The problem therefore resolves to calculating the probability $P(z_{t,i} = dyn | z_t, m_t)$ of a point $i \in z_t$ at time t being dynamic using the current sensor measurement z_t and map m_t. This information is sufficient, since all points in z_t which are not captured as being occupied in m_t, most likely belong to a fully-dynamic object.

In the remainder of this chapter two different methods for semi-dynamic object detection will be presented together with a rudimentary approach for grid-map based fully-dynamic object detection, leveraging the LTS benefits. First the representation of the training data, i.e. input and output of

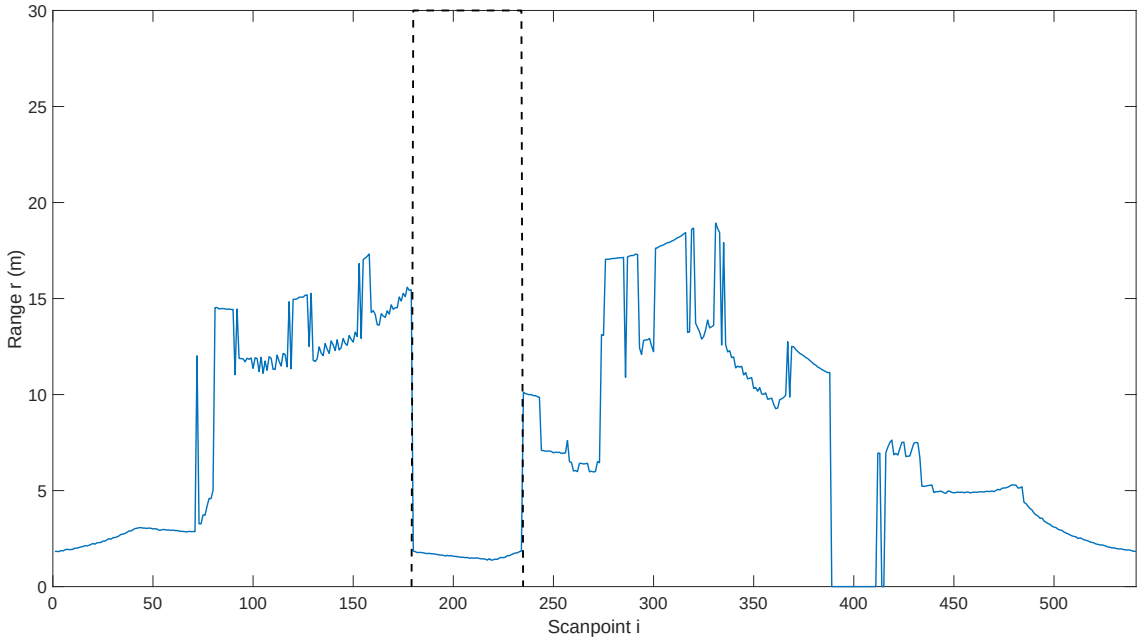


Figure 4.1: A plot of a 2D LIDAR scan with 541 scan points at one time step. The dotted line marks the scan points belonging to a dynamic object.

the networks, is described. Afterwards the general approach of the networks together with their architecture and parameter choices are presented. Finally the training the final networks is briefly described.

4.2 Data Representation

4.2.1 1D Dynamic Object Detection

One advantage of 2D LIDAR data is its low dimensionality and the associated small amount of data being processed (compared to 3D LIDAR or vision data). The downside that comes with this benefit is the lack of unique features. The visualization of a 2D LIDAR scan from one time step is shown in figure 4.1. Although this scan is composed of 2 dimensions (angle, range), we will refer to this representation in the following as being 1D, since the angular step size is constant and the data representation of such a scan is therefore a 1D array. Finding features or indicators of an object in this representation is a difficult task for people as well as machines. The dotted line shows the part of the scan which captured the silhouette of a robot, but without knowing it a guess would be almost impossible. One idea to retrieve more features is to stack multiple consecutive scans into one array. The result of such an operation is shown in the plot of figure 4.2 for the same scan including 9 previous scans. Although this representation does not really help to detect specific objects, it reveals spatio-temporal features. The path that the robot took while this scan was captured is easily visible to a human observer - the dark blue, step-like line of measurements in the center of the image. The scans for those figures were captured using a simulated environment, the image therefore looks clean since only artificial Gaussian noise is introduced in the scans.

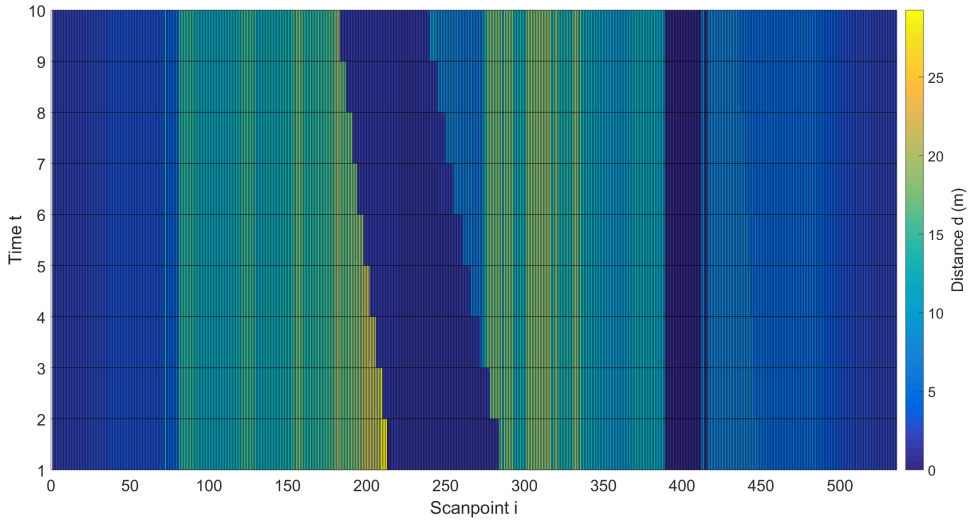


Figure 4.2: A plot of multiple (10) consecutive 2D LIDAR scans with each 541 scan points gathered from a static sensor. The blue measurements in the center around scan point 250 belong to a moving object. The changing location of the measured object form spatio-temporal features if stacked as in this example.

Although this representation reveals dynamic objects, it also comes with a major disadvantage. The sensor used for the data gathering in this scenario was placed on a static platform which explains the clear and consistent distance measurements. If the sensor is mounted onto a moving platform, the ego motion of the platform distorts the data representation completely. An example of the resulting data is shown in figure 4.3. As a second approach, the data set used for training the network could be transformed into a consistent coordinate frame for each k consecutive scans. This yields a representation as seen in figure 4.4, which now contains 720 scan points to represent the 360 degree environment around the robot. All previously shown images used scans with a angular range of 270 degree but since they are now transformed into one coordinate frame, it can happen that a scan point breaks the initial 270 degree boundaries. Due to uncertainty in the sensor measurements as well as the ego vehicles localization, this representation does not look as accurate as the static measurements in figure 4.2. Additionally, it is not possible to maintain a consistent representation when transforming the data into one coordinate frame, since there could be multiple distance measurements for the same angular increment. This approach was therefore tested but will be not further described or evaluated, since it didn't show any promising results. Instead we decided to continue with different approaches, as introduced in the following.

The input for the 1D static sensor approach is a txn dimensional array, where t is the number of stacked scans and n the number of scan points retrieved from one scan. The output is a 1D Array with ones for scan points containing semi-dynamic objects and zeros for all others.

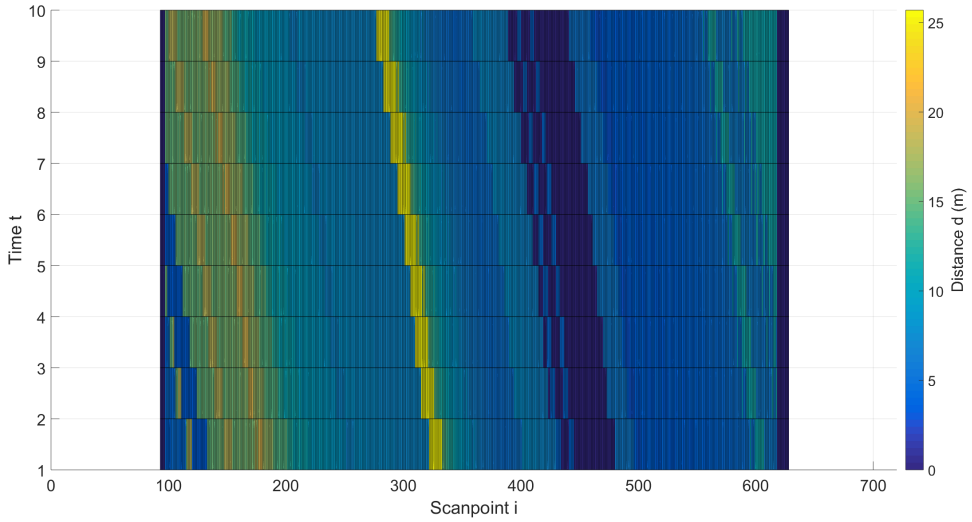


Figure 4.3: A plot of multiple (10) consecutive 2D LIDAR scans with each 541 scan points gathered from a sensor on a moving platform. The two light blue measurements around scan point 400 that continue to stay in the dark blue area throughout the 10 consecutive scans belong to a moving object. Due to the movement of the ego vehicle, the object can't be distinguished from the static background as in figure 4.2

4.2.2 2D Dynamic Object Detection

Due to the issues in handling the raw 1D sensor data a second approach and data representation are introduced for the purpose of dynamic object detection. Most CNNs work on image data and some non-learning approaches leverage spatio-temporal features of dynamic objects [QCS+16] to improve the detection. Hence, the second data representation used in this thesis transforms all scan points from polar coordinates (range, angle) into a map using cartesian coordinates (x, y). This discretized view of the environment, which is basically an occupancy grid-map, is shown in figure 4.5a for one sample scan. The image shows a $224x224$ pixel map with a resolution of $0.05m$ per pixel and the sensor being at the center of the map. This is the default setup for generated grid-maps used throughout this thesis. Using grid-maps as an input for the network reveals more features on the one hand and leads to more structure in the data representation on the other, but it also produces significantly more data to process in the network. A single scan is transformed into an mxm dimensional grid-map. We only use square sized images (maps) with an edge length of m, since the robot carrying the sensor is located at the center of the image and has a 360 degree view. In a square sized grid-map the range of the robots perception is therefore uniform into all directions.

Although this representation already includes a lot more features and details than the previous 1D representation (see 4.1) there are no spatio-temporal features visible yet. Instead of stacking the data (3D) or using multiple consecutive scans as the input for the network, we implement another idea presented in Qin et al. [QCS+16] based on accumulating the input. Compare to the other two approaches, this one results in a significantly lower computational complexity. By accumulating the grid-map of k consecutive scans, a resulting map is obtained where higher values are an indication

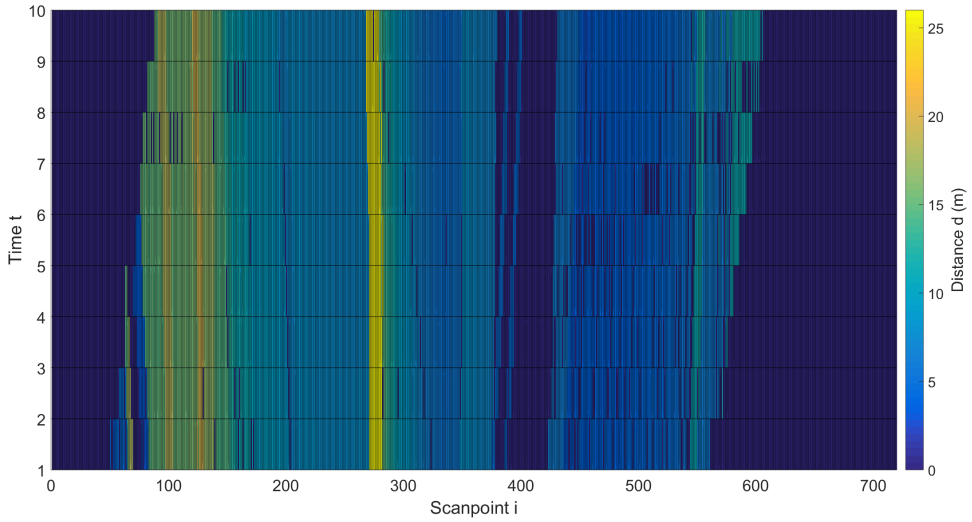
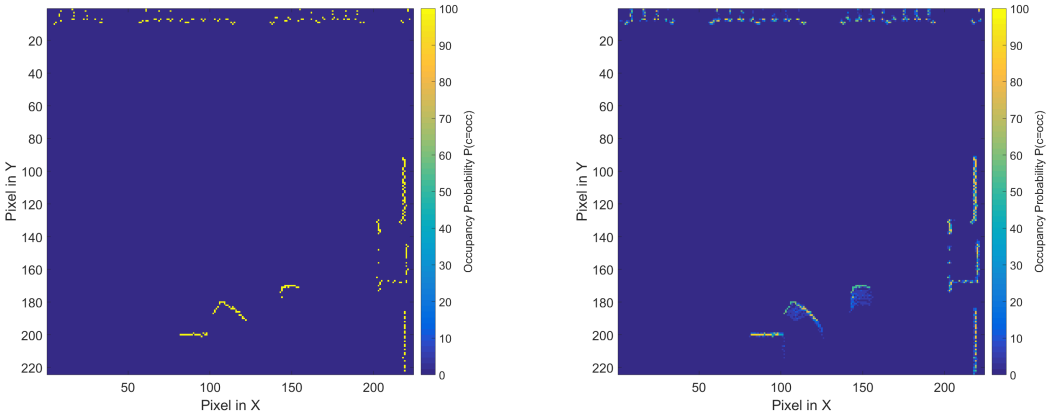


Figure 4.4: A plot of multiple (10) consecutive 2D LIDAR scans with each 541 scan points gathered from a sensor on a moving platform. All scans are transformed into the current frame ($t = 10$). The scans are the same as in figure 4.3. The light-blue moving object is now easier to detect, since the surrounding background does not shift over time as much as in the non-transformed data.

for static objects since these objects are always detected at the exact same position (with some uncertainty). Due to the ego-motion the grid-maps have to be transformed into one coordinate frame again. This induces some error as well. The result of such a map for $k = 10$ is shown in figure 4.5b. Beyond summing all scans from t to $t - 9$, the most current scan at time t here is also multiplied by a factor of 10 to make it clearly identifiable as the most current one for the network.

The map shows two moving robots, both in the bottom left part of the image. The rectangular shape on the left just became visible to the robot with the sensor, which is again at the center of the map. Therefore there is no trace behind it as it is for the semi-circular robot on its right. This robot is detectable as a moving object through the clear spatio-temporal features - the trace of decreasing values behind it. This trace, however, also occurs when the sensor platform moves e.g. around a corner or a box like the one in the left center of the image. Here the right side of the box is still visible and therefore has a high value, the upper side is not visible anymore, but was before in the past k scans. Nevertheless the pattern looks different in these two cases, since moving objects have wave-like decreasing traces and objects like these static boxes only have a line with smaller values.

The input to the network used in this approach is an array containing a grid-map of size mxm, where m is the number of pixels in the grid-map into one direction. The output is a two dimensional array of the same size as the input with zeros for pixels/positions which do not contain a semi-dynamic object and ones for all that do.



(a) **(b)**

Figure 4.5: The grid-map generated from the laser scan for one time-step t (a). The color indicates whether the space (pixel) is occupied or free (blue). The same scene with accumulated data of the past 10 time-steps (b). The grid-map at the current time t is additionally multiplied with 10 to highlight it.

4.2.3 2D Dynamic Object Detection With Grid-Map

Both of the so far introduced approaches leverage spatio-temporal information for the detection of semi-dynamic objects in a laser scan without including any other information. In the majority of the cases, a mobile robot, however, does have more information available that can aid in finding dynamic objects, e.g. an occupancy grid-map. This map usually contains fully-static objects that were detected during a SLAM process. But since the robots which will be using this object detection are also running an LTS (see 1.3), they continuously map the environment and therefore it also includes semi-static objects that are not moving during the time the robot captures them. We utilize this behavior to calculate the difference between the grid-map m_t and the scan-map x_t at time t. Hence the input to the network is an image with two frames (scan, map) instead of one, as in the previous approach. The output is illustrated in figure 4.6, together with the two inputs. As in the previous approach it is an image of the same size as the input ones. Again a 1 denotes a (semi-)dynamic object and a 0 everything else.

4.3 Approach

For each of the data representations presented in the previous section, an approach based on CNNs will be introduced in the following. Since the input and output of the networks differ from each other the presented network are different as well. The purpose of implementing three different approaches to solve the same problem is to compare and evaluate them against each other in order to find the most promising approach.

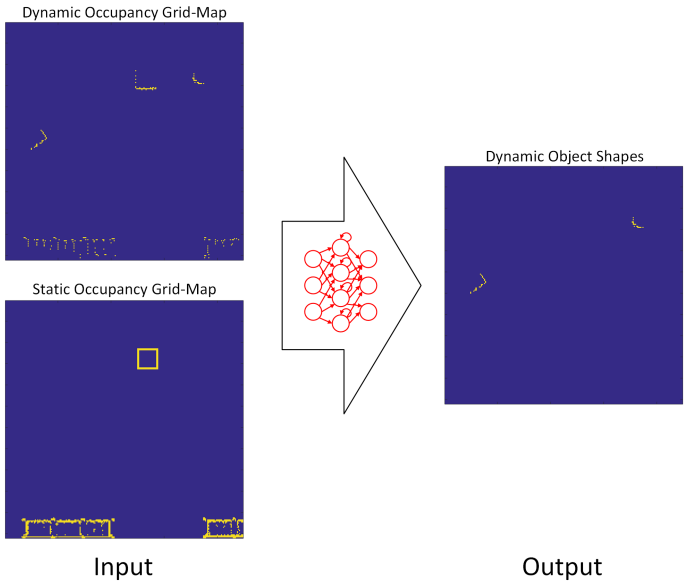


Figure 4.6: The input of the network using dynamic scan data together with a static grid-map is shown on the left. The right side displays the output of the same size where only the pixels belonging to dynamic objects are marked.

4.3.1 1D Dynamic Object Detection

After introducing the in- and output of the algorithms, we will now elaborate on the idea as well as the network architecture used to solve the problem. The input data for the first approach using stacked 1D sensor data contains spatio-temporal features on a high level. Compared to an RGB image for example, which has different features on different levels of abstraction (colors > pattern of a carpet > a dogs face), the input for this approach only has features on one level of abstraction - the difference in range measurements over time. Therefore a relatively shallow network architecture has been chosen, with only three max-pooling operations. The whole architecture can be seen in figure 4.7. The nxk dimensional input, with n being the number of scan points in one scan measurement and k being the number of stack scans, is processed by three consecutive convolutions, each followed by a max pooling operation. This allows the network to learn features on three different levels of abstraction to then map this information on the n dimensional output. In the up-sampling process, only one dimension is up-sampled, since the goal is to predict a dynamic object labeling only for the most current scan. In order to avoid over-fitting, a dropout with rate 0.1 is applied after each convolutional layer, except for the final one. This neural network architecture is similar to the SegNet [BKC15] architecture, which itself is based on the idea of autoencoders, described in section 2.1.4. Normally the features learned in the encoder phase are processed afterwards using fully-connected layers. This network, however, replaces these with a step-wise up-sampling and convolutional layers to keep the spacial information of the features. The final output is then a one dimensional array after a sigmoid function, generating values between 0 (static) and 1 (dynamic).

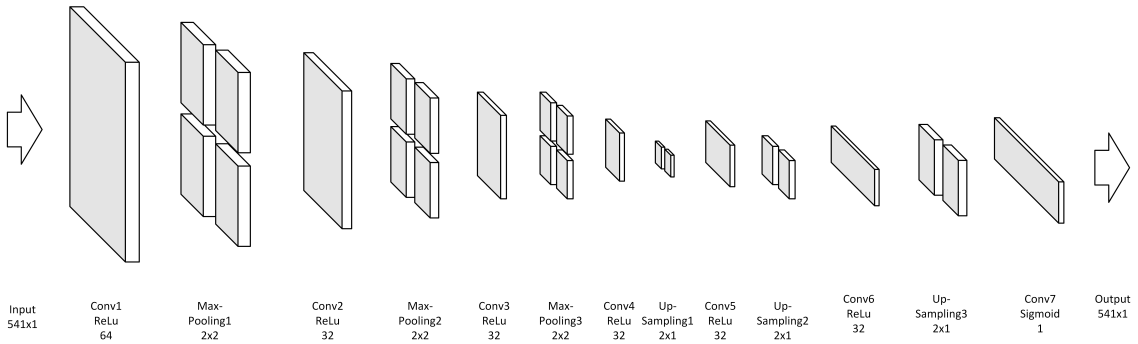


Figure 4.7: The architecture of the network using stacked 1D scans as input and with a 1x541 output for each scan point, specifying whether it belongs to a dynamic object (1) or not (0). The numbers below the descriptions denote the size of the input, output and the pooling layers as well as the number of hidden layers in the convolutional layers. The variables n and k are defined by the number of scan points for one time frame and the number of scans stacked on top of each other, respectively.

4.3.2 2D Dynamic Object Detection

The idea behind the first 2D approach is that the network learns to detect spatio-temporal features, as the tail-like shapes of the robots in figure 4.8. For this purpose we present an architecture inspired by the networks introduced in [OP16], [ODWP16] and [DRO+16]. The network consisting of two consecutive ConvLSTM layers followed by a final convolutional layer for the output is visualized in figure 4.9. We use the ConvLSTM layers as presented in SHI et al. [SCW+15], without a static memory as implemented by Ondruska et al. [ODWP16]. Since the sensor is mounted on a moving vehicle in our scenario, the static memory would not increase the power of the network to identify moving objects. A dilation rate as used by Dequaire et al. [DRO+16] is not necessary in our case as well an even leads to worse performance, as will be shown later in section 6.5.1. A dilation rate increases the receptive field of stacked convolutions. This is especially useful if objects with high velocities are tracked or a sensor with a low publishing frequency is used. In our setup we use a sensor with a publish rate of $12Hz$ and the dynamic objects do not move faster than $2m/s$. The network additionally applies a dropout with rate 0.1 to the convolution of the input at each ConvLSTM layer. A dropout with rate 0.05 is used for the output of the convolution applied to the recurrence in the network.

The ConvLSTMs used in this network are *stateful*. This means, that instead of feeding a fixed length sequence of data into the network, it is trained by receiving one data example per time-step. The state of the network is not reset after each example but after the complete training set. The layers therefore keep their state, i.e. the input to the ConvLSTM at time t depends on the hidden state h_{t-1} calculated by the previous data sample. This technique allows to train a network that is able to work on continuous data, as received in real-world examples by the system. Instead of explicitly storing the past n input samples, the network learns to remember this data. This reduces the computational time of a forward pass through the system as well as the memory requirements during the online usage of the network.

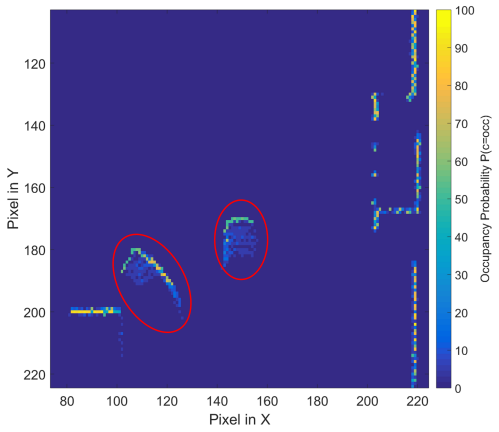


Figure 4.8: A snippet of the networks input visualizing spatio-temporal features of two moving robots with different shapes (red ellipses).

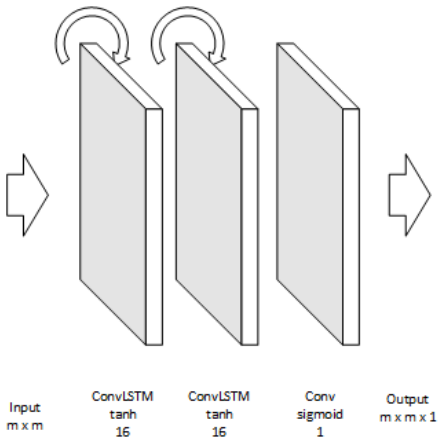


Figure 4.9: The architecture of the network using accumulated 2D grid-maps as input.

The convolutional operations of the ConvLSTMs detect the accumulated spatio-temporal features and the recurrence stores the state of these objects to increase the prediction accuracy of the network at each time-step.

4.3.3 2D Dynamic Object Detection With Grid-Map

The downside of the previously introduced architecture and the given input/output data is the computational effort associated with learning it, as well as running a forward-pass through the system. However, the system, that the algorithm presented in this thesis will be integrated into, already provides parts of this information in form of a long-term occupancy-grid map. The last approach uses this information to generate dynamic object information about the environment. If there would be no uncertainty in the robots localization and its sensor measurements, this could be easily done by calculating the difference between the long-term occupancy grid-map and the sensor based map. Since there is always an uncertainty in these two estimates, using a basic CNN, simplifies the task of retrieving the points in the sensor map that do not belong to points in the long-term occupancy grid-map. The CNN is somewhat robust to small shifts or turns in the sensor data compared to a simple substitution of the static grid-map from the sensor grid-map. Due to the convolutional operations used in the layers, the network is still able to associate the related pixels. This robustness against noisy input data is achieved by using training data of a robot using its own localization instead of the ground-truth. This is elaborated on in detail in section 6.7.1.

The network implemented to achieve this purpose only contains two convolutional layers without any pooling. One for calculating the difference between the two input frames and a second one generating an output. The reason for that is, that we don't need to learn any specific features or shapes. The network is only supposed to compare the two input frames and output the difference between them. The first convolutional layer uses the ReLu activation function and the output layer

Network	Learning Rate	Decay	Dropout	Batch Size	Seq. Length	Epochs
1-D CNN	0.0001	0.00001	0.1	20	10	100
2-D LSTM	0.0001	0.00001	0.1	1	10	100
2-D CNN	0.0001	0.00001	0.1	10	1	10

Table 4.1: An overview of the most important hyperparameters for learning the networks. The sequence length refers to the number of consecutive scans stacked/accumulated for the first and second approach, respectively.

sigmoid. The main layer is composed of 32 convolutional filters. To account for higher uncertainty in the input data, i.e. a larger shift/turn of the data, a kernel size of $5x5$ was chosen. As in the previous approaches a dropout with rate 0.1 is applied to the output of the first convolutional layer.

4.4 Training

The two approaches presented for the detection of semi-dynamic objects are trained on pre-labeled data, where an object is defined as being semi-dynamic if it has exceeded a certain movement threshold during the past Δt scans. All scan points belonging to this object are thus marked as being dynamic (1). The third network is trained on data where the labels are generated by defining all scan points that belong to a fully-dynamic object as being dynamic (1). In both cases, the rest of the output array/map is defined as being non-dynamic (0) in the output.

Although the presented approaches differ in network architecture and data in-/output, they were all trained in a similar manner - using supervised training. All networks were trained using the Adam optimizer instead of the standard SGD. The reason for this being the reduction in time until the network converges with Adam due to the adaptive learning rates used for training the network. It further outperforms SGD in dealing with sparse gradients, which specifically applies to the presented networks using grid-maps. Since the output of the networks is binary (dynamic/not dynamic), independently from its shape, all networks use a standard binary cross-entropy loss function (see section 2.1.3).

An overview of the most important hyperparameters for the learning process can be seen in table 5.1. The sequence length specifies the number of stacked scans for the 1D approach as well as the number of accumulated scans in the first 2D approach. The batch size is 1 for the network using ConvLSTM layers due to the fact that these layers are stateful, as introduced in section 4.3.2. If a larger batch size was chosen, the data examples with the same index in each batch must be consecutive scans. This data representation does not make sense for the introduced application, since we need a network that is able to work with continuous data.

Further details on the exact training data and network specific parameters, will be described as part of the evaluation in chapter 6.

5 Mobile Robot Classification and Position Estimation

5.1 Problem Definition

The detection and classification of mobile robots in a 2D LIDAR scan is a challenging task. The previous chapter 4 introduced several techniques for pre-processing the data in order to reduce the complexity of the problem for the classifier. Within this chapter, the approach for classifying the different robot shapes, while simultaneously estimating a position for each hypothesis is presented. The problem can be visualized using a graphical model, as illustrated in figure 5.1. It can be defined as calculating $P(x_{r,t}|z_t, h_t)$, the probability of the position hypotheses x_t for each robot type r at time t based on the current laser-scan of the ego robot z_t as well as the hidden state of the previous time-step h_t. Since the calculation only depends on the current state, this is a first order Markovian problem.

In the following, two different data representations used for training the network are introduced. Afterwards the approach is presented including the network architecture and the subsequent position estimation algorithm. Finally the training of the network is elucidated including a custom loss function.

5.2 Data Representation

The impacts of the data representation of 2D LIDAR data were already discussed in section 4.2. A transformation of the data into a 2D occupancy grid-map was introduced for a better richness of features. The input for training the classification network is therefore similar to the already presented types. Although in the final system, the dynamic object detection can be used as a pre-processing step, this was not done for training the network, since it was not necessary. Using labeled data, it was possible to learn the network either with only those scan points belonging to dynamic objects or with the complete scan data. This will be elucidated as well as evaluated further in the experimental results 6.10.

The first data representation used for training the network is the 2D occupancy grid-map obtained by transforming the polar coordinates obtained by the sensor into cartesian coordinates. A visualization of this can be seen in figure 5.3a with ones specifying occupied and zeros free space. A drawback of this representation and also a disparity to the standard occupancy-grid, is the fact, that only free and occupied space is defined. There is no measure for unknown space. To understand this better, figure 5.2 illustrates two shapes which look almost identical to a convolutional network using this representation. However, the semantic interpretation of these two shapes is completely different from each other. Still the network wouldn't differentiate between those two shapes, since

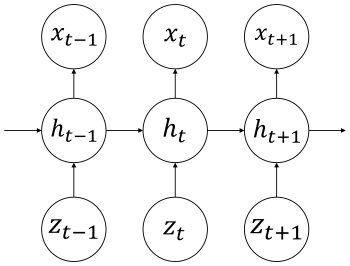


Figure 5.1: The classification problem represented as a graphical model with z being the sensor input, h the hidden state of the system and x the desired positions of the different robot types.

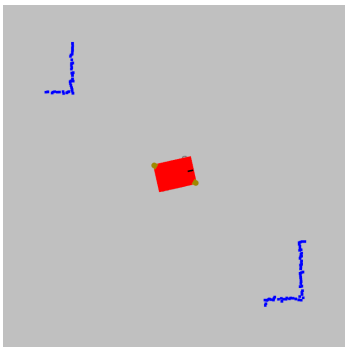
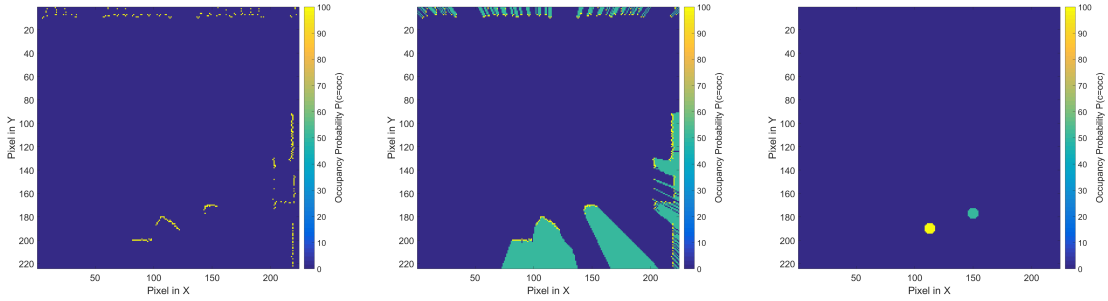


Figure 5.2: Two identically looking shapes detected by the 2D LIDAR. The shape in the top left corner is a rectangular shaped robot, while the one in the bottom right is an open corner of two arranged walls. Without the information about the sensor position, these two shapes are semantically identical.

the convolutional operations are applied to each part of the image and objects are detected no matter where they are and independent from an overall context (in this case the relative position to the sensor). A second version of the occupancy grid-map is therefore used for training the network, where the space that was neither hit by a sensor beam (occupied) nor is part of the path from the sensor to the object (free) is defined as being unknown. This increases the networks ability in distinguishing shapes as the ones discussed before.

The occupancy grid-map including unknown space is created by ray-tracing a beam from the center of the image to every single pixel on the border of the image. First the points on the line connecting the center pixel with the border pixel are identified by using Bresenhams line algorithm [Bre65]. Afterwards each point on the line is checked whether or not it belongs to an object. All points not belonging to an object are denoted with a 0, all points belonging to an object with a 2 and unknown pixels are identified by a 1. The resulting map is visualized in figure 5.3b. In the remainder of this thesis we will refer to this map as a *fully (occupancy) grid-map*, opposing to the previous version, which is referred to as a *binary (occupancy) grid-map*.

Independently from the input representation (binary/fully grid-map) the output of the network is always a $kxkxc$ dimensional array, with k being the number of pixels in x/y for the input image and c being the number of classes. The number of classes is in this case not equal to the number of different robot types that are supposed to be detected. It is equal to the number of different



(a) **(b)** **(c)**

Figure 5.3: The in- and output of the classification networks. The input without marked unknown space (a). The input with marked unknown space (b). The output for both networks, where each colored spot marks the position of a robot and the blue colored space everything else.

robot types with an additional type added for everything in the image that neither belongs to e.g. robot A or robot B. We could distinguish further between e.g. free space and static obstacles or even label more objects but for a simplification of the learning phases, this has not been taken into consideration for this thesis. Each array in the third dimension of the output is a heatmap over the possible locations for objects (or non-objects for index 0). An example output of the network is shown in figure 5.3c for the respective inputs in figures 5.3a and 5.3b. To take a wider range of pixels around the actual robot position into consideration, the real robot position is marked by a surrounding, filled circle. All pixels that lie within this circle have a value of 1 for this class. In the first approach the output of the network was chosen to be a multi-modal gaussian around the robots position. The performance of the network, however, was much better when instead of using these partial probabilities (e.g. 0.4) only 0 for non-robot pixels and 1 for robot pixels were used.

5.3 Approach

In order to solve the introduced classification problem, a network architecture based on a combination of CNN and ConvLSTM layers was developed, as shown in figure 5.4. The first part of the network, the encoder, projects the input map down to a smaller feature space. The two convolutional layers in between the encoder and the decoder learn different features of the robots shapes, e.g. arcs and lines of certain lengths. After projecting these features back into the original space (decoder) the two stateful ConvLSTM layers learn to memorize important information about the objects dynamic state. This increases the complexity but also the power of the network. The ConvLSTM layers improve the detection rate of dynamic objects that are once identified as a robot over a longer time-span. Finally a heatmap is predicted by a softmax activation over multiple classes. The kernel size of the last convolutional layer is chosen to be $5x5$ because it increases the networks ability to generate the circle around the robots position, since the receptive field is larger. To keep the network from overfitting, dropouts with a rate of 0.1 are applied after each convolutional layer except for the third and the final convolution. The ConvLSTM layers use a dropout rate of 0.1 for the convolution of the input data and a rate of 0.05 for the recurrence. These parameters were chosen based on cross-validating the network with different choices.

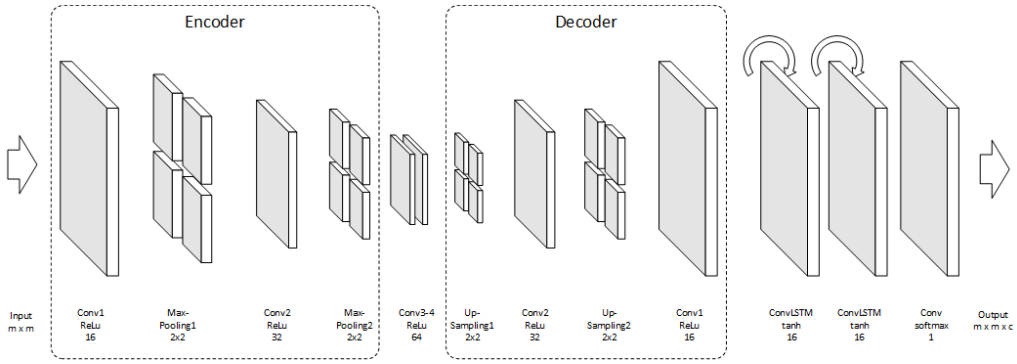


Figure 5.4: The network architecture of the classification network. It is composed of an encoder-decoder part with additional layers in between. Two ConvLSTM layers follow on the decoder. The final output layer is a convolutional one with a sigmoid activation function.

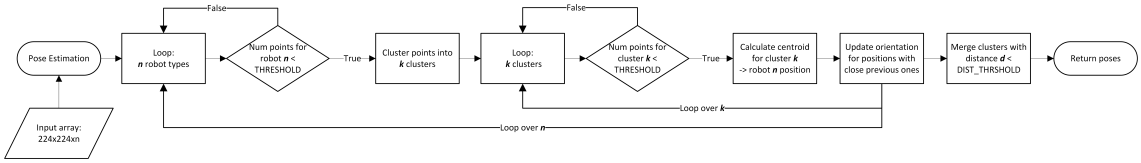
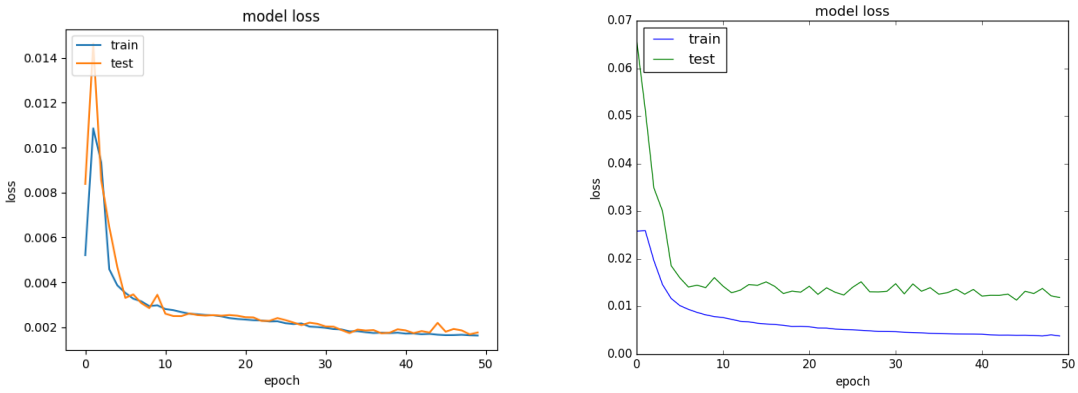


Figure 5.5: The process according to which a position estimation is performed for an output of the network.

Some of these alternative parameter choices will be evaluated in section 6.5.2. The evaluation further supports the assumptions made in this section by comparing the performance of different architectures.

As shown in the previous section, potential robot locations are marked in the output data with a filled circle of ones around the actual robot position. This means, that ideally, the network outputs exactly the same, i.e. the robots predicted position is located at the center of this circle. The next step in the position estimation is therefore depicted of a clustering and center calculation algorithm. The flow diagram for the algorithm is shown in figure 5.5. The input for this procedure is the predicted heatmap of the network over multiple classes. The algorithm first loops over each robot type and checks whether enough points in the array exceed a specified threshold. If this evaluates true, all of these points are clustered based on their location in the 2D heatmap using a hierarchical clustering[1] algorithm. In a loop over all clusters, another check verifies that the cluster contains enough points to be considered a robot hypothesis. Finally the centroids for all clusters that passed this check are calculated based on their 2D cartesian coordinates in the real world. These centroids are then returned as potential robot positions for the respective robot type. For each hypothesis an additional probability value could be calculated based on the number of points detected for the robot and the predicted value of the network.

[1] Hierarchical clustering is a method for partitioning data into optimal groups/clusters based on a defined distance measure (e.g. nearest point) [Joh67].



(a) **(b)**

Figure 5.6: The loss during 50 epochs of training with a standard categorical cross-entropy loss function (a) and the adapted weighted version (b).

There can be multiple hypotheses for a single robot type. One approach to minimize the number of hypotheses would be to merge close ones of the same type. However, the LTS system which will be using these robot positions is able to handle multiple hypotheses. Therefore this representation is sufficient for the purpose of this thesis.

5.4 Training

The training of the network is conducted in a supervised manner, similarly to the procedures described for dynamic object detection in section 4.4. As before an Adam optimizer is used for training the network. Since in this case, a multi-class classification is performed, a categorical cross-entropy (see section 2.1.3) loss function needs to be used. The problem that arises with this setup is, that only few cells of the output array actually belong to the shape of a robot. The majority of the cells is either free-space or belongs to another static obstacle. Therefore, the standard categorical cross-entropy (see equation 2.4) is modified to account for this imbalance as follows:

$$H_{categorical}(y, \hat{y}) = -\sum_{i} y_i log \hat{y}_i w_i. \tag{5.1}$$

In this *weighted categorical cross-entropy*, an additional class-weight w_i is multiplied with the calculated loss for each data point. This allows the network to converge faster. The improvement can be seen in figure 5.6a and 5.6b, visualizing the loss of the network after a training of 50 epochs using the standard categorical cross-entropy and the modified version, respectively. The loss value cannot be used in this scenario to compare both functions, since the weighted version manipulates the loss. A higher loss value therefore doesn't mean that the network learned a worse model than without the weights applied. The positive influence of the weights is, however, clearly visible in the gradient of the plotted data. After about 10 epochs of training, the test loss using the weighted loss function converged. The test loss using the standard categorical cross-entropy on the other hand is still decreasing at the end of the training (epoch 50). The network with the weighted function

Network	Learning Rate	Decay	Dropout	Batch Size	Epochs
2-D Class	0.0001	0.00001	0.1	1	50/50

Table 5.1: An overview of the most important hyperparameters for learning the networks. The sequence length refers to the number of consecutive scans stacked/accumulated for the first and second approach, respectively.

therefore sooner reaches a point where the performance of the network does not increase anymore by continue learning. An evaluation of the performance of networks trained on both loss functions will be given in section 6.5.2.

An overview of the most important hyperparameters for the learning process can be seen in table 5.1. To increase the trained performance of the network and decrease the time till convergence, the classification networks are first trained for 50 epochs solely on the scan points belonging to a robot. This allows the network to learn features of the robot shapes more easily. Afterwards the network is trained another 50 epochs using the full scan including other dynamic and static objects as an input for the network. A batch size of one is chosen due to the statefullness of the ConvLSTM layers (see section 4.4).

Further details on the exact training data and network specific parameters, will be described as part of the evaluation in chapter 6.

6 Evaluation

Throughout this thesis, different approaches for dynamic object detection as well as object classification and relative position estimation were presented. After introducing the hardware and robots together with the environments used for the conducted experiments, there will be an extensive evaluation. This includes an evaluation of the algorithms' architectural choices as well as an investigation of their performances on unknown as well as real-world data. Furthermore the impact of the target robots distance to the ego robot is evaluated together with the impact of the ego robots localization error.

The entire data-set used within this chapter was collected specifically for the purpose of evaluating this thesis. A comparison of the algorithms' performance against a state-of-the-art object classification algorithm was therefore not performed. In addition to that, most of the related algorithms are designed for similar but not identical use cases, e.g. in an outdoor environment or with a static sensor.

6.1 Implementation & Hardware

The networks which were introduced throughout this thesis were all implemented using the Python framework Keras with TensorFlow as backend. Except for the adapted loss function presented in section 5.4, all networks were constructed using only Keras layers. Since the overall LTS system as well as the entire navigation stack used for the robots is implemented in the Robot Operating System (ROS) framework, for the purpose of live evaluation we implemented an online classifier for ROS using Python. This classifier was tested on a notebook with an Intel Core i7-6600U CPU running at 2.60 GHz and $16GB$ of DDR3 RAM but without any additional graphics card except for the integrated Intel graphics in the processor. We achieved a detection rate of around $3 - 5Hz$ with pre-processing of the data. This is acceptable for robot detection using a sensor which commonly publishes at $10Hz$. It is not necessary in this scenario to publish information about the robots' position at $10Hz$. The training of the networks was performed on a computer equipped with an AMD Ryzen Threadripper 12 core CPU with max. $3.5GHz$ an Nvidia 1080Ti with 12 GB of GDDR5X RAM as well as $32GB$ of DDR4 RAM. The training of the classification networks as well as the dynamic object detection LSTM took around 10 hours (100 epochs) until convergence. The 1D stacked approach as well as the approach using static grid-maps as a second input only took around 4 and less than 1 hour, respectively.

6.2 Robots & Sensors

Two mobile robots were used for data gathering in a simulated or real world. The Care-O-bot[1] and the Rob@Work[2] both have been developed by the Fraunhofer IPA. The first one can be seen in figure 6.5b and its semi-circular shape in figure 6.5d. Its domain is mainly set in personal, service and domestic robotics. It can be used in retirement homes, hospitals or stores to serve with information or observe its environment. In its full configuration, it consists of an omnidirectional base, with a torso that has two 7 degrees of freedom arms attached to it. Within the base there are three safety laser scanners of the model Sick S300[3] covering 360 around the robot together. Each of them has an angular range of 270 with a resolution of 0.5 scan points per angle (541 overall) and a distance of up to $30m$.

Two of these laser scanners are also built into the Rob@Work which is shown in figures 6.1b. The Rob@Work is equipped with an omnidirectional drive as well and is primarily built for industrial environments, such as car manufacturing or pharmaceutical factories, hence the simple design with one (optional) arm. This arm can grab objects and put them onto the robot itself to deliver them to another place or person. Compared to the almost round shape of the Care-O-bot this robot has a strict rectangular shape as visualized in figure 6.1d. These two robots were used for all data gathering as well as evaluations conducted throughout this thesis.

6.3 Environments & Labeling

The collection of training as well as evaluation data was performed in simulation as well as real-world environments. The Gazebo[4] simulator, which is part of the ROS[5] framework, was used for gathering all the simulation data for training and evaluation. An important aspect for the training data collection is the object diversity in the different environments to allow the network to generalize well across many environments. Hence, a large environment was designed for the data collection including multiple rooms with different *themes*. A visualization of the world in gazebo can be seen in figure 6.2a. It includes many different objects with varying shapes. The data collection has been conducted using three robots in one of the rooms marked with blue in figure 6.2b. One robot is used as a moving source scanner and the two other robots are a Care-o-Bot and a Rob@Work, respectively. The robots receive random goals within the blue boundary of a room and collect data for five minutes (3000 samples). Further, static obstacles (square boxes and columns) are added to the room to increase the shape diversity in the data.

The data for the evaluation is collected in the room marked as red in figure 6.2b and the upper left blue room. The latter one is a machine-room already used for training the networks. The red room (warehouse) has not been used for training the network at all. Three separate cases of evaluation data are distinguished in the following to compare the networks performance in an environment it

[1] https://www.care-o-bot.de/en/care-o-bot-4.html
[2] https://www.care-o-bot.de/en/rob-work.html
[3] https://www.sick.com/de/en/opto-electronic-protective-devices/safety-laser-scanners/s300-professional/c/g187237
[4] http://www.gazebosim.org/
[5] http://www.ros.org/

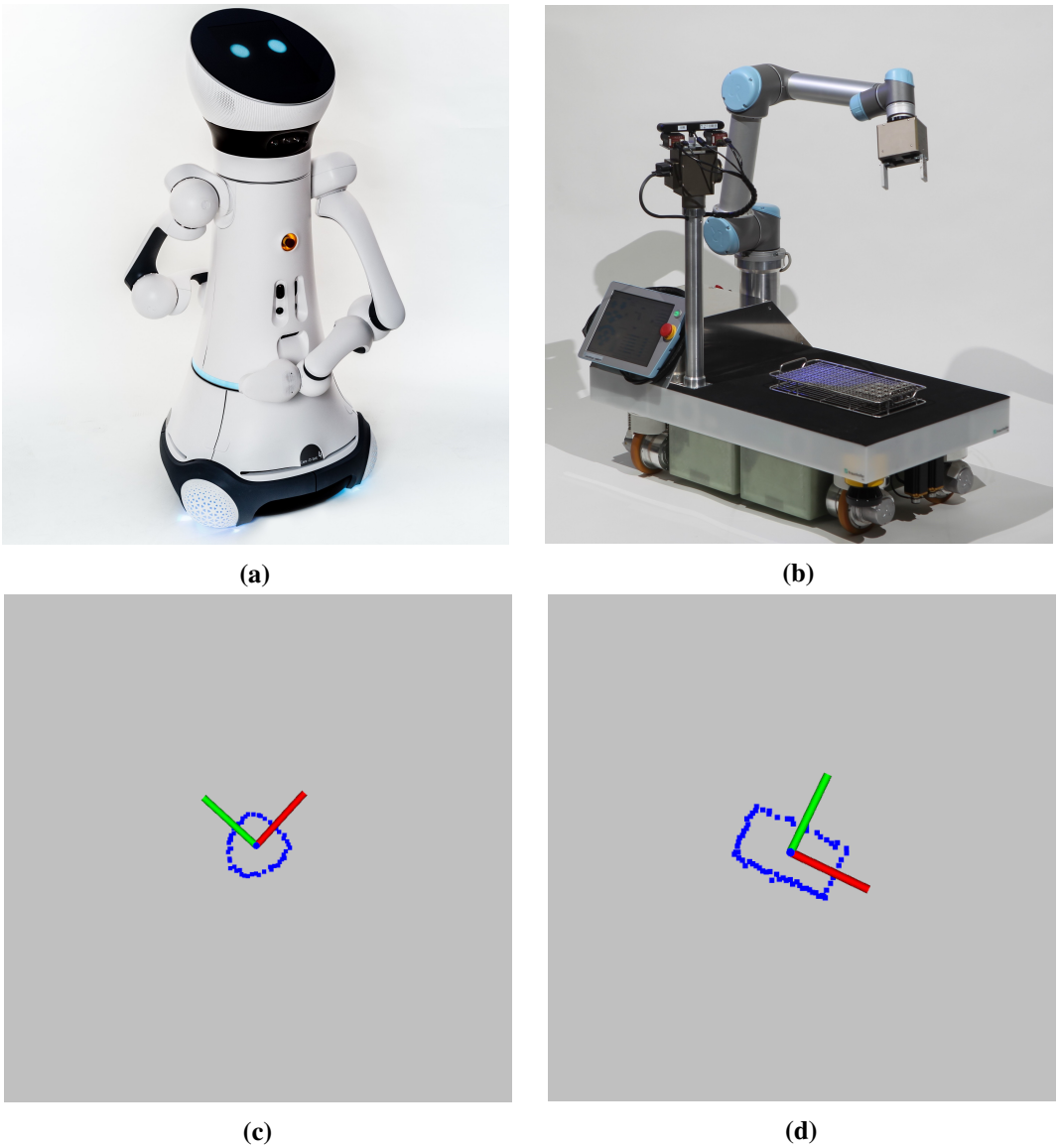
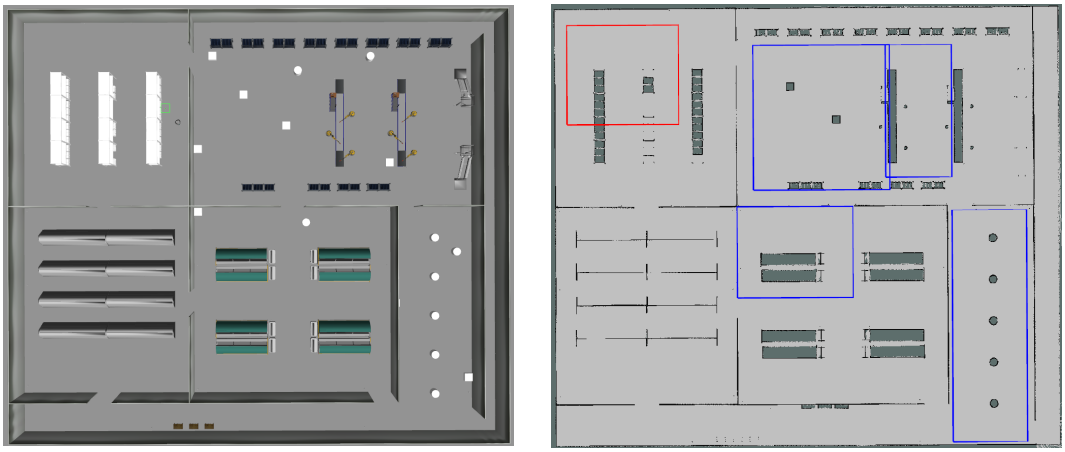


Figure 6.1: The two robots used for the evaluation. A Care-o-Bot with a semi-circular shape (a,c) and a Rob@Work with a rectangular shape (b,d).

already knows, to an environment with slight changes as well as a totally unknown environment. For the first and second one previously unknown static obstacles that haven't been in the environment for the training data collection are spawned in the machine-room. The data collected in evaluation the warehouse is then used for assessing the networks' performance in a completely unknown environment.

To evaluate the different approaches in real-world conditions, two different rooms have been used for the data gathering. The first on is a laboratory, shown in figure 6.3a. The second one is similar to an industrial setting and visualized in 6.3b. The lab is simple as it does not contain many complex objects, however it does contain many rectangular shaped objects. In contrast the industrial



(a) (b)

Figure 6.2: The simulation environment with five different rooms used for training (blue) and evaluation (red) of the data, visualized in Gazebo (a) and rviz (b).

hall contains many machine-like objects which also look different in a 2D LIDAR scanner from different points of view. This makes it harder to recognize objects. The data collected in these two environments involves the same three robots as for the data collection in simulated rooms.

The labeling of the data was performed using the localization of the robot or its ground truth position in some simulation cases. All points closer than a certain threshold around the robot are being labeled as the robots' type. This results in labeling error, however, the amount is negligible and also influences the results of all evaluations equally. Therefore the error may only reduce the overall performance of the different approaches but it does not affect the experiments and comparisons conducted between the different networks.

6.4 Data & Models

For the purpose of the evaluation, multiple data-sets were collected from simulated and real-world environments. This section introduces the different input data-sets and models together with the metrics and parameters used for evaluating the networks performance.

To increase the clarity of the presented data throughout this chapter, we will introduce some abbreviations for the properties of the data-sets and models. These will be used in the tables showing the results of the experiments as well as for describing the respective data-set or model. The data was collected either in SIM or REA environments. The two different rooms used for the SIM data will be referred to as machine-room (MR) and warehouse (WH), respectively. For some parts of the evaluation data-sets with additional unknown and unmapped static objects (STC-OBJ) will be uses as well as static robots (STC-ROB) and dynamic objects (DYN-OBJ). The REA data-sets are gathered either in the APC or LAB. The data for the 1D dynamic object detection approach was gathered using a STC sensor with 270 degree angular range. The data for the other approaches was collected using multiple sensors, fused together to obtain a 360 degree view around the robot. Both scan models maintain an angular increment of 0.5 scan points per degree.

(a) **(b)**

Figure 6.3: The two real-world environments used for evaluating the networks, a laboratory (a) and an industrial environment (b).

The models used for the evaluation are the DYN-STACK using stacked 1D data, the DYN-LSTM using ConvLSTMs and the DYN-MAP-LSTM using additional static-map data. Each of these networks is trained using different data for various evaluation purposes. All networks except for the DYN-STACK are trained on GRT data without uncertainty in the localization and EKF data with uncertainty in the localization due to the used Kalman Filter. For evaluating the networks on real-world data, all networks are NEW on APC as well as CTD on a pre-trained EKF model.

To compare the performance of the different dynamic object detection models against each other we will use the precision and recall of the detected dynamic pixels, defined as follows:

$$Precision_{dyn} = \frac{TP}{TP + FP}, \tag{6.1}$$

$$Recall_{dyn} = \frac{TP}{TP + FN}, \tag{6.2}$$

where TP is the number of correctly labeled dynamic pixels, FP and FN the number of wrongly classified dynamic and static pixels, respectively. Due to the low number of dynamic pixels in an array, the accuracy is always close to one and therefore impractical. The precision and recall are able

to demonstrate the trade-off between correctly identifying most examples and misclassifying only few. Due to these reasons these metrics are used for the classification networks as well. Although they do not measure a per-pixel value anymore but a per robot value. The precision and recall in this case are defined as follows:

$$Precision_{cls} = \frac{TP_c}{TP_c + FP_c}, \tag{6.3}$$

$$Recall_{cls} = \frac{TP_c}{TP_c + FN_c}, \tag{6.4}$$

where TP_c is the number of correctly classified robots of class c, FP_c and FN_c the number of wrongly classified robots of class c, respectively. In the different parts of the evaluation we will either use a per class precision/recall or the mean precision of all classes. If not marked by a class, we will refer to the precision as being the mean precision and the recall as being the mean recall of all classes.

6.5 Network Architecture

Choosing the right architecture for a deep neural network is a difficult but crucial task in order to solve a problem. Evaluating all possible architecture and parameter choices against each other would be infeasible. Nevertheless it is important to understand and evaluate the different behaviors of the various architectures and parameters. Throughout this section we therefore evaluate the chosen network architectures against obvious alternatives. This is only done for the DYN-LSTM and the CLS approach. The approach for dynamic object detection using 1D stacked input data cannot be altered without loss of functionality. The DYN-MAP-LSTM network is straightforward since it is only one layer for calculating the difference between two frames. The CLS-RAY approach on the other hand is using the exact same network as the CLS and is therefore not evaluated separately.

6.5.1 Dynamic Object Detection

The GRT model of the DYN-LSTM approach is evaluated in this section against a couple of alternative architectures and without accumulated input data. The results of the different models is shown in table 6.1. In the first two rows the performance of the *state-of-the-art* network is shown learned on accumulated (GRT) and plain grid-map input without spatio-temporal features. Without these features the network is not able to detect dynamic objects. The low precision indicates that it just outputs almost all scan data instead of only the dynamic pixels. Changing the number of consecutive ConvLSTM layers decreases the precision as well as the recall. This is surprising in case of the increased number of layers, since the network is more powerful but performs worse. The deeper network may need to train for a longer period of time. A network consisting of only convolutional layers (the encoder decoder part of the CLS approach, see section 5.3) results in a model which is not capable of making any correct predicions. Using a similar network as the CLS, with convolutional layers followed by ConvLSTM layers on the other hand leads to a network with similar performance as the GRT model. Overall the GRT model easily outperforms all other models which is the reason why we chose this architecture.

Model	Input Data	Prec	Recall
GRT	360 SIM-MR GRT	95.09	89.33
No Accumulated Input	360 SIM-MR GRT	3.03	84.60
1 ConvLSTM Layer	360 SIM-MR GRT	68.84	78.06
3 ConvLSTM Layers	360 SIM-MR GRT	68.52	17.24
Only Conv Layers	360 SIM-MR GRT	0.00	0.00
Conv & ConvLSTM Layers	360 SIM-MR GRT	86.18	75.78

Table 6.1: A comparison of different input data and architectures for the DYN-LSTM network.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
GRT	360 SIM-MR GRT	93.63	81.86	0.093
No Weights	360 SIM-MR GRT	75.17	26.96	0.171
Accumulated Input	360 SIM-MR GRT	0.00	0.00	-
Only Conv Layers	360 SIM-MR GRT	0.00	0.00	-
Only ConvLSTM Layers	360 SIM-MR GRT	38.81	1.01	0.142
Last Layer 3x3	360 SIM-MR GRT	80.98	84.41	0.095
All Layers 5x5	360 SIM-MR GRT	89.77	83.81	0.070

Table 6.2: A comparison of different input data and architectures for the CLS network.

6.5.2 Robot Classification & Position Estimation

The results of the architecture and input comparison of the CLS approach are shown in figure 6.2, with the GRT model as the *state-of-the-art*. Using a standard categorical cross-entropy loss function decreases the performance of the learned model significantly compared to the model using a weighted loss function (GRT). The network is not able to classify many of the robots correctly. Using an accumulated input to the network, as done in the DYN-LSTM approach, leads to a model which has not learned anything. The same result is achieved with a network using only convolutional layers. The network does not seem to be powerful enough to learn the right features for the robot classification. The same applies to a network using only ConvLSTM layers. In this case, however, the network is at least able to learn a weak classifier. Reducing the kernel size of the output layer to $3x3$ increases the recall slightly while decreasing the precision by around 13%. Better performance is achieved by increasing the kernel size of all layers to $5x5$. This decreases the mean error in the position by almost 25%. Since the accuracy in the position estimate is one of the most important performance measures of the network, this is indicated a likely area for future work. Using an increased kernel size also increases the complexity of the network and with that the time for learning it or running a forward pass through it. Therefore we decided to go with the GRT architecture, since it still achieves accurate position estimates while keeping the number of false positives low (high precision).

6.6 Impact of Unknown Data

The ability of a neural network to generalize to unknown data is one of its most important and difficult properties. This section evaluates the performance of all presented approaches in (un)known environments, with unknown dynamic objects and additional known static objects. The objective is to determine whether the networks generalize well with regards to their environment, i.e. do the networks actually learn to label dynamic data or do they just learn to represent the environment. Furthermore, the experiments with additional dynamic objects and static objects used as dynamic ones during the learning phase are used to demonstrate the networks ability to detect dynamic objects independent from their shape instead of detecting objects as dynamic due to their shape.

6.6.1 Dynamic Object Detection

The results of the impact of environmental changes on the networks performance for the dynamic object detection networks is shown in table 6.3. The data is sorted by the degree of unknownness of the environment in ascending order. The first part of the table clearly shows, that the network using stacked 1D data is not able to adjust to unknown environments. While the recall stays constantly high, the precision decreases dramatically with an increasingly unknown environment. This means that more pixels of the input map are falsely labeled as dynamic objects (false positive). Therefore when a pixel is labeled as being dynamic, the probability that it actually is dynamic decreases (precision). Since this is an important property of the network, the reliability in unknown environments decreases dramatically due to these results.

The second approach using spatio-temporal features in a 2D grid-map as an input achieves significantly better results. There is a slight decrease in the precision again. This could, on the other hand, be also due to the differences in the data-sets. This could be the reason as well for the increase in the recall with a decreased knowledge of the environment. The results demonstrate, that the trained network is not affected by unknown environments and generalizes well in these scenarios. The differences in the metrics are negligible, since they are most likely just variations due to different data-sets. There is no continuous decrease between the different data-sets as seen for the previous approach.

Using the static map of the environment as an input to detect dynamic objects requires a map including all static obstacles. In this experiment, only the first data-set fulfills this prerequisite. The results of the evaluation therefore show whether or not the trained network does more than just calculating the difference between the two input frames. The recall for all three data-sets is again almost equally high. The precision on the other hand drops by more than 50% in the second example. The reason for that is that the map in the second data-set contains unmapped static objects, which the network is not able to neglect. This is an intended behavior, since the LTS normally provides a map including these objects. The third data-set shows, that the network is actually able to generalize to unknown environments. A requirement is of course a provided map including all static obstacles.

The results for the second experiment conducted to determine the generalization capabilities of the networks is shown in table 6.4. The first part of the table displays the results for the 1D network. It shows a decrease in the precision and the recall in all three examples. In the second run, the network is still able to identify most of the dynamic objects (recall) while not forfeiting too much of

Model	Input Data	Prec	Recall
DYN-STACK	270 SIM-MR STC	93.16	93.67
DYN-STACK	270 SIM-MR STC-OBJ STC	68.72	86.91
DYN-STACK	270 SIM-MR STC	39.38	82.05
DYN-LSTM	360 SIM-MR GRT	95.09	89.33
DYN-LSTM	360 SIM-MR STC-OBJ GRT	88.94	89.49
DYN-LSTM	360 SIM-WH GRT	88.84	91.28
DYN-MAP	360 SIM-MR GRT	98.73	94.50
DYN-MAP	360 SIM-MR STC-OBJ GRT	37.76	97.95
DYN-MAP	360 SIM-WH GRT	90.03	97.72

Table 6.3: The evaluation data comparing the dynamic object detection networks on data-sets collected in a known (1), partially known (2) and unknown (3) environment.

its precision. In the last test, the network on the other hand shows, that it does detect the dynamic objects to some extend based on their shape and not solely based on spatio-temporal features. The reason for that is the decrease in the precision. The recall stays high, hence the algorithm still detects most of the dynamic object pixels. The low precision on the other hand suggests that the network detects the static robots spawned in the environment as dynamic objects due to their known shape. This reduces the networks reliability in detecting dynamic objects and makes it almost unfeasible for real-world applications.

The DYN-LSTM network on the other hand performs well in all three experiments. There is no major difference between the values of precision and recall. The small variations that exist are again due to the differences in the data-sets, e.g. more or less dynamic objects from different points of view. This confirms that the network is not only capable of generalizing to different environments but also to different dynamic objects. This also shows that it does not detect dynamic objects based on their shape but on general spatio-temporal features.

The map-based approach achieves equally high results on the first data-set with unknown dynamic objects. On the second data-set, however, the precision drops again to about 50% while the recall stays high at around 98%. These results show again, that the network does generalize to different environments or shapes of objects, as in this case. It further demonstrates the networks (intended) inability to cope with unmapped static objects as the robots in the third data-set.

Overall the second approach for detecting spatio-temporal features in 2D grid-maps achieves the most promising and reliable results. The first approach using stacked 1D scan data does neither generalize to different environments, nor does it detect dynamic objects solely based on spatio-temporal features. It is therefore impractical for real-world applications, since the network would need to be retrained, whenever a static obstacle is placed in a known environment. The last approach demonstrates its generalizability with regards to unknown environments as well as unknown dynamic objects. It fulfills all expectations, including the drawback of being dependent on an accurate and current grid-map of the environment including all static objects. Since the 2D approach based on spatio-temporal features does not require this information while still achieving similar results, it seems like the most promising.

Model	Input Data	Prec	Recall
DYN-STACK	270 SIM-MR STC	93.16	93.67
DYN-STACK	270 SIM-MR DYN-OBJ GRT	85.09	89.02
DYN-STACK	270 SIM-MR STC-ROB GRT	51.47	88.46
DYN-LSTM	360 SIM-MR GRT	95.09	89.33
DYN-LSTM	360 SIM-MR DYN-OBJ GRT	95.56	86.63
DYN-LSTM	360 SIM-MR STC-ROB GRT	95.21	89.12
DYN-MAP	360 SIM-MR GRT	98.73	94.50
DYN-MAP	360 SIM-MR DYN-OBJ GRT	97.75	96.80
DYN-MAP	360 SIM-MR STC-ROB GRT	50.20	98.43

Table 6.4: The evaluation data comparing the dynamic object detection networks performances on the default data-set (1) with the performances on data-sets including additional unknown dynamic objects (2) and additional static objects, encountered as dynamic objects during the learning phase.

6.6.2 Robot Classification & Position Estimation

The evaluation of the classification networks performances only includes the different environments from the previous section together with additional unknown dynamic objects. The third input data type, including static robots, was specifically chosen for the dynamic object detection and is therefore not used in this case.

The results of the evaluation is shown in table 6.5. The first part highlights the performance of the network using a grid-map without ray-tracing and the second part with ray-tracing. The CLS performs constantly good independent from the environmental changes. The precision decreases a bit when the network is applied to a completely unknown environment but the difference could be explained by the differences in the data-sets (as described in section 6.6.1). The performance of the network when used on the data-set including unknown dynamic objects are even better, even though it's just a small increase which is most likely due to differences in the data-sets. In all cases, the precision of the network is significantly higher than the recall. This is desired behavior, since it is more important to be certain about a positive robot classification than to classify all robots at any time-step. For the algorithms purpose, supporting another robots localization, it is not necessary to detect the robot at every single time-step but rather to be certain about its class and its position. The error in the position estimation does not change significantly in this scenario with different input data.

The second network used (CLS-RAY) maintains a high precision over all examples while only achieving a relatively low recall. The network seems less powerful than the one presented before. The position estimates, however, are almost 30% more accurate than the ones from the CLS network. The reason for that could be the more detailed representation of the input. By incorporating unknown space, the position is probably easier to estimate because the shape is better to identify. This could also explain the high precision. The low recall on the other hand could be explained by the fact, that the robots shape now has more different variations which makes it hard to always classify it but on the other hand also easy to distinguish from other objects, hence the high precision.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
CLS	360 SIM-MR GRT	93.63	81.86	0.093
CLS	360 SIM-MR STC-OBJ GRT	92.87	89.02	0.077
CLS	360 SIM-WH GRT	84.33	80.16	0.088
CLS	360 SIM-MR DYN-OBJ GRT	96.58	90.96	0.072
CLS-RAY	360 SIM-MR GRT	83.28	56.88	0.066
CLS-RAY	360 SIM-MR STC-OBJ GRT	83.02	61.32	0.062
CLS-RAY	360 SIM-WH GRT	75.86	42.91	0.066
CLS-RAY	360 SIM-MR DYN-OBJ GRT	91.14	72.29	0.054

Table 6.5: The evaluation data comparing the robot classification networks performances on the default data-set (1) as well as on data with different levels of unknownness (2-4).

In general, the approach without ray-tracing outperforms the other network in all cases and metrics, except for the position estimation. However, it is in general more important to correctly classify the robots than retrieving a position estimate which is $3cm$ more accurate but for the wrong robot.

6.7 Impact of Sensor Localization

The localization of a mobile robot is always uncertain, independent from the algorithmic choice (EKF, Particle Filter, etc.). The uncertainty in sensor measurements, the environmental conditions and make it impossible to estimate an exact position. Classical approaches for robot detection as Bayesian Filters model this uncertainty explicitly. In neural networks, however, this is not possible. The network has to learn and adapt to the uncertainty in the world.

In this section we therefore evaluate the performance of the different approaches given different levels of localization uncertainty. The grid-map representation of the environment, generated for most approaches, can shift especially during sudden changes in the robots angular velocity. An evaluation is conducted for all approaches except for the one using stacked 1D input data, since it only works on static scan data. All other networks are trained on two different data sets gathered in simulation. The first one was collected using the ground-truth pose of the robot for localization. The second one is using an EKF-based localization for the robot, introducing some uncertainty. The networks will be evaluated against data from two different rooms in the simulation environment to determine, whether the networks trained using a probabilistic localization performs better on data with localization uncertainty than the networks trained on ground-truth data.

This section mainly tries to find out, whether training a network with uncertainty in the localization improves the networks performance under uncertainty. The networks are therefore evaluated on data from a simulated room without any unmapped objects (least localization error), another one with unmapped objects(some localization error) and a real-world environment (most localization error). All data sets were gathered using EKF localization. In the real-world there is much more uncertainty due to e.g. people moving around.

6.7.1 Dynamic Object Detection

In order to determine the impact of the ego-vehicles localization error on the performance of the networks, we will evaluate the predictions of the 2D-LSTM as well as the 2D-Map approach on the introduced data-sets. The 1D-Stacked approach will not be evaluated, since it only works with a static sensor.

The results for the 2D-LSTM network are shown in table 6.6, where each part of the table contains the evaluation data for another environment. For the first two parts (simulation) in addition to the EKF data-set, the performance of the GRT network on GRT data is included to allow a better comparison of the networks. The first two parts of the network highlight a significant loss in precision when using the GRT network on EKF data. It is still able to correctly label almost the same amount of dynamic objects. It incorrectly labels more static objects than dynamic ones. The network trained on EKF data performs significantly better, with regards to the precision. The recall, however, is lower compared to the GRT network. This behavior can be explained by the introduced uncertainty. The GRT model only knows data without uncertainty, therefore some of the static obstacle shapes look similar to learned spatio-temporal features with increased uncertainty.

The reason for this uncertainty is the accumulation of the scan data. If there is e.g. a rotational shift in consecutive scans due to uncertainty in the orientation of the robot, this may create tail-like shapes for static obstacles similar to the ones learned as spatio-temporal features (see figure 4.8). Hence, the precision drops due to an increase in falsely labeled dynamic objects. If the a network is trained on the data including uncertainty (EKF), the precision of the networks predictions is high, since it is able to adapt to this uncertainty and the ambiguous shapes. Due to these shapes, the network presumably only learns to detect the ones with unambiguous shapes, which are clearly identifiable as dynamic objects. Therefore the network achieves a high precision while not being able to identify as many of the dynamic objects as without the uncertainty in the data (lower recall). The last part of the table visualizes the networks ability to generalize to real-world data. In both cases the EKF network outperforms the GRT network because the real-world introduces more uncertainty than the EKF data and the EKF model already adapted to this uncertainty.

The evaluation results for the network using additional static map information is shown in 6.7. They are similar to the results of the DYN-LSTM network. The differences in the precision, however, are significantly smaller. Compared to the DIF model results, which were obtained by simply calculating the difference between both input frames (static grid-map/sensor grid-map), the convolutional approach clearly works better on all data-sets. The DIF approach labels too many pixels as being dynamic due to small shifts in the data. The ability of the convolutional operations to cope with these small shifts between the sensor grid-map and the static one allow the network to perform much better than this straightforward approach. The DYN-MAP-LSTM approach is further just composed of two convolutional layers, resulting in a quick forward pass through the network.

The low precision on the real-world data can be explained by the fact, that the grid-map of a real-world is almost always less accurate than the one of a simulation environment. The mapping process introduces additional uncertainty into the map which the network is not able to deal with, since it only computes the difference between the static grid-map and the dynamically generated scan map. Using the approach together with the LTS would increase the precision. However, this was not possible to evaluate within this thesis due to time constraints but it will be part of the future work presented in chapter **??**.

Model	Input Data	Prec	Recall
GRT	360 SIM-MR GRT	95.09	89.33
GRT	360 SIM-MR EKF	71.09	87.09
EKF	360 SIM-MR EKF	92.61	75.05
GRT	360 SIM-WH GRT	88.84	91.28
GRT	360 SIM-WH EKF	81.46	91.36
EKF	360 SIM-WH EKF	90.69	82.54
GRT	360 REA-BIO	38.43	75.68
EKF	360 REA-BIO	60.00	69.82
GRT	360 REA-APC	46.39	82.99
EKF	360 REA-APC	63.04	71.74
GRT	Average	59.34	84.28
EKF	Average	76.59	74.79

Table 6.6: The evaluation data comparing a DYN-LSTM-network trained on GRT with another one trained on EKF data. The input data was collected in two different simulation rooms and two real-world environments. The simulated data is collected using both, GRT and EKF localization.

All in all, both network types demonstrate, that using training data with uncertainty in the robots localization, i.e. the sensor data, makes them somewhat robust to this or additional uncertainty (real-world). In average, the networks trained on EKF data outperform the ones trained on GRT data. When evaluating the performance of the approaches on real-world data in section 6.8.1 we therefore only use the EKF networks, since they achieved the most promising results.

6.7.2 Robot Classification & Position Estimation

The evaluation of the classification approaches is performed using the same input data as for the dynamic objects detection. The results are shown in tables 6.8 and 6.9 for the network without and with ray-tracing, respectively.

The CLS approach achieves similar results to the DYN-LSTM network before. The precision generally increases, as does the recall to a lesser degree. The network is able to cope better with the increased uncertainty than the DYN-LSTM, since not only the precision but also the recall increases in all cases except for one. This means, that the network is not only capable of adapting to the uncertainty, i.e. to learn more general classifiers in order to correctly classify robots with more variations in their shapes. It is also able to maintain the ratio of correctly classified robots out of all robots of that type. The reason for that is the fact that the accumulated data used in the DYN-LSTM model creates different shapes with an increased localization error (see section 6.7.1). The CLS network on the other hand uses the plain grid-map as an input where localization error leads to shifts or turns in the data but no differences in the shapes. The network is therefore better able to adapt to these conditions. Beyond that, using EKF instead of GRT models decreases the error in the position estimation, especially when applied to real-world data. The reason for that could be the more accurate output of the network due to the adaption to the uncertainty.

Model	Input Data	Prec	Recall
GRT	360 SIM-MR GRT	98.73	94.50
GRT	360 SIM-MR EKF	96.06	96.58
EKF	360 SIM-MR EKF	98.25	98.30
DIF	360 SIM-MR EKF	15.60	99.96
GRT	360 SIM-WH GRT	90.03	97.72
GRT	360 SIM-WH EKF	76.78	96.18
EKF	360 SIM-WH EKF	78.89	96.35
DIF	360 SIM-WH EKF	11.24	99.13
GRT	360 REA-BIO	21.05	96.23
EKF	360 REA-BIO	22.79	96.59
DIF	360 REA-BIO	10.20	99.48
GRT	360 REA-APC	25.78	90.50
EKF	360 REA-APC	28.44	90.95
DIF	360 REA-APC	9.82	98.71
GRT	Average	54.92	94.87
EKF	Average	57.09	95.55
DIF	Average	11.71	99.32

Table 6.7: The evaluation data comparing a DYN-MAP-LSTM-network trained on GRT with one trained on EKF data. Additionally the results of simply calculating the difference between both input frames are listed (DIF). The input data was collected in two different simulation rooms and two real-world environments. The simulated data is collected using both, GRT and EKF localization.

The CLS-RAY approach does not seem to be able to adapt to the uncertainty in the data as well as the CLS network. The overall increase in precision, recall and position accuracy of the networks is quite small. There is no significant improvement in the networks performance, especially when applied to the real-world data. Only the error in the position estimate is here significantly lower for the EKF model. The more complex structure of the ray-tracing grid-map might be the reason for weak performance increase. The incorporation of the unknown parts of the map introduce uncertainty as well, since in a discretized world they are not always 100% accurate.

The unknown data might also disguise the uncertainty of the localization because only small parts of the map change when a pixel (cell) marking occupied space is e.g. shifted by one. This scenario is displayed in figure 6.4. If the cell is close to the robot, there is a large unknown space behind it (figure 6.4a). Once an occupied cell is shifted by one, the shape of the unknown space does not change dramatically (figure 6.4b). Most cell keep their values even after the shift compared to the scenario applied to the CLS approach visualized in figure 6.4c and 6.4d. Since in this map occupied cells are the only ones with a value $c_i \neq 0$, a third of these cells change in this example. This could explain the ineffectiveness in training a CLS-RAY network on EKF data in order to make it more robust to uncertainty.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
GRT	360 SIM-MR GRT	93.63	81.86	0.093
GRT	360 SIM-MR EKF	92.43	85.12	0.099
EKF	360 SIM-MR EKF	92.16	81.87	0.092
GRT	360 SIM-WH GRT	84.33	80.16	0.088
GRT	360 SIM-WH EKF	81.90	82.50	0.083
EKF	360 SIM-WH EKF	91.07	86.68	0.105
GRT	360 REA-BIO	46.11	45.75	0.310
EKF	360 REA-BIO	63.10	59.47	0.280
GRT	360 REA-APC	38.20	36.73	0.298
EKF	360 REA-APC	45.66	44.50	0.202
GRT	Average	64.66	62.53	19.75
EKF	Average	73.00	68.13	16.99

Table 6.8: The evaluation data comparing the CLS model trained on GRT with one trained on EKF data. The input data was collected in two different simulation rooms and two real-world environments. The simulated data is collected using both, GRT and EKF localization.

The CLS network outperforms the CLS-RAY network again in this discipline. The network is able to adapt to the uncertainty by using training data with uncertainty in the localization. The performance of the CLS-RAY on the other hand could not be improved by this technique. The results of both networks on the real-world data are not satisfying on the other hand. The robots shapes in the real-world data differ from the ones in the simulation data quite a bit, especially due to small tilts in the environments. The following section therefore trains the networks of all approaches on real-world data and compares their performance to the simulation only networks.

6.8 Simulated vs. Real-World Environments

The algorithms presented in this thesis will be integrated into a real-world multi-robot system. One of the most important aspects of the evaluation is therefore the performance and generalizability of the networks on real-world data. In the following three different kinds of models for each network are evaluated on three different environments. The first model is the EKF model of all networks (except for the 1D network), the second one is the CTD and the last one the NEW model as introduced in 6.4. These models are evaluated on an unknown simulation room (WH) as well as two real-world rooms (LAB, APC). The purpose of these evaluations is to determine whether the networks trained on pure simulation data are able to perform as well on real-world data as the models trained on real-world data. Further it is examined, whether the models trained on real-world data are able to generalize to the simulation data.

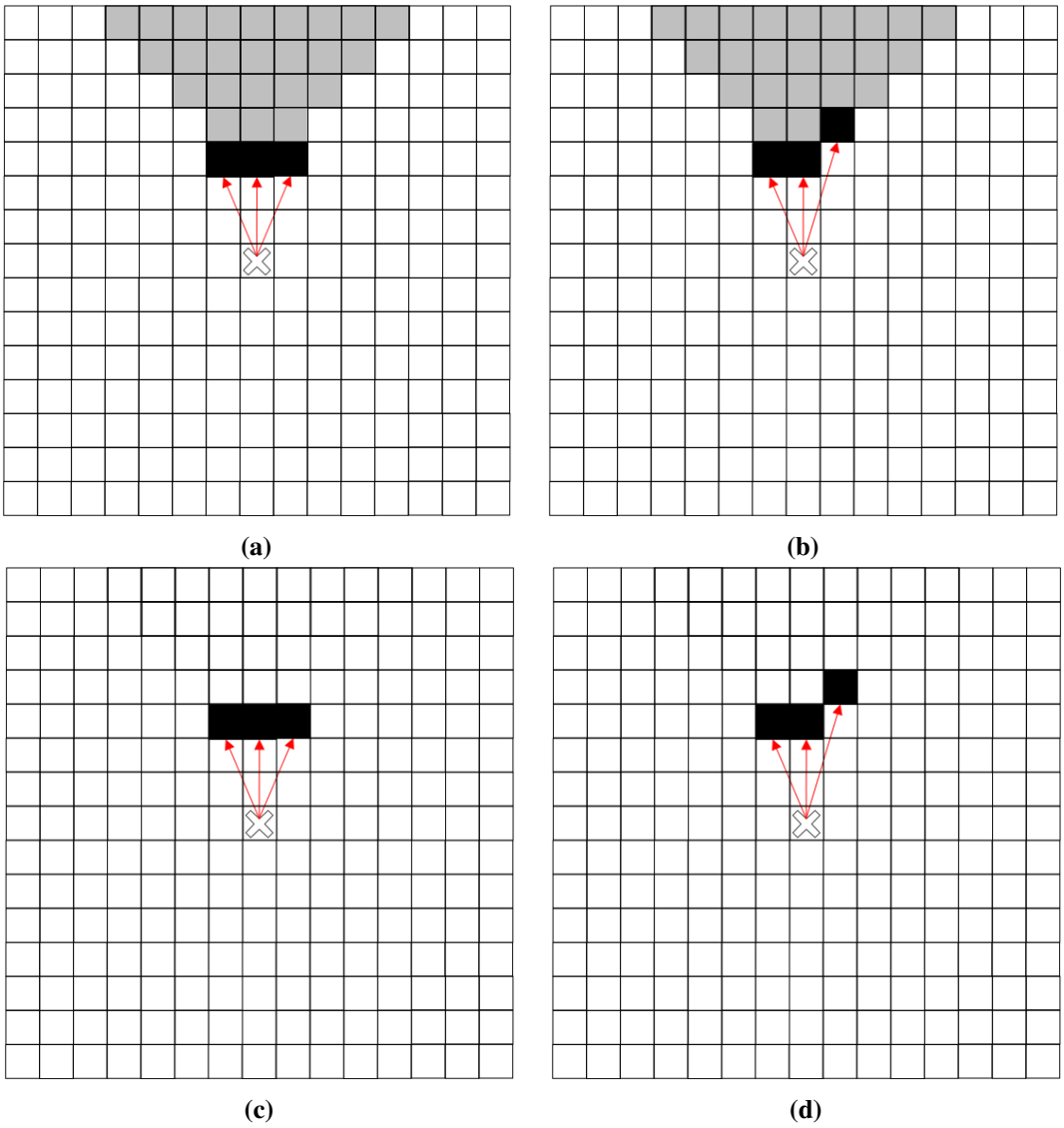


Figure 6.4: A scenario demonstrating the difference of the influence of a pixel-shift in the CLS-RAY data (a, b) compared to the CLS data (c, d). The images show the grid-map before (a, c) and after (b, d) the shift of the pixel.

6.8.1 Dynamic Object Detection

The evaluation of the approach using stacked 1D scan data was performed similarly to the other networks, but with a static sensor. The results can be seen in table 6.10. As before in section 6.6.1, the network again doesn't generalize well to different environments. The performance of the network learned on simulation data when applied to real-world environments is better than the performance on unknown simulation data. Especially in the case of the LAB data. This can be explained by the simplicity of this environment. The networks trained on real-world data on the other hand are not able to perform well on simulation data either. This is again due to the networks inability to generalize to unknown environments.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
GRT	360 SIM-MR GRT	93.63	81.86	0.093
GRT	360 SIM-MR EKF	87.59	66.20	0.094
EKF	360 SIM-MR EKF	87.35	65.01	0.083
GRT	360 SIM-WH GRT	75.86	42.91	0.066
GRT	360 SIM-WH EKF	81.60	46.77	0.080
EKF	360 SIM-WH EKF	88.15	48.04	0.109
GRT	360 REA-APC	38.10	37.25	0.270
EKF	360 REA-APC	39.44	43.57	0.196
GRT	360 REA-BIO	52.72	56.32	0.290
EKF	360 REA-BIO	49.00	52.47	0.254
GRT	Average	65.00	51.64	18.36
EKF	Average	65.99	52.27	16.03

Table 6.9: The evaluation data comparing the CLS-RAY model trained on GRT data with one trained on EKF data. The input data was collected in two different simulation rooms and two real-world environments. The simulated data is collected using both, GRT and EKF localization.

The table also demonstrates, that the networks trained on data from the APC are able to perform well in this known environment. Both the NEW and the CTD network achieve a precision and recall of around 95%. This performance, however, does again not generalize to the LAB room. These results show that this network is capable of achieving high precision and recall rates in a known environment. This performance is equally high on simulated and real-world data. The network is on the other hand not able to generalize to different environments which makes it impractical for most use-cases.

The DYN-LSTM approach has so far proven that it is capable to generalize to different environments with only a minor reduction of precision and recall. The results of the DYN-LSTM evaluation in table 6.11, show, that this does apply for the models trained on simulation data but not for the ones trained on real-world data. The EKF model still performs well when applied to the two different real-world environments with only a decrease of 30% in precision and recall. At least when compared to the performance of the two real-world networks on the simulation data and the unknown real-world room. Both networks perform poorly when applied to a different environment.

The results also show that the CTD network outperforms the NEW one in each scenario. When applied to a known real-world domain, the CTD network achieves twice as much precision and recall as the NEW network. The results in this table suggest that the networks trained on real-world data weren't trained on enough data samples. Both networks have just been trained on one set of data from the APC. The outcome clearly shows that this is not enough for the network to even learn to work in one environment. It was, however, infeasible to gather data from four different real-world environments for learning the network plus an additional one for evaluating it. Due to temporal and spatial restrictions, we were not able to perform this kind of evaluation.

Model	Input Data	Prec	Recall
STC	270 SIM-MR STC	39.38	82.05
NEW	270 SIM-MR STC	18.48	32.93
CTD	270 SIM-MR STC	25.45	53.23
STC	270 REA-APZ STC	61.64	82.45
NEW	270 REA-APZ STC	51.80	81.02
CTD	270 REA-APZ STC	36.95	86.71
STC	270 REA-BIO STC	37.93	82.76
NEW	270 REA-BIO STC	96.15	94.34
CTD	270 REA-BIO STC	95.94	96.61
STC	Average	46.32	82.42
NEW	Average	55.48	69.43
CTD	Average	52.78	78.85

Table 6.10: The evaluation data comparing a DYN-STACK network trained on SIM data (STC) with one trained on REA data (NEW) and a third one which trains the STC model additionaly on REA data. These models are evaluated on SIM and REA (two rooms) data-sets.

Nevertheless, especially results of the CTD network are promising, since it is able to detect almost 80% of the pixels belonging to dynamic objects correctly. This rate decreases just slightly when applied to different environments. Since the dynamic object detection is a pre-processing of the data for the classification, the recall is much more important than the precision for these networks. It is more important to classify many of the dynamic object pixels correctly in order to process as much of the robots shape as possible. Detecting and processing 50% of the pixels belonging to static obstacles as dynamic ones is not as bad as wrongly labeling 50% of all dynamic object pixels as being static. The classification network is able to deal with this static information as shown in the evaluation so far. This behavior will be further evaluated in section 6.10 when the classification network is tested on the dynamic network output.

Applying the DYN-MAP-LSTM approach to the introduced real-world scenarios leads to the results shown in table 6.12. As stated before, the simulation model does not generalize well to real-world environments with regards to the precision. Using models trained on real-world data (NEW/CTD) on simulation data results in an increase in the precision. The recall for these two models is relatively low on all data-sets, compared to the simulation model.

One reason for this behavior could be the additional unmapped objects which are always part of real-world environment. Due to these objects, which are not part of the map the network might learn additional features for detecting the dynamic objects based on their shape. Therefore the precision is high because the network learns to identify fewer objects than the ones that result from simply calculating the difference between the static map and the scan map. The recall is low because the network is not powerful enough to detect all dynamic objects with just one convolutional layer for feature learning. One way to solve this issue would be to only learn the network on data where the map is up-to-date and contains all static objects. This would result in the intended behavior of the network calculating only the approximate difference between the static and the scan grid-map.

Model	Input Data	Prec	Recall
EKF	360 SIM-WH EKF	90.69	82.54
NEW	360 SIM-WH EKF	16.15	37.49
CTD	360 SIM-WH EKF	43.59	64.80
EKF	360 REA-BIO	60.00	69.82
NEW	360 REA-BIO	20.14	38.50
CTD	360 REA-BIO	34.42	75.36
EKF	360 REA-APC	63.04	71.74
NEW	360 REA-APC	26.87	41.00
CTD	360 REA-APC	58.84	78.68
EKF	Average	71.24	74.70
NEW	Average	21.05	39.00
CTD	Average	45.62	72.95

Table 6.11: The evaluation data comparing a DYN-LSTM network trained on SIM data (EKF) with one trained on REA data (NEW) and a third one which trains the EKF model additionaly on REA data. These models are evaluated on SIM and REA (two rooms) data-sets.

Overall the most promising results were shown by the DYN-LSTM approach again. Besides the fact that much more data is needed to train a network that is able to achieve satisfying results on real-world data, the network demonstrated that a pre-learned (simulation) model is able to detect almost 80% of all dynamic pixels correctly in both real-world environments. The results of the 1D approach showed again that it works well in known but not at all in unknown environments. The performance of the DYN-MAP-LSTM network on the other hand illustrated how sensitive this approach is with regards to the integrity of the input data.

6.8.2 Robot Classification & Position Estimation

The evaluation of the classification approaches is performed using the same input data as for the dynamic objects detection. The results are shown in tables 6.13 and 6.14 for the network without and with ray-tracing, respectively.

As demonstrated before in section 6.7.2 the DYN-LSTM network trained on simulation data is able to generalize to real-world scenarios. The precision and recall decreases but in average it still classifies around 50% of all robots correctly while only 50% of the predicted classes are wrong. These values are not sufficient for a real-world application. They are, however, a great basis for learning a model on real-world data (CTD). The results clearly show, that the model which uses the simulation model as a basis for its training performs significantly better than the model learned from scratch. Although it is only trained on the one data-set from this one room the network is capable of achieving a precision of 90% in a known and 72% in an unknown environment. The network trained from scratch performs significantly worse (61%/53%).

Model	Input Data	Prec	Recall
EKF	360 SIM-WH EKF	78.89	96.35
NEW	360 SIM-WH EKF	91.35	41.38
CTD	360 SIM-WH EKF	91.85	55.26
EKF	360 REA-BIO	22.79	96.59
NEW	360 REA-BIO	42.29	40.48
CTD	360 REA-BIO	45.56	56.02
EKF	360 REA-APC	28.44	90.95
NEW	360 REA-APC	80.37	40.82
CTD	360 REA-APC	82.96	58.18
EKF	Average	43.37	94.63
NEW	Average	71.34	40.90
CTD	Average	73.46	56.49

Table 6.12: The evaluation data comparing a DYN-MAP-LSTM network trained on SIM data (EKF) with one trained on REA data (NEW) and a third one which trains the EKF model additionaly on REA data. These models are evaluated on SIM and REA (two rooms) data-sets.

We focus on the precision here, since as already described this metric is far more important for the purpose of the developed algorithm. Both network do not perform well on simulated data which stresses the differences in the robot shapes between simulation and real-world data. It is on the other hand not necessary to train a model in order to perform well on both, simulation and real-world data. In general it is desirable that a network pre-trained on simulation data does not need much real-world data in order to generalize to the real-world scenarios it is used for. The network demonstrated that it is able to achieve this property.

In contrast to the CLS network, the CLS-RAY network is not able to generalize well to the real-world environments as discussed before. The networks trained on real-world data outperform the CLS approaches in all three environments. This is counterintuitive, since so far the CLS-RAY networks performed worse with increased uncertainty in the localization as well as in unknown environments compared to the CLS approach. This still holds for the model trained on simulation data only. The other two models perform significantly better than the CLS models. The CTD network is able to generalize to perform well on data from the LAB room although it is only trained on one data-set gathered in the APC. While the precision drops from 94% to 82%, the recall almost stays the same. This means that the network is able to correctly classify the same amount of robots in an unknown as in a known environment.

The difference in the precision can be explained by new, unseen objects in the unknown environment which might be classified as one of the robots due to similar looking shapes. The NEW model generalizes equally well, however, it does not achieve the initial performance of the CTD. This shows, that similarly to the CLS approach the pre-trained model using simulation data supports the learning of real-world models. Beyond that the accuracy of the position estimates is also significantly better than in the CLS results. These accurate estimates are partially linked to the precision of the model. The more certain a prediction is, the more accurate is its position estimate.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
EKF	360 SIM-WH EKF	91.07	86.68	0.105
NEW	360 SIM-WH EKF	19.05	28.28	0.225
CTD	360 SIM-WH EKF	38.33	42.70	0.107
EKF	360 REA-BIO	63.10	59.47	0.280
NEW	360 REA-BIO	53.01	40.00	0.229
CTD	360 REA-BIO	72.03	54.39	0.197
EKF	360 REA-APC	45.66	44.50	0.202
NEW	360 REA-APC	61.43	38.54	0.171
CTD	360 REA-APC	89.58	64.40	0.142
EKF	Average	66.61	63.55	0.196
NEW	Average	44.50	35.60	0.208
CTD	Average	66.65	53.83	0.149

Table 6.13: The evaluation data comparing a CLS network trained on SIM data (EKF) with one trained on REA data (NEW) and a third one which trains the EKF model additionaly on REA data. These models are evaluated on SIM and REA (two rooms) data-sets.

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
EKF	360 SIM-WH EKF	88.15	48.04	0.109
NEW	360 SIM-WH EKF	34.57	23.54	0.127
CTD	360 SIM-WH EKF	37.83	27.82	0.096
EKF	360 REA-BIO	49.00	52.47	0.254
NEW	360 REA-BIO	65.50	51.32	0.177
CTD	360 REA-BIO	82.04	59.64	0.175
EKF	360 REA-APC	39.44	43.57	0.196
NEW	360 REA-APC	83.21	54.88	0.119
CTD	360 REA-APC	94.06	59.96	0.105
EKF	Average	58.86	48.03	0.186
NEW	Average	61.09	43.25	0.141
CTD	Average	71.31	49.14	0.126

Table 6.14: The evaluation data comparing a CLS-RAY network trained on SIM data (EKF) with one trained on REA data (NEW) and a third one which trains the EKF model additionaly on REA data. These models are evaluated on SIM and REA (two rooms) data-sets.

The evaluation of the two classification approaches revealed, that the CLS-RAY is able to perform much better in real-world scenarios than the CLS networks. Although the CLS-RAY approach performed weak in previous evaluations, especially with regards to the recall, it demonstrated its ability to generalize in real-world environments and achieve high precision rates of more than 90%. Beyond that the error in the position estimates is about $3cm$ lower than for the CLS models. Since this is another important property in order to support another robots localization, the CLS-RAY approach seems to be more suitable for real-world applications of this algorithm.

Model	Input Data	$Prec_{COB}$	$Prec_{RAW}$	$Recall_{COB}$	$Recall_{RAW}$	μ_{COB}	μ_{RAW}
CTD-CLS	360 REA-APC	98.20	80.96	82.68	46.12	0.110	0.174
CTD-CLS	360 REA-BIO	98.86	45.21	90.01	18.77	0.180	0.213
CTD-RAY	360 REA-APC	98.40	89.71	65.93	53.99	0.095	0.116
CTD-RAY	360 REA-BIO	92.82	71.25	86.42	32.86	0.168	0.182

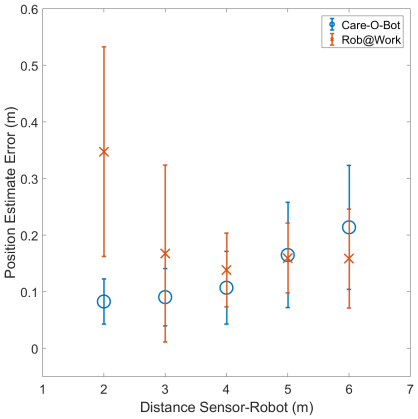
Table 6.15: The evaluation data comparing the influence of the robot shapes on the classification performance of both CTD classification models on two different REA environments.

6.9 Impact of Object Distance & Shape

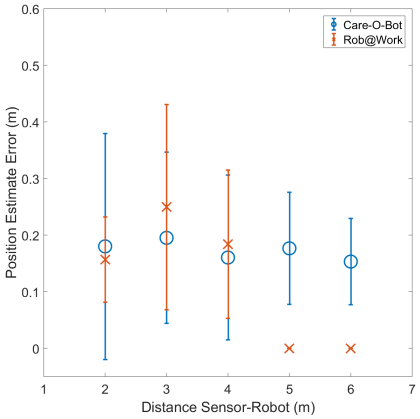
In the experiments conducted so far the distance and shape of the object to be classified were not taken into consideration. This section focuses on the question, whether the distance to the detected object influences the accuracy of the position estimate. It is also examined whether the precision, recall and position estimation accuracy for the robot classification is dependent on the robots shape. This entire section is therefore only about the robot classification.

The results of the evaluation regarding the robots distance are shown in figure 6.5 for the CLS/CLS-RAY CTD models. For both models the position estimation error for Care-O-Bots increases with the distance to the robot. The error in estimating a position for a Rob@Work, however, is similarly independent from the distance to the ego-vehicle as the results of both robots in the LAB environment. For the parts of the plot where the mean as well as the standard deviation of the error is both zero, no robots of that type where detected in the respective distance during the data gathering. These results show that especially the position estimation of the Care-O-Bot is dependent on the distance to the ego-vehicle. This applies to both approaches CLS and CLS-RAY. The different results in the LAB compared to the APC are most likely due to the small size of the room. To further investigate this another data-set would need to be collected in a different larger room. Furthermore the shape does not seem to make a big difference when it comes to estimating the position accurately. The error in estimating a position for the Rob@Work is in most cases slightly higher, but there are also a few where the Care-O-Bots error is higher.

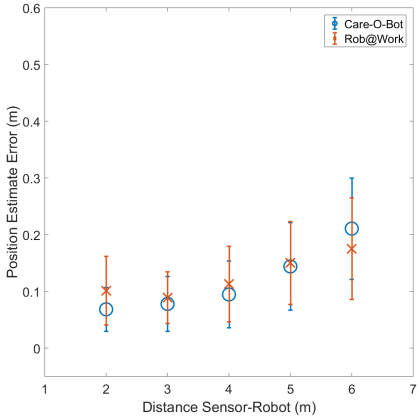
To further compare the influence of the two robots shapes on the evaluation data another experiment was conducted using the same models and input data as before. Table 6.15 visualizes the results. These clearly demonstrate, that the Care-O-Bot is significantly easier to detect than the Rob@Work. In all metrics and all experiments, the Care-O-Bot outperforms the Rob@Work. The reason for that is the shape of the two robots. Which is also an explanation for the increased position estimation error for the Rob@Work, since it is larger the error in estimating a position is thus also larger. The Care-O-Bot shape is unique compared to the most common object shapes in 2D LIDAR scans (rectangular), it is therefore easier to detect and distinguish it from other objects in the environment. Many objects or even walls have a rectangular shape, which is difficult to distinguish from the Rob@Works shape. This is the reason why the pre-processing of the data was introduced. By detecting dynamic objects first these networks eliminate many potential misclassifications. This will be further elucidated in the following section comparing the classification networks performance on the pre-processed data to data-sets without pre-processing.



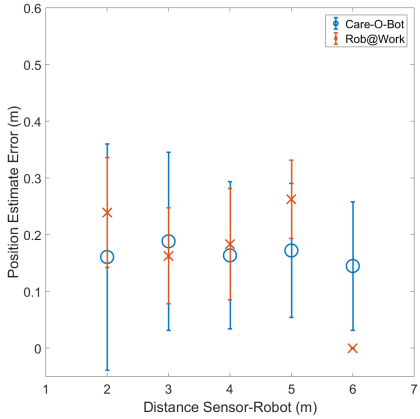
(a) CLS model on APC data.



(b) CLS model on LAB data.



(c) CLS-RAY model on APC data.



(d) CLS-RAY model on LAB data.

Figure 6.5: The mean error and standard deviation of the position estimation using the CLS/CLS-RAY CTD model on data-sets from the LAB and APC rooms. Each bar at x represents all position estimates between $x-1$ and x. There are no position estimates for $x < 1$ because the size of their respective shapes prevents such close constellations.

6.10 Pre-processing Improvement

In the last part of the evaluation the CTD models of the CLS and the CLS-RAY are compared against each other when applied to the pre-processed data only including the pixels of dynamic objects. Four different scenarios in both of the familiar real-world environments are considered. The first one uses the same data as in the experiments before including static and dynamic pixels. The second one uses only dynamic pixels as labeled manually using the robots position and bounding boxes. The last two scenarios feed the information calculated by the DYN-LSTM and DYN-MAP-LSTM models into the respective classification network.

The results for the CLS model are shown in table 6.16. Using the standard input leads in both environments to the highest and second highest precision, respectively. Only in the LAB it is possible to increase the precision by pre-processing the data. We expected that with the pre-processing

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
CLS-CTD	360 REA-APC BOTH	89.58	64.40	0.142
CLS-CTD	360 REA-APC	88.90	67.64	0.143
CLS-CTD	360 REA-APC LSTM	84.92	74.41	0.128
CLS-CTD	360 REA-APC MAP	75.14	69.02	0.167
CLS-CTD	360 REA-BIO BOTH	72.03	54.39	0.197
CLS-CTD	360 REA-BIO	81.91	50.04	0.166
CLS-CTD	360 REA-BIO LSTM	67.60	86.39	0.218
CLS-CTD	360 REA-BIO MAP	59.93	75.00	0.225
CLS-CTD	Average	77.50	67.66	17.31

Table 6.16: The evaluation data comparing the performance of the CLS-CTD model using the DYN-LSTM and DYN-MAP-LSTM approaches together with a labeling based on the robots positions for pre-processing the data.

step, the precision would increase, since the number of false positives would decrease due to the reduction of unnecessary data (static pixels) which might be falsely labeled as a robot. On the other hand we expected the recall to decrease, because the pre-processing might eliminate some important information of the robots shapes. However, since the model used in this scenario was trained on both static and dynamic data, we assume that the reduced information due to the pre-processing leads to the decrease in the precision in almost all of the cases. The network adapted to the information (static & dynamic) which has now changed. The recall in these cases increases due to the reduced information as well. The new information does lead to more false positives but also to a reduced number of false negatives. The reason for that is that the network can separate the robot shapes more clearly from the fewer static pixels. The impact of the pre-processing on the position estimation is relatively small in both environments. There is no recognizable pattern in the results.

In contrast to the CLS model, the CLS-RAY model is able to maintain a high precision while increasing the recall for the APC in all cases by around 15% as shown in table 6.17. In general the model shows a similar behavior to the previously described results for the CLS model. There are two main differences, however. First, the recall in the LAB environment decreases with the pre-processed data compared to an increase for the CLS model. The reason for that is the reduced generalizability of the CLS-RAY approach due to the differences in the data representation. This has already been discusses in the previous parts of the evaluation. The second disparity from the CLS results are the accurate position estimates in the known environment (APC) compared to the 1.7 times higher error when using pre-processing. The reason for that is the issue of generalization again, since with uncertainty in the classifications the uncertainty in the position estimate rises as well.

Overall we can conclude, that the pre-processing of the data increases the recall of the model in most cases, independently from the chosen input (CLS/CLS-RAY). The difference in the performance of the network using either the output of the DYN-LSTM or DYN-MAP-LSTM model is only minor. The DYN-LSTM model, however, outperforms the DYN-MAP-LSTM in each of the given scenarios, although it only detects semi-dynamic objects. This could be at least somehow due to the unmapped objects in the environments. A deeper analysis of these two networks using more complete grid-maps as an input for the DYN-MAP-LSTM network could produce more insightful

Model	Input Data	Prec	Recall	$\mu_{pos-error}$
CLS-RAY-CTD	360 REA-APC BOTH	94.06	59.96	0.105
CLS-RAY-CTD	360 REA-APC	94.78	79.88	0.100
CLS-RAY-CTD	360 REA-APC LSTM	91.56	74.31	0.111
CLS-RAY-CTD	360 REA-APC MAP	84.95	75.61	0.109
CLS-RAY-CTD	360 REA-BIO BOTH	82.04	59.64	0.175
CLS-RAY-CTD	360 REA-BIO	81.79	67.50	0.141
CLS-RAY-CTD	360 REA-BIO LSTM	63.36	58.68	0.176
CLS-RAY-CTD	360 REA-BIO MAP	53.50	56.49	0.191
CLS-RAY-CTD	Average	80.75	66.51	13.88

Table 6.17: The evaluation data comparing the performance of the CLS-RAY-CTD model using the DYN-LSTM and DYN-MAP-LSTM approaches together with a labeling based on the robots positions for pre-processing the data.

results. Such a comparison using much more real-world data would also clarify whether the CLS or the CLS-RAY network performs better. In average, the CLS-RAY network outperforms the CLS one in precision and position accuracy while maintaining almost the same recall as the CLS network. The differences are only minor and could be examined in more detail by training the networks using data from multiple different real-world environments. This would especially enlighten the true ability of the CLS-RAY approach to generalize to different environments. This will be further discussed in the future work of this thesis (see section 7.3). The superiority of the CLS-RAY in most scenarios, especially in precision and position accuracy, indicated that this approach is the most promising for a real-world application.

7 Conclusion

7.1 Summary

The mutual detection and classification of robots in a heterogeneous fleet using only 2D LIDAR data is a complex task. The shape of a robot as perceived by a 2D LIDAR scanner can change dramatically between different scans especially in real-world scenarios. The slightest changes in roll and pitch of the robot carrying the sensor often leads to a completely different shape in the scan.

In this work a novel approach for mutual robot detection and relative position estimation based on a combination of convolutional and ConvLSTM layers was presented in order to solve this issue. The algorithm learned an end-to-end classification of robot shapes using only 2D LIDAR information transformed into a grid-map. Subsequently multiple hypotheses for each robot type are extracted out of the heatmap output of the network using a hierarchical clustering algorithm combined with a centroid calculation for each hypothesis. These position hypotheses are used in an overall multi-robot localization in order to increase the localization of the robots by sharing this information. Due to the similarities that many robot shapes in 2D LIDAR data share with static objects in the environment (rectangles, circles), a pre-processing of the data was implemented in order to improve the algorithms' performance.

Three different end-to-end approaches for semi- and fully-dynamic object detection were introduced. The first one used stacked laserscans with a CNN to detect spatio-temporal features of moving objects. The second one transforms the sensor data into a 2D grid-map and accumulates multiple consecutive maps to create a map with spatio-temporal features for moving objects. This map is then used as an input for a ConvLSTM network with two layers. Both approaches only detect semi-dynamic objects, since the spatio-temporal features in both cases are only visible for robots moving at the currently processed time-span. Another simple approach was presented for extracting fully-dynamic objects by calculating the difference between the static grid-map provided by the robots LTS and the gird-map calculated from the laserscan. This approach only uses a network of two convolutional layers but relies on a grid-map including recent information about static objects.

7.2 Conclusion

The evaluation of these approaches was conducted in simulation and real-world environment to determine the impact of the architecture, unknown data, uncertainty in the localization, the objects distance and shape as well as the improvement of the pre-processing of the data. Furthermore the performance of the different networks in simulation were compared to the performance on real-world data. These experiments have shown that the 1D stacked approach for detecting dynamic objects does not generalize at all to changes in the environment. Although trained on data gathered

in completely different rooms, only the slightest changes within the static environment reduce the precision and recall of the network dramatically. This behavior renders the algorithm impractical for most real-world applications, since the assumption that the static objects in the environment don't change are not feasible in most cases. On known, unchanged environments, however, the algorithm outperformed the other two approaches.

The ConvLSTM-based dynamic object detection performed much better in the conducted experiments when applied to unknown environments or dynamic objects. It demonstrated its ability to detect the spatio-temporal features and not just the shapes of the dynamic objects used in the training. The approach further showed that it is better able to cope with uncertainty when trained on data including uncertainty (in the localization). This combined with the fact that a model pre-trained on simulation data and further trained on real-world data performs better in real-world scenarios then a model trained from scratch, makes the approach suitable for real-world applications. Data for real-world scenarios is generally much harder to collect than simulation data. Therefore the ability of a network trained on simulation data to perform well on real-world data using only a small number of real-world training examples is crucial.

The last approach for dynamic object detection demonstrated in the evaluation, that it relies heavily on an accurate and complete static grid-map including all static objects in the environment. This is generally provided by the overall localization system of the robot. However, the map is never 100% accurate and the results in e.g. a room with only a few additional, unmapped objects show that the performance of the network then drops significantly. This is obvious because the network solely computes the difference between the two input frames, i.e. it outputs all pixels belonging to unmapped objects. One advantage of using this convolutional operations instead of a plain subtraction of the arrays is the robustness of convolutions to small shifts in the input data. With an increasing kernel size, the network is able to cope with larger shifts in the data, since the receptive field of the neurons increases. The experimental results showed that this property allows the convolutional approach to outperform the subtraction approach in all cases. Hence, as long as a accurate representation of all static objects in the environment is provided, this approach works well in all scenarios.

The network introduced for classifying the robots was evaluated on two different kinds of grid-maps. One with and the other one without labeling the unknown space behind an object. The different evaluations showed that both approaches are able to build a precise model that generalizes well to different environments even without pre-processing the data. Both are able to cope well with induced uncertainty in the localization while the one with unknown space in general achieves more precise position estimates. Despite the fact, that the model learned without unknown space in the input performs much better in all simulation scenarios, the model with unknown space achieves a higher precision and lower error in the position estimates when applied to real-world environments. This is most likely due to the increased information contained in the input data. The networks achieved a precision of 90% (without unknown) and 94% (with unknown) on real-world data while maintaining a recall of about 60%. This is sufficient for the purpose of mutual robot detection and position, since it generally decreases the rate at which a robot is detected by about 40%. The precision of the robot classification is more important than the recall, since for the purpose of mutually increasing the localization of multiple robots decreasing the number of false positives is of higher importance than decreasing the number of false negatives. A wrong classification is worse than not classifying a robot at all at a given time-step, since the frequency of a LIDAR is usually high enough ($10Hz$). Together with the an average position estimation error of 13% and 15% for

the network using data with and without marked unknown space, both networks seem capable of being able to improve the localization of the robots within a heterogeneous fleet. The evaluation of this, however, was not part of this thesis. For a complete evaluation of the full multi-robot localization system we refer the reader to Dörr [Dör].

The distance of the robots only made a small impact on the classification performance while the robots shape on the other hand can make a huge difference. Some shapes as the one of the Rob@Work (rectangular) look similar to many static objects like boxes, hence leading to an increased number of false positives. To reduce this number the networks were evaluated on the pre-processed data obtained by the dynamic object detection networks. The results of these experiments confirmed the results from the real-world comparison, the model which incorporates unknown data performed on average better than the one which doesn't. The difference compared to using the whole scan data as an input is small. There is no significant benefit in pre-processing the data. The precision of both classification approaches even decreases in all cases. The reason for that is most likely, that the networks are trained using the full scan data. By training the networks only on data which was predicted by a dynamic object detection network, the network might adapt to the different data and increase in performance.

Overall both, the dynamic object detection and the classification networks achieve promising and satisfying results in simulation as well as real-world environments. Using spatio-temporal features in combination with a stateful LSTM led to a powerful dynamic object detector even when used on a moving platform. Using the same accumulated input data for the classification network decreased its performance on the other hand. The stateful LSTM layers used in both approaches achieved a great performance on continuous data. Compared to using a sequential input of size n on a stateless LSTM, the unnecessary overhead of storing the consecutive scan data is no longer necessary to perform. Furthermore the computational time of a forward pass through the network is reduced since not a sequence but a single data sample is processed. This approach is therefore much easier to apply to real-world detection, classification and tracking problems like the ones presented in this thesis.

The weak generalizability of the 1D dynamic object detection compared to the 2D LSTM-based approach demonstrates that using CNNs on grid-maps instead of plain 2D LIDAR data (1D array) increases the performance of the network. The network trained to identify the accumulated spatio-temporal features is able to generalize to different environments without a problem while the 1D approach decreases significantly in performance. This leads to the assumption that the 2D grid-maps simply contain more features that can be easier learned by the respective networks. At the beginning of this work we also tested a few network architectures using the plain sensor data in order to classify the robots. None achieved promising results. This idea was therefore abandoned, especially since the following tests using grid-maps as an input performed much better.

7.3 Future Work

The results and insights gained throughout this work on the representation of LIDAR data and the performance of CNNs on this data are promising on the one hand but also show that there are still a lot of open questions and open research to do. Probably the most crucial task in future work on this topic is to gather much more data, in simulation but especially in real-world environments. The presented approaches have shown promising results, especially when applied to real-world data,

but they have also demonstrated the lack of samples used for training the network in these cases. Training the networks on 10-20000 examples from 3-4 different rooms instead of 3000 examples from one room will allow a more precise estimate of their performance. This will help in choosing the right data representation (with/without unknown data) for the final model.

The pre-processing of the data can be evaluated again as well. In chapter 6 we already came to the conclusion, that the classification networks need to be trained on the output of the dynamic networks in order to achieve a valid evaluation result. To realize this, two more sets of real-world data from multiple rooms needs to be collected. The dynamic object detection will be trained on the first set of the data. Afterwards we apply the resulting models to the second set and use the output data to train the classification networks. These will be also trained on the plain second data set without pre-processing. Finally, the result can be evaluated on the previously collected real-world evaluation data from different rooms. This will hopefully give a better insight on the networks performance using pre-processed data.

In order to enable a more accurate assessment of the performance of the classification networks, related state-of-the-art methods using a traditional, model-based approach (ICP) as well as another learning based approach (SVM) can be implemented. They first need to be adapted to the specific problem of this thesis (robot shapes, etc.). The classification performance of these methods can then be compared against the ones from our networks. This will set the results of this thesis into a better context.

Bibliography

[18a] *Long Short-Term Memory Cell*. July 2018. URL: https://i.stack.imgur.com/SjiQE.png (cit. on p. 32).

[18b] *RNN-Architecture Types*. May 2018. URL: http://karpathy.github.io/assets/rnn/diags.jpeg (cit. on p. 31).

[18c] *RNN-Unfold*. May 2018. URL: http://www.wildml.com/wp-content/uploads/2015/09/rnn.jpg (cit. on p. 30).

[AA08] J. Almeida, R. Araujo. "Tracking multiple moving objects in a dynamic environment for autonomous navigation". In: *2008 10th IEEE International Workshop on Advanced Motion Control*. Mar. 2008, pp. 21–26. DOI: 10.1109/AMC.2008.4516035 (cit. on pp. 19, 36).

[AAA05] J. Almeida, A. Almeida, R. Araujo. "Tracking multiple moving objects for mobile robotics navigation". In: *2005 IEEE Conference on Emerging Technologies and Factory Automation*. Vol. 1. Sept. 2005, 8 pp.–210. DOI: 10.1109/ETFA.2005.1612521 (cit. on pp. 15, 36).

[APPN16] A. Asvadi, C. Premebida, P. Peixoto, U. Nunes. "3D Lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes". In: *Robotics and Autonomous Systems* 83 (Sept. 2016), pp. 299–311. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.06.007. URL: http://www.sciencedirect.com/science/article/pii/S0921889016300483 (cit. on p. 38).

[BA04] G. A. Borges, M.-J. Aldon. "Line Extraction in 2D Range Images for Mobile Robotics". In: *J. Intell. Robotics Syst.* 40.3 (July 2004), pp. 267–297. ISSN: 0921-0296. DOI: 10.1023/B:JINT.0000038945.55712.65. URL: https://doi.org/10.1023/B:JINT.0000038945.55712.65 (cit. on p. 40).

[Bal87] D. H. Ballard. "Modular Learning in Neural Networks". In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*. AAAI'87. Seattle, Washington: AAAI Press, 1987, pp. 279–284. ISBN: 978-0-934613-42-2. URL: http://dl.acm.org/citation.cfm?id=1863696.1863746 (cit. on p. 28).

[BD06] A. Bachmann, T. Dang. "Multiple Object Detection under the Constraint of Spatiotemporal Consistency". In: *2006 IEEE Intelligent Transportation Systems Conference*. Model Free segmentation, tracking. Sept. 2006, pp. 295–300. DOI: 10.1109/ITSC.2006.1706757 (cit. on p. 39).

[Ber18] L. Bertinetto. *siamese-fc: Arbitrary object tracking at 50-100 FPS with Fully Convolutional Siamese networks*. original-date: 2016-08-30T16:13:13Z. Apr. 2018. URL: https://github.com/bertinetto/siamese-fc (cit. on p. 44).

[BKC15] V. Badrinarayanan, A. Kendall, R. Cipolla. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *arXiv:1511.00561 [cs]* (Nov. 2015). arXiv: 1511.00561. URL: http://arxiv.org/abs/1511.00561 (cit. on pp. 18, 21, 28, 43, 53).

[BM92] P. J. Besl, N. D. McKay. “Method for registration of 3-D shapes”. In: *Sensor Fusion IV: Control Paradigms and Data Structures*. Vol. 1611. International Society for Optics and Photonics, Apr. 1992, pp. 586–607. DOI: 10.1117/12.57955. URL: https://www.spiedigitallibrary.org/conference-proceedings-of-spie/1611/0000/Method-for-registration-of-3-D-shapes/10.1117/12.57955.short (cit. on p. 37).

[Bre65] J. E. Bresenham. “Algorithm for computer control of a digital plotter”. In: *IBM Systems Journal* 4.1 (1965). 02161, pp. 25–30. ISSN: 0018-8670. DOI: 10.1147/sj.41.0025 (cit. on p. 58).

[BRWW17] G. Brunner, O. Richter, Y. Wang, R. Wattenhofer. “Teaching a Machine to Read Maps with Deep Reinforcement Learning”. In: *arXiv:1711.07479 [cs, stat]* (Nov. 2017). arXiv: 1711.07479. URL: http://arxiv.org/abs/1711.07479 (cit. on p. 43).

[BSF94] Y. Bengio, P. Simard, P. Frasconi. “Learning long-term dependencies with gradient descent is difficult”. eng. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181 (cit. on p. 31).

[BVH+16] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, P. H. S. Torr. “Fully-Convolutional Siamese Networks for Object Tracking”. In: *arXiv:1606.09549 [cs]* (June 2016). arXiv: 1606.09549. URL: http://arxiv.org/abs/1606.09549 (cit. on p. 43).

[CA16] R. O. Chavez-Garcia, O. Aycard. “Multiple Sensor Fusion and Classification for Moving Object Detection and Tracking”. In: *IEEE Transactions on Intelligent Transportation Systems* 17.2 (Feb. 2016), pp. 525–534. ISSN: 1524-9050. DOI: 10.1109/TITS.2015.2479925 (cit. on p. 39).

[CC89] Y. L. Cun, Y. L. Cun. *Steels (eds) ’Connectionism in perspective’, Elsevier 1989. Generalization and Network Design Strategies*. 1989 (cit. on p. 21).

[CJB+89] Y. L. Cun, L. D. Jackel, B. Boser, J. S. Denker, H. P. Graf, I. Guyon, D. Henderson, R. E. Howard, W. Hubbard. “Handwritten digit recognition: applications of neural network chips and automatic learning”. In: *IEEE Communications Magazine* 27.11 (Nov. 1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400 (cit. on pp. 21, 42).

[DAG+15] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, T. Darrell. “Long-Term Recurrent Convolutional Networks for Visual Recognition and Description”. In: 2015, pp. 2625–2634. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Donahue_Long-Term_Recurrent_Convolutional_2015_CVPR_paper.html (cit. on p. 33).

[DCTB16] A. Dewan, T. Caselitz, G. D. Tipaldi, W. Burgard. “Motion-based detection and tracking in 3D LiDAR scans”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. Bayesian approach, NO prior shape data. May 2016, pp. 4508–4513. DOI: 10.1109/ICRA.2016.7487649 (cit. on p. 38).

[DHS11] J. Duchi, E. Hazan, Y. Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. en. In: (2011), p. 39 (cit. on p. 26).

[Dör] S. Dörr. “Cloud-based Cooperative Long-Term SLAM for Mobile Robots in Industrial Applications”. Doctoral Thesis. Print-pending: ISW, University of Stuttgart (cit. on p. 91).

[DRO+16] J. Dequaire, D. Rao, P. Ondruska, D. Wang, I. Posner. “Deep Tracking on the Move: Learning to Track the World from a Moving Vehicle using Recurrent Neural Networks”. In: *arXiv:1609.09365 [cs]* (Sept. 2016). arXiv: 1609.09365. URL: http://arxiv.org/abs/1609.09365 (cit. on pp. 18, 19, 45, 54).

[DUV+14] B. Douillard, J. Underwood, V. Vlaskine, A. Quadros, S. Singh. “A Pipeline for the Segmentation and Classification of 3D Point Clouds”. en. In: *Experimental Robotics*. Ed. by O. Khatib, V. Kumar, G. Sukhatme. Vol. 79. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 585–600. ISBN: 978-3-642-28571-4 978-3-642-28572-1. DOI: 10.1007/978-3-642-28572-1_40. URL: http://link.springer.com/10.1007/978-3-642-28572-1_40 (cit. on p. 41).

[DZMS14] J. Deng, Z. Zhang, E. Marchi, B. Schuller. “Sparse Autoencoder-Based Feature Transfer Learning for Speech Emotion Recognition”. In: *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction(ACII)*. Sept. 2014, pp. 511–516. DOI: 10.1109/ACII.2013.90. URL: doi.ieeecomputersociety.org/10.1109/ACII.2013.90 (cit. on p. 28).

[ELSG09] A. Ess, B. Leibe, K. Schindler, L. v. Gool. “Moving obstacle detection in highly dynamic scenes”. In: *2009 IEEE International Conference on Robotics and Automation*. Pedestrians. May 2009, pp. 56–63. DOI: 10.1109/ROBOT.2009.5152884 (cit. on p. 39).

[FB17] H. Farazi, S. Behnke. “Online visual robot tracking and identification using deep LSTM networks”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Sept. 2017, pp. 6118–6125. DOI: 10.1109/IROS.2017.8206512 (cit. on p. 44).

[FB81] M. A. Fischler, R. C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: http://doi.acm.org/10.1145/358669.358692 (cit. on p. 38).

[FBKT00] D. Fox, W. Burgard, H. Kruppa, S. Thrun. “A Probabilistic Approach to Collaborative Multi-Robot Localization”. en. In: *Autonomous Robots* 8.3 (June 2000), pp. 325–344. ISSN: 0929-5593, 1573-7527. DOI: 10.1023/A:1008937911390. URL: https://link.springer.com/article/10.1023/A:1008937911390 (cit. on pp. 16, 41).

[FHM02] A. Fod, A. Howard, M. A. J. Mataric. “A laser-based people tracker”. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 3. X. 2002, pp. 3024–3029. DOI: 10.1109/ROBOT.2002.1013691 (cit. on p. 36).

[FOS09] A. Franchi, G. Oriolo, P. Stegagno. “Mutual localization in a multi-robot system with anonymous relative position measures”. In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Oct. 2009, pp. 3974–3980. DOI: 10.1109/IROS.2009.5354560 (cit. on p. 16).

[GBC16] I. Goodfellow, Y. Bengio, A. Courville. *Deep learning*. Adaptive computation and machine learning. Cambridge, Massachusetts: The MIT Press, 2016. ISBN: 978-0-262-03561-3 (cit. on pp. 21–24, 28, 29, 31).

[GDDM13] R. Girshick, J. Donahue, T. Darrell, J. Malik. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *arXiv:1311.2524 [cs]* (Nov. 2013). R-CNN. URL: http://arxiv.org/abs/1311.2524 (cit. on p. 42).

[GGZC15] Q. Gan, Q. Guo, Z. Zhang, K. Cho. "First Step toward Model-Free, Anonymous Object Tracking with Recurrent Neural Networks". In: *arXiv:1511.06425 [cs]* (Nov. 2015). RNN-Tracking. URL: http://arxiv.org/abs/1511.06425 (cit. on p. 44).

[Gir15] R. Girshick. "Fast R-CNN". In: *arXiv:1504.08083 [cs]* (Apr. 2015). Fast R-CNN. URL: http://arxiv.org/abs/1504.08083 (cit. on p. 42).

[Gra12] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. en. Vol. 385. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-24796-5 978-3-642-24797-2. DOI: 10.1007/978-3-642-24797-2. URL: http://link.springer.com/10.1007/978-3-642-24797-2 (cit. on pp. 29, 31).

[Hin12] G. Hinton. *Neural Networks for Machine Learning*. 2012. URL: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf (cit. on p. 26).

[Hoc91] S. Hochreiter. "Untersuchungen zu dynamischen neuronalen Netzen". In: (Apr. 1991) (cit. on p. 31).

[HZRS15] K. He, X. Zhang, S. Ren, J. Sun. "Deep Residual Learning for Image Recognition". In: *arXiv:1512.03385 [cs]* (Dec. 2015). Microsoft ResNet. URL: http://arxiv.org/abs/1512.03385 (cit. on pp. 18, 42).

[JKRL09] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th International Conference on Computer Vision*. Sept. 2009, pp. 2146–2153. DOI: 10.1109/ICCV.2009.5459469 (cit. on p. 25).

[Joh67] S. C. Johnson. "Hierarchical clustering schemes". en. In: *Psychometrika* 32.3 (Sept. 1967), pp. 241–254. ISSN: 0033-3123, 1860-0980. DOI: 10.1007/BF02289588. URL: http://link.springer.com/10.1007/BF02289588 (cit. on p. 60).

[KB14] D. P. Kingma, J. Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980. URL: http://arxiv.org/abs/1412.6980 (cit. on p. 26).

[KMM15] S. E. Kahou, V. Michalski, R. Memisevic. "RATM: Recurrent Attentive Tracking Model". In: *arXiv:1510.08660 [cs]* (Oct. 2015). RNN-Tracking. URL: http://arxiv.org/abs/1510.08660 (cit. on p. 44).

[KO11] B. Karlik, A. V. Olgac. "Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks". en. In: (2011), p. 13 (cit. on p. 25).

[KSH12] A. Krizhevsky, I. Sutskever, G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on pp. 21, 42).

[LAS17] K. G. Lore, A. Akintayo, S. Sarkar. "LLNet: A deep autoencoder approach to natural low-light image enhancement". In: *Pattern Recognition* 61 (Jan. 2017), pp. 650–662. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2016.06.008. URL: http://www.sciencedirect.com/science/article/pii/S003132031630125X (cit. on p. 28).

[LE01] M. Lindstrom, J. O. Eklundh. "Detecting and tracking moving objects from a mobile platform using a laser range scanner". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 3. X. 2001, 1364–1369 vol.3. DOI: 10.1109/IROS.2001.977171 (cit. on pp. 15, 37).

[LSD15] J. Long, E. Shelhamer, T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: 2015, pp. 3431–3440. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Long_Fully_Convolutional_Networks_2015_CVPR_paper.html (cit. on p. 42).

[MBN04] A. Mendes, L. C. Bento, U. Nunes. "Multi-target detection and tracking with a laser scanner". In: *IEEE Intelligent Vehicles Symposium, 2004*. ! Classification of all objects. June 2004, pp. 796–801. DOI: 10.1109/IVS.2004.1336486 (cit. on p. 41).

[MDU11] P. Morton, B. Douillard, J. Underwood. "An evaluation of dynamic object tracking with 3D LIDAR". In: *Proc. of the Australasian Conference on Robotics & Automation (ACRA)*. 2011 (cit. on p. 38).

[MNM+13] C. Mertz, L. E. Navarro-Serment, R. MacLachlan, P. Rybski, A. Steinfeld, A. Suppé, C. Urmson, N. Vandapel, M. Hebert, C. Thorpe, D. Duggins, J. Gowdy. "Moving object detection with laser scanners". en. In: *Journal of Field Robotics* 30.1 (Jan. 2013), pp. 17–43. ISSN: 1556-4967. DOI: 10.1002/rob.21430. URL: http://onlinelibrary.wiley.com/doi/10.1002/rob.21430/abstract (cit. on pp. 15, 36, 38).

[MPS05] A. Martinelli, F. Pont, R. Siegwart. "Multi-Robot Localization Using Relative Observations". en. In: IEEE, 2005, pp. 2797–2802. ISBN: 978-0-7803-8914-4. DOI: 10.1109/ROBOT.2005.1570537. URL: http://ieeexplore.ieee.org/document/1570537/ (cit. on p. 16).

[MSRD06] M. Mahlisch, R. Schweiger, W. Ritter, K. Dietmayer. "Sensorfusion Using Spatio-Temporal Aligned Video and Lidar for Improved Vehicle Detection". In: *2006 IEEE Intelligent Vehicles Symposium*. 2006, pp. 424–429. DOI: 10.1109/IVS.2006.1689665 (cit. on p. 39).

[MTW02] M. Montemerlo, S. Thrun, W. Whittaker. "Conditional particle filters for simultaneous mobile robot localization and people-tracking". In: *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. Vol. 1. 2002, 695–701 vol.1. DOI: 10.1109/ROBOT.2002.1013439 (cit. on pp. 15, 37).

[ODWP16] P. Ondruska, J. Dequaire, D. Z. Wang, I. Posner. "End-to-End Tracking and Semantic Segmentation Using Recurrent Neural Networks". In: *arXiv:1604.05091 [cs]* (Apr. 2016). arXiv: 1604.05091. URL: http://arxiv.org/abs/1604.05091 (cit. on pp. 15, 18, 19, 36, 43, 44, 54).

[OP16] P. Ondruska, I. Posner. "Deep Tracking: Seeing Beyond Seeing Using Recurrent Neural Networks". In: *arXiv:1602.00991 [cs]* (Feb. 2016). arXiv: 1602.00991. URL: http://arxiv.org/abs/1602.00991 (cit. on pp. 18, 19, 44, 54).

[PC15] P. O. Pinheiro, R. Collobert. "From image-level to pixel-level labeling with convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 1713–1721 (cit. on p. 42).

[PN06] C. Premebida, U. Nunes. "A Multi-Target Tracking and GMM-Classifier for Intelligent Vehicles". en. In: IEEE, 2006, pp. 313–318. ISBN: 978-1-4244-0093-5. DOI: 10.1109/ITSC.2006.1706760. URL: http://ieeexplore.ieee.org/document/1706760/ (cit. on pp. 18, 40, 41).

[PSJV15] S. Paisitkriangkrai, J. Sherrah, P. Janney, A. Van-Den Hengel. "Effective Semantic Pixel Labelling With Convolutional Networks and Conditional Random Fields". In: 2015, pp. 36–43. URL: https://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W13/html/Paisitkriangkrai_Effective_Semantic_Pixel_2015_CVPR_paper.html (cit. on p. 42).

[QCS+16] B. Qin, Z. J. Chong, S. H. Soh, T. Bandyopadhyay, M. H. Ang, E. Frazzoli, D. Rus. "A Spatial-Temporal Approach for Moving Object Recognition with 2D LIDAR". en. In: *Experimental Robotics*. Springer Tracts in Advanced Robotics. Springer, Cham, 2016, pp. 807–820. ISBN: 978-3-319-23777-0 978-3-319-23778-7. URL: https://link.springer.com/chapter/10.1007/978-3-319-23778-7_53 (cit. on pp. 18, 39, 50).

[RB02] S. I. Roumeliotis, G. A. Bekey. "Distributed multirobot localization". In: *IEEE Transactions on Robotics and Automation* 18.5 (Oct. 2002), pp. 781–795. ISSN: 1042-296X. DOI: 10.1109/TRA.2002.803461 (cit. on p. 16).

[RHGS15] S. Ren, K. He, R. Girshick, J. Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *arXiv:1506.01497 [cs]* (June 2015). Faster R-CNN. URL: http://arxiv.org/abs/1506.01497 (cit. on pp. 18, 21, 42).

[RHW86] D. E. Rumelhart, G. E. Hinton, R. J. Williams. "Learning representations by back-propagating errors". en. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://www.nature.com/articles/323533a0 (cit. on p. 28).

[Rud16] S. Ruder. "An overview of gradient descent optimization algorithms". In: *arXiv:1609.04747 [cs]* (Sept. 2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (cit. on p. 26).

[SCW+15] X. SHI, Z. Chen, H. Wang, D.-Y. Yeung, W.-k. Wong, W.-c. WOO. "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, R. Garnett. Curran Associates, Inc., 2015,

pp. 802–810. URL: http://papers.nips.cc/paper/5955-convolutional-lstm-network-a-machine-learning-approach-for-precipitation-nowcasting.pdf (cit. on pp. 33, 54).

[SEZ+13] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *arXiv:1312.6229 [cs]* (Dec. 2013). arXiv: 1312.6229. URL: http://arxiv.org/abs/1312.6229 (cit. on pp. 18, 21, 42).

[SHK+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Over tting". en. In: (2014), p. 30 (cit. on p. 28).

[SLD17] E. Shelhamer, J. Long, T. Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.4 (Apr. 2017). 01385, pp. 640–651. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2572683 (cit. on pp. 18, 21, 42).

[SWY+15] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich. "Going deeper with convolutions". en. In: GoogLeNet. IEEE, June 2015, pp. 1–9. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298594. URL: http://ieeexplore.ieee.org/document/7298594/ (cit. on pp. 18, 42).

[SZ14] K. Simonyan, A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]* (Sept. 2014). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (cit. on pp. 21, 42).

[TBF05] S. Thrun, W. Burgard, D. Fox. *Probabilistic robotics*. 06654. MIT press, 2005. URL: https://books.google.com/books?hl=de&lr=&id=wjM3AgAAQBAJ&oi=fnd&pg=PR7&dq=probabilistic+robotics&ots=21oZtjqqnG&sig=KA074nDJ1TIVlZ72YhL-vM7ZEnE (cit. on p. 16).

[TL09] M. Thuy, F. P. Leon. "Non-linear, shape independent object tracking based on 2D lidar data". In: *2009 IEEE Intelligent Vehicles Symposium*. ! Car segmentation, ICP, prob. association. June 2009, pp. 532–537. DOI: 10.1109/IVS.2009.5164334 (cit. on pp. 15, 19, 37).

[TMT+16] F. Tschopp, J. N. Martel, S. C. Turaga, M. Cook, J. Funke. "Efficient convolutional neural networks for pixelwise classification on heterogeneous hardware systems". In: *Biomedical Imaging (ISBI), 2016 IEEE 13th International Symposium on*. IEEE, 2016, pp. 1225–1228 (cit. on p. 42).

[TSM15] K. S. Tai, R. Socher, C. D. Manning. "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *arXiv:1503.00075 [cs]* (Feb. 2015). arXiv: 1503.00075. URL: http://arxiv.org/abs/1503.00075 (cit. on p. 33).

[VJ01] P. Viola, M. Jones. "Rapid object detection using a boosted cascade of simple features". en. In: vol. 1. IEEE Comput. Soc, 2001, pp. I–511–I–518. ISBN: 978-0-7695-1272-3. DOI: 10.1109/CVPR.2001.990517. URL: http://ieeexplore.ieee.org/document/990517/ (cit. on p. 42).

[Wil01] S. B. Williams. "E cient Solutions to Autonomous Mapping and Navigation Problems". en. In: (2001), p. 212 (cit. on p. 38).

[WPN15] D. Z. Wang, I. Posner, P. Newman. “Model-free detection and tracking of dynamic objects with 2D lidar”. en. In: *The International Journal of Robotics Research* 34.7 (June 2015). !, pp. 1039–1063. ISSN: 0278-3649. DOI: 10.1177/0278364914562237. URL: https://doi.org/10.1177/0278364914562237 (cit. on pp. 15, 38).

[XPC+05] J. Xavier, M. Pacheco, D. Castro, A. Ruano, U. Nunes. “Fast Line, Arc/Circle and Leg Detection from Laser Scan Data in a Player Driver”. en. In: IEEE, 2005, pp. 3930–3935. ISBN: 978-0-7803-8914-4. DOI: 10.1109/ROBOT.2005.1570721. URL: http://ieeexplore.ieee.org/document/1570721/ (cit. on pp. 18, 40).

[ZC88] Y. T. Zhou, R. Chellappa. “Computation of optical flow using a neural network”. In: *IEEE 1988 International Conference on Neural Networks*. July 1988, 71–78 vol.2. DOI: 10.1109/ICNN.1988.23914 (cit. on p. 26).

[ZS05] H. Zhao, R. Shibasaki. “A novel system for tracking pedestrians using multiple single-row laser-range scanners”. In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 35.2 (Mar. 2005), pp. 283–291. ISSN: 1083-4427. DOI: 10.1109/TSMCA.2005.843396 (cit. on p. 36).

[ZSKS06] H. Zhao, X. W. Shao, K. Katabira, R. Shibasaki. “Joint Tracking and Classification of Moving Objects at Intersection Using a Single-Row Laser Range Scanner”. In: *2006 IEEE Intelligent Transportation Systems Conference*. Sept. 2006, pp. 287–294. DOI: 10.1109/ITSC.2006.1706756 (cit. on pp. 18, 40).

All links were last followed on July 26, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature