Machine Learning in Robotic Navigation

Shantia, Amir

DOI:
10.33612/diss.157435654

Machine Learning in Robotic Navigation

Deep Visual Localization and Adaptive Control

Amirhossein Shantia

Printed by - GVO drukkers & vormgevers B.V.

university of groningen

Machine Learning in Robotic Navigation

Deep Visual Localization and Adaptive Control

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Friday 19 February 2021 at 9.00 hours

by

Amirhossein Shantia

born on 9 August 1986
in Tehran, Iran

Supervisor
Prof. L.R.B. Schomaker

Co-supervisor
Dr. M.A. Wiering

Assessment Committee
Prof. R. Babuska
Prof. M. Cao
Prof. P. G. Plöger

Acronyms

A2C	Advantage Actor-Critic
ADAS	Advanced Driver Assistant Systems
AMCL	Adaptive Monte Carlo Localization
BRIEF	Binary Robust Independent Elementary Features
BVLC	Berkeley Vision and Learning Center
CNN	Convolutional Neural Networks
CUDA	Compute Unified Device Architecture
CVPR	Computer Vision and Pattern Recognition
DA	Denoising Autoencoder
DARPA	Defense Advanced Research Projects Agency
DQN	Deep Q-Network
DWA	Dynamic Window Approach
FA	Function Approximator
FANN	Fast Artificial Neural Network Library
FAST	Features from Accelerated Segment Test
GAN	Generative adversarial networks
GPS	Global Positioning System
GPU	Graphical Processing Unit
HOG	Histogram of Oriented Gradients
ICP	Iterative Closest Point

ILSVRC	ImageNet large scale visual recognition competition
LSTM	Long Short Term Memory
MLP	Multilayer Perceptron
MPC	Model Predictive Control
ORB	Oriented FAST and Rotated BRIEF
RBM	Restricted Boltzmann Machine
RC	Remote Control
RL	Reinforcement Learning
SDA	Stacked Denoising Autoencoder
SHOT	Unique Signatures of Histograms
SIFT	Scale Invariance Feature Transform
SUN	Scene Understanding
TRP	Trajectory Rollout Planner
URDF	Unified Robot Description Format

Glossary

A	Average Image of cost maps belonging to a cluster
a_t	Action at time t
$b, b^{'}$	Neural network layer bias
C_i	Set of points belonging to cluster i
$c, \hat{c}$	Number of clusters
D	Normalized distance function
I_x	Cost map image of data point x
$i, \hat{i}, j, \hat{j}$	Image coordinates
L	Loss Function
m	Robot's map
N	Number of data points in a cluster
O	Output of the neural network layer with corrupted input $\tilde{\Omega}$
Q_D	Corruption Function
Q_k	Q-value at step k
r_t	Reward at time t
s_t	State at time t
T	Bolztmann temperature
TD_{err}	Temporal Difference Error
t	Time
u_t	Robot's motion command at time t

v	Robot velocity
$W, W^{'}$	Weight Matrix
X	The X direction in the robot's Cartesian reference frame
x	The x coordinate in the robot's Cartesian reference frame
x_t	The robot position at time t
Y	The Y direction in the robot's Cartesian reference frame
y	The y coordinate in the robot's Cartesian reference frame
z_t	Sensors observation at time t
α	Learning Rate
γ	Discount Factor
ϵ	The randomness probability value of ϵ-greedy method
η	Normalizing factor
θ	The robot's angle in the Cartesian reference frame
$g_{\theta'}$, f_θ	Reconstruction and Forward Function
ς	Non-linear activation function
Ω	Neural network input vector
$\tilde{\Omega}$	Corrupted neural network input vector
$\bar{\Omega}$	Reconstructed input vector using corrupted vector $\tilde{\Omega}$
ω	The robot's angular velocity

Contents

Chapter 1

Introduction

We, humans, are often considered to be the apex of the evolution of life on earth, and perhaps, a large number of neighboring solar systems around us. Our abilities to recognize intricate patterns, explore and navigate the surroundings, share experiences with others, and learn from our mistakes seem to be the building blocks of our success. The learning process of living beings such as humans can be categorized into three fields. The foremost is evolutionary or reinforcement learning (RL), where living beings fight for their survival based on the environment's feedback. An early example is perhaps the emergence of proteorhodopsin protein in early ocean bacteria to react to sunlight for better survival (Gómez-Consarnau et al. 2007). The second is unsupervised learning, which can be attributed to how individuals react to various sensory readings during their lifetime. For example, when a human child plays with a color ring sorting toy, he looks at similarities between the objects and creates a mental model without any supervision. We make decisions based on the observations that we take from the world we live in rather than being told what to do at each step (LeCun et al. 2015). However, the importance of supervised learning should not be ignored. An individual's experience is limited throughout his lifetime, and it will die with the individual if it is not transferred to the community by supervised learning.

In the field of artificial intelligence, we have been trying to understand human abilities in learning, movement, manipulation, and communication, in order to recreate them in robots so they can help us in our daily lives, assist us in manufacturing and accompany us in space exploration in the future. While we have achieved significant progress in industrial robotics, commercial robot home cleaners, autonomous vehicles, biped and four-legged robot movement control, and space exploration robots, there is still a large gap between our current progress and the capability of an animal. The same applies to robot navigation. Robots have reached adequate movement control for either wheel-based or leg-based systems, but any operation outside of a confined environment can be problematic. In addition to sensory and physical limitations, the models that robots use to solve these problems are usually fixed and have strict boundary conditions. Using learning makes it possible to improve the current state-of-the-art methods and design models that can learn and adapt over

time.
In this thesis, we focus on robotic navigation as a whole and propose multiple methods to address some of the known problems through the use of machine learning techniques.

1.1 Robotics and Automation

The first semi-intelligent robot, Shakey, was developed by researchers from the Stanford research institute (Nilsson 1984). Shakey could dissect written commands into a list of required actions, plan paths and move around in a controlled environment without collision, and push objects. In 1989, Kuperstein and Rubinstein published their work on an adaptive neural controller for sensory-motor coordination (Kuperstein and Rubinstein 1989). Their experiments consisted of a stereo camera, a 5 degree of freedom arm, and an elongated object. Without any supervision or knowledge of the underlying mechanics of the arm, the system learned to move the end effector to grasp this object and was resistant to hardware faults and object position, length, and radius change. This was an important step toward continuous learning in robotics. While the use of industrial and laboratory robots increased, their operation domain remained confined to fixed assembly lines or controlled laboratory environments. The main reason was that the used methods to model the environment, such as the surroundings, objects, and robot movement was quite simplistic and would not hold in more complex and always changing settings. It was not until this decade that perception started to have an active role in robots that operate in complex environments. The combination of machine learning and accurate models for motion, navigation, and manipulation have allowed us to develop robots that perform complicated tasks in very complex, but still limited, environments. Google's research on robotics manipulation (Gu et al. 2017), Waymo and Tesla's autonomous vehicles, and Boston Dynamic's humanoid Atlas robot (Feng et al. 2015) are examples of such achievements. Although many of the more recent examples of (industrial) robots are impressive, essential components of the behavioral repertoire are hard coded. The scientific community is yet to provide solutions for robust performance, high-level understanding, resource awareness, and task-driven inference for robotic navigation. In brief, autonomous navigation is a key skill and the reason that it is the center of attention of this thesis.

Figure 1.1: The developed robotic platforms by the Borg team of the University of Groningen from 2011 to 2015 (van Elteren et al. 2013, Shantia, Mulder, Wolf, Timmers, van der Mark, Sandor, Knigge, van der Struijk, Vienken, Bidoia and Luneburg 2015).

1.2 Robotic Navigation

Navigation is the process or activity of accurately ascertaining one's position and planning and following a route. Therefore, we can define a robotic navigation task by answering the following questions:

- Where should the robot go?
- How should the robot move?
- Where is the robot?

Where should the robot go? The answer to this question relies on the higher-level task at hand, which is true both for living organisms and machines. If the environment is unknown, exploration becomes essential. This thesis, was part of a larger research project focusing on developing service and assistive robot technology. As part of this project, we developed several service robotic platforms (Figure 1.1) and participated in Robocup@Home competitions (Wisspeintner et al. 2009). In this competition, a set of benchmark tests is used to evaluate the robots' abilities and performance in a realistic non-standardized home environment setting with

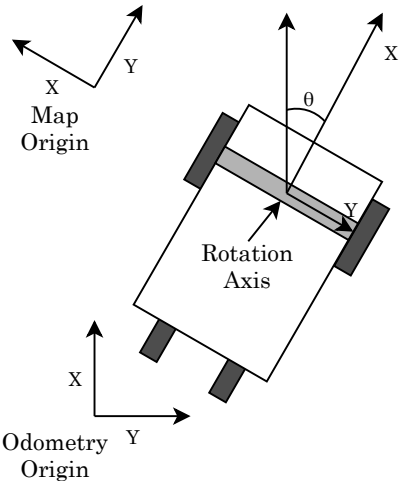


Figure 1.2: The robot's various frames of reference.

the focus on topics such as Human-Robot-Interaction and Cooperation, Navigation and Mapping in dynamic environments, Computer Vision and Object Recognition under natural light conditions, Object Manipulation, Adaptive Behaviors, Behavior Integration, Ambient Intelligence, Standardization and System Integration. Since navigation is the initial building block of our service robots, we investigate how machine learning, especially neural networks and reinforcement learning (Sutton and Barto 1998), can help us to improve the navigation of indoor robots in this thesis.

How should the robot move? In animals, muscular cells, joints, and skeletal structures enable the movement. They move by controlling their muscles through contraction and extension. For ground mobile robots, wheels are the dominant locomotion form while there has been significant research on biped or four-legged robots. This thesis focuses on wheel-based differential drive (non-holonomic) robots where the robot moves by setting a longitudinal and angular velocity.
The motion of a differential drive is constrained in a way that the translational velocity v always drives the robot in the direction of θ (Figure 1.2). Considering discretized time interval and constant velocity, the goal is to solve the general Equations 1.1 and 1.2:

$$x(t_n) = x(t_0) + \sum_{i=0}^{n-1}(F_x^i(t_i + 1))$$

where (1.1)

$$F_x^i = \begin{cases} \frac{v_i}{\omega_i}(\sin\theta(t_i) - \sin(\theta(t_i) + \omega_i \cdot (t - t_i))), \omega_i \neq 0 \\ v_i \cos(\theta(t_i)) \cdot t, \omega_i = 0 \end{cases}$$

$$y(t_n) = y(t_0) + \sum_{i=0}^{n-1}(F_y^i(t_i + 1))$$

where (1.2)

$$F_y^i = \begin{cases} -\frac{v_i}{\omega_i}(\cos\theta(t_i) - \cos(\theta(t_i) + \omega_i \cdot (t - t_i))), \omega_i \neq 0 \\ v_i \sin(\theta(t_i)) \cdot t, \omega_i = 0 \end{cases}$$

where x and y are the robot coordinates, t_i is the discretized time, and v_i and ω_i are the translational and angular velocities.

The controller should generate velocities that satisfy criteria such as smooth acceleration, absence of oscillation, and collision avoidance. This can be achieved through reactive methods such as the dynamic window approach (Fox et al. 1997), potential fields (Khatib 1986), and velocity obstacle (Alonso-Mora et al. 2018) or model predictive control (MPC) approaches (Maciejowski 2002) such as active steering (Borrelli et al. 2005) and model predictive contouring (Lam et al. 2010). The above requirements result in the importance of the connection between perception and motion of a system. A system with a lower level of input noise due to oscillation of the visual system or the inertial measuring unit can allocate a process to the task at hand rather than correcting the input noise. Just imagine reading a book in a bumpy car. The same applies to a robot.

Where is the robot? Humans have an excellent notion of their current surroundings. Even if somebody kidnaps us and puts us in an unknown location, we can describe the characteristics of this new place without much effort. Our knowledge and experience of our world allow us to identify shapes, geometry, and objects that are around us. Besides, we can correctly connect a series of observations from our senses to build a spatial map. We are so good at it that even without vision, and with some training, we can move around a block and find our starting position. We do get lost, but usually, with a hint of a landmark, a passerby, or our phone's GPS, we adjust our estimated location. We can program robots to partially perform the localization and mapping tasks by using a variety of sensors such as cameras and range

finders through detecting robust features in the image or obtained range information such as sharp edges, stable landmarks, and the topology of the place. The challenge becomes difficult when the robot moves. The robot movement is erroneous (same as us, close your eyes and try to reach the door of your apartment without touching anything), and this error adds up over time. We need to have an accurate movement model of the robot that we can associate with the robot's perception to build this spatial map. The sensor noise also plays a significant role here. There is a strong relationship between the interpretation of measurements from sensors and the movement of the robot, which makes the measurements statistically dependent. The robot should also be able to recognize the place where it traversed before for the correct creation of the map. This is called the correspondence test.
Therefore, the robot localization is formulated by Markov localization (a variant of Bayes Filter) in Equation 1.3 to track the position of the robot based on the sensory reading and robot movement:

$$P(x_t|z_{1:t}, u_{1:t}) = \alpha \cdot p(z_t|x_t, m) \cdot \int P(x_t|u_t, x_{t-1}) \cdot P(x_{t-1}) dx_{t-1} \tag{1.3}$$

where $z_{1:t}$ is the sensory reading vector, $u_{1:t}$ is the robot control vector, $x_{1:t}$ is the robot trajectory, and m is a given occupancy or landmark map. If the map is not given, then the robot has to create the spatial map online through simultaneous localization and mapping (SLAM) techniques, which are generally formulated by Equation 1.4

$$P(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{1.4}$$

that can be factorized to Equation 1.5 if we separate the trajectory estimation and the mapping steps.

$$P(x_{1:t}, m|z_{1:t}, u_{1:t}) = P(m|x_{1:t}, z_{1:t}) \cdot P(x_{1:t}|z_{1:t}, u_{1:t-1}) \tag{1.5}$$

The created map can represent the environment in a metric or semantic way. For metric map models, either landmarks and distinctive features (Torr and Zisserman 1999) , (Mur-Artal and Tardós 2017) are extracted from the sensors to create the map, or the raw dense sensor information (Lu and Milios 1997), (Nüchter 2009) is utilized for map creation. One main drawback of metric maps is the abundance of information that is not used but consumes memory. We humans do not remember all the exact details of an environment. Depending on the task at hand, we use semantic cues, and we decide the importance of objects during our navigation.

We can quickly move through a hallway or drive on the road without calculating each point's exact distance. This is perhaps why semantic maps are becoming more popular (Salas-Moreno et al. 2013), where the robot associates semantic concepts to geometrical entities in its surroundings. For a detailed overview, we encourage reading the survey papers by Cadena et al. (Cadena et al. 2016) and Saputra et al. (Saputra et al. 2018). In this thesis, we rely on metric maps and propose two neural network-based localization methods for indoor localization.

1.3 Artificial Neural Networks and Learning

The artificial neural network research is rooted in the early works by Donald Hebb (Hebb 1949), Frank Rosenblatt (Rosenblatt 1958), and Misky et al. (Minsky and Papert 1969). However, it did not grow until the maturation and implementation of Rumelhart, Hinton, and Williams's backpropagation algorithm in 1986 (Rumelhart et al. 1986). One drawback of multilayer perceptrons (MLP) is that its input is always a vector and that all the neurons are fully connected, which produces two problems. First is the sheer amount of parameters in the network that need to be trained for large input vectors (the curse of dimensionality). The second is the loss of topological information for inputs such as images or sound spectrums. It was not until 1990 when LeCun introduced the first convolutional neural network (CNN) for character recognition (LeCun et al. 1990) (Figure 1.3) that these problems were (partially) solved.

This architecture, however, still had two main problems for solving complex tasks; Increasing the number of filters would lead to overfitting, while an increase in the number of layers would cause the error not to reach the weights in the first layers (vanishing gradients). In 2010, Nair and Hinton curbed the vanishing gradient problem by introducing a new piecewise linear activation function (Nair and Hinton 2010). Shortly after, Hinton et al. proposed a dropout scheme to avoid overfitting (Hinton et al. 2012). With a considerable increase in parallel computing capability on graphical processing units (GPU) and the removal of obstacles in CNN learning, Krizhevsky et al. were able to demonstrate the learning capability of CNNs for the ImageNet large scale visual recognition competition (ILSVRC) (Krizhevsky et al. 2012). The next important invention was Google's Inception architecture (Szegedy et al. 2015). This network applies convolution filters of variable sizes in inception layers (1x1, 3x3, and 5x5). The 1x1 filter reduces dimensionality while the other filters learn to extract useful features in variable resolutions (Figure 1.4).

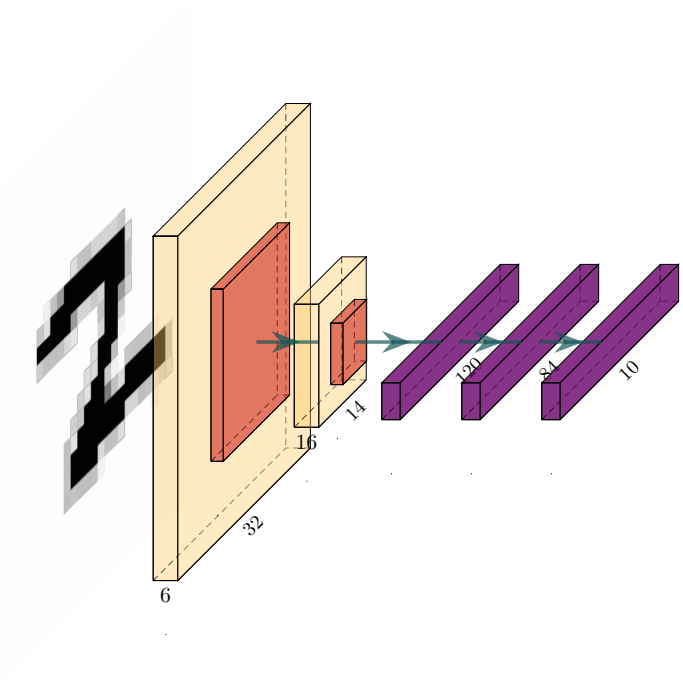


Figure 1.3: Lecun's seven layer CNN architecture for character recognition. The input topology is retained by using a set of convolution kernels (trainable filters) that repaint the image from the previous layer into a 'feature map', which is subsequently sub sampled and filtered again, until a final strand of traditional, fully connected MLP layers.

The next innovation for deep learning came with the design of ResNets, with hundreds of layers, by He et al. (He et al. 2016). To avoid the problem of vanishing gradients for very deep networks, he suggested the concept of skip connections where the output of particular layers in the network is copied and used later in the upper layers. The network can then learn a residual mapping in which it decides what part of the feature maps should be processed and skipped. In parallel, there has been numerous research that expands artifical neural networks' learning capabilities to better support unsupervised learning (Kingma and Welling 2014), capture the relation of neighboring pixels (Chen et al. 2018), facilitate high-resolution image training (Wang et al. 2020), and support data augmentation (Goodfellow et al. 2014). The performance of CNNs in object detection and recognition, lane detection, and

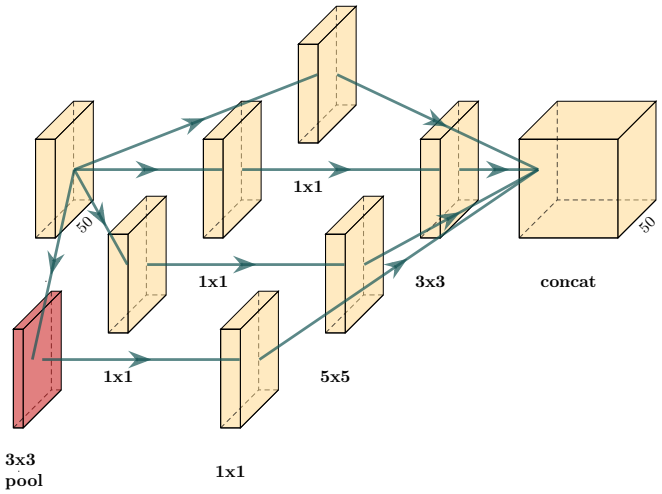


Figure 1.4: The inception part of the GoogLeNet architecture. The depth-wise 1x1 filters significantly reduce dimensions by combining feature maps. The 3x3 and 5x5 filters capture information with different spatial resolutions.

depth perception opened a path toward extensive commercial use such as medical image analysis, advanced driver assistance systems, and face recognition.
It is due to this exciting progress that we chose to apply artificial neural networks as the primary learning method in this thesis for robot localization.

1.4 Reinforcement Learning

Reinforcement learning is the process of optimizing the policy of an agent by trying out a different set of actions through exploration and map the agent's observations to its actions to maximize an expected reward (Sutton and Barto 1998). This optimization goal becomes very challenging when dealing with highly complex real-world problems such as robotic navigation where the robot's sensor (e.g., cameras or range finders) describes the current state of the robots and its wheel movements describe the possible set of actions. We categorize the challenges in robot RL as follow ((Kober and Peters 2014)):

Curse of dimensionality: The readings from the robot's sensors and its actuators' movement represents a high dimensional space. Without a compressed state representation and an adequate level of granularity in the action space, the robot

would need to explore endlessly to find meaningful relations between the state-action pairs (Bellman 1957). Function approximation (Sutton and Barto 1998) and macro-action (Barto and Mahadevan 2003), or options (Hart and Grupen 2011) are examples of remedies used to solve this problem in robot RL applications, such as state-action discretization in one degree of freedom (DoF) ball-in-a-cup (Nemec et al. 2010) and deep neural network function approximator in robotic manipulation (Levine et al. 2016).

Curse of real-world samples: Reinforcement learning requires a large of number of trials to converge. In the real world, this is time consuming and can also affect the robot's performance over time since the actuators and sensors of the robots are subject to wear and tear. One approach to reducing the cost of real-world interaction is using simulation to bootstrap the learning process. In theory, it should be possible for the agent to learn the behavior in simulation and transfer it to the real world. However, based on the state space representation, the simulation needs to represent the real world accurately. Otherwise, these modeling errors can accumulate, and the simulated robot can diverge from the real-world setting (Atkeson 1994).

Difficulty of goal and reward specification: The desired behavior of a reinforcement learning algorithm relies heavily on the reward function in addition to the state representation and possible set of actions. While it is possible to have a binary reward upon task achievement, a robot may rarely attain such a reward in a real-world scenario. Therefore, it is often necessary to include intermediate rewards in the reward function to speed up the learning process. This process is known as reward shaping (Laud 2004). Learning co-operative micromanagement in a real-time strategy game (Shantia et al. 2011), ball-in-a-cup through demonstration (Peters and Schaal 2008), and robot manipulation through reward shaping (Vecerik et al. 2017) are examples of this approach.

The application of reinforcement learning in real-world scenarios is not straightforward. Albeit its challenges, however, reinforcement learning offers powerful methods to solve complex problems with minimum supervision. In this thesis, we apply reinforcement learning to the robot navigation problem and demonstrate the benefit of transfer and simultaneous goal learning.

1.5 Scope of this Thesis

The fascinating challenges of robotic navigation and perception were the inspiration for writing this thesis. Therefore, we start by revisiting the required components for autonomous navigation and describing the challenges and the current state of the art solutions.

We would like to find out which part of an existing and established navigation system can be optimized by using on-the-shelf learning algorithms with minimum supervision effort. This leads to the following research question:
"Is it possible to improve an established navigation system using an available learning method?"
Therefore, in Chapter 2, we propose a method for dynamic parameter selection for a robot by analyzing the surrounding environment in a semi-supervised manner.

Our next step was to address perhaps the most crucial part of a navigation system, localization. The current models used for localization create a partially reliable spatial map of the environment, but the generalizability and their resistance to change is limited. Therefore, we investigated whether it is possible to use artificial neural networks to solve the localization problem. This leads to the research question:
"Can artificial neural networks be used to solve the localization problem in navigation?"
In Chapter 3, we present our research regarding the capability of stacked denoising autoencoders in retaining localization information in a 3D simulated environment.

Then, we turned our attention toward the scalability of our approach and included exploration and goal selection of a navigation system to examine whether it is possible to use a self-learning approach to navigate in an environment. This leads to the following research questions:
"Are convolutional neural networks better than stacked denoising autoencoders in solving the localization problem in larger environments, and are they scalable?"
"How can reinforcement learning be improved to let a robot learn to navigate to many goal positions?"
In Chapter 4 , we first compare the scalability and accuracy of the stacked denoising autoencoders with convolution neural networks in different 3D simulation environments. Then, we propose a reinforcement learning technique that can learn to navigate to multiple goals at the same time while using a goal selection technique based on temporal difference errors.

We finally summarize the findings of this thesis in Chapter 5 and discuss the possible future outlook.

Chapter 2

Dynamic Parameter Update using Unsupervised Situational Analysis

Abstract

A robot's local navigation is often done through forward simulation of robot velocities and measuring the possible trajectories against safety, distance to the final goal and the generated path of a global path planner. Then, the computed velocities vector for the winning trajectory is executed on the robot. This process is done continuously through the whole navigation process and requires an extensive amount of processing. This only allows for a very limited sampling space. In this chapter, we propose a novel approach to automatically detect the type of surrounding environment based on navigation complexity using unsupervised clustering, and limit the local controller's sampling space. The experimental results in 3D simulation and using a real mobile robot show that we can increase the navigation performance by at least thirty percent while reducing the number of failures due to collision or lack of sampling.

2.1 Introduction

The use of autonomous robots in our daily lives are on the rise. From autonomous drones delivering packages (Floreano and Wood 2015), mobile security robots (Everett and Gage 1999), autonomous vehicles (Levinson et al. 2011) to domestic robots that monitor elderly and help them in their activity of daily living (SILVER project 2016). The first and foremost responsibility of all these robotic systems is to navigate safely and efficiently in their environments. The navigation stack usually consists of a global localization module and path planner, a local base controller, and a set of sensor processing systems. The global localization is usually done through a mapping and localization process (Thrun et al. 2002). When a map is made, these robots estimate their position based on the erroneous movement of the base and precise sensory readings using probabilistic methods such as adaptive Monte Carlo localization (AMCL) (Fox et al. 1999). At this point, the only remaining task is to calculate a set of velocities to make sure that the robot reaches its goal safely. For this process, generally a two or three dimensional cost map is made. In the case of three dimensions, all sensor readings, such as rotating lasers, infrared devices, and sonars are added to a three dimensional pointcloud structure called voxel/octo map (Hornung et al. 2013). For a two dimensional cost map, all sensors information is projected down into a two dimensional array. In each control cycle, the system has to mark or clear these cells using ray tracing techniques (Glassner 1989). Finally, the robot searches for the best local trajectory, taking into account all the obstacles on the way and the robot's footprint and shape. This process is accomplished through either reactive collision avoidance methods such as the dynamic window approach (Fox et al. 1997), potential fields (Khatib 1986), and velocity obstacle (Alonso-Mora et al. 2018) or methods that rely on model predictive control (Maciejowski 2002) such as contouring in dynamic environments (Brito et al. 2019). This process is done multiple times a second to achieve a required control frequency for the robot. This requirement can be different depending on the type and application of the robot. All parts of the navigation stack have parameters to be tuned: Precision of the cost maps, number of the particles in AMCL, velocity sample rate, simulation time, scoring parameters, etc. It is important to note that these parameters have significant effects on the processing load. Therefore, one can conclude that it is hard to select one set of parameters for all navigation tasks that may differ in complexity. For example, in cluttered environments, tight corners, or doors, a higher resolution cost map and velocity sample rate will help the robot to navigate more safely and efficiently, while in larger hallways, the system can use faster speeds and a lower number of samples.

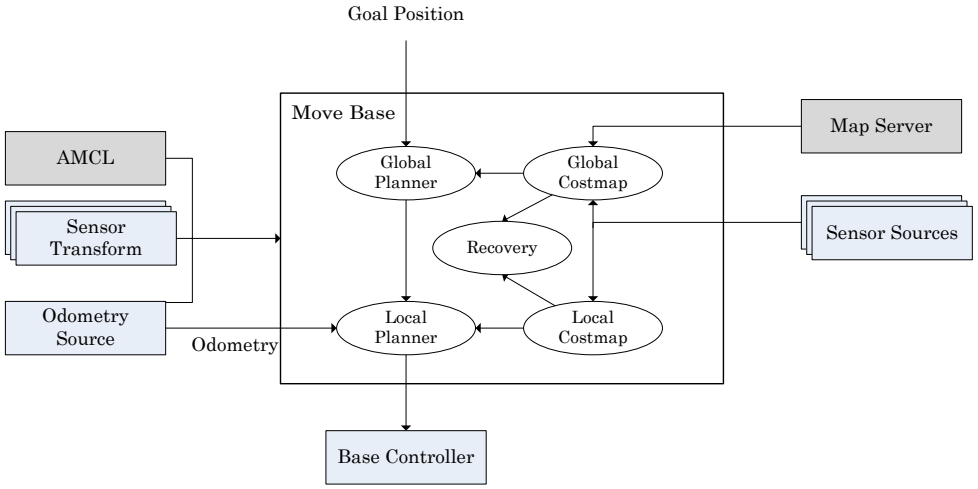


Figure 2.1: The structure of the navigation system.

Contributions: In this chapter, we propose a novel approach to automatically identify the complexity of scenes by extracting a customized histogram of oriented gradients (HoG) (Dalal and Triggs 2005) from a two dimensional projection of three and two dimensional sensory readings (cost maps) and clustering them using multiple unsupervised methods such as K-means and agglomerative clustering. In addition, we identify a tuned set of parameters for each of these clusters. Our experiments show that the clustering methods successfully separate the situations into meaningful and human-understandable clusters. Therefore, our contributions can be summarized as follows:

- Automatic navigational complexity classification
- Dynamic parameter update for the local navigation system and the cost maps
- Performance increase of the navigation system

Structure of the chapter: In Section 2.2 we describe the full navigation system in detail. In Section 2.3 we present the customized HoG feature extractor, the used clustering methods, and our approach to extract the best possible parameters for the navigation system. We describe the experiments and the results in Section 2.4. Finally, we conclude the chapter in Section 2.5 and discuss possible future work.

2.2 Preliminaries

Before continuing with the methodology section, we would like to depict the navigational structure that we use in detail. The used navigational stack is based on the work by David Lu (Lu 2014). This work was implemented as a package for the Robot Operating System (ROS) (Quigley et al. 2009). The general structure of the navigation stack is depicted in Figure 2.1. This stack requires certain inputs to be able to function correctly. The inputs outside the rectangle in Figure 2.1 require:

- A pre-generated 2D map of the environment.
- A localization method, in our case AMCL.
- Odometry information
- Base velocity control
- Sensor sources
- A complete transformation function to relate all the robot links in real-time.

Having a pre-generated map, we can use the AMCL method to localize and track the position of the robot. When a destination goal is sent to move base, the global planner will attempt to find a path towards the selected goal using either the $A^{\star}$ (N. J. Nilsson and B. Raphael 1968), or Dijkstra's shortest path (Misa and Frana 2010) algorithm. The calculated path is then sent to the local planner that will use the projected cost map to find the best set of velocities to approximately follow the global trajectory to reach the goal. In case of failures due to incorrect sensor readings or possible deadlocks, the system can initiate recovery behaviours. Our optimization is done using the dynamic window approach local planner (Fox et al. 1997).

Dynamic Window Approach

We selected the dynamic window approach (DWA) because it performs well on robots with good acceleration rates (Figure 2.2). However, recent model predictive control and reactive methods such as (Brito et al. 2019) and (Kapitanyuk et al. 2017) have shown better performance than the traditional DWA. Nevertheless, this chapter's objective is to limit the search space of any control algorithm through situational analysis, and therefore, can be applied to new algorithms.

The DWA needs the local cost map, a window section of the calculated global path, the projected footprint of the robot, robot base capabilities, and a set of simulation parameters. The robot base capabilities are the achievable acceleration and velocities in X, Y, and θ directions (Figure 2.3). In Table 2.1, the important simulation parameters are described. We omitted parameters with low importance and the ones that we do not optimize[1]. If all the above requirements are fulfilled, the robot navigates safely and reliably. However, different environmental circumstances require different parameters for optimal performance. Simulating unnecessary velocity trajectories in an empty hallway is a waste of resources, and lack of this sampling can be dangerous in crowded and narrow hallways. In Section 2.3, we explain how we solve this problem by applying an unsupervised situation detector.

2.3 Methodology

In this section we present our novel approach for autonomous situation analysis of the environment and a method to select suitable parameters for each situation.

2.3.1 Unsupervised Environmental Situation Analysis

The cost map is a representation of the surrounding environment of the robot. Therefore, we believe that we can find certain patterns in these costmaps and classify different situations on this basis. The first step is to collect data points. This is done through moving the robot through the environment several times. The next step is to come up with a feature set that can help us in the classification procedure.

Customized Histograms of Oriented Gradients

The Histogram of oriented gradients (HoG) (Dalal and Triggs 2005) is a suitable method to represent the structure in images. It has been used extensively in both two and three dimensional data. The three dimensional case is called the unique signatures of histograms (SHOT) (Tombari et al. 2010). The two dimensional case (HoG) has been applied to human detection (Dalal and Triggs 2005), indoor localization (Shantia, Timmers, Schomaker and Wiering 2015), object recognition (Tombari et al. 2010), and many other applications. Therefore, we believe that it is also a suitable method for our application. The local cost map is centered on the robot. However, this cost map is in the odometry coordinate system and is invariant to the robot rotation. We need the cost map in the robot's coordinate system. Therefore, we calculate a combined rotation and translation matrix M to transfer the cost

[1]The full list of parameters can be found in `http://wiki.ros.org/dwa_local_planner`.

Figure 2.2: The robot used for the experiments. The laser in the front of the robot is used for localization and obstacle avoidance. The two 3D sensors on both ends of the top bar are for 3D obstacle avoidance. The front sensor rotates based on the current speed of the robot to better detect obstacles.

map from the odometry coordinate system to the robot's coordinate system using equation 2.1.

$$M = \begin{bmatrix} \alpha & \beta & (1-\alpha) \cdot center.x - \beta \cdot center.y \\ -\beta & \alpha & \beta \cdot center.x + (1-\alpha) \cdot center.y \end{bmatrix} \tag{2.1}$$

where $center.y$ and $center.x$ are the center of the cost map and rotation axis of the robot, and

$$\alpha = \cos(\theta)$$
$$\beta = \sin(\theta)$$

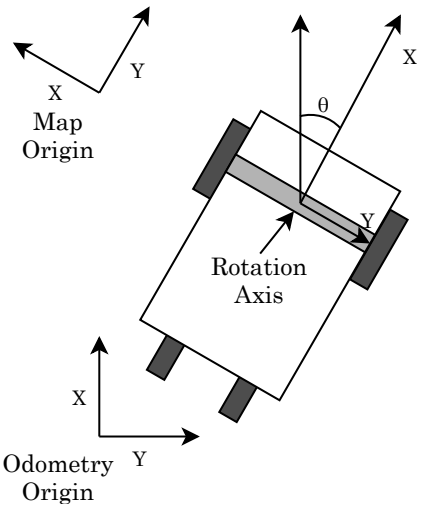


Figure 2.3: Various frames of reference in robotic Navigation. The robot's velocity in X, Y, and θ direction represents forward, sideward, and angular movement in the robot base frame.

where θ is the rotation angle of the robot in the odometry coordinate system (Figure 2.3).

Then, we multiply each image pixel (i, j) of the cost map with the M matrix. The result is the new cost map image with pixels calculated by equation 2.2.

$$\begin{bmatrix} \hat{i} \\ \hat{j} \end{bmatrix} = M \times \begin{bmatrix} i \\ j \\ 1 \end{bmatrix} \tag{2.2}$$

We lose a part of the image during these operations, but since this information lies on the corners of the cost map, the effect is minimal and can be neglected. Note that the center of the image is not necessarily the center of the robot. It is the main rotation axis of the robot. In addition, we also altered the windowing mechanism of the HoG feature set. We changed the window approach to emphasize this characteristic. Figure 2.4 shows the new window selection.

Category	Parameter	Description
Robot Configuration	Acceleration Limits	Rotational and translational acceleration in $\frac{m}{s^2}$.
	Velocity Limits	Rotational and translational speed in $\frac{m}{s}$.
Goal Tolerance	Yaw Tolerance	Yaw threshold for the goal in radians.
	X-Y Tolerance	X and Y distance threshold to the goal in meters.
	Latch X-Y	Only rotation will be checked after position is reached.
Forward Simulation	Time & Granularity	Simulation resolution (meters) and length of time (seconds).
	Sampling rates (X-Y-θ)	The number of velocity samples for simulation.
	Controller Frequency	Planner's desired loop for driving
Trajectory Scoring	Path Distance	Higher score if the simulated path is close to global path.
	Goal Distance	Higher score if the simulated path is closer to the local goal.
	Collision Distance	Lower score if the simulated path is close to obstacles.

Table 2.1: DWA parameter structure.

Clustering and Center Selection

We use K-means clustering (MacQueen 1967) to separate the data into multiple clusters. The problem is to select a good value for the number of clusters. There are various methods to tackle cluster validity. Theodoridis et al. (Theodoridis and Koutroumbas 2009) categorized these methods into three classes. The First is based on external criteria, where class labels are required to assess the methods' performance. The second approach is based on internal criteria, which focuses on the clustering algorithm itself to evaluate the results. The third approach is based on

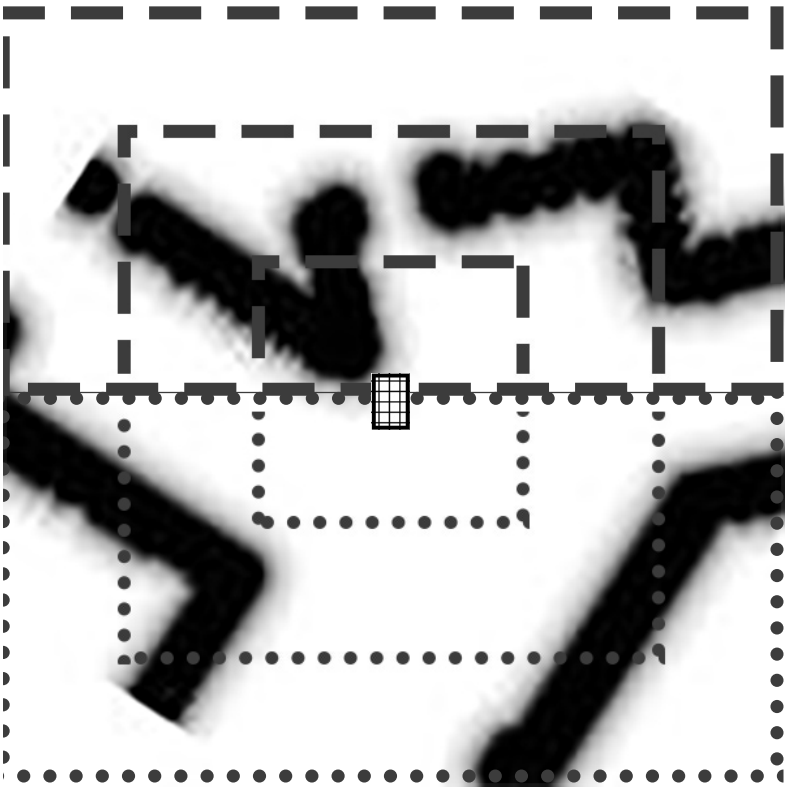


Figure 2.4: The window structure of the HoG. The underlying picture is a sample projected cost map. The middle rectangle is the robot footprint, and its rotational axis point is the middle of the image. The white pixels in the background are free space, the black pixels are obstacles, and the gray pixels are the inflation of obstacles where the robot may have difficulty navigating. The dotted and dashed lines emphasize the separation of HoG windows. Each rectangular section is used to calculate the HoG features. This way, the resulting descriptors are more suitable for our problem.

relative criteria, where the same algorithm is run with different parameters (e.g., the number of clusters) to find the valid number of clusters. We focus on the latter and perform agglomerative hierarchical clustering using the single linkage (Gower and Ross 1969) method to find the best number of clusters (Algorithm 2.1).

We set the cluster number c to one to extract the dendrogram based on the single linkage result, shown in Figure 2.5. Finally, we analyze the similarity values and select a cluster number where we detect a large similarity gap between layers.

1: **begin initialize** $c, \hat{c} \leftarrow n, C_i \leftarrow \mathbf{x}_i, i = 1, \ldots, n$
2: **repeat**
3: $\hat{c} \leftarrow \hat{c} - 1$
4: Find nearest clusters, $C_i \; and \; C_j$
5: Use $d_{min}(C_i, C_j) = \min_{x \in C_i, x' \in C_j} \|x - x'\|$
6: Merge $C_i, and \; C_j$
7: **until** $c = \hat{c}$
8: **end**

Algorithm 2.1: Agglomerative Hierarchical Clustering using Single Linkage, where $\hat{c}$ is the current number of clusters to obtain in each iteration, c is the desired number of clusters (with $c = 1$, we extract the dendrogram), and C_i and C_j are set of points belonging to a cluster where the distance measure is used to merge them.

However, since this step requires human observation, it is best to test different cluster numbers and compare the final experiment results. Due to the large required number of experiments, we only observed the members of each cluster for cluster numbers between 3 to 6. With five clusters, the results were most intuitive, which can be seen in Figure 2.6. It is also notable that the current dendrogram uses a binary approach to separate clusters. However, a study by Louis Vuurpijl et al. (Vuurpijl and Schomaker 1997) shows that the underlying data structure can hinder the effectiveness of a binary approach. Instead, they propose an N-ary approach by comparing the distance of the child clusters to the parent cluster considering the standard deviation between all the child clusters and their respective parent. Malika et al. (Malika et al. 2014) further investigated the cluster validity problem and created a tool to mix several methods to enhance the results. We left the analysis of these methods for future work.

2.3.2 Parameter Selection

When the clustering is finished, we can analyze the results by means of an average image for each cluster using equation 2.3:

$$A = \frac{1}{N} \sum_{x}^{N} I_x \times \frac{1}{1 + D_{I_x}} \tag{2.3}$$

where A is the average image for each cluster, N is the number data points in each cluster, I is a cost map image, x is the index of the data point, and D_{I_x} is the normalized distance of the x^{th} data point to the cluster center.

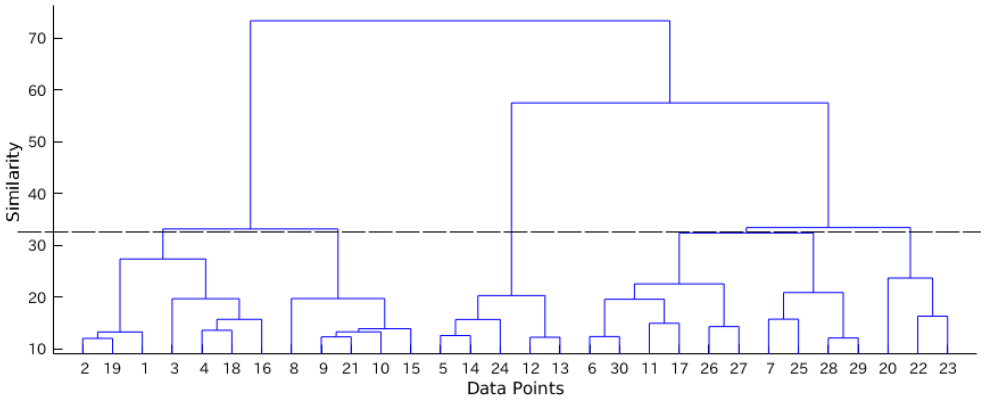
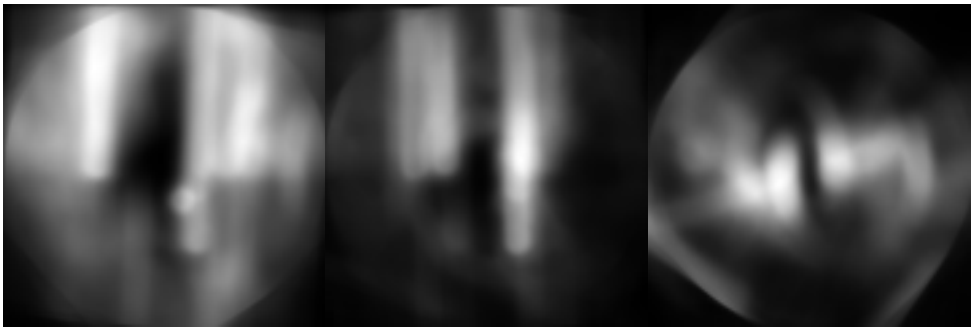


Figure 2.5: The dendogram constructed from the single linkage algorithm. There is a large similarity gap for all the joint clusters between $c = 6$ and $c = 5$, and also another gap between $c = 4$, and $c = 3$. By inspecting the average image of each cluster number, we found out that five clusters are the most intuitive number for the k-means clustering.

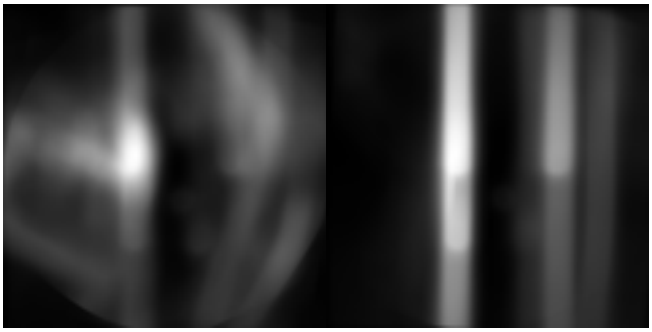
The procedure is to compare the distance between each recorded cost map image to the center of the cluster. Using a non-linear function, we give higher weights to the closer points, and lower weights to the points far from the center. The final result is an average image which describes the surrounding environment (Figure 2.6). Using these images, we arrange the clusters based on danger level. This parameter selection, however, is dependent on the type of the robot used. The requirement is to have one safe parameter setting with a low maximum speed and high simulation sampling, and a fast parameter set with a higher maximum speed and lower sampling rate of velocities. From these two parameters, we can extrapolate the rest of the parameters for clusters based on their danger level.

2.3.3 Parameter Update

Every time the cost map is updated, our algorithm analyses and determines the environmental situation. Before proceeding to update the optimal navigation parameters, we make sure that the analysis is correct and coherent. This is done by accepting only results which are coherent over three consecutive cost map updates. This also prevents continuous parameter updates due to outliers which leads to latency in the navigation control loop. For the actual update of the navigation parameters, the method sends the chosen parameters to the navigation stack by calling



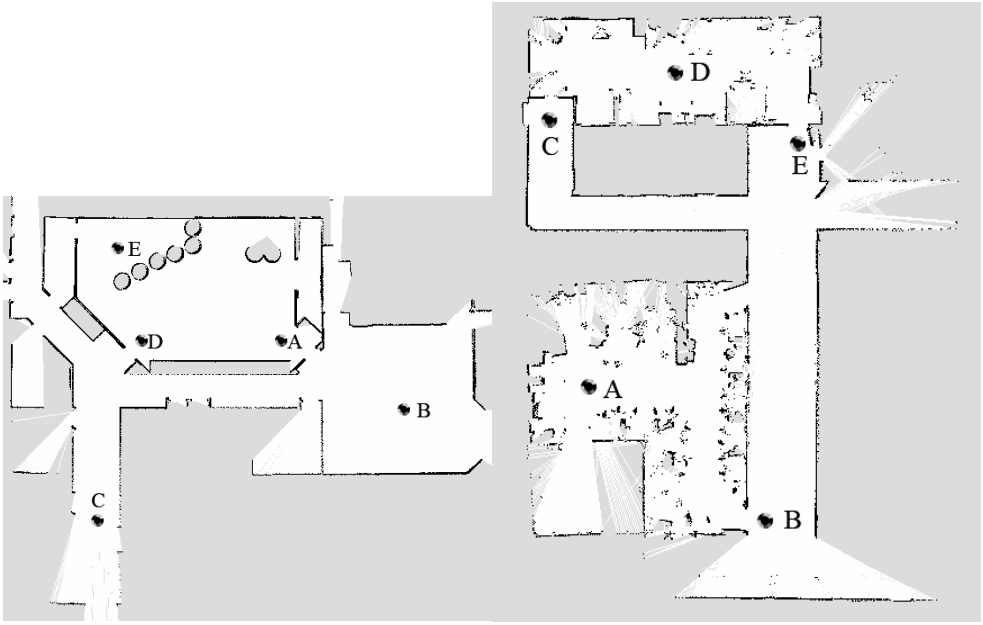
(a) Open area (b) Approach Corridor (c) Doors



(d) Corner on the left (e) Hallway

Figure 2.6: The weighted average image of all the clusters. Black pixels mean free space, white pixels means obstacles. The intensity level of white pixels show how close they are to the cluster center. (a) shows low congested areas and open spaces, (b) shows that the robot is approaching corridors, (c) shows very dense areas such as doors, (d) shows close proximity to corners, specially on the left side, and (e) shows hallways.

a specific ROS service. This service updates the parameters in the interval between the navigation control loops. This generates a latency that could lead to the control loop missing the desired control frequency, causing the robot to stop.



(a) Gazebo Environment (b) Real Environment

Figure 2.7: The map of the environment in Gazebo (a) and for the real experiments (b). The robot starts from checkpoint A, and continues through all the checkpoints. The robot collects the time data when it reaches checkpoint E. If a failure or collision occurs, a penalty of one thousand seconds is recorded, and the experiment is restarted.

2.4 Experiments

For the experiments, we use both a 3D simulator and a real robot. The 3D simulator used is an open source program called Gazebo (Koenig and Howard 2004) which simulates physics and allows us to have access to sensors, actuators, etc. This simulator is selected because of its wide spread use in robotics, such as the DARPA robotic challenges[2]. The robot is controlled through the robot operating system (ROS) framework (Quigley et al. 2009). The environment used for the simulation and real life can be seen in Figure 2.7. We use OpenCV, MATLAB and Numpy libraries for feature extraction and clustering (Bradski 2000), (MATLAB 2014), (Dubois et al. 1996).

[2] `http://www.darpa.mil/program/darpa-robotics-challenge`

The robot in Figure 2.2 is used for our experiments. The robot uses a differential drive base, and a frame which carries the manipulator, two RGB and depth sensors, and a laser range finder. We modeled this robot in Gazebo using the Unified Robot Description Format (URDF), which is an XML format for representing a robot model. Our goal is to find a suitable number of clusters and select adequate control parameters for each cluster respectively in simulation, and apply it to a simulated and real-world navigation experiment without any further changes.

2.4.1 Clustering Results

We gathered nine thousand cost map data points by driving the robot through the simulated environment (Figure 2.7a) and calculated the customized HoG features. A single linkage was done on this set, and we extracted a dendrogram based on the distance values (Figure 2.5) and visually analyzed the result for 5 and 6 clusters. We selected 5 clusters since the similarity distance is considerable between this level and the previous one, and the visual analysis showed that one more cluster does not add additional information. We used this number for our K-means clustering method, and Figure 2.6 shows the average images of these clusters. Hallways, doors, and the situation when approaching doors are evident in these images. When a cluster is detected, the navigation parameters will be changed on the fly.

2.4.2 Base Parameter Selection

In order to select the best base parameters, we first hand tuned the values to match the robot's differential drive base capabilities. Then, we sampled exploration values for each of the DWA parameters in Table 2.1. We calculated all possible combinations of these values, and compared the time it took to reach a point. The timeout to reach a point is 200 seconds, if the robot collides with an object or cannot reach the goal in time, it receives a penalty of 1000 and the trial is reset. For each combination, we simulate 100 rounds. The best selected base parameters are depicted in Table 2.2.

2.4.3 Simulation

In order to test the performance of our method, we made sure that the navigation environment has different varieties such as narrow hallways, open space, doors, etc. which can be seen in Figure 2.7a. The robot starts from point A, and continues through all checkpoints until it reaches point E. We measure the time between each pair, and average the results for comparison. We performed 500 trials for each of the best static settings (safe setting), fast parameter and our dynamic parameter setting. If the robot was not able to reach a point, we added it to the number of failures

Category	Parameter	Value
Robot Config	Acceleration X-θ	1.0 $\frac{m}{s^2}$, 1.0 $\frac{rad}{s^2}$
	Velocity X-θ	0.4 $\frac{m}{s}$, 1.0 $\frac{rad}{s}$
Forward Simulation	Simulation granularity, time	0.05 m, 3.5 s
	Sampling rate X-θ	15, 45
	Controller Frequency	5 Hz
Trajectory	Path Distance	0.2 m
	Goal Distance	0.3 m
	Collision Distance	0.05 m

Table 2.2: Best calculated parameters for the DWA method.

Param Type	Mean (w/o Penalty)	Standard Dev.	Failures
Dynamic	**56.32 (37.37) s**	19.22 s	10
Best Static (Safe)	85.08 (60.76) s	30.36 s	13
Fast	89.29 (61.44) s	28 s	15

Table 2.3: The simulation results. On average the dynamic parameter sets perform 33.8% better than the best static parameter, and a high speed set. In addition, the number of failures is also lower than that of the rest.

and marked that trial length as one thousand seconds. We then calculated the average time without penalty, and calculated the standard deviation not considering the failures. Table 2.3 shows the time average, standard deviation and the number of failures due to collision or lack of sampling in the simulated environment. It is evident that our dynamic situation analysis performs superior to a single parameter set. The reasoning is straightforward, the clustering method correctly detects the majority of situations, and sets the best parameters. For example, close to the doors, the system uses the safest approach. With a limited velocity space, the system can simulate enough velocity points to find the best path in a congested area. On the other hand, in an open space, the system can use a lower number of sample points and reach higher speeds without reliability and safety issues.

	A to B		
Parameter Type	Mean (w/o Penalty)	Standard Dev.	Failures
Dynamic	137.71 (26.5) s	16.9 s	2
Best (Safe)	**120.59 (22.9) s**	5.3 s	2
Fast	372.96 (35.44) s	26.22 s	8

	B to C		
Parameter Type	Mean (w/o Penalty)	Standard Dev.	Failures
Dynamic	**90.51 (42.51) s**	5.14 s	1
Best (Safe)	111.52 (64.27) s	7.0 s	1
Fast	144.58 (43.9) s	7.31 s	2

	C to E			Full Path
Parameter Type	Mean (w/o Penalty)	Standard Dev.	Failures	Mean (w/o Penalty)
Dynamic	**94.64 (47.0)** s	23.32 s	1	**326.06 (116.01) s**
Best (Safe)	196.73 (55.0) s	11.57 s	3	893.151 (118.82) s
Fast	375.63 (39.4) s	7.18 s	8	428.38 (142.13) s

Table 2.4: The real experiment results. On average the dynamic parameter sets perform 18% better than the best static set of parameters, and a high speed set. In addition, the number of failures is significantly lower than that of the rest.

2.4.4 Real Experiments

We used a similar approach to that of Section 2.4.3. The selected path included narrow and large hallways, congested areas, and doors which can be seen in Figure 2.7b. We performed 20 trials for each of the base, fast, and our dynamic parameters for A-B, B-C, and C-D-E points. Table 2.4 shows the average time, standard deviation, and the number of failures for each method and their respective pair of points.

There were some complications during the real experiments. Unlike the simulator, after online parameter changes of the navigation system, the sensor buffers were frozen for approximately one second which resulted in a full stop of the robot. Depending on the complexity of the scene, the number of parameter change operation varied. However, our dynamic method is still superior (18% faster than the safest method) in comparison to the fast and safest set of parameters with the total time of $116.01s$ without considering collision penalty. We estimate that this number will be closer to that of our simulation results if this sensor buffer failure did not occur. If we take a look at point to point results, we see that the best static method is performing better than our dynamic approach between point A and B. The main reason here was indeed the sensor buffer freezing problem because of high number of parameter changes. In the rest of the path, however, the dynamic parameter system performs better. It is notable that the fast parameter set is very unreliable in crowded areas with 18 total collisions, some of which actually damaged one of our sensors. We can conclude that using our approach, we can achieve a safer, faster, and more reliable navigation.

2.5 Conclusion

In this chapter, we introduced a novel approach to automatically analyze a robot's surrounding environmental situation using HoG features and unsupervised clustering. The feature extraction and clustering were done on an obstacle matrix (cost map) which is a two dimensional projection of realtime sensory readings. We first extracted the best safe and fast parameter sets without using our dynamic approach. Then, using our method, we determined the congestion and danger level of these clusters by calculating a scaled average image for each cluster. Finally, we tuned multiple parameters for the local planning module of the robot in order to navigate with a higher performance and reliability. Using the clusters and danger level scores, we extrapolated new parameters and performed simulation experiments. In simulation, we had a performance increase of 33%, and a reliability increase of 28% in terms of navigation failure. We then performed the experiments on a real mobile robot with the same cluster centers that were learned during the simulation experiments. The performance and reliability increase were 18%, and 30% respectively. We can conclude that by understanding the surrounding environment, we can dynamically change navigational parameters to allow for a faster and more reliable movement.

Future Work: There are several improvements that can be done to further enhance the reliability and performance of the system. Currently, the number of clusters and the danger levels are calculated by manually observing the dendrogram and average images of the clusters. This procedure can be automated by calculating similarity values between cluster levels (Figure 2.5), and assigning scores to different sections of our windowing approach (Figure 2.4). In addition, in this research, we only optimize the parameters for the dynamic window approach. However, we can apply this method to other types of dynamic controllers such as the guiding vector field (Kapitanyuk et al. 2017) and model predictive contouring control (Brito et al. 2019) in cluttered environments to assess how well the dynamic parameter tuning can increase the performance. Finally, we can use reinforcement learning or other optimization algorithms applied to robot navigation and manipulation path planning, such as the proposed automatic parametrization of path planning methods by Burger et al. (Burger et al. 2017) to tune the parameters for each of the detected environments.

Chapter 3

Localization using Stacked Denoising Auto Encoders

Abstract

Robotic mapping and localization methods are mostly dominated by using a combination of spatial alignment of sensory inputs, loop closure detection, and a global fine-tuning step. This requires either expensive depth sensing systems, or fast computational hardware at run-time to produce a 2D or 3D map of the environment. In a similar context, deep neural networks are used extensively in scene recognition applications, but are not yet applied to localization and mapping problems. In this chapter, we adopt a novel approach by using denoising autoencoders and image information for tackling robot localization problems. We use semi-supervised learning with location values that are provided by traditional mapping methods. After training, our method requires much less run-time computations, and therefore can perform real-time localization on normal processing units. We compare the effects of different feature vectors such as plain images, the scale invariant feature transform and histograms of oriented gradients on the localization precision. The best system can localize with an average positional error of ten centimeters and an angular error of four degrees in 3D simulation.

3.1 Introduction

The commercial availability of robots is currently limited to small service robots such as vacuum cleaners, lawn mowers, and experimental robots for developers. Recently, however, there is a clear rise of consumer interest in home robotics. One example is the development of service robots that operate in home and office environments. These robots must be priced congruent with their abilities. In this chapter, we join the current trend of focusing more on smart software solutions combined with low-cost hardware requirements to facilitate the process of creating cheaper robots with reliable functionalities. With this goal in mind, we focus on a very basic functionality of a sophisticated robot, localization. Currently, the most well-known and commonly used approaches for solving the localization problem are through a precise process of mapping the environment using 2D or 3D sensory data and their required algorithms. These methods generally consist of three major parts, spatial alignment of data frames, loop closure detection, and a global fine-tuning step (Thrun et al. 2002, Grisetti et al. 2007). The hardware requirements for these methods, however, are quite expensive.
For example, the cheapest 2D laser range finder roughly costs several hundreds of Euros, and the price exponentially increases when more precision and robustness are added to the system. There has been valuable research done on this topic in the recent years, especially by using the fairly inexpensive Primesense sensor [1] found on Microsoft Kinect devices. Henry et al. (Henry et al. 2012), introduced a 3D indoor mapping method using a combination of RGB feature-based methods and depth images to connect the frames and close the loops. The results of this work are a very good approximation of challenging environments visualized in a 3D map. The calculations however, took approximately half a second per frame to compute using their more precise combined RGB-D and iterative closest point (ICP) algorithm, and three hundred milliseconds using their two-stage RGB-D ICP optimization. Whelan et al. (Whelan et al. 2013), on the other hand, promised a real time delivery of mapping and localization by the use of high performance GPUs and expensive hardware.

Another issue that has not yet been addressed by researchers is the effect of luminance from outside sources to the process of mapping. In (Khoshelham and Elberink 2012), the authors mention that the lighting condition influences the correlation and measurement of disparities. The laser speckles appear in low contrast in the infrared image in the presence of strong light. Sun light through windows, behind a cloud, or its reflection through the walls and the floor can introduce non-existing pixel

[1] Primesense Ltd. Patent No.US7433024 (Garcia and Zalevsky 2008)

patches or gaps to the depth image. Therefore, additional research is required to measure these effects, and perhaps to introduce new methods that require less computational power at run-time which are robust to external sources and changes to the environment. In a similar context, there has been significant research on image classification and scene recognition using deep neural networks in the recent years (Hinton 2006, Vincent et al. 2010, Krizhevsky et al. 2012, Schmidhuber 2015). This has allowed the community to achieve high classification performance in character, scene and object recognition challenges (Krizhevsky 2009, Deng et al. 2009, Quattoni and Torralba 2009).

In this chapter, we adopt a novel approach to the localization problem by using denoising autoencoders (DA)(Vincent et al. 2008) with HSV information. We use a semi-supervised approach to learn the location values provided by traditional mapping and localization methods such as Adaptive Monte Carlo localization (AMCL), and Grid Mapping (Fox et al. 1999, Grisetti et al. 2007). This approach has a significant run-time advantage over the traditional ones due to low computational requirement of a feed-forward neural network. It is only the initial training of the network that requires a significant amount of computation which can be done offline and delegated to other sources such as cloud computing centers (e.g. Amazon Web Services). Another benefit is the general ability of neural networks to cope with sensor noise, and changes in the environment. We combine multiple feature vectors with the DA and compare them to other scene recognition methods such as histograms of oriented gradients (HoG) (Dalal and Triggs 2005). In a commercial scenario, the manufacturer can temporarily install a depth sensor during the product delivery/installation and perform a traditional mapping to record the approximate ground truth for each captured image. The robot will then start the training of the network, and the depth sensor can be removed. Finally, the robot can continue localizing with acceptable error rate.

In short, our method:

- is a novel approach for localization using denoising autoencoders with semi-supervised learning
- has low run-time computation requirements
- has inexpensive hardware requirements

We also compare the effect of different feature vectors on localization precision and conclude that the positional and angular errors can compete with those of traditional methods.

In Section 3.2, we explain the features, the networks, and the training methods in detail. We discuss our experimental setup and the achieved results in Section 3.3, and finally conclude the chapter in Section 3.4 and discuss future work in Section 3.5.

3.2 Methodology

In this section we first discuss the different feature vectors used for training with denoising autoencoders (DA), and the reasons for selecting them for localization purposes. Next, we continue with an explanation of the pre-training of the DA network, and a second layer neural network that is used for learning the metric location values. A block diagram of the complete pipeline is depicted in Figure 3.1.

3.2.1 Feature Sets

Indoor localization requirements are slightly different compared to those for normal scene recognition where the goal is to distinguish totally different types of scenes from each other. For example, in the SUN (Xiao et al. 2010) and indoor CVPR (Quattoni and Torralba 2009) databases, there are different types of scenes such as offices, trees, zoos, etc. In each of these classes the details in the scenery are different, but they share a set of similar objects or landscapes (Figure 3.2). For robot navigation in indoor environments, however, the robot needs to distinguish similar scenes with different scale and rotation from each other. In some cases (Figure 3.3), moving forward for a couple of meters does not change the scenery, or there are views that look the same, but are in fact, taken from different locations. In order to select a good feature set, we use some established feature vectors and carry out the experiments using them.

Sub-sampled Plain Image

The original raw image contains a lot of information about the scene. Due to large size of images, applying machine learning techniques become difficult on the basis of the curse of dimensionality theory (Bellman 1961). However, it is possible to extract a sub-sample of the original image, which results in a much smaller feature vector. The downside of this method is that we may lose fine grained cues that are present in the original image. Krizhevsky et al. in (Krizhevsky et al. 2012), and (Krizhevsky and Hinton 2011), demonstrated the learning ability of deep neural networks using plain images. In (Krizhevsky et al. 2012), the authors used a combination of convolutional neural networks (CNN) (LeCun and Bengio 1995) and

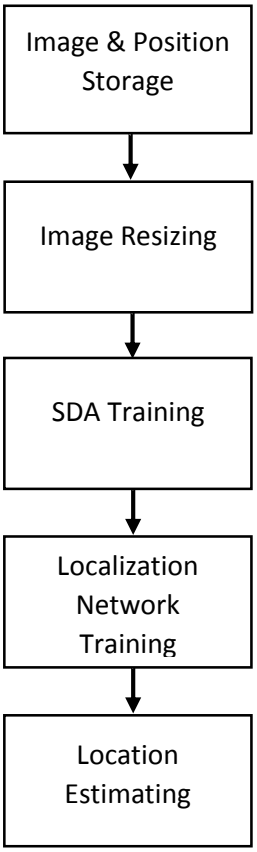


Figure 3.1: The block diagram of required steps for training and testing the proposed approach.

traditional neural networks to solve a one thousand class problem with more than a million training images (Deng et al. 2009). In (Krizhevsky and Hinton 2011), the authors used a very deep autoencoder with plain images to retrieve a set of images from the CIFAR (Krizhevsky 2009) database. Since their experiments showed promising results using only plain images, we also use a sub-sampled plain image in our feature set.

On this basis, we use a flattened gray-scale and HSV image with a fixed size of 28×28. We perform the re-sampling using pixel area relation, which is a preferred method for image decimation. This results in 784 input dimensions for the gray scale

Figure 3.2: Samples from the ICVPR dataset. The office environments share the same characteristics, but are from different environments.

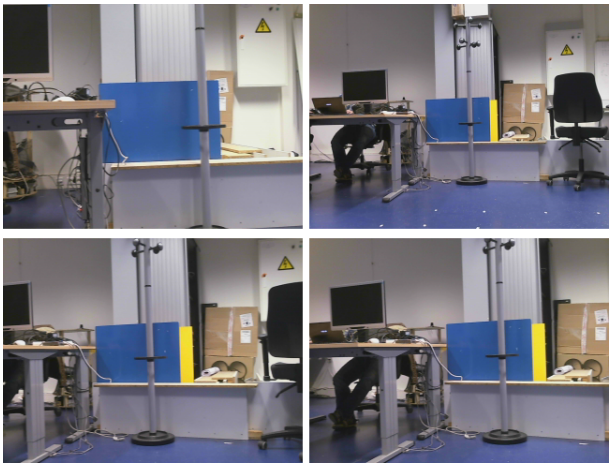


Figure 3.3: Sample images from our robotics laboratory. The pictures taken show the same scenery, but the robot was positioned in different locations.

image and 2352 input dimensions for the HSV image. The HSV coloring system is preferred to RGB because of its resistance to lighting changes. We will use the sub-sampled plain image feature as a baseline for our comparisons.

Sub-sampled image with top SIFT keypoints

On the one hand, sampling down the camera image has the advantage of retaining the scene structure, and also reducing noise. It also allows the system to cope better with color and light variations. On the other hand, it has a disadvantage of losing fine, high resolution sections of the image which can be essential in detailed scene recognition and localization. For example, a key switch in the kitchen, or a prominent door handle can give hints to the robot about its actual location. In order to scale down this effect, we decided to use a combination of sub-sampled plain images, and prominent SIFT features (Lowe 2004) extracted from each image. For each image, we calculate the SIFT features for the original high-resolution image. The top four keypoints are selected from each image based on their SIFT response value, and their descriptors are added to the plain image feature set. This approach may help the system to retain prominent edge structure while learning from general information of the sub-sampled image.

Histograms of Oriented Gradients

We decided to use yet another method while bypassing the autoencoder to compare the possible computation performance gain in comparison with the other two proposed methods. We used the idea of histogram of oriented gradients which was previously applied to human detection (Dalal and Triggs 2005), and for indoor localization (Kosecka et al. 2003). We, however, decided to calculate the HoG of additional cells with varying sizes to capture both general and detailed edge information of the scene. The image is divided into separate cells of 8×8, 4×4, and 2×2. The gradients for all of the cells and the image itself are then calculated. This results in 680 values for the feature vector.

3.2.2 Denoising Autoencoder Training

We decided to perform indoor scenery recognition using the denoising autoencoder (Vincent et al. 2008) because of the simplicity of the training procedure compared to CNNs. Stacked formation of DAs and its strong learning ability and low error rate compared to support vector machines, the restricted Boltzmann machine (RBM), and traditional autoencoders (Vincent et al. 2010), (Vincent et al. 2008), (Tan and Eswaran 2008), are other reason for selecting this approach.

Pre-training

Pre-training is the part that distinguishes a denoising autoencoder from a traditional autoencoder. The input of the network is corrupted by either random masking, salt and pepper, Gaussian noise, etc. (eq. 3.1).

$$\Omega \rightarrow \tilde{\Omega} \sim q_D(\tilde{\Omega}|\Omega) \tag{3.1}$$

Then, the corrupted input is fed through a matrix product with the first layer of weights, and the bias vector is added to the hidden layer neuron activations. Finally the results go through a sigmoid function (eq. 3.2).

$$O = f_\theta(\tilde{\Omega}) = \varsigma(\mathbf{W}\tilde{\Omega} + b) \tag{3.2}$$

In the next step, the network attempts to reconstruct the original input vector (eq. 3.3).

$$\bar{\Omega} = g_{\theta'}(O) = \varsigma(\mathbf{W}'O + b') \tag{3.3}$$

The cost function used is the reconstruction cross-entropy depicted in equation 3.4:

$$L(\Omega, \bar{\Omega}) = -\sum_{k=1}^{d}[\Omega_k \log \bar{\Omega}_k + (1 - \Omega_k) \log(1 - \bar{\Omega}_k)] \tag{3.4}$$

In which Ω is the original input which is normalized between 0 and 1, $\bar{\Omega}$ is the reconstructed output from the corrupted input, and k denotes the index of the input vector. By using this equation, the error and the updates of one stochastic gradient descent can be calculated. After sufficient epochs, the training of the first layer is stopped. Figure 3.4a shows this process.

The training of a stacked denoising autoencoder (SDA) is similar to that of a DA, but with a small difference in the higher layers. Consider that the network is trained up to layer $n-1$, and we want to train the final layer n. For input Ω, output f_θ of the layer $n-1$ is extracted using traditional feed-forward neural network algorithms. Then, this output vector is corrupted using $q_D(\tilde{\Omega}|\Omega)$, and after that the output of the n^{th} layer is calculated. An attempt is made to reconstruct the original n^{th} layer output using the corrupted n^{th} layer input (Figure 3.4b). The type, and amount of the corruption has a large effect on the result of pre-training. Parameters used in our experiments will be discussed further in section 3.3.

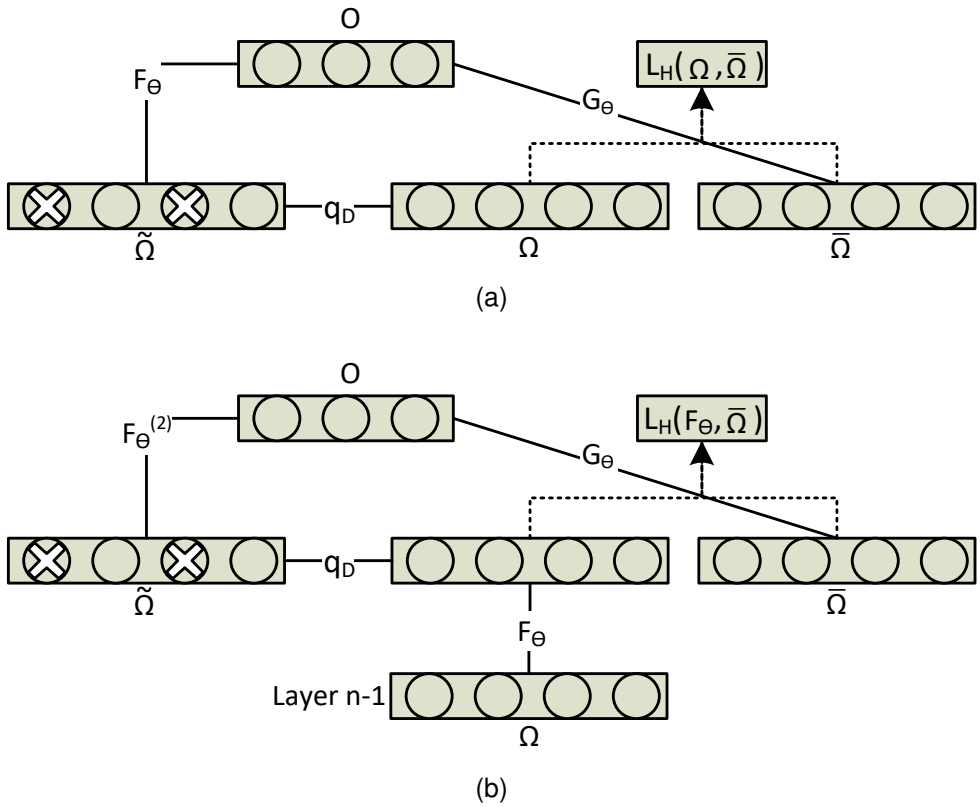


Figure 3.4: (a) The input Ω is corrupted using the q_D function. Then the result is fed forward to the next layer using the f_θ of equation 3.2. Finally, a reconstruction attempt to the original input is made using the g_θ of equation 3.3. For the stacked version (b), the non-corrupted input is fed forward up to level n of the network before applying the corruption.

Semi-supervised Location Estimation using Ground Truth

In order to associate the ground truth location values to the encoded result of the neural network, we connect a secondary two layer feed forward neural network to the learned encoding of the input. In our experiments, we record the ground truth data using the perfect robot odometry in 3D simulation.

The relation between the scenes and the odometry readings is a non-linear function, and we can learn this by using a feed forward neural network with non-linear activation functions. In order to achieve this goal, the following procedure is followed. After the pre-training phase of the SDA is completed, we add a separate

neural network with one hidden layer on top of the denoising autoencoder. The hidden layer activation function is a sigmoid, and the output layer has a linear activation. The outputs of the neural network consist of normalized metric X and Y values, and the $\sin(\theta)$ and $\cos(\theta)$ with θ being the robot angle in radians. Reasons for selecting sine and cosine of the radian are that first the values are bounded and between $[-1, 1]$, and can be easily normalized to $[0, 1]$, and we want to give a hint to the neural network that the angular input is not continuous, but has a periodic property. The network can learn this by observing the 3rd and 4th output. The normalization factors of X and Y come from the ground truth map available through the simulation.

Fine-tuning layers in classification problems such as in (Vincent et al. 2008, Krizhevsky et al. 2012, Krizhevsky and Hinton 2011), have a one layer network where their outputs were either the class label or the softmax probability of each class. Our problem, however, is of the regression type, and a hidden layer is required to non-linearly map the encoded outputs to the position and angular values. In addition, in those papers, the class labels are used to further reduce the reconstruction error of the network. In our case, however, we merely aim at associating location values to the encoded outputs. Therefore, we do not propagate the error through the whole network. Consequently, using the mean squared error as our error function, the back propagation rule is used to train the weights of the network.

3.3 Experiments

We carried out our experiments using the Gazebo (v3.0) 3D physics simulator (Koenig and Howard 2004), and used the ROS framework to develop our software (Quigley et al. 2009). We used Gazebo with the Bullet 3D physics engine which allows for the use of GPUs to facilitate a real-time simulation and high frame rates for all the robot sensory inputs using an ordinary PC. Environments used for the experiments can be seen in Figure 3.5[2]. We used FANN, OpenCV, and Theano libraries to perform the SDA, and location estimation neural network training phases, (Nissen 2003), (Bergstra et al. 2010), (Bradski 2000). Theano uses multiple cores and a GPU to perform the SDA training steps.

The simulated robot, which can be seen in Figure 3.6 is used for our experiments. The robot consists of a moving base with differential drive, and a top frame which carries its essential depth and RGB sensors, a laser range finder, manipulator and an interface for human robot interaction. We modeled our laboratory robot in Gazebo

[2]https://3dwarehouse.sketchup.com.

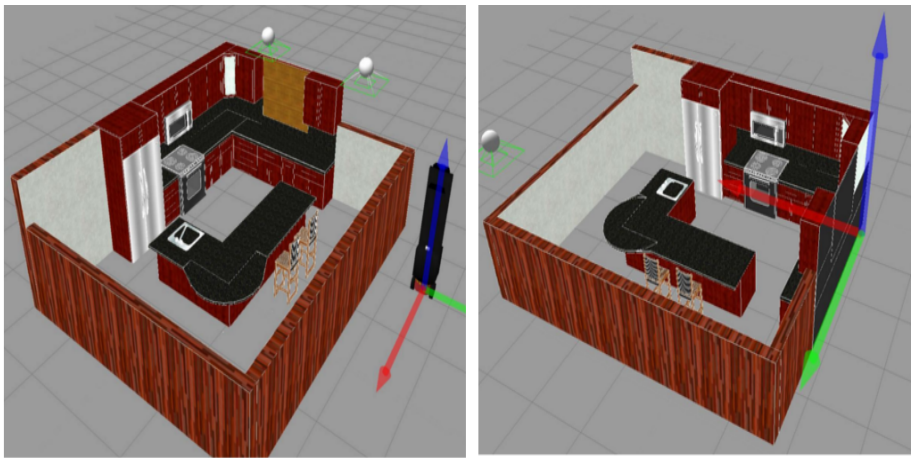


Figure 3.5: The 3D model of the environment. The model is a modified version of the Cherry Kitchen model in Google's 3D warehouse.

using Unified Robot Description Format (URDF), which is an XML format for representing a robot model. The model shapes and meshes used in simulation were partly generated by the vendors and partly designed by ourselves.

All the neural network training simulations are repeated 10 times with different initializations of the network. The baseline, which is the sub-sample plain image, is not given to the SDA for high level feature extraction. The HoG features are also not processed by the SDA because these features are already a representation/compression of the full image. After the autoencoder training is finished, training of the location estimation neural network starts. We only update the weights of the location estimation neural networks without propagating it back to the SDA network. We explored multiple network and training configurations in the 3D simulation and selected the best ones based on localization performance and computational requirements.

We found out that the number of required neurons in each layer can be one third of the number of inputs. The results would increase slightly if the number of neurons increases. However, the speed of calculation decreases during training and prediction phases which is not suitable for a real-time robot. Therefore, we decided to limit the number of neurons as much as possible. The denoising autoencoder network with gray scale input images uses 256 hidden neurons, and the networks with HSV and HSV+SIFT input images use 750 hidden neurons per layer. For subsampled plain image with top SIFT keypoints, we kept the number of neurons in each layer to make sure the comparison is valid. The HoG feature vector, however,

Figure 3.6: 3D model of the robot used for the experiments.

does not require the autoencoder so it is left out of this part of training. The learning rate for training the SDA was set to 0.001.

The neural network used for estimation of the location of the robot is a single hidden layer neural network with sigmoid functions in the hidden layer and linear activation functions in the output layer. There are hundred neurons in the hidden layer and four outputs which represent $X, Y, \sin(\theta), \cos(\theta)$ respectively. Finally, for the sub-sampled plain image vectors, the encoding of the images are processed and then fed to the localizing neural network. The HoG feature vector is directly fed to the localizing neural network. The training of the localizing neural network is done by traditional back propagation with the learning rate of 0.001 and $10,000$ epochs with no stopping criteria. However, we record the network every hundred epochs and only use the network with highest validation performance for the final experiment with the test dataset. This will allow us to avoid over-fitting of the neural network during the training process.

3.3.1 3D Simulation

The simulation environment is a kitchen with a dimension of 5×7 meters. It contains both detailed and coarse objects. The lighting is provided by a unified directional beam (simulated sun), and two additional light bulbs. Only walls and object shadows are visible, because the current texture properties of the environment do not include light reflection properties.

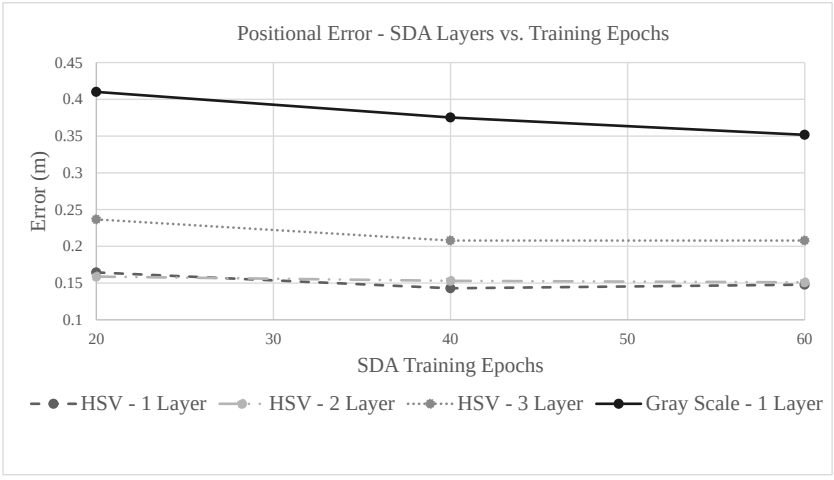
The data gathering is done through automatic and manual processes to collect training, validation, and test data. In the automatic method, the robot takes random discrete actions. Before performing an action, the robot saves the current ground truth location by using odometry in simulation. It then starts to take pictures while rotating from minus to plus five degrees around that point. This is to make sure that the robot has similar training data for the approximate same location in the environment. The robot is equipped with an obstacle avoidance system, and will change trajectory if it is close to an obstacle. In order to make sure that all locations are traversed, we let the robot operate for several hours and take a data set with 150 thousand training pictures. These pictures are used to train the denoising autoencoder.

For training the location estimation layer, we gather a more structured data set of 10 thousand training, and 2 thousand validation and testing pictures. A human operator controls the robot, and takes steps of $0.5 - 1.5$ meter. After each step he/she will rotate the robot in a full circle to capture the surroundings. This operation is repeated until the environment is covered. Since the lighting effects are incomplete in the 3D simulation, we avoid taking pictures of only walls. One side of the kitchen has continuous walls with the same texture, and no lighting changes. We make sure that another part of the kitchen is also in the view. Otherwise, it would be impossible for the robot to correctly estimate its position. For the validation and testing set, however, the operator will move the robot in a path for which the overlap with the previous run is minimal.

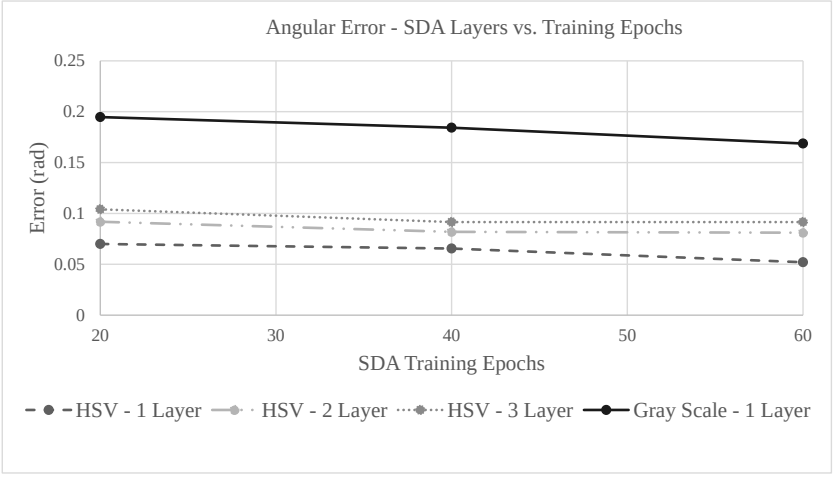
3.3.2 Results

We first discuss the results of different network architectures and training configurations. Next, we elaborate the final performance of these networks using the features mentioned in section 3.2.1. We tested a 1 layer DA with gray scale images, and different SDA architectures (1-3 layers) with HSV images using multiple training epochs in 3D simulation. The binomial corruption levels were started at 0.1. For each additional layer, the corruption level was also increased by 0.1. We selected the corruption levels similar to other applications of SDA for scene and character recognition in (Vincent et al. 2008). Figure 3.7a, and 3.7b show that the HSV results

are significantly superior in comparison to the gray-scale. The reason is that the HSV images contain more information about the scene.



(a)



(b)

Figure 3.7: The positional and angular localization error for HSV feature vectors with 1-3 denoising autoencoder layers. We also included the gray scale image with a 1 layer denoising autoencoder. The one layer network with HSV images is the best considering computational requirements and the localization error.

The three layer HSV network performs worse in comparison to the two and one layer networks. It is possible that the type and amount of corruption is not suited for this number of layers in the SDA and our application. The two layer network performs better using a smaller number of epochs, but the one layer DA network catches up when more training is done on the denoising network. However, computationally the one layer network is superior because it takes much less time to train the network. Since the positional and angular errors of two and one layer network are similar, we decided to select the one layer networks for the rest of the experiments. In addition to the localization results, a reconstruction attempt of the scenes is depicted in Figure 3.8.

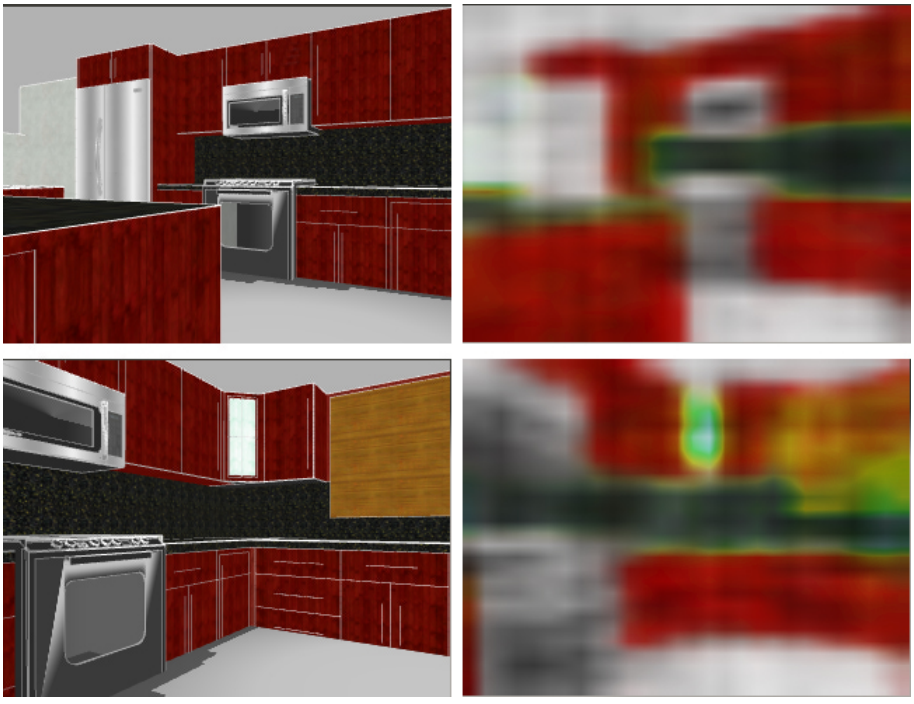


Figure 3.8: The pictures on the left are original images from the camera, and the right side pictures are the reconstructions from a one layer denoising autoencoder. The number of pictures used for training this network is 150 thousand.

We compared the final location performance of all network configurations with their respected features. The sub-sampled plain HSV images which were not given to the DA networks are selected as our baseline. The HoG features were also directly connected to the location estimating feed forward MLP. The HSV image features

Error	Baseline	HoG
X(m)	$1.5 \times 10^{-1} \pm 2.0 \times 10^{-1}$	$1.0 \pm 8.0 \times 10^{-1}$
Y(m)	$1.5 \times 10^{-1} \pm 1.2 \times 10^{-1}$	$4.0 \times 10^{-1} \pm 2.0 \times 10^{-1}$
$\cos(\theta)$	$9 \times 10^{-2} \pm 1.3 \times 10^{-1}$	$5.6 \times 10^{-1} \pm 3.0 \times 10^{-1}$
$\sin(\theta)$	$9 \times 10^{-2} \pm 1.1 \times 10^{-1}$	$7.4 \times 10^{-1} \pm 3.0 \times 10^{-1}$

Error	DA-HSV	DA-HSV+SIFT
X(m)	$9 \times 10^{-2} \pm 1.0 \times 10^{-1}$	$8 \times 10^{-2} \pm 1.0 \times 10^{-1}$
Y(m)	$6 \times 10^{-2} \pm 6 \times 10^{-2}$	$8 \times 10^{-2} \pm 1.0 \times 10^{-1}$
$\cos(\theta)$	$6 \times 10^{-2} \pm 7 \times 10^{-2}$	$8 \times 10^{-2} \pm 1.0 \times 10^{-1}$
$\sin(\theta)$	$6 \times 10^{-2} \pm 7 \times 10^{-2}$	$8 \times 10^{-2} \pm 1.0 \times 10^{-1}$

Table 3.1: The metric and angular errors (with standard deviations) of all the feature vectors in 3D simulation. The number of training examples for the autoencoder is 150 thousand images, and for the location estimation layer we used the training set of 10 thousand images, and 2 thousand for validation and test.

that were given to the DA networks are named DA-HSV. The HSV image features including the top four prominent SIFT features are named DS-HSV+SIFT. Table 3.1 shows the results of all the methods against each other in 3D simulation. As can be seen, the lowest performance is for the HoG feature. It seems that the compressed information about the gradients in the scene is not unique enough to approximate metric position of the robot. The baseline clearly outperforms the HoG feature, but it fails to reach the performance of features with the DA network with a large margin. The sub-sampled HSV image average error on X and Y is less than 10cm, with negligible error on θ which corresponds to approximately 4 degrees. Surprisingly, the results of the sub-sampled HSV image plus the prominent SIFT keypoints performs worse than the normal HSV. This is because the keypoints are resilient against rotation and scale, and therefore they can be present in multiple views of the same scene. Perhaps, including the location of the keypoint in the image would help reducing the error of the system. The run-time performance of our neural network is on average 120Hz.

3.3.3 Computational Performance and Costs

We used a standard desktop PC running Ubuntu 12.04 with an Intel Core i7-4790 CPU at 3.60 GHz, 8 GB of RAM, and an nVidia GeForce 980GTX GPU with 4GB of memory with the price of 1500 Euros. The GPU was only used for training of the SDA networks using the Theano library, and we only used one CPU core to test the final trained network. The average training time for SDAs was about 2.5 hours for 150,000 images. Training of the localizing network took on average 1 day for 10,000 epochs using the FANN library. The runtime speed of the complete pipeline on one CPU core is on average 0.008 seconds.

On the other hand, Whelan et al. (Whelan et al. 2013) achieved a run-time speed of 0.03 seconds using CUDA implementation of visuo odometry for dense RGBD mapping on an Intel Core i7-3960X CPU at 3.30 GHz, and an nVidia 680GTX with the approximate price of 1800 Euros. Our approach has the advantage of higher speed and requires minimal hardware during runtime. The computational disadvantage of our method is the long training procedure which can be neglected since it is only done once, and can be delegated to cloud services to reduce the hardware costs.

3.4 Conclusion

In this chapter, we adopted a novel approach to tackle the robotic localization problem by using denoising autoencoders with image information and assistance of traditional mapping methods. We first experimented with multiple network architectures and training configurations. Next, we trained the autoencoders in 3D simulation using multiple feature vectors. We finally compared the localization error by attaching a two layer neural network, and associating ground truth values to the compressed autoencoder output. Our experiments showed very promising autoencoder reconstruction results in addition to low localization error. The error rates were approximately 10 centimeters and 4 degrees for 3D simulation using a one layer denoising autoencoder with sub-sampled HSV images. We can conclude that denoising neural networks perform well in retaining image structure, and can be used to both compress the image and associate location values to the compressed results. In addition, the network run-time computational requirements are so low that we were able to achieve 120Hz on a conventional processing unit.

3.5 Future Work

Despite the SDA results in unsupervised reconstruction of the scene and supervised location estimation through regularization of the latent space through denoising, variational autoencoders (VAE) (Kingma and Welling 2014) surpassed SDA's capability and performance in compression capability, robust latent space representation, and convolutional structure support in the encoding and decoding part (Kulkarni et al. 2015). VAEs have been applied in classification and robotics contexts such as Semi-supervised learning scenarios for image classification (Pu et al. 2016), generative zero-shot transfer in reinforcement learning (Higgins et al. 2017), and learning sampling distribution in robotic motion planning (Ichter et al. 2018). It is possible to replace the SDA with a VAE using a CNN back-bone to increase the location estimation precision and reduce the number of required labeled images.

In the context of increasing the performance of the current network, we can first examine the error propagation of the location estimation layer to the full network. This can be done after the initial training of the location estimation layer to avoid disrupting the SDA pre-trained weights. In addition, the relation between the size of the environment and the performance of the network is still unknown. Therefore, we plan to first carry out extensive tests on bigger simulated environments, and then move the experiments to real scenes and report the performance and requirements of the network. Although the results on the SIFT features were not promising, it is also clear that a part of the location information may come from sharp details of landmark objects, in addition to the overall scene appearance provided by the HSV full image. To incorporate the spatial layout of SIFT keypoints of relevant landmark objects, it will be conducive to explore the method of attentional patches (Sriman and Schomaker 2015) in future work. We also did not use the informative depth features such as the images given by an RGB-D sensor. It may further reduce the localization errors. We plan to reconstruct the scenes from the estimated positions using the full network, and attempt to build a 3D representation of the memory of the system. Finally, we are trying to combine the odometry of the robot and the neural network estimations by using an extended Kalman filter in order to increase the localization performance and exclude outliers.

Chapter 4

Two-Stage Visual Navigation by Deep Neural Networks and Multi-Goal Reinforcement Learning

Abstract

In this chapter, we propose a two-stage learning framework for visual navigation in which the experience of the agent during exploration of one goal is shared to learn to navigate to other goals. We train a deep neural network for estimating the robot's position in the environment using ground truth information provided by a classical localization and mapping approach. The second simpler multi-goal Q-function learns to traverse the environment by using the provided discretized map. Transfer learning is applied to the multi-goal Q-function from a maze structure to a 2D simulator and is finally deployed in a 3D simulator where the robot uses the estimated locations from the position estimator deep network. In the experiments, we first compare different architectures to select the best deep network for location estimation, and then compare the effects of the multi-goal reinforcement learning method to traditional reinforcement learning. The results show a significant improvement when multi-goal reinforcement learning is used. Furthermore, the results of the location estimator show that a deep network can learn and generalize in different environments using camera images with high accuracy in both position and orientation.

4.1 Introduction

Learning by reward and punishment is one of the fundamental learning methods in nature. This learning process is intrinsic in most of the species living on Earth, especially the ones with higher levels of cognitive abilities such as humans (Shteingart and Loewenstein 2014). This type of learning, reinforcement learning (Sutton and Barto 1998), has been a subject of research for a long time. Its modern form, which is highly based on Markov decision processes, started to emerge in the 1980s (Witten 1977), and became popular in the second half of the '90s (Sutton and Barto 1998), (Bertsekas and Tsitsiklis 1995).

There are two main learning methods in reinforcement learning (RL): model-based and model-free. The model-based approach requires a deep knowledge of the environment which allows us to build decision processes that connect the states with consequences of actions (Howard 1960), (Bellman 1957). However, the model grows substantially when the number of states and actions increases. In addition, it is often very complex to learn a model of the tasks that the agent needs to solve. Model-free RL plays an important role by allowing the exploration of unknown state spaces and using function approximators (FA) (Wiering 1999), (Reynolds 2002), (Busoniu et al. 2010). These approximators estimate values of actions in a state through a linear or non-linear mapping. The linear FAs are proven to converge, but they lack the ability to map complex states (e.g. camera images as input) into meaningful value estimates for actions. This is the reason that non-linear FAs are often used to tackle complex problems. One suitable and frequently used non-linear FA is an artificial neural network. The main problem with neural networks and other non-linear FAs is that they can diverge from the optimal solution due to forgetting past experiences or instabilities in the learning process (Wiering 1999). Therefore, careful thinking should be done during the design of the system. Types of feature inputs, the rewarding mechanism, and the propagation of these rewards are some of the important topics to consider. Due to these problems, the use of model-free RL methods was for a long time restricted to reasonably small sized problems such as robot gait control (Kohl and Stone 2004), unit control in games such as StarCraft (Shantia et al. 2011), etc. However, with the recent progress in training deep neural networks, the prospects have changed for RL. Google's DeepMind Research on Atari games (Mnih et al. 2013) was, perhaps, a substantial landmark and more influential than TD-Gammon (Tesauro 1995) towards a large scale RL deployment that can solve difficult tasks that matches or surpasses human performance. Later, in 2014, with Google's DeepMind research on the complex board game Go (Silver et al. 2016), we saw yet another hurdle being removed from the world of non-linear approaches in RL. The trend has not stopped there; we have seen many RL publications ranging from completing objectives in 2D games such as Doom (Kempka et al. 2016), besting top tier players at StarCraft II (Vinyals et al. 2017) to more real-world applications such as the Google research on robotic manipulation (Gu et al. 2017).

All the above applications have one thing in common, the need for hundreds of thousands of epochs which translates to significant training time. This can, however, be reduced by using a combination of simulation, transfer learning, or multiple agents. For example, in order to learn to avoid obstacles in the environment with a robot, one can first train it in a simulator, and then use the same network to continue the training in real life (Kahn et al. 2018). However, avoiding obstacles using a camera requires learning the optical flow. While it is possible to partially learn

optical flow in simulation and later complete the learning on a real robot, the same does not apply to place recognition, which is necessary for navigation. Nevertheless, it is possible to use simulations to speed up parts of the learning process. For this reason, we propose a method that can tackle robotic navigation tasks using a deep neural network architecture and model-free RL benefiting from simplified and complex simulations.

In this chapter, we propose a novel two-stage approach to alter the common end-to-end RL scheme and reduce the required time to learn to navigate in the environment. In the first stage, we train a deep neural network to localize the robot in the environment using supervised learning, and in the second stage, a multi-goal RL method is used which uses the estimated positions given by the deep network to drive the robot towards the given goals. In stage one, we use a traditional grid-mapping algorithm (GMapping) to extract the geometrical topology of the environment for the supervised training section of our approach (Grisetti et al. 2007), (Grisetti et al. 2005). A set of images is recorded during the data gathering phase that are tagged with the estimated location from the GMapping algorithm. In our previous research (Shantia, Timmers, Schomaker and Wiering 2015), we showed that a stacked denoising autoencoder (SDA) can learn the geometrical relation between the images and the robot location in a small 3D simulated environment. In this paper, we use deep convolutional neural networks (CNNs) (LeCun et al. 1998) to test the scalability and precision of this approach and compare the results to that of SDAs in order to select the best type of network architecture. In parallel, we use a grid-based approach to train our second stage multi-goal RL method. In the first step, we extract a maze from the given map, and train a multi-goal Q-function. When the training is finished, we continue the training of the same Q-function in a 2D simulator using the same map. Finally, we combine the position estimator network, which provides the global location, and the multi-goal Q-function, which was trained in the maze and 2D simulator, to navigate the robot to different destinations in the 3D simulated environment.

We summarize our contributions as follows:

- Proposing a novel framework for robot navigation.
- Testing the scalability and localization performance of convolutional deep neural networks that learn to map camera images to positions.
- Proposing a multi-goal reinforcement learning framework to learn to navigate to several different goals at once.
- Transfer learning from maze to the 2D, and 3D physics simulator.

- Comparison of the results of the proposed multi-goal framework with traditional reinforcement learning.

In Section 4.2, we further investigate the state of the art in robotic localization and the advances in deep RL. In Section 4.3, we describe the different methods used in this chapter for position estimation and multi-goal RL. In Section 4.4, we portray in detail the experiments done in which different types of position estimator neural networks are compared, and further elaborate on the environments that are used to carry out the tests. We continue the section by demonstrating the results for the multi-goal RL method. Finally, we discuss the benefits and shortcomings of our proposed method and conclude the chapter in Section 4.5.

4.2 Previous Work

Extensive research has been done on robot navigation, from indoor mobile robots (Thrun et al. 2002), to drones (Bristeau et al. 2011), and vehicles (Levinson et al. 2011). Most of the research is focused on creating a map and localizing the robot in this map using a variety of sensors, such as 2D/3D LIDARs (Hornung et al. 2013), (Bristeau et al. 2011), cameras (Bonin-Font et al. 2008), or a combination of both (Whelan et al. 2013). Often these maps are created based on the notion of grid cells. These methods extract geometrical information from the scene and stitch them together by combining the robot motion model and the information that comes from the sensors using probabilistic approaches (Thrun 2002). In the end, the robot has to be able to plan a path (e.g. $A*$, or Dijkstra's algorithm, (Misa and Frana 2010)) and avoid static and dynamic obstacles to maintain safe and reliable trajectories. This is done mostly through different control algorithms which often, in a predictive manner, forward simulate the movement of the robot and check the sampled path versus a variety of criteria (e.g., proximity to obstacles, distance to global path, oscillation, etc.) (Fox et al. 1997), (Gerkey and Konolige 2008). The performance and scalability of these methods, however, is directly related to the precision of the sensors, and the available computational power. This is one of the reasons that most mobile robots or vehicles use dedicated 3D sensors such as laser range finders or time-of-flight cameras to map and navigate in the environment. This, however, comes with a price, the range finders are very expensive.

There have been attempts to solve this problem by using cheaper sensors such as cameras to map and navigate the environment(Visual SLAM). In these methods, either feature extraction and matching are used to find correspondences between multiple images using corner detectors (Torr and Zisserman 1999), (Nister et al. 2004), SIFT descriptor (Schonberger and Frahm 2016), ORB descriptor (Mur-

Artal and Tardós 2017), or featureless approaches are used that generate a global map using direct image alignment, and probabilistic depth maps (Engel et al. 2015). By applying geometric and motion constraints, theses algorithms separate static and moving features from one another to localize the robot and build a map. A recent survey paper by Saputra et al. (Saputra et al. 2018) gives a comprehensive view of these methods.

With the popularity of deep neural networks (Schmidhuber 2015), different approaches were made to tackle the navigation problem using deep neural networks. Previously, we transferred the knowledge from a traditional map to a stacked denoising autoencoder (SDA) in which the robot used grid mapping for training data, and could localize its position using a camera after training in a small environment (Shantia, Timmers, Schomaker and Wiering 2015). Bidoia et al. (Bidoia et al. 2018), used a semi-supervised approach to create a graph map using deep CNNs. QR codes were scattered in the environment while the robot randomly moved throughout the environment. Later, using these codes, several graph nodes were created which allowed the network to predict its location. Moving through the graph was done by remembering the geometrical distance between the nodes.

Wang et al. (Wang et al. 2017), proposed a deep visual odometry method using a recurrent convolutional neural network architecture that receives a pair of images in each forward pass and outputs the poses. In this architecture, the CNN learns to extract features from pairs of images while a long short-term memory (LSTM) block learns the motion model and movement in the environment. Zhou et al. (Zhou et al. 2017), presented an unsupervised learning framework for monocular depth and camera motion estimation from unstructured video sequences. While the results are promising and have comparable performance to supervised methods, the current framework requires intrinsic camera parameters and has difficulties with dynamic scenes. In our approach, we evaluate how well our architecture can learn positions using only one image before applying additional recursion complexity. In addition, our proposed architecture tackles the kidnapped robot problem.

Researchers have also used end-to-end learning schemes to solve this problem. Kemkpa et al. (Kempka et al. 2016) used RL to solve different sub-tasks in the game of Doom. Although this research lacks the constraints of a mobile robot due to an unconstrained movement in the virtual world, it shows how well the deep networks can condense information in the form of images and make decisions to reach certain goals. Kulhánek et al. (Kulhanek et al. 2019) used a long short term memory (LSTM) as part of their CNN network in combination with a modified batched advantage actor-critic (A2C) algorithm (Wu et al. 2017) to solve end-to-end visual navigation. The LSTM part of the network was added to solve the partial observability of the navigation task. In addition, the CNN network predicted the depth map and image

segmentation of the current observation and the goal image to enhance the training process and increase the generalizability of the network. Kahn et al. (Kahn et al. 2018) developed a generalized computational graph using deep recurrent neural networks to navigate a remote control (RC) robot in a hallway. The focus of this research was mainly to train the local control mechanism of the robot to perform movements without hitting obstacles using a fixed longitudinal and a controllable angular velocity. The researchers showed that by pretraining their model in simulation, they can achieve faster convergence and better results in the real world.
There is one main issue regarding these end-to-end approaches, and that is the difficulty of extracting meaningful and human understandable data from the system. It would be very difficult to tell the RC robot to go to a certain place (e.g. next to the kitchen counter), or extract information from the system where it thinks it is. Therefore, the training procedure has a hit-miss characteristic. One may never know whether the system will converge, nor which problems are causing the divergence.

Our focus is to make the navigation process goal oriented and explainable. We would like to know in which location the robot has difficulties reaching or localizing itself, and why. In addition, we would like to make the robot learn multiple objectives at the same time while exploring the space for the current objective. If the robot is traversing a home environment looking for the kitchen, it may gain knowledge about reaching the bedroom as well, or by using reverse logic, it can know which new places it has to explore.

There has been prominent research in the past years regarding the utilization of experiences and exploration, but the initial idea has been around for a long time (Lin 1993). The main drive to use the past experiences is to avoid spending a lot of time searching for different goals repeatedly while avoiding that the neural networks forget the old experiences. From this perspective, any attempt to reuse experiences is an advantage. The more recent reuse of this idea combined with deep neural networks was done by Mnih et al. (Mnih et al. 2016) which allowed multiple agents to share their experiences while a single server computes the gradients and sends the newest neural network function approximator back to the agents. As we mentioned before, neural networks as function approximators have problems with forgetting past experience. Therefore, in DQN (Mnih et al. 2013) the agent uses a replay buffer as well. Every now and then, this buffer is sampled from, and the state-action pairs are used to train the system.

While using neural networks as function approximators, one can wonder how to create one network that remembers all the Q-values for different goals. The universal value function approximator (Schaul et al. 2015) showed that this can be done by augmenting the feature input with the position of the goal. This way, the network will not have any problem remembering and assigning different Q-values to

different goals.

Although this approach allows us to use one function approximator for multiple goals, it solely relies on the generalizability of the FA for providing correct Q-values for unseen goals. However, every action results in reaching a new state which can become a goal in the future. For example, if we are driving for the first time to the supermarket, we may also see the gas station on the way. Next time, if we want to go to the gas station, we do not have to search for it. This is the idea behind hindsight experience replay (Andrychowicz et al. 2017).

Veeriah et al. (Veeriah et al. 2018) continued this trend to apply multi-goal RL using an unsupervised mastery in scenarios where there are no apparent goals. For example, in most of the Atari games, the goal is to increase the reward intake, and therefore, no specific single goal can be set.

Our research focuses on the specific navigation problem with obstacles and environmental boundaries in place. The simulated robot should be able to keep track of certain fixed goals and be able to use the experiences to learn to navigate to a new goal. In addition, due to the two-stage training style of the proposed approach, our method does not have to use the universal value function approximator and includes goals in the input. Instead, look-up tables are used in our architecture in which each goal has a separate Q-function.

4.3 Methodology

In this section, we address in detail the methodology that we used to design and test the robot navigation pipeline. First, we elaborate on how we solve the exploration and path finding problem through our proposed multi-goal RL method, and then explain our position estimator network.

4.3.1 Multi-Goal Reinforcement Learning

The first step of our algorithm is to learn the correlation between the estimated position of the robot and the robot's utilities of different actions in different states. In end-to-end deep RL, the system learns to optimize (state, action) pair Q-values using an image as input. This approach has several drawbacks. The most prominent is the required number of trials for a complex task such as navigation. In order to learn to navigate in the environment, one would need a deep network to be able to extract meaningful information from raw images. In the case of model-free methods such as Q-learning (Watkins 1989), (Watkins and Dayan 1992) and Sarsa (Rummery and Niranjan 1994), (Sutton 1996), exploration of possible new states adds to the

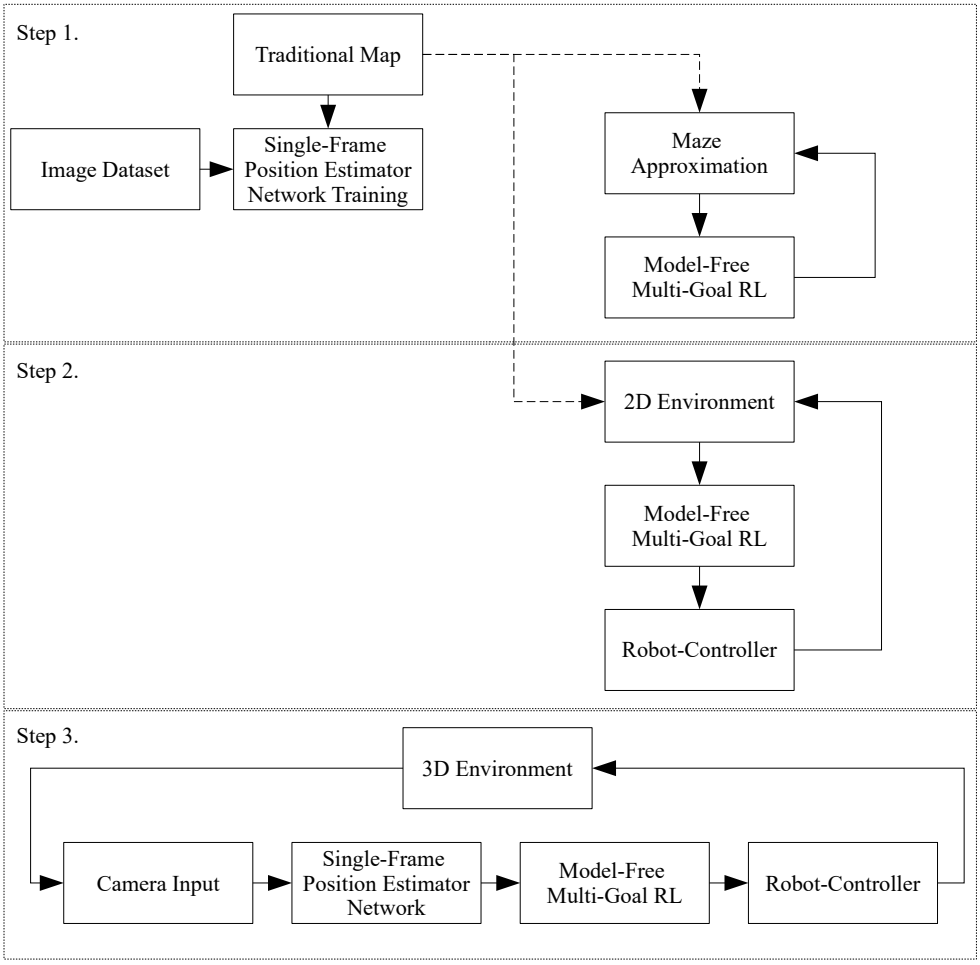


Figure 4.1: The complete proposed framework in a nutshell. In the first step, the position estimator is trained with positions extracted from a traditional mapping method while the model-free RL agent learns the environment using an approximated maze extracted from the map. In the second step, the RL agent continues to learn the effects of the robot controller on its actions in a 2D simulated environment. Finally, in the 3D environment, the agent uses position estimation from the porposed CNN and continues learning.

difficulty of the task at hand. Parallelizing the experiences in deep RL and multi-objective approaches, however, has allowed to reduce this number significantly in applications such as robot manipulation (Levine et al. 2016). Our proposed method

focuses on the idea of sharing the experiences between multiple goals and using a goal selection technique that gives us the most useful information from exploration throughout the trials.

In robotic navigation, the state space has six dimensions with three position and three orientation axes. Most robots, apart from drones and robots that operate in uneven terrain navigate only in three dimensions (X, Y, and θ). One can define this state space as a discrete set of blocks or a continuous set of numbers. In practice, low level control of the robot base is done in a continuous space while the location estimation and general path planning use a discrete approach.

We use a discrete separation of locations with fixed resolution. While the selected action of the robot is decided through the multi-goal RL method, the movement of the robot between the cells is done using the dynamic window approach (DWA) (Fox et al. 1997). We only use the X and Y dimensions in order to reduce the size of the state space. Our localization method, as described in section 4.3.2, accurately estimates the orientation of the robot which allows the local controller to reach the desired orientation. Figure 4.2 shows a discrete and down-sampled maze extracted from a higher resolution 2D map.

We consider that the agent's knowledge of the environment is incomplete and requires exploration. Therefore, we select a model-free RL approach. In the model-free domain, we can either select on-policy or off-policy RL. On-policy RL algorithms, such as Sarsa are suitable when we want to evaluate the policy that also generated the current outcome of the agent's actions. Off-policy RL methods such as Q-learning, on the other hand, allow to train policies that did not generate the current data. This is the main reason why RL methods that share experiences with one another should always use off-policy RL. Google's DeepMind DQN (Mnih et al. 2013), and a promising robotic manipulation system (Levine et al. 2016) are examples of such approaches. Our proposed approach uses a model-free multi-goal off-policy RL algorithm. For example, in Figure 4.2b, the agent can use the experiences during exploration of goal 1 and use it later to reach goal 4.

Q-Learning

Q-learning requires a way to store Q-values for each state-action pair (s_t, a_t). These values estimate how good the given action is to reach the goal. The Q-values are updated based on rewards that are given to the agent when it reaches a goal or rewards that are given during the navigation phase. Equation 4.1 shows the general

Figure 4.2: Figure 4.2a is the 2D generated map of the big apartment using the Rao-Blackwellized grid mapping method with $5cm$ resolution. Figure 4.2b shows the down-sampled approximated maze of the map with $40cm$ resolution, which needs to be learned by the RL algorithm. The red squares show the initial positions, and the yellow squares show the goals for the RL experiments.

update rule for the Q-learning method (Watkins 1989).

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha\left(r_t + \gamma \max_a Q_k(s_{t+1}, a) - Q_k(s_t, a_t)\right) \tag{4.1}$$

After each action, the agent moves and receives a reward r_t. The Q-value of the previously selected action is updated based on the reward of the current state,

the performed action, and the prospect of the next state based on its highest Q-value estimation. The γ parameter is the discount factor which determines the importance of future rewards versus immediate rewards and α is the learning rate. The function that approximates the Q values from each state-action pair can either be linear or non-linear. For our problem, since the position estimation network provides the location of the robot, the state space of the RL can be a grid, hence we use a look-up table. For problems in which the state space and the connection to actions are not trivial, a more complex non-linear approximator is required. Considering the grid state space, the available actions to perform in each state are up, down, left, and right movements. We discard the rotation dimension to reduce the size of the state space. This is again possible because the position estimation network gives global coordinates, and before the robot needs to go to the next position, it can rotate to the desired angle.

Due to exploration with Q-learning and the required time to move a robot, the convergence to the optimal solution takes a considerable amount of time. We speed up the convergence by using previous experiences of the agent in the environment. We accumulate all the state, action, reward, and next state experience tuples (s_t, a_t, r_t, s_{t+1}) in a replay buffer. After a certain number of online updates, we recalculate target values. Then, we shuffle the buffer to remove correlations between data points that are next to each other in time. We take a mini batch from the dataset and train the Q-function. This process is continued until all the past experiences in the replay buffer are used.

Using the above approach, we can initialize a separate lookup table for each goal. After the Q-function for the first goal is optimized, we go to the next one until all the goals can be found. The downside is that for each Q-function, we must reset the values to the initial ones since these Q-values are optimized for one goal.

Multi-Goal Q-learning with Experience Replay

Using the Q-learning method allows us to train multiple policies at once. This means that while the agent is searching for one goal, it can learn about reaching other goals at the same time. To this end, the first step is to update the Q-values for all the goal Q-functions at the same time. In the worst-case scenario, none of the other goals will be traversed during the exploration of the current goal. However, the Q-functions for all the other goals can learn from all the negative results due to obstacles in the state space. In the best case, some of the other goals will be traversed during the exploration phase of the first goal, and therefore, the robot can learn to optimize its path toward them. With multi-goal Q-learning, multiple Q-functions are trained each time step which all have their own reward function that emits a reward r_t^g

(Equation 4.2):

$$Q^g_{k+1}(s_t, a_t) = Q^g_k(s_t, a_t) + \alpha\left(r^g_t + \gamma \max_a Q^g_k(s_{t+1}, a) - Q^g_k(s_t, a_t)\right) \tag{4.2}$$

The selection of which policy to use or on which goal to train is an important factor and determines the total time of convergence. We use the notion of temporal difference (TD) errors to measure how many times a secondary goal was traversed. The TD error is the squared difference between the new target of a Q-value of a state-action pair, and its previous value (Equation 4.3).

$$TD^g_{err} = (r^g_t + \gamma \max_a Q^g(s_{t+1}, a) - Q^g(s_t, a_t))^2 \tag{4.3}$$

When the TD error is high for some Q-function, it shows that the specified goal location has not been fully learned by the policy. When the TD value is low, it means that the robot has visited this state several times, and the Q-values are converging to an optimal value. We set up a TD-error matrix for each of the Q-functions with a high initial value and we update the elements of the matrix that the agent passes through. We only keep the last value of the TD-error. After reaching and meeting the convergence criteria for the current goal, we select the new goal with the highest TD error by averaging over all the actions for the next possible goal position. This makes sure that the robot traverses the environment in an efficient way and increases the chances of reaching other goals on the way.

Robot Movements and Transfer Learning

We discussed our approach toward solving a maze navigation problem using the multi-goal approach. However, learning to solve a maze is straightforward compared to a robot moving in an environment. A robot has a footprint which is a projected polygon of the 3D shape of the robot on the 2D floor plan. For easier calculations, all the concave footprints are often changed to a convex version. The robot also has an axis of rotation which is often on the center of the robot. Movements of the robot are given through a velocity vector. In a 2D navigation scenario this velocity consists of a positional speed in X, Y, and a rotational speed in θ. If the robot has a differential drive base, the positional speed in X direction, and rotational speed in θ can be used, while an omni-directional robot can freely move in all directions. The robot base program will translate these velocities to wheel speeds. In order to have a safe navigation system, we need to apply the speeds given to the robot and change the position and orientation of the footprint. Using obstacle

detection sensors, such as infrared, sonar, or laser, the robot can identify its position in relation to these obstacles and processes the movement commands in such a way that the edges of its footprint polygon never touch the obstacles on the way. We use the dynamic window approach (DWA) method to control the robot locally. In this method, the robot has a global path to the sub-goal, the location of the sub-goal, and a cost-map that shows the surrounding obstacles and the current footprint of the robot. The global path in our case is very short because the actions just move the robot from one cell to another neighboring cell (sub-goal). The location of the set sub-goal is therefore one step away from the robot in either X or Y direction. The cost-map is a matrix with a selectable resolution. Based on the sensor, if a certain cell is occupied, it will be marked. Multiple sensors can mark and clear cells in the cost maps. DWA will sample velocities in the available dimensions based on the type of robot. Then, it will forward simulate the robot movements for a short amount of time based on these samples. In the end, a trajectory will be selected which causes no collision, and keeps the robot close to the designated path with a safe distance from obstacles. This accomplishes having the robot move to neighboring cells.

In order to connect the maze navigation results to a more realistic setting, we must adapt the continuous movement of the robot in the 3D simulated environment, so it matches the cell centers in the maze. To this end, our first step is to move from the approximated maze to a 2D simulator. The 2D simulator has a simpler physical model and does not require any rendering, allowing us to speed up the simulation by a factor of ten in comparison to the 3D simulator. In the 2D simulator, we use the map that was made from the 3D simulation.

The actions in the simulator do not necessarily have the same results as in the maze due to the higher resolution of the map, and the nature of the DWA method. Actions can fail, especially in corners, and the RL method needs to be able to cope with it. We use the trained Q-function from the maze, and continue learning in the 2D simulator. It is notable that the robot position in the 2D simulator is always correct. Therefore, the robot is exactly at the location that it thinks it is. However, in the 3D simulator this is not the case, because the position is inferred from a camera image and this brings us to our next challenge, dealing with approximation errors.

Each robot has an odometry error model. There are 5 different noise types that affect the odometry of the robot. Three of which are the direct deviations in X, Y, and θ dimension. The other two are co-dependent errors across these dimensions. When the robot moves forward, it can slightly rotate, or when the robot rotates, the position of the robot can also change due to wheel skid. These 5 error types make a fully functional navigation based on odometry impossible. This problem is solved by estimating the global position of the robot using a variety of sensors and combining it with the odometry model through filtering. Non-linear Kalman

filters (e.g. extended (Jazwinski 1970), and Unscented Kalman Filter (Wan and Van Der Merwe 2000)), and particle filters (Gordon et al. 1993) are the mostly used methods. The idea is to predict the new position of the robot after a series of motion commands (Equation 4.4) and update this belief with the estimation of the position from any sensor (Equation 4.5).

$$\hat{bel}(x_t) = \int p(x_t|x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1} \tag{4.4}$$

$$bel(x_t) = \eta p(z_t|x_t) \hat{bel}(x_t) \tag{4.5}$$

Where x_t is the state vector (e.g. X, Y, and θ), u_t is the given motion command, and z_t is the sensor reading of the robot at time t. η is a normalization factor since the multiplications of the beliefs of the robot at time t by the probability $p(z_t|x_t)$ may integrate to a value unequal to one. By knowing the noise model of the robot movement, and the probability of being in the current location by the position estimator, one can keep a good track of the location of the robot for small movement steps, such as used in our methods.

For the final step of transfer learning, the trained Q-functions in the 2D simulator for all the goals will be further trained in a 3D simulator. In the 3D simulator, the position estimator networks give the estimated position of the robot, while the RL method must deal with wrong location estimations. Since the position estimation of the neural networks is accurate and the distance between the cells are small, the odometry error can be ignored. Therefore, we leave out Equation 4.4, and use the global estimation from the network to determine the grid-cell of the robot's position. We are curious to see whether the RL method is able to handle the measurement errors of the network without the use of motion prediction and learns to navigate the robot to all goals along the shortest path.

4.3.2 Position-Estimator Networks

In our previous research, we have shown that SDA can map images to positions in a small 3D simulated environment with adequate generalizability (Shantia, Timmers, Schomaker and Wiering 2015). However, the question is how well the network scales with the size of the environment. SDAs have shown promising results since their introduction in academia in 2009 (Vincent et al. 2010). Due to the fully connected architecture of the network, the number of weights to train increases significantly when the input vector is large. Each additional layer will increase the number of weights significantly which requires a large amount of training data in order to learn a good position estimator. In addition, by flattening the two-dimensional

image into a one-dimensional vector, we lose topological information of the pixels, which is crucial in the process of learning. Perhaps, these are the main reasons that SDAs were outdated quickly with the advent of faster learning and more accurate CNNs. The CNN was first proposed by LeCun et al. (LeCun et al. 1990) but they became widely popular after Alex Krizhevsky et al. (Krizhevsky et al. 2012) bested the ImageNet Large Scale Visual Recognition Competition (ILSVRC) in 2012. Combining the processing power of GPUs with the new ReLU activation function, dropout (Hinton et al. 2012), and contrast normalization of the images in the CNN architecture allowed them to perform much better than all other approaches in the object recognition field. Since then, researchers have been working to find ways to optimize the training and design of the networks — creating deeper and deeper networks — such as the Google Inception architecture (Szegedy et al. 2015), (Szegedy et al. 2017) while others focused on broadening the application of CNNs in different fields such as object detection (Ren et al. 2015) and semantic segmentation of the scene (Long et al. 2015). By changing parts of the CNN architecture, we use its potential for scalability and generalizability, and compare the results of CNNs and SDAs to select the best network to estimate the position of the robot in the environment.

Convolutional Neural Networks

When we talk about CNN architectures, it is important to see what type of problem we are trying to solve. CNNs are used extensively in the area of object recognition, where one would like to recognize the main object that can be displayed in different parts of the image. Therefore, one would like the networks to have scale and translation invariance. This is where the pooling layers play an important role. The pooling layers not only reduce the size of the feature map of a hidden unit, they also carry either the maximum or the average response over the input window. When applied in multiple layers, it works as a scale, and shift invariant feature for the network. This property, although useful for object recognition, is detrimental for precise location estimation (Bidoia et al. 2018). We would like the network to observe exactly where an edge was in the image and with what size, since this is important for localizing the agent. Therefore, we do not use pooling layers in our architecture. This has a drawback as well, the input resolution should be limited, because the number of parameters to optimize are greatly higher than that of networks with pooling layers. We experimented with different depths of the network, and single or multiple kernel sizes in the same layer which we will explain in detail in the next section.

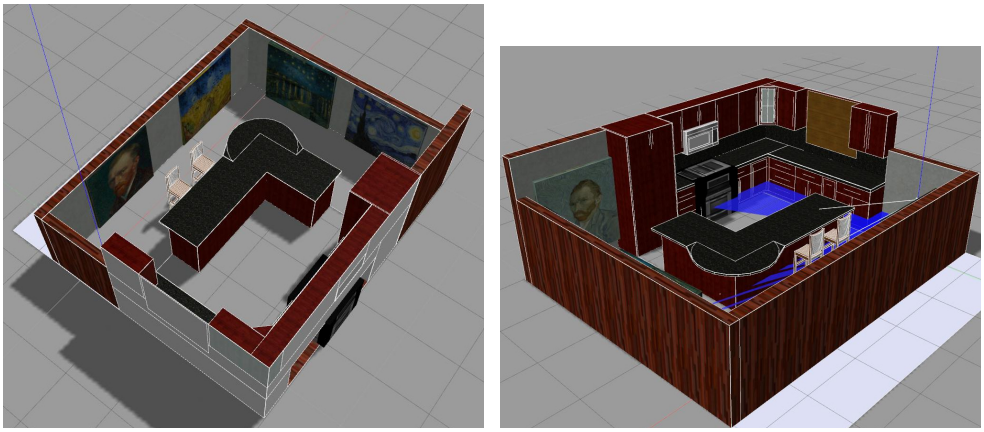


Figure 4.3: Small kitchen room of size 7×5 meters.

Figure 4.4: Big apartment of size 14×9 meters.

4.4 Experiments and Results

In this section, we describe the experiments for the position estimator networks, and the reinforcement learning algorithms in the maze, 2D, and 3D simulation. For the environments, we use one maze simulator, the 2D Stage simulator (Vaughan 2008), and the 3D Gazebo physics simulator (Koenig and Howard 2004). We used the robot operating system (ROS) framework (Quigley et al. 2009) to develop our software. The environments for the experiments are shown in Figures 4.3 and 4.4. We used OpenCV, Theano, and Tensorflow to extract images and train the SDA and CNNs (Bradski 2000), (Bergstra et al. 2010), (Abadi et al. 2016).

4.4.1 Data Gathering

We first extract the map of the 3D simulated environment (see Figure 4.3 & 4.4) using the Rao-Blackwellized grid mapping approach (Grisetti et al. 2007). In this method, the map-building process starts from the first laser reading. When the robot moves, the position of the robot is updated by using scan matching of the new laser reading with the old one. If this scan matching fails, the odometry model of the robot is used to update the location and merge the laser reading from the existing map with the previous map. This process led to the map shown in Figure 4.2.

We divide the map in 25-centimeter cells. The center of each cell is a location that the robot should traverse in order to gather the training and test data. In order to do this in an automatic manner, we use Dijkstra's shortest-path algorithm to plan a path to the center of the cell (Misa and Frana 2010). We consider the footprint of the robot in order to make sure that it fits in the destination cell. The unknown locations, occupied locations, and locations where the robot does not fit are ignored. Finally, we send the robot to each of these positions, and record images and corresponding positions given by the grid mapping method while rotating the camera 1 degree at a time in each position.

After data gathering, we create a test, validation, and training set. We split the images from each location into thirty-six sections (using 10-degree steps). For each section, half of the positions and rotations together with the images are assigned to the training set. For the remaining five images, two go to the validation set, and three to the test set. Therefore, 50% of the data is used for training, 20% for validation, and 30% for testing. The total data set size for the small kitchen and the big apartment are 24,000 and 195,000 images, respectively.

4.4.2 Deep Networks and Localization

We performed a set of experiments to find the best architecture for the position estimator network. As mentioned in section 4.3, we use SDAs and CNNs to estimate the position and orientation of the robot. We want to find out which of these networks has the best global localization performance in the environment and how well they scale with the size of the room.

Layers	Units per Layer	Corruption per Layer
1	$[4000]$	$[0.2, 0.2]$
1	$[1500]$	$[0.2, 0.2]$
2	$[4000, 4000]$	$[0.2, 0.2]$
2	$[1500, 1500]$	$[0.2, 0.2]$
3	$[4000, 4000, 4000]$	$[0.2, 0.2, 0.2]$
3	$[1500, 1500, 1500]$	$[0.2, 0.2, 0.2]$

Table 4.1: SDA Architectures for Experiments

Stacked Denoising Autoencoder Architecture

For the SDAs, we selected a number of architectures that can be seen in table 4.1. The number of SDA layers, the number of neurons in each layer, and the size of the position estimator multi-layer perceptron (MLP) on top of the SDA layers are the subject of our tests. We train each network for 10,000 epochs, and save the validation results. The network with the lowest validation error for each architecture is used for testing. In our previous research, we did experiments in a room very similar to the small kitchen in Figure 4.3. However, there are two main differences in the setup of the localization experiment of this paper in comparison to the previous one. In our previous paper, we had a large pre-training data set of 150,000 unlabeled images. However, since we want to compare the results of SDAs to CNNs, we decided to use the pre-training phase on the smaller labeled data set. We estimate that the performance of the network will drop due to the importance of the unsupervised phase of SDA training. In addition, the environment used in our previous research had a large number of texture-less walls, which forced us to reduce the data gathering locations to positions with some textures in the field of view of the camera. In this paper, however, we added paintings to the walls to allow for complete data gathering of the environment, since the agent has access to meaningful information of the whole environment to localize itself.

The input size of the images for the SDA network are HSV color images of size 36×36 which led to the best results in our previous paper. Due to full connectivity of the layers, increasing the image size results in very high testing errors.

Layers	Convolution Type	Kernel
7	Convolution - No Pooling	5×5
7	Inception Convolution - Pooling	5×5, 3×3, and 1×1
9	Convolution - No Pooling	5×5
9	Inception Convolution - No Pooling	5×5, 3×3, and 1×1
9	Inception Convolution - Pooling	5×5, 3×3, and 1×1

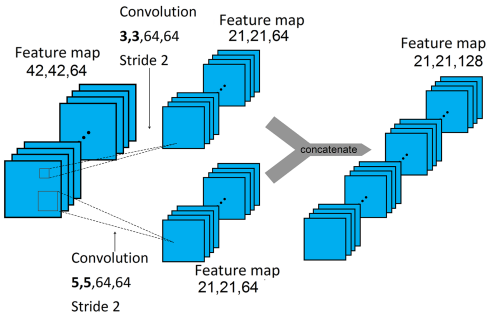
Table 4.2: CNN Architectures for Experiments. The dense layer size of all networks have 512 neurons, and the optimizer used for all the networks is Adam optimizer.

Convolutional Neural Networks

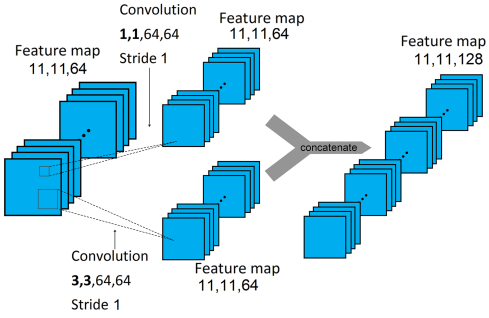
The same procedure applies to training the CNNs. Table 4.2 shows the CNN architectures that were trained and evaluated. For the CNNs, presence and absence of pooling layers, the depth of the network, and Convolution type (Inception vs. Normal) were selected as the main criteria for testing different architectures. We use strides in networks without pooling layers to reduce the number of parameters. We compare the use of RGB and HSV images to train and validate the CNNs on the date from the small kitchen, and will train the best performing network on the data from the big apartment. The input size for the CNNs are images of size $84 \times 84 \times 3$. The detailed design of the third architecture from Table 4.1 is shown in Figure 4.5.

4.4.3 Reinforcement Learning

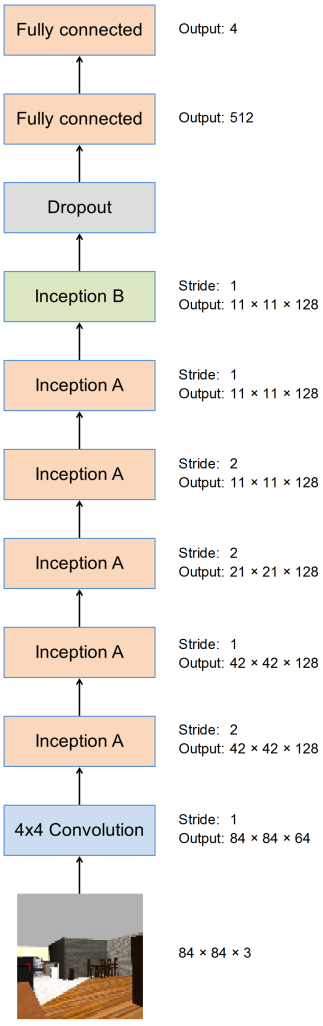
We first compare the normal and multi-goal Q-learning approach on random mazes. The mazes are grids of size 10×10 surrounded by walls. The randomization of the maze is as follows. The $(0, 0)$ cell is empty, and we call it the starting point. For each column, there is a thirty percent chance to have obstacles inside. When the column has obstacles, a random number will generate how long the obstacles will be. Another random number selects the position of the obstacles in the column. The length of the obstacle cannot exceed more than half of the maze. This procedure is repeated for all columns, and we also repeat it for all the rows. After this operation, there may exist obstacles with hollow cells inside. From the starting point, we perform connected component analysis, to find out all the cells that are reachable from this point. We fill the non-blocked remaining cells inside with obstacles. Then, we generate $X = 10$ random goals, the distance of these goals to the starting point $(0, 0)$ should be more than 7 steps. If the random agent cannot find these goals after several iterations, we reject the maze and start over. Finally, we randomize $X - 1$ more initial positions to create in total, 10 mazes with 10 goals and 10 initial positions.



(a) Inception A module architecture



(b) Inception B module architecture



(c) CNN Architecture

Figure 4.5: The nine-layer position estimation convolutional neural network architecture with inception. The input to the network is an 84×84 image; The output is estimated as X, Y, $\sin\theta$, $\cos\theta$. Images (A) and (B) show two different Inception modules and (C) shows the complete CNN architectures.

Reaching the goal gives the agent a reward of 100 while hitting blocked cells gives a reward of -2. All other actions receive a fixed punishment of -0.1. We initialize the Q-values to 80 to encourage exploration. The learning rate for the update of Q-values is 1.0 for the maze, 0.1 for the 2D simulator, and 0.8 for the Gazebo 3D simulator. The maze learning rate is set to 1.0 because the environment and actions are deterministic. For the 2D simulator, a smaller value is selected to avoid large changes in Q-values during training due to the stochasticity of the robot control. In the 3D simulator, however, we increased the learning rate because the simulation speed is slower and due to the position estimation errors the Q-values should be updated faster.

The agent uses Boltzmann exploration for the selection of actions for all the maze scenarios which can be seen in Equation 4.6.

$$P(a_i|s) = \frac{\exp(\frac{Q(s_t,a_i)}{T})}{\sum_{j=1}^{N} \exp(\frac{Q(s_t,a_j)}{T})} \tag{4.6}$$

T is the temperature, and for each action in state s_t, we compute action probabilities for the N actions. When the temperature T is high, actions will be assigned similar probabilities. When T drops, higher Q-values will have a higher probability to be selected. We initialize the temperature T to 2 with a decay value of 0.998. After each trial, the temperature is multiplied by the decay value. For the 2D and the 3D environment, however, we use the $\epsilon-$greedy method to reduce exploration because the Q-functions have already been trained and too much additional exploration costs more time in the simulators. The ϵ value is set to 0.1.

For the maze experiments, we consider a goal learned when the success ratio at reaching the goal is 100 percent for fifty consecutive trials from each starting position, and the average difference in number of steps to reach the goal is smaller than 10.

The results on the mazes allow us to reliably measure the performance of the multi-goal RL approach versus the sequential goal selection method. To assess the temporal difference goal selection, we perform a separate set of maze experiments with different convergence criteria. In these experiments, the required success ratio of 100 percent is only necessary for fifteen consecutive trials. We also remove the average difference in the number of steps to show the difference between the two approaches better. However, the final goal is to use our method to navigate a robot in a realistic environment. Therefore, we perform experiments in the 2D and 3D simulator as well. For these environments, we use the same big apartment map as during the data gathering phase. We have to, however, downscale the map and convert it to a maze with the correct cell sizes. We start the map approximation by

selecting the top left corner of the map images as $(0, 0)$. We divide the width and height of the map with the required cell resolution to extract rows and columns. Any cell with an occupied map pixel will be considered as an obstacle, and the rest will be free cells. We apply the same closing method as we perform for the random mazes.

Big Apartment and Transfer Learning

For the big apartment in Figure 4.4, the map size has a width of 17 and length of 9 meters. The cell resolution is 40 centimeters, and therefore the approximated maze size is 47×24. We run the RL tests on the approximated maze (Figure 4.2b), 2D simulator (Figure 4.2a), and 3D simulator (Figure 4.4). The location of the robot in the maze is just a cell index. For the 2D simulator, we use the ground truth pose of the robot to determine the location of the robot in the cells. In the 3D simulator, the position estimator network outputs the predicted position of the robot. These experiments are repeated 10 times.

In order to test the performance of transfer learning, we train the agent in the maze using RL, and then use the trained Q-function in the 2D simulator to cope with the possible problems of local navigation. Finally, the trained Q-function in the 2D simulator is used for the 3D simulation where the position estimator neural network predicts the location of the robot, and the RL method deals with the errors in the predictions. We expect to see a sharp decrease in the required time to converge to reliably finding the goals. These experiments are done on the big apartment (Figure 4.4).

Small Kitchen - HSV						
Layer Size	No. Layer	Corruption	Error			
			Position (m)		Angular (Degree)	
			Mean	Std	Mean	Std
1500	1	20%	0.218	0.213	12.6	14.0
4000	1		0.217	0.207	12.8	13.6
1500	2		0.204	0.221	11.6	13.7
4000	2		0.193	0.205	11.3	12.8
1500	3		0.198	0.214	11.6	13.9
4000	**3**		**0.182**	**0.205**	**10.7**	**12.5**
Big Apartment - HSV						
4000	**3**	20%	**0.560**	**0.869**	**28.8**	**48.9**

Table 4.3: The experimental results for position and orientation approximations with the stacked denoising autoencoders.

4.4.4 Experiment Results

In this section, we discuss the results of the position estimation networks and the RL experiments.

Position Estimator Results

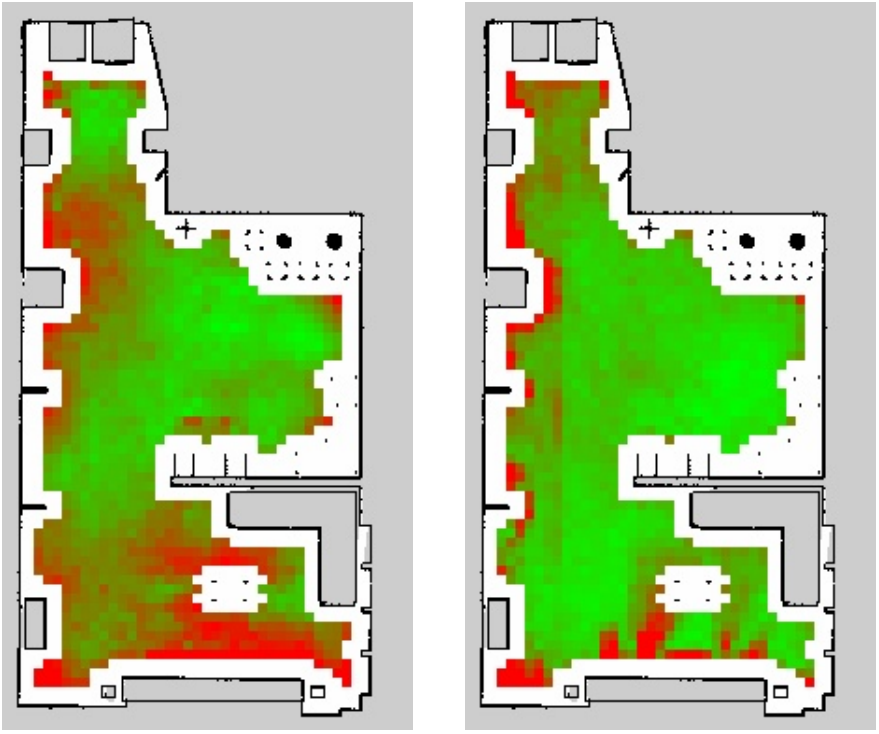
Table 4.3 shows the results of the SDA experiments. The best results for the small kitchen room are for the network with 3 layers and 4000 hidden units with 20 percent corruption of the input data. We used the same network for the big apartment as well. The high position and angular errors in the big simulated room show that these networks cannot scale well with the size of the environment.

Figure 4.6a shows a better overview of the error throughout the environment for the big apartment. For each location, the robot estimates the position for each angular rotation of the camera. Then, we average this error for each cell. The green color shows the minimum mean cell error value from the network and the shades toward red mean higher errors. Positions in the center of the map have a better localization in comparison to locations close to walls and corners. The main reason is that the robot has a wider view and can see a larger part of the room when it is in the center. This gives the network more information to distinguish its location robustly. In the corners and next to the walls on the other hand, the estimations are poor. Plain looking walls or objects do not have much information, and therefore the estimations suffer from this lack of information.

Small Kitchen RGB				
Network Type	Position Error (m)		Angle Error (Degree)	
	Mean	Std	Mean	Std
7 Layer	0.066	0.087	4.9	1.96
9 Layer	0.069	0.101	6.1	2.59
inception 9 Layer	0.056	0.079	4.9	2.07
Small Kitchen HSV				
Network Type	Position Error (m)		Angle Error (Degree)	
	mean	std	mean	std
7 Layer	0.065	0.086	6.1	2.46
9 Layer	0.069	0.101	6.2	2.60
Inception 9 Layer	0.056	0.080	5.1	2.10
Big Apartment RGB				
Network Type	Position Error (m)		Angle Error (Degree)	
	mean	std	mean	std
7 Layer	0.156	0.190	3.9	2.67
9 Layer	0.113	0.125	3.1	2.21
Inception 9 Layer	**0.076**	**0.067**	**3.8**	**16.3**
Inception Pool 7 Layer	0.324	0.087	5.6	1.94
Inception Pool 9 Layer	0.276	0.174	4.5	2.51

Table 4.4: The experimental results for the convolutional neural networks.

Table 4.4 shows the results of the position estimation for the CNNs. The best results are achieved by the Inception architecture in both rooms. In addition, the results for the RGB color space is slightly superior to that of the HSV space. The best 9-layer network with the RGB color space has 0.056 cm position error and 4.9 degrees angular error with 0.079 cm and 20.7 degrees standard deviation. The use of the RGB color space works slightly better with the CNNs which is confirmed by other researches as well (Sachin et al. 2018). Having more layers allows the system to encapsulate the environment better, while using the inception architecture in each level allows the network to use coarse and fine information at the same time. The smaller 3×3 kernels only use the surrounding pixel information while the bigger 5×5 kernels also include a larger neighborhood which adds more global informa-



(a) Position errors for the best SDA network. Green color represent minimum errors of 0.43 meter per location, and red color has the maximum error of 1.6 meter per location.

(b) Position errors for the best CNN network. Green color represent minimum errors of 0.12 meter per location, and red color has the maximum error of 0.56 meter per location.

Figure 4.6: The error heat map for the position of the robot in the big simulated room.

tion. The result of the inception network on the bigger room is also interesting. The positional and angular error has remained similar while the size of the room is doubled. This shows that the CNNs can scale with the size and type of the environment.

The results from table 4.3 and 4.4 clearly show that not only the CNN network performs better in a small environment, but it also scales much better when the environment is larger. For this reason, the 9 layer inception network is used to estimate the positions for the RL method in the 3D simulator.

Figure 4.6b shows the distribution of the errors in the big environment. The same procedure for the SDA heat map is repeated here. The higher error values are

Random Mazes						
	Maze Number					
RL Method	1	2	3	4	5	6
Normal RL	184k	209k	204k	185k	255k	212k
Multi-Goal RL	76k	82k	90k	73k	10k	91k

Random Mazes					
	Maze Number				
RL Method	7	8	9	10	**Average**
Normal RL	186k	172k	203k	201k	**201k**
Multi-Goal RL	83k	75k	78k	93k	**84k**

Table 4.5: The experiment with 10 different random mazes. The values in the table are the total number of steps required to solve the maze from each initial location to each goal.

strictly for the positions that are closer to walls. In addition, we also observe that the network gives higher position estimation errors when the scene has a large depth. Since the network uses only a single image, it will be quite hard for it to correctly estimate the depth, and the appearances of the objects that are further away do not change a lot when the camera moves toward them.

Reinforcement Learning Results

Table 4.5 shows the results of experiments with the normal and multi-goal approach for random mazes. The data show the average required number of steps to learn to navigate to all the goals from all the initial positions. The multi-goal approach on average has a 239% faster convergence time. Table 4.6 shows the comparison between random and temporal difference based goal selection with different starting temperatures for the Bolzmann exploration. Note that we relaxed the convergence criteria for this experiment. When the starting temperature and initial Q-values are high, the agents are encouraged to explore the complete environment, and most of the new goals do not need additional training. However, with a reduced starting temperature and initial Q-values, the agents will not encounter all the goals where smart goal selection can positively impact the convergence time. With this setting, The TD-based goal selection performs on average 9 percent better than random goal selection.

Random Mazes - Multi-Goal RL						
T = 1.0, Q = 0	Maze Number					
Goal Selection	1	2	3	4	5	6
Random	23k	31k	28k	27k	46k	29k
TD-Error	23k	30k	23k	26k	43k	28k
T = 2.0, Q = 80	Maze Number					
Goal Selection	1	2	3	4	5	6
Random	26k	39k	31k	27k	26k	33k
TD-Error	33k	41k	29k	27k	28k	31k

Random Mazes - Multi-Goal RL					
T = 1.0, Q = 0	Maze Number				
Goal Selection	7	8	9	10	**Average**
Random	32k	52k	38k	29k	**34k**
TD-Error	31k	41k	40k	28k	**31k**
T = 2.0, Q = 80	Maze Number				
Goal Selection	7	8	9	10	**Average**
Random	26k	50k	26k	31k	**32k**
TD-Error	21k	47k	28k	34k	**32k**

Table 4.6: Comparison between temporal difference error based goal selection and random goal selection with 10 different random mazes with different starting temperatures and initial Q-values.

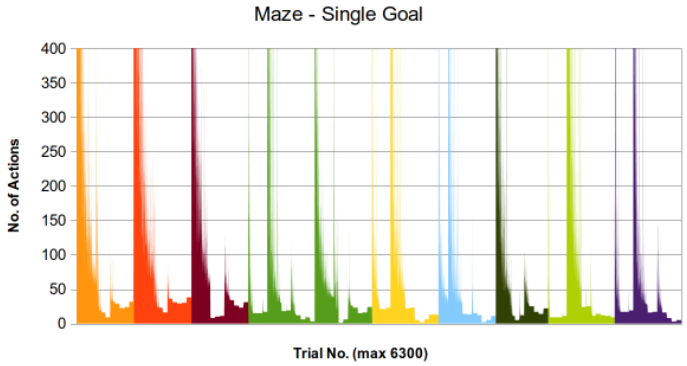
Table 4.7 and 4.8 show the results of the big apartment maze for the single and multi-goal approach. The multi-goal approach requires 40% less trials and requires half the number of actions as well. The large difference between the random mazes and the approximated maze can be explained by the position of the goals. In random mazes, it is possible that two goals are very close to each other which benefits the multi-goal approach. In our approximated maze, the goals and initial positions are scattered fairly, hence the improvement is less substantial. Figures 4.7a and 4.7b depict the differences between the single and the multi-goal approach more clearly. The single goal network has to learn to navigate to goals from scratch and therefore the number of trials and actions to learn the optimal path is higher compared to the multi-goal approach. For the multi-goal approach, however, all the goals that were close to the exploration range of the first goal were learned almost immedi-

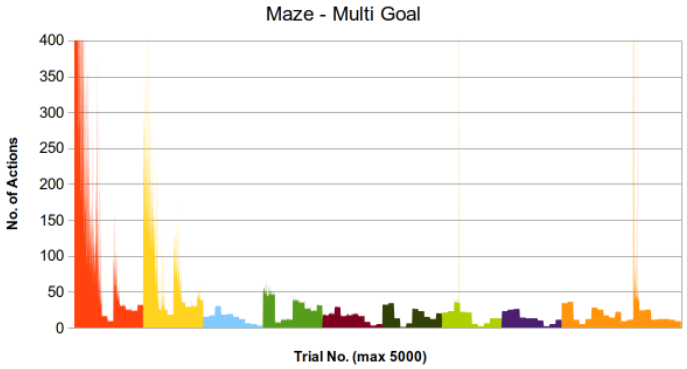
	Single Goal		Multi Goal	
Goal No.	Mean	Std	Mean	Std
1	112.1	50.8	112.6	50.6
2	123.2	62.9	107.7	48.6
3	89.6	16.4	50.0	0.0
4	87.2	28.2	50.1	0.1
5	107.2	49.8	50.4	0.3
6	99.7	22.6	51.3	0.9
7	87.3	17.8	50.7	0.5
8	88.4	31.6	51.0	0.7
9	96.0	21.9	52.4	1.6
10	98.9	32.2	50.1	0.1
Average	98.9	32.2	**62.6**	**10.3**

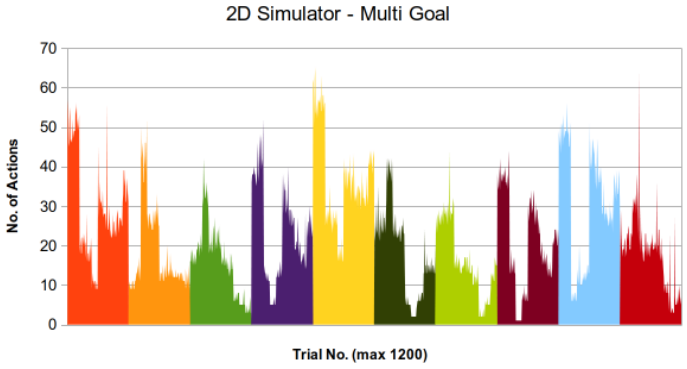
Table 4.7: The average number of trials to reach the goals from all initial positions in the approximated maze of the big simulated apartment using the single and the multi-goal approach. The multi-goal approach requires 40 percent less trials for convergence.

ately, while due to the experience sharing between the Q-functions and smart goal selection using the TD error the agent learned to navigate to all the goals faster. The TD-based goal selection obtained a similar overall performance as the random goal selection in this scenario. While the average number of trials is similar, the TD-based error has a significantly lower standard deviation, making it a suitable candidate for the rest of the multi-goal experiments as well.

Table 4.9 shows the performance of the multi-goal method in the 2D and 3D simulator. For the 2D simulation, the robot could reach all the goals. The required number of actions to reach the goals were slightly higher than that of the maze simulations. The stochasticity of the ϵ-greedy method, and non-deterministic behaviour of the controller of the robot can explain the higher number of actions. The local navigation system considers the robot footprint and must locally navigate from cell to cell. This can introduce problems when the robot is in tight corners due to insufficient sampling of the velocity space by the DWA approach which may result in failed or incomplete actions.


Maze - Single Goal
No. of Actions
400
350
300
250
200
150
100
50
0
Trial No. (max 6300)


Maze - Multi Goal
No. of Actions
400
350
300
250
200
150
100
50
0
Trial No. (max 5000)


2D Simulator - Multi Goal
No. of Actions
70
60
50
40
30
20
10
0
Trial No. (max 1200)

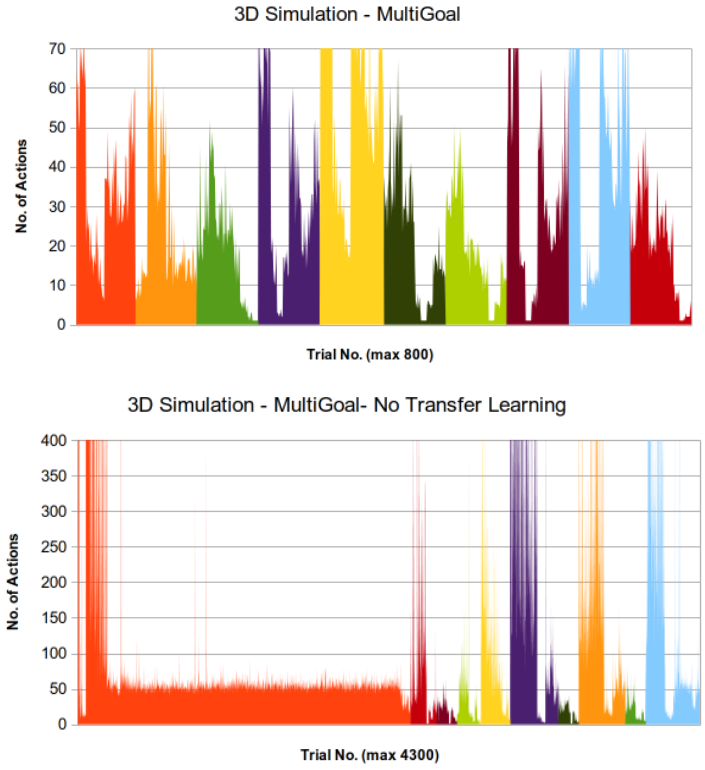


Figure 4.7: The number of actions versus the number of trials for the multi-goal and the single goal RL method in the approximated maze, and for the multi-goal approach in the 2D and 3D simulation of the big apartment. Figure 4.7a shows the result for the single goal approach on the approximated maze. Figure 4.7b shows the multi-goal results for the approximated maze. Note that the Q-functions have almost converged after the second goal. Figure 4.7c shows the 2D simulator results with the multi-goal approach. Figure 4.7a shows the results of the 3D simulation with the multi-goal approach. Note that the number of actions to reach some of the goals are considerably higher than for the 2D simulation due to the errors of the CNN position estimation. Figure 4.7b shows the results of the multi-goal approach without transfer learning in the 3D simulation.

	Single Goal		Multi Goal - TD		Multi Goal - Random	
Goal No.	Mean	Std	Mean	Std	Mean	Std
1	115.0	73.4	117.5	2.8	120.6	20.7
2	103.5	59.7	54.7	5.0	28.3	13.7
3	55.2	15.7	26.6	0.1	19.8	7.6
4	68.0	37.9	17.1	3.2	21.9	7.8
5	79.7	49.0	17.4	2.4	17.7	3.4
6	65.5	23.7	20.8	5.4	24.3	11.4
7	56.0	17.9	16.8	4.1	18.9	7.6
8	75.0	44.2	18.1	2.9	22.7	5.8
9	53.4	18.3	20.3	4.2	24.6	11.2
10	59.0	16.3	17.7	5.9	23.0	9.8
Average	73.0	35.6	32.7	3.6	32.2	9.9

Table 4.8: The average number of actions to reach the goals from all initial positions in the approximated maze of the big simulated room using the single and the multi-goal approach. The multi-goal's required number of actions is two times less compared to the single goal method. This shows that goals were quite often reached during exploration to other goals. The temporal difference goal selection has a slightly higher mean but a significantly lower standard deviation in this specific maze.

The 3D simulation results show a higher number of actions to reach the goals (also in Figure 4.7a). In the 2D simulator, the location of the robot is always correct. However, in the Gazebo 3D simulation, the inception CNN was used to estimate the position of the robot. Therefore, due to erroneous estimations, cell selection will be incorrect at some positions in the room. As can be seen in Figure 4.8, it is possible that the actual location of the robot is slightly different than the estimated position. This problem becomes larger if the robot faces plain looking walls, or texture-less surfaces, where the output of the CNN is usually the average of all the different positions with the same input. However, the ϵ-greedy method allows the robot to get out of these situations but with a cost of a higher number of actions.

Table 4.10 shows the results of training the agents directly in the 3D simulator without any transfer learning. From the large standard deviations and higher means compared to Table 4.9, and the number of agents that failed to learn all the goals, it is evident that the two-stage transfer learning significantly speeds up the learning process.

	2D Simulation		3D Simulation	
Goal No.	Mean	Std	Mean	Std
1	35.3	7.3	43.6	10.3
2	42.4	7.8	73.4	16.3
3	19.7	2.6	28.0	4.6
4	25.2	6.4	46.7	15.7
5	33.7	7.9	44.5	11.1
6	23.8	4.0	28.9	5.0
7	22.1	4.2	26.6	5.3
8	27.6	6.3	43.0	12.5
9	19.2	4.4	27.7	9.4
10	21.2	2.1	29.0	3.7
Average	**27.0**	5.3	**39.1**	9.4

Table 4.9: The average number of actions (after convergence) to reach the goals from all initial positions in the 2D and 3D simulator using the multi-goal approach. The 2D simulator used ground truth positions for the robot localization while the 3D simulation used the CNN to estimate the positions.

4.5 Discussion

In this chapter, we introduced a two-stage visual navigation system using deep convolutional neural networks and multi-goal reinforcement learning. Our goal was to design a system that is explainable, robust, and resilient to localization errors. Therefore, we first investigated whether a deep convolutional neural network is capable of learning position information based on an image and whether it can generalize well for locations that are outside of the training set. We performed several comparisons between different CNN architectures and our previously proposed SDA architectures on a small and a large 3D simulated environment. The proposed inception CNN architecture performed best with 0.076m position error and 3.8 degrees angular error in the large room and 0.056m position error and 4.9 degrees angular error in the small room. CNNs proved superior to SDA in both accuracy and the ability to scale with a more considerable amount of data. However, CNNs, similar to other methods that rely on visual input, suffer greatly from lack of texture, and therefore their output values should be used in combination with the motion model of the robot. Based on the results, we can argue that the proposed CNN has good potential in robot localization. We then tested our proposed multi-goal RL method to see

	Trials		Actions		Not Reached
Goal No.	Mean	Std	Mean	Std	No. of Agents
1	6849.6	5154.0	74.8	8.4	0
2	376.6	253.2	87.9	40.3	3
3	181.1	104.5	46.9	63.5	1
4	141.8	1.8	24.0	3.8	0
5	432.0	60.0	144.4	45.2	4
6	142.4	5.6	21.3	6.4	1
7	153.1	15.8	33.8	5.8	1
8	473.1	181.5	189.7	31.6	1
9	3141.5	5343.4	131.1	60.9	3
10	160.4	23.9	372.7	26.1	0
Average	1205.2	1114.4	80.1	29.2	

Table 4.10: The average number of trials and actions (after convergence) to reach the goals from all initial positions in the 3D simulator using the multi-goal approach without transfer learning. The right most column is the number of agents that couldn't learn the respective goals after 14 thousand trials. These experiments took approximately a month to complete.

whether a combination of dynamic goal selection, experience sharing, and transfer learning can reduce learning time. We first tested the multi-goal and single-goal approach in 10 different random mazes with 10 random goals and 10 random initial positions. Our multi-goal approach learned to solve all navigation tasks around, on average, 240 percent faster than the traditional method. We then focused on robot RL experiments and tested how transfer learning can reduce learning time. We compared RL agents on the approximated maze of the large simulated room using the multi-goal and the single-goal approach. The multi-goal agent was able to learn to navigate to all the goals using half of the total number of actions required by the single-goal method. We transferred the trained multi-goal agent to the 2D simulation of the large room in which we evaluated the effects of the stochasticity of robot base movements. After learning to navigate to all goals in the 2D simulator, we transferred the agent to the 3D simulated environment. We showed that while the agent could guide the robot to the goals, the number of actions for convergence was slightly higher in the 3D simulator in comparison to the 2D simulator. The higher number of actions was caused by the position estimation errors of the CNN and the stochasticity of the robot moving platform. We can conclude that CNNs can learn

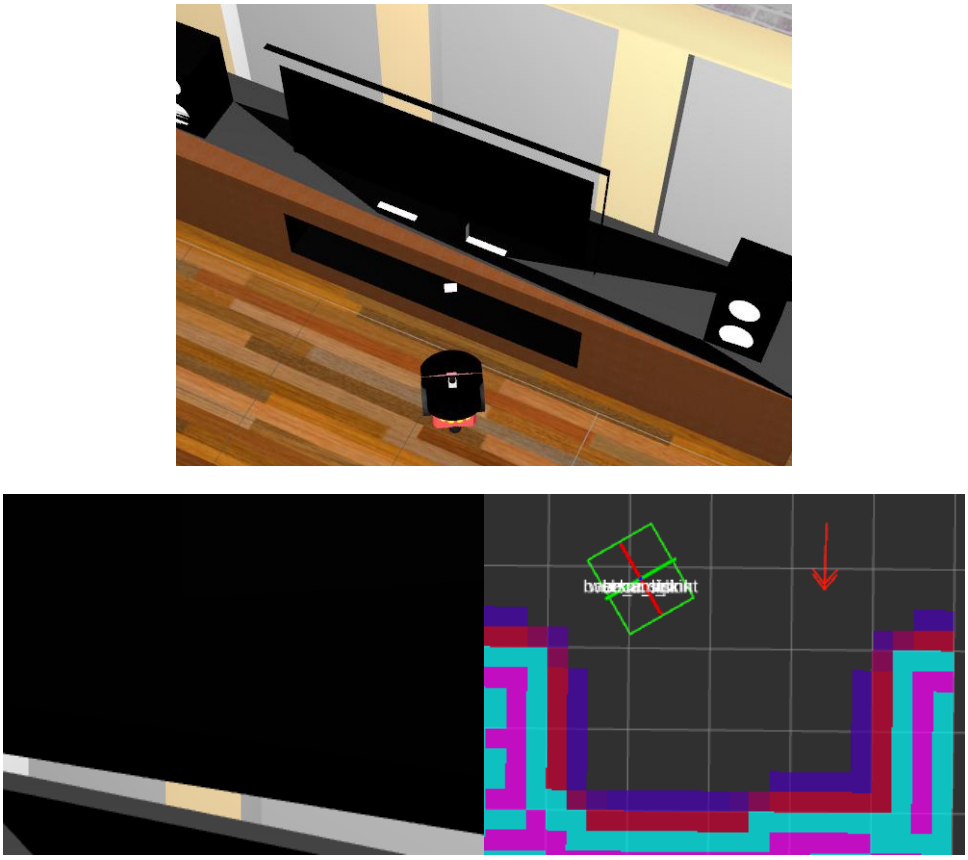


Figure 4.8: An example view of erroneous position estimation from the CNN. Figure 4.8a shows one of the positions in the big apartment where the CNN estimation is incorrect due to the limited view of the robot. Figure 4.8b shows the robot's camera view. Figure 4.8c shows the actual position of the robot versus the estimated pose. The estimated pose is shown by the red arrow, while the robot footprint is the green rectangle. The other colored pixels are the inflated cost maps with increased range to better visualize the situation.

and transform images into reliable coordinates for localization and that a multi-goal RL agent can achieve faster convergence by sharing the experiences of one goal with all the others using transfer learning, and smart exploration to reduce the required number of actions to navigate to different goals.

For future work, the first improvement step could be to learn continuous control of the robot. It is possible to use a neural network with the position, angle, velocity,

and acceleration information of the robot as inputs, and longitudinal and angular velocity as outputs. This would simplify the method and since no discretization would be needed, the generalization performance would increase, which should lead to less training time. Also, it is possible to expand the multi-goal system to contain realistic, conflicting affordances. Similar to Van Moffaert et al. paper (Moffaert and Nowé 2014) on pareto dominant policies in multi-objective RL, besides finding goals, the robot should minimize battery usage and exploration time.

We also want to research how we can train the position estimator network without needing so much accurate position data. It might be possible to use the locally accurate robot odometry model for this purpose. This would allow to scale up our system to learn to navigate in even larger environments.

Chapter 5

Discussion

In this thesis, we addressed several parts of a robotic navigation system. A navigation system consists of three main modules; mapping and localization, control, and planning. State of the art navigation systems rely on a variety of sensors and actuators to navigate safely in an environment. Often through occupancy grid mapping methods, a spatial map of the environment is created by combining visual or range sensory readings with the movement of the robot. Additional sensors can assist the robot in more reliable sensing of the obstacles. The robot can then plan a global path through the environment. Finally, a controller produces trajectories which moves the robot toward the goal while satisfying criteria such as smoothness and collision avoidance.

In Chapter 2, we reviewed the control algorithm of a widely used navigation pipeline. The dynamic window approach, like any other robot control system, produces trajectories based on a number of parameters and criteria such as:

- Longitudinal, lateral and angular acceleration, and velocity range
- Number of samples for acceleration and velocity during prediction
- Horizon of prediction
- Controller Frequency
- Distance to goal, obstacles, and the given path

For each controller loop, the robot forward simulates trajectories based on the available samples, and checks for safety, and closeness to the given path and the local goal. To reach the desired controller frequency, and because of computational limitations, the number of acceleration and velocity samples are limited. Therefore, either the robot speed, grid resolution, simulation time, or controller frequency should be reduced. Each of these either increases the navigation time or reduces the smoothness and reliability of the navigation system. We showed that a single set of parameters is not sufficient to guarantee high and reliable performance and answered the first research question in our thesis.

1. **Is it possible to improve an established navigation system using an available learning method?**

Our proposal to use a custom histogram of oriented gradients (HoG) on the occupancy grid of the robot proved that it is possible to distinguish various scenarios through unsupervised hierarchical clustering of these feature sets in both simulation and real environments. We also proved that it is sufficient to learn these clusters in simulation and use the results in real-life experiments without additional data gathering. By using a different set of parameters for each possible scenario, we can make sure that the robot operates with maximum performance. For example, in cluttered locations with bottlenecks such as doors, the robot can limit the acceleration and velocity range for a more careful approach while it can increase the number of samples to assure safety. There are several possible improvements to our method. We can additionally change the occupancy grid resolution during operation to enhance the tuning process — lower resolutions for areas that are open, and higher resolutions for cluttered environments. Besides, we can automate the parameter optimization process for each cluster. Another shortcoming of our feature descriptor is that it only encapsulates topological information. At this moment, we do not capture other characteristics such as speed and direction of dynamic objects and the robot itself. We can implement an additional feature descriptor that focuses on these characteristics. The hierarchical clustering can then use the combination of all the feature sets. With these additions, we can extend the application of our method to a broader range of robotics applications.

In Chapter 3, we investigated the robotic localization problem to address the next research question.

2. **Can artificial neural networks be used to solve the localization problem in navigation?**

We evaluated the accuracy and generalizability of stacked denoising autoencoders. Unlike the usual occupancy grid belief system which can suffer from initialization and the kidnapped robot problem, the SDA results showed that the network was able to recreate similar images of the environment after the unsupervised pretraining phase and that this initialization of the network weights allowed the backpropagation to achieve better results compared to a normal MLP with similar design. Since the network is not relying on the robot's motion, it is resistant to the kidnapped robot problem. We observed that a neural network is capable of encoding images and the respective positions. However, our proposed SDA suffers from sparse-texture views; therefore, the position is unreliable when the robot faces a

plain-looking wall. Additionally, there is no indication of the reliability of the output. Perhaps, by training an ensemble of networks, or using the dropout approach during network operation, we can extract a reasonable standard deviation. By using this as a noise value for the update section of a non-linear Kalman filter, we can track the position of the robot in sparse-texture places without problems.
In Chapter 4, we went further with testing the scalability of the SDA networks in larger environments to answer the third research question.

3. **Are convolutional neural networks better than stacked denoising autoencoders in solving the localization problem in larger environments, and are they scalable?**

The SDA results for a larger environment showed that this type of network is not capable of handling a larger and more complex problem. Increasing more nodes in each layer also didn't help, although the training did not show any signs of overfitting. Therefore, we switched to convolutional neural networks with deeper layers to evaluate their performance in these scenarios. We proposed a new CNN architecture based on the GoogLeNet Inception network (Szegedy et al. 2017). We reused the inception modules and removed pooling layers to avoid scale and rotation invariance. The CNN performed significantly better than the SDA in both environments. The reason for this superiority is that the CNN retains the topological structure of the image, and allows for deeper layers with less number of parameters. However, the problem with sparse-texture views remains. In addition, we observed that the propagation of the error is non-linear throughout the environment. The results show an auspicious learning system for localization. We should note that our research did not investigate how resilient the CNN is against occlusion of the view. Our preliminary analysis, shown in Figure 5.1, shows that only significant disruptions to the view can greatly affect the position data. Therefore we believe that this is a valid research area for future work. Nevertheless, the question of the usefulness of mapping only with convolutional neural networks is still an open question. We believe that the result of mapping an environment using neural networks is slightly different than the current grid-based approaches. For self-supervised training of a deep CNN, we ought to involve the prediction of the robot motion into the design. Therefore, on the one hand, the network should be able to learn discriminative features from the environment by minimizing a reconstruction loss function, which can be done through deconvolution layers in the network. On the other hand, we not only expect to learn the current state, but we want to predict the future based on a given movement command. Therefore, we can concatenate robot motion information to the flattened section of the network during the exploration and data gathering phase (Figure 5.2).

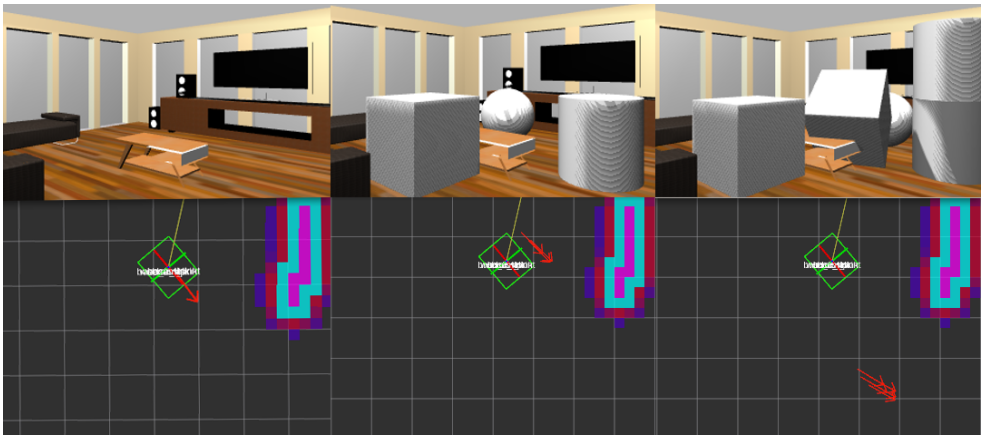


Figure 5.1: An example of the effect of scene occlusion to the position estimation of our proposed CNN Inception architecture. Preliminary analysis show resilience to partial occlusion. The green rectangle is the actual position of the robot, and the red arrows are the estimated positions.

However, it is challenging for the network to associate a couple of numerical physical attributes to the required geometrical changes in the image. Our suggestion is to also calculate the optical flow from the previous frame and concatenate the information in the layer before the deconvolution (Figure 5.3).

The goal of the network is to learn to predict the next image based on the current robot motion. The trained network should contain the inherently learned map without the conventional two-dimensional coordinates. Any positive results in this area can open new paths in the robotic mapping and localization field.

In the second part of Chapter 4, we used the trained deep CNN position-estimator network for the reinforcement learning experiments to answer the final research question of this thesis.

4. **How can reinforcement learning be improved to let a robot learn to navigate to many goal positions?**

We know that reinforcement learning methods are capable of solving mazes. The additional question that we wanted to answer was whether it is possible to train it promptly in a robot application. Our proposed goal exploration method by using the goal states' temporal difference error and the multi-goal experience sharing was our solution to this question. We considered the maze learning, robot movement stochasticity, and the non-linear CNN position estimation errors. The multi-goal approach with the TD-error goal selection strategy demonstrated the efficiency of

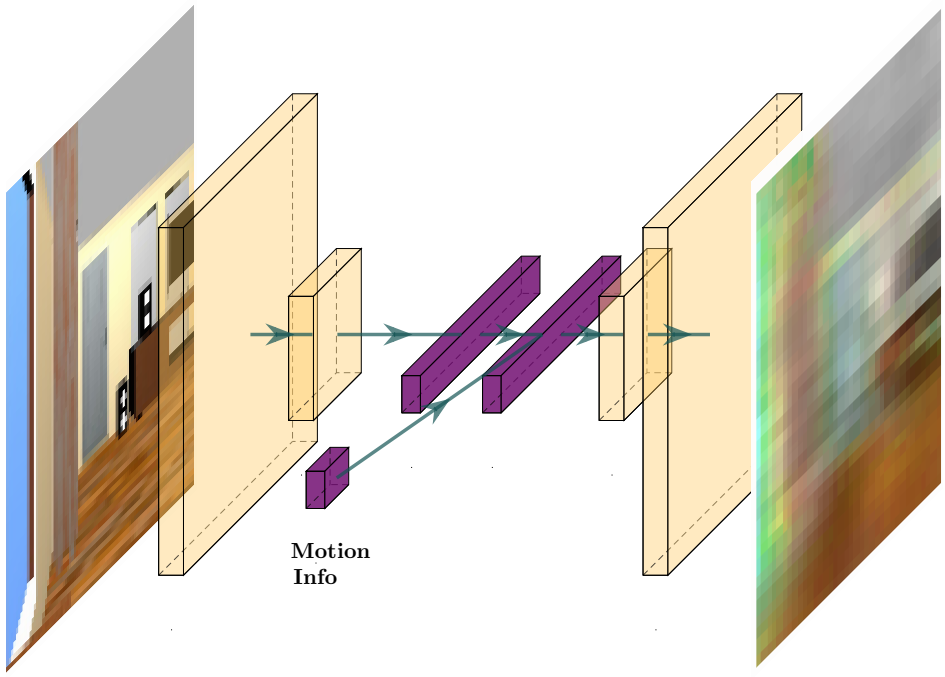


Figure 5.2: Convolution neural network concept for prediction of next state using the robot's motion information and deconvolution.

the methods for faster learning. The use of transfer learning from the maze to 2D and then 3D physics simulator proved valuable due to the reduced learning time on the robot. Our proposed multi-goal RL method could consistently reach all the goals. However, we propose to move from a discrete action space to a continuous one by using a non-linear function approximator for the Q-values. In this setting, we can use the robot's current longitudinal and angular acceleration and velocity information in combination with its estimated position. The selected actions are the robot's longitudinal and lateral velocity. Additionally, we should keep in mind to select a stopping velocity range for the action space. Finally, it is possible to expand the multi-goal system to contain realistic, conflicting affordances such as further reduction of exploration time and minimizing battery usage.

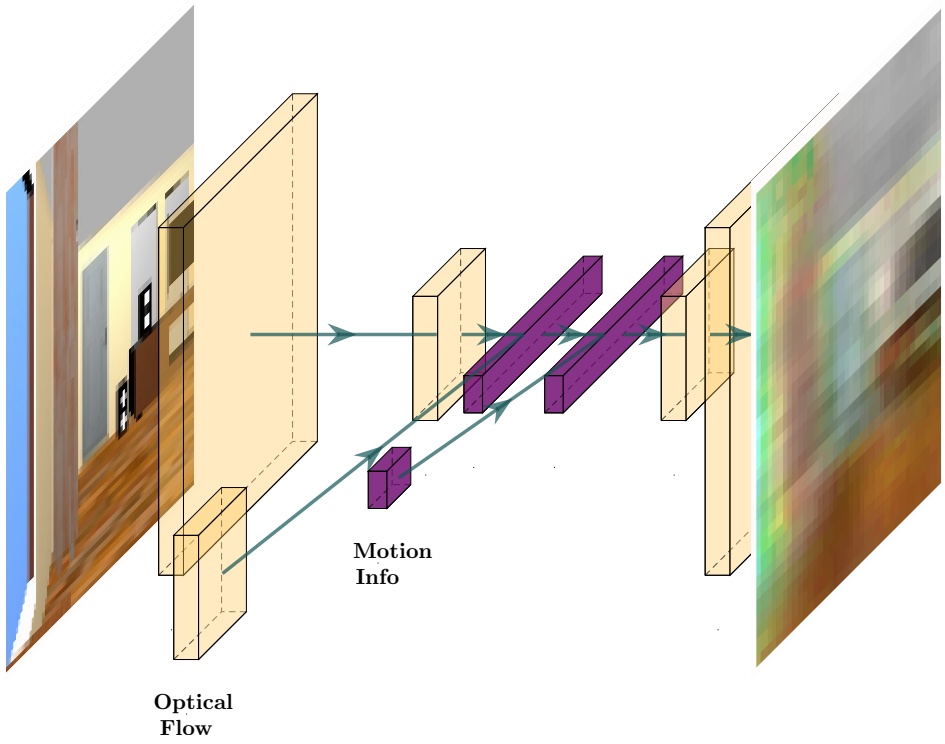


Figure 5.3: Convolutional neural network concept for prediction of next state using the optical flow in addition to plain robot motion information.

Epilogue

The results and experiences obtained in this project have allowed us to apply these methods in the autonomous driving industry. In (Shantia et al. 2019), we designed a framework in which we predict the direction of the travel of a vehicle by detecting and tracking the front wheel(s) using convolutional neural networks. The CNN is responsible for estimating the wheel angle based on previous images. We use the wheel angles to extract the change in vehicle yaw-rate, which allows us to calculate and track a trajectory using a Kalman filter with simplified bicycle motion model. In (Pathak et al. 2020), we proposed a reinforcement learning framework where the agent learns whether it is safe to perform a lane-change operation using the vehicle's current path planning system. In this framework, the road and object data

is projected into a multi-channel semantic image, each channel carrying a specific type of data (e.g., lane marking, object acceleration, and object speed channels). The agent encounters various scenarios in a simulation environment where it learns to decide whether a lane change is possible and beneficial. When the simulation results are satisfactory, we move the agent to an actual vehicle and continue the learning process.

The proposed machine learning methods for improving robotic navigation have been iteratively improved upon based on field experiences, case studies, and peer reviews. Nevertheless, no algorithm ever reaches perfection. This thesis is only a small part of the ever-growing robotics and machine learning field. Potential future directions include connected and explainable deep reinforcement learning frameworks, allowing transfer learning from simulated environments to the real world or new neural network architectures that can better encapsulate objects and their frame of reference in the world.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X.: 2016, Tensorflow: Large-scale machine learning on heterogeneous distributed systems, *CoRR* **abs/1603.04467**.
URL: *http://arxiv.org/abs/1603.04467*

Alonso-Mora, J., Beardsley, P. and Siegwart, R.: 2018, Cooperative collision avoidance for nonholonomic robots, *IEEE Transactions on Robotics* **34**(2), 404–420.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P. and Zaremba, W.: 2017, Hindsight experience replay, *Advances in Neural Information Processing Systems*, pp. 5048–5058.

Atkeson, C. G.: 1994, Using local trajectory optimizers to speed up global optimization in dynamic programming, *in* J. D. Cowan, G. Tesauro and J. Alspector (eds), *Advances in Neural Information Processing Systems 6*, Morgan-Kaufmann, pp. 663–670.
URL: *http://papers.nips.cc/paper/788-using-local-trajectory-optimizers-to-speed-up-global-optimization-in-dynamic-programming.pdf*

Barto, A. G. and Mahadevan, S.: 2003, Recent advances in hierarchical reinforcement learning, *Discrete Event Dynamic Systems* **13**(4), 341–379.
URL: *https://doi.org/10.1023/A:1025696116075*

Bellman, R.: 1957, A markovian decision process, *Journal of Mathematics and Mechanics* pp. 679–684.

Bellman, R.: 1961, *Adaptive control processes: a guided tour*, Princeton University Press, Princeton, New Jersey, USA.
URL: *http://books.google.nl/books?id=hIP5oAEACAAJ*

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D. and Bengio, Y.: 2010, Theano: a CPU and GPU math expression compiler, *Proceedings of the Python for Scientific Computing Conference (SciPy)*. Oral Presentation.

Bertsekas, D. P. and Tsitsiklis, J. N.: 1995, Neuro-dynamic programming: an overview, *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, Vol. 1, IEEE, pp. 560–564.

Bidoia, F., Sabatelli, M., Shantia, A., Wiering, M. A. and Schomaker, L.: 2018, A deep convolutional neural network for location recognition and geometry based information, *Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2018, Funchal, Madeira - Portugal, January 16-18, 2018.*, pp. 27–36.
URL: *https://doi.org/10.5220/0006542200270036*

Bonin-Font, F., Ortiz, A. and Oliver, G.: 2008, Visual navigation for mobile robots: A survey, *Journal of intelligent and robotic systems* **53**(3), 263–296.

Borrelli, F., Falcone, P., Keviczky, T., Asgari, J. and Hrovat, D.: 2005, Mpc-based approach to active steering for autonomous vehicle systems, *International journal of vehicle autonomous systems* **3**(2-4), 265–291.

Bradski, G.: 2000, OpenCV, *Dr. Dobb's Journal of Software Tools* .

Bristeau, P.-J., Callou, F., Vissiere, D. and Petit, N.: 2011, The navigation and control technology inside the ar. drone micro uav, *IFAC Proceedings Volumes* **44**(1), 1477–1484.

Brito, B., Floor, B., Ferranti, L. and Alonso-Mora, J.: 2019, Model predictive contouring control for collision avoidance in unstructured dynamic environments, *IEEE Robotics and Automation Letters* **4**(4), 4459–4466.

Burger, R., Bharatheesha, M., van Eert, M. and Babuska, R.: 2017, Automated tuning and configuration of path planning algorithms, *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4371–4376.

Busoniu, L., Babuska, R., Schutter, B. D. and Ernst, D.: 2010, *Reinforcement Learning and Dynamic Programming Using Function Approximators*, 1st edn, CRC Press, Inc., USA.

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I. and Leonard, J. J.: 2016, Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age, *IEEE Transactions on Robotics* **32**(6), 1309–1332.

Chen, L., Papandreou, G., Kokkinos, I., Murphy, K. and Yuille, A. L.: 2018, Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **40**(4), 834–848.

Dalal, N. and Triggs, B.: 2005, Histograms of oriented gradients for human detection, *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, Vol. 1, pp. 886–893 vol. 1. ID: 1.

Deng, J., Dong, W., Socher, R., jia Li, L., Li, K. and Fei-fei, L.: 2009, Imagenet: A large-scale hierarchical image database, *CVPR*.

Dubois, P. F., Hinsen, K. and Hugunin, J.: 1996, Numerical Python, *Computers in Physics* **10**(3).

Engel, J., Stückler, J. and Cremers, D.: 2015, Large-scale direct slam with stereo cameras, *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942.

Everett, H. R. and Gage, D. W.: 1999, From laboratory to warehouse: Security robots meet the real world, *The International Journal of Robotics Research* **18**(7), 760–768.

Feng, S., Xinjilefu, X., Atkeson, C. G. and Kim, J.: 2015, Optimization based controller design and implementation for the atlas robot in the darpa robotics challenge finals, *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pp. 1028–1035.

Floreano, D. and Wood, R. J.: 2015, Science, technology and the future of small autonomous drones, *Nature* **521**(7553), 460–466.

Fox, D., Burgard, W., Dellaert, F. and Thrun, S.: 1999, Monte carlo localization: Efficient position estimation for mobile robots, *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, American Association for Artificial Intelligence, Menlo Park, CA, USA, pp. 343–349.
URL: *http://dl.acm.org/citation.cfm?id=315149.315322*

Fox, D., Burgard, W. and Thrun, S.: 1997, The dynamic window approach to collision avoidance, *Robotics Automation Magazine, IEEE* **4**(1), 23 –33.

Garcia, J. and Zalevsky, Z.: 2008, Range mapping using speckle decorrelation. US Patent 7,433,024.
URL: *https://www.google.com/patents/US7433024*

Gerkey, B. P. and Konolige, K.: 2008, Planning and control in unstructured terrain, *Workshop on Path Planning on Costmaps, Proceedings of the IEEE International Conference on Robotics and Automation (ICRA*.

Glassner, A. S. (ed.): 1989, *An Introduction to Ray Tracing*, Academic Press Ltd., England.

Gómez-Consarnau, L., González, J. M., Coll-Lladó, M., Gourdon, P., Pascher, T., Neutze, R., Pedrós-Alió, C. and Pinhassi, J.: 2007, Light stimulates growth of proteorhodopsin-containing marine flavobacteria, *Nature* **445**(7124), 210–213.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C. and Bengio, Y.: 2014, Generative adversarial networks, *CoRR* **abs/1406.2661**.
URL: *http://arxiv.org/abs/1406.2661*

Gordon, N. J., Salmond, D. J. and Smith, A. F. M.: 1993, Novel approach to nonlinear/non-gaussian bayesian state estimation, *IEE Proceedings F - Radar and Signal Processing* **140**(2), 107–113.

Gower, J. C. and Ross, G. J. S.: 1969, Minimum spanning trees and single linkage cluster analysis, *Journal of the Royal Statistical Society. Series C (Applied Statistics)* **18**(1), 54–64.
URL: *http://www.jstor.org/stable/2346439*

Grisetti, G., Stachniss, C. and Burgard, W.: 2005, Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling, *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 2432–2437.

Grisetti, G., Stachniss, C. and Burgard, W.: 2007, Improved techniques for grid mapping with rao-blackwellized particle filters, *Robotics, IEEE Transactions on* **23**(1), 34–46.

Gu, S., Holly, E., Lillicrap, T. and Levine, S.: 2017, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, pp. 3389–3396.

Hart, S. and Grupen, R.: 2011, Learning generalizable control programs, *IEEE Transactions on Autonomous Mental Development* **3**(3), 216–231.

He, K., Zhang, X., Ren, S. and Sun, J.: 2016, Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

Hebb, D. O.: 1949, *The organization of behavior; a neuropsychological theory*, Vol. 65, Wiley, New York, NY, USA.

Henry, P., Krainin, M., Herbst, E., Ren, X. and Fox, D.: 2012, Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments, *The International Journal of Robotics Research* **31**(5), 647–663.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C. and Lerchner, A.: 2017, DARLA: Improving zero-shot transfer in reinforcement learning, *in* D. Precup and Y. W. Teh (eds), *Proceedings of the 34th International Conference on Machine Learning*, Vol. 70 of *Proceedings of Machine Learning Research*, PMLR, International Convention Centre, Sydney, Australia, pp. 1480–1490.
URL: *http://proceedings.mlr.press/v70/higgins17a.html*

Hinton, G. E.: 2006, Reducing the dimensionality of data with neural networks, *Science* **313**(5786), 504–507.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R.: 2012, Improving neural networks by preventing co-adaptation of feature detectors, *CoRR* **abs/1207.0580**.
URL: *http://arxiv.org/abs/1207.0580*

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C. and Burgard, W.: 2013, Octomap: An efficient probabilistic 3d mapping framework based on octrees, *Autonomous Robots* **34**(3), 189–206.

Howard, R. A.: 1960, *Dynamic Programming and Markov Processes*, MIT Press, Cambridge, MA.

Ichter, B., Harrison, J. and Pavone, M.: 2018, Learning sampling distributions for robot motion planning, *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7087–7094.

Jazwinski, A.: 1970, Stochastic process and filtering theory, academic press, *A subsidiary of Harcourt Brace Jovanovich Publishers* .

Kahn, G., Villaflor, A., Ding, B., Abbeel, P. and Levine, S.: 2018, Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation, *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, IEEE, pp. 1–8.
URL: *https://doi.org/10.1109/ICRA.2018.8460655*

Kapitanyuk, Y. A., Proskurnikov, A. V. and Cao, M.: 2017, A guiding vector-field algorithm for path-following control of nonholonomic mobile robots, *IEEE Transactions on Control Systems Technology* **26**(4), 1372–1385.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J. and Jaśkowski, W.: 2016, Vizdoom: A doom-based ai research platform for visual reinforcement learning, *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–8.

Khatib, O.: 1986, Real-time obstacle avoidance for manipulators and mobile robots, *Autonomous robot vehicles*, Springer, pp. 396–404.

Khoshelham, K. and Elberink, S. O.: 2012, Accuracy and resolution of kinect depth data for indoor mapping applications, *Sensors* **12**(2), 1437–1454.
URL: *http://www.mdpi.com/1424-8220/12/2/1437*

Kingma, D. P. and Welling, M.: 2014, Auto-encoding variational bayes, *in* Y. Bengio and Y. LeCun (eds), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.
URL: *http://arxiv.org/abs/1312.6114*

Kober, J. and Peters, J.: 2014, *Reinforcement Learning in Robotics: A Survey*, Springer International Publishing, Cham, pp. 9–67.
URL: *https://doi.org/10.1007/978-3-319-03194-1˙2*

Koenig, N. P. and Howard, A.: 2004, Design and use paradigms for Gazebo, an open-source multi-robot simulator., *IROS*, IEEE, pp. 2149–2154.

Kohl, N. and Stone, P.: 2004, Policy gradient reinforcement learning for fast quadrupedal locomotion, *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, Vol. 3, pp. 2619–2624 Vol.3.

Kosecka, J., Zhou, L., Barber, P. and Duric, Z.: 2003, Qualitative image based localization in indoors environments, *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, Vol. 2, pp. II–3–II–8 vol.2.

Krizhevsky, A.: 2009, *Learning Multiple Layers of Features from Tiny Images*, Master's thesis, University of Toronto.
URL: *http://www.cs.toronto.edu/˜kriz/learning-features-2009-TR.pdf*

Krizhevsky, A. and Hinton, G. E.: 2011, Using very deep autoencoders for content-based image retrieval, *ESANN 2011, 19th European Symposium on Artificial Neural Networks, Bruges, Belgium, April 27-29, 2011, Proceedings*.

Krizhevsky, A., Sutskever, I. and Hinton, G. E.: 2012, Imagenet classification with deep convolutional neural networks, *in* P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou and K. Q. Weinberger (eds), *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pp. 1106–1114.
URL: *http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks*

Kulhanek, J., Derner, E., de Bruin, T. and Babuska, R.: 2019, Vision-based navigation using deep reinforcement learning, *2019 European Conference on Mobile Robots (ECMR)*, IEEE, pp. 1–8.

Kulkarni, T. D., Whitney, W. F., Kohli, P. and Tenenbaum, J.: 2015, Deep convolutional inverse graphics network, *in* C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett (eds), *Advances in Neural Information Processing Systems*, Vol. 28, Curran Associates, Inc., pp. 2539–2547.

Kuperstein and Rubinstein: 1989, Implementation of an adaptive neural controller for sensory-motor coordination, *International 1989 Joint Conference on Neural Networks*, Vol. 2, pp. 305–310.

Lam, D., Manzie, C. and Good, M.: 2010, Model predictive contouring control, *49th IEEE Conference on Decision and Control (CDC)*, IEEE, pp. 6137–6142.

Laud, A. D.: 2004, *Theory and Application of Reward Shaping in Reinforcement Learning*, PhD thesis, University of Illinois, USA. AAI3130966.

LeCun, Y. and Bengio, Y.: 1995, Convolutional networks for images, speech, and time series, *The handbook of brain theory and neural networks* **3361**.

LeCun, Y., Bengio, Y. and Hinton, G.: 2015, Deep learning, *nature* **521**(7553), 436–444.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., Jackel, L. and Baird, H.: 1990, Constrained neural network for unconstrained handwritten digit recognition, *in* C. Suen (ed.), *Frontiers in Handwriting Recognition, Montreal, 1990*, CENPARMI, Concordia University.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P.: 1998, Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11), 2278–2324.

Levine, S., Pastor, P., Krizhevsky, A. and Quillen, D.: 2016, Learning hand-eye coordination for robotic grasping with large-scale data collection, *International Symposium on Experimental Robotics*, Springer, pp. 173–184.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., Kammel, S., Kolter, J. Z., Langer, D., Pink, O., Pratt, V., Sokolsky, M., Stanek, G., Stavens, D., Teichman, A., Werling, M. and Thrun, S.: 2011, Towards fully autonomous driving: Systems and algorithms, *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 163–168.

Lin, L.-J.: 1993, *Reinforcement Learning for Robots Using Neural Networks*, PhD thesis, Carnegie Mellon University, Pittsburgh.

Long, J., Shelhamer, E. and Darrell, T.: 2015, Fully convolutional networks for semantic segmentation, *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA*, pp. 3431–3440.
URL: *https://doi.org/10.1109/CVPR.2015.7298965*

Lowe, D. G.: 2004, Distinctive image features from scale-invariant keypoints, *International journal of computer vision* **60**(2), 91–110.

Lu, D. V.: 2014, *Contextualized Robot Navigation*, PhD thesis, Washington University in St. Louis.

Lu, F. and Milios, E.: 1997, Globally consistent range scan alignment for environment mapping, *Autonomous robots* **4**(4), 333–349.

Maciejowski, J.: 2002, *Predictive Control with Constraints.*, Prentice Hall, England.

MacQueen, J. B.: 1967, Some methods for classification and analysis of multivariate observations, *in* L. M. L. Cam and J. Neyman (eds), *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, Vol. 1, University of California Press, pp. 281–297.

Malika, C., Ghazzali, N., Boiteau, V. and Niknafs, A.: 2014, Nbclust: an r package for determining the relevant number of clusters in a data set, *J. Stat. Softw* **61**, 1–36.

MATLAB: 2014, *version 8.4.0 (R2014b)*, The MathWorks Inc., Natick, Massachusetts, USA.

Minsky, M. and Papert, S.: 1969, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, Cambridge, MA, USA.

Misa, T. J. and Frana, P. L.: 2010, An interview with Edsger W. Dijkstra, *Commun. ACM* **53**(8), 41–47.
URL: *http://doi.acm.org/10.1145/1787234.1787249*

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D. and Kavukcuoglu, K.: 2016, Asynchronous methods for deep reinforcement learning, *International conference on machine learning*, pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. and Riedmiller, M. A.: 2013, Playing atari with deep reinforcement learning, *CoRR* **abs/1312.5602**.
URL: *http://arxiv.org/abs/1312.5602*

Moffaert, K. V. and Nowé, A.: 2014, Multi-objective reinforcement learning using sets of pareto dominating policies, *Journal of Machine Learning Research* **15**(107), 3663–3692.
URL: *http://jmlr.org/papers/v15/vanmoffaert14a.html*

Mur-Artal, R. and Tardós, J. D.: 2017, Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras, *IEEE Transactions on Robotics* **33**(5), 1255–1262.

N. J. Nilsson, P. and B. Raphael: 1968, A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems, Science, and Cybernetics* **SSC-4**(2), 100–107.

Nair, V. and Hinton, G. E.: 2010, Rectified linear units improve restricted Boltzmann machines, *Proc. International Conference on Machine Learning*, pp. 807–814.

Nemec, B., Zorko, M. and Zlajpah, L.: 2010, Learning of a ball-in-a-cup playing robot, *19th International Workshop on Robotics in Alpe-Adria-Danube Region (RAAD 2010)*, pp. 297–301.

Nilsson, N. J.: 1984, Shakey the robot, *Technical report*, SRI INTERNATIONAL MENLO PARK CA.

Nissen, S.: 2003, Implementation of a fast artificial neural network library (fann), *Technical report*, Department of Computer Science University of Copenhagen (DIKU). http://fann.sf.net.

Nister, D., Naroditsky, . and Bergen, J.: 2004, Visual odometry, *Proc. CVPR*, pp. 652–659.

Nüchter, A.: 2009, *3D Robotic Mapping: The Simultaneous Localization and Mapping Problem with Six Degrees of Freedom*, 1st edn, Springer Publishing Company, Incorporated.

Pathak, S., Shantia, A. and Veronese, L.: 2020, Autonomous lane change. EP3667556.
URL: *https://register.epo.org/application?number=EP18212102*

Peters, J. and Schaal, S.: 2008, 2008 special issue: Reinforcement learning of motor skills with policy gradients, *Neural Netw.* **21**(4), 682–697.
URL: *https://doi.org/10.1016/j.neunet.2008.02.003*

Pu, Y., Gan, Z., Henao, R., Yuan, X., Li, C., Stevens, A. and Carin, L.: 2016, Variational autoencoder for deep learning of images, labels and captions, *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, Curran Associates Inc., Red Hook, NY, USA, pp. 2360–2368.

Quattoni, A. and Torralba, A.: 2009, Recognizing indoor scenes, *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 413–420. ID: 1.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T. B., Leibs, J., Wheeler, R. and Ng, A. Y.: 2009, ROS: an open-source robot operating system, *ICRA Workshop on Open Source Software*.

Ren, S., He, K., Girshick, R. and Sun, J.: 2015, Faster r-cnn: Towards real-time object detection with region proposal networks, *Advances in neural information processing systems*, pp. 91–99.

Reynolds, S. I.: 2002, *Reinforcement learning with exploration*, PhD thesis, University of Birmingham.

Rosenblatt, F.: 1958, The perceptron: a probabilistic model for information storage and organization in the brain., *Psychological review* **65**(6), 386.

Rumelhart, D. E., Hinton, G. E. and Williams, R. J.: 1986, Learning representations by back-propagating errors, *nature* **323**(6088), 533–536.

Rummery, G. A. and Niranjan, M.: 1994, *On-line Q-learning using connectionist systems*, Vol. 37, University of Cambridge, Department of Engineering Cambridge, England.

Sachin, R., Sowmya, V., Govind, D. and Soman, K. P.: 2018, Dependency of various color and intensity planes on cnn based image classification, *in* S. M. Thampi, S. Krishnan, J. M. Corchado Rodriguez, S. Das, M. Wozniak and D. Al-Jumeily (eds), *Advances in Signal Processing and Intelligent Recognition Systems*, Springer International Publishing, Cham, pp. 167–177.

Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J. and Davison, A. J.: 2013, Slam++: Simultaneous localisation and mapping at the level of objects, *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359.

Saputra, M. R. U., Markham, A. and Trigoni, N.: 2018, Visual slam and structure from motion in dynamic environments: A survey, *ACM Computing Surveys (CSUR)* **51**(2), 1–36.

Schaul, T., Horgan, D., Gregor, K. and Silver, D.: 2015, Universal value function approximators, *International Conference on Machine Learning*, pp. 1312–1320.

Schmidhuber, J.: 2015, Deep learning in neural networks: An overview, *Neural Networks* **61**, 85–117.
URL: *https://doi.org/10.1016/j.neunet.2014.09.003*

Schonberger, J. L. and Frahm, J.-M.: 2016, Structure-from-motion revisited, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113.

Shantia, A., Begue, E. and Wiering, M.: 2011, Connectionist reinforcement learning for intelligent unit micro management in starcraft, *The 2011 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1794–1801.

Shantia, A., Mulder, A., Wolf, B., Timmers, R., van der Mark, R., Sandor, L., Knigge, L., van der Struijk, S., Vienken, G., Bidoia, F. and Luneburg, N.: 2015, Team description paper 2015, *Technical report*, University of Groningen.

Shantia, A., Thorsten, W. and Wedel, A.: 2019, Method for predicting a change in the direction of travel of a vehicle. EP3543086A1.
URL: *https://register.epo.org/application?number=EP18163369*

Shantia, A., Timmers, R., Schomaker, L. and Wiering, M.: 2015, Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment, *International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1–7.

Shteingart, H. and Loewenstein, Y.: 2014, Reinforcement learning and human behavior, *Current Opinion in Neurobiology* **25**, 93 – 98. Theoretical and computational neuroscience.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. et al.: 2016, Mastering the game of go with deep neural networks and tree search, *nature* **529**(7587), 484–489.

SILVER project: 2016, *Supporting Independant living for the elderly through robotics*, SILVER consorsium, European Union.
URL: *http://www.silverpcp.eu/newsletter-july-2016/*

Sriman, B. and Schomaker, L.: 2015, Object attention patches for text detection and recognition in scene images using sift, *Proceedings of the International Conference on Pattern Recognition Applications and Methods*, pp. 304–311. ID: 1.

Sutton, R. S.: 1996, Generalization in reinforcement learning: Successful examples using sparse coarse coding, *Advances in neural information processing systems*, pp. 1038–1044.

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

Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A. A.: 2017, Inception-v4, inception-resnet and the impact of residual connections on learning, *in* S. P. Singh and S. Markovitch (eds), *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*, AAAI Press, pp. 4278–4284.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S. E., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A.: 2015, Going deeper with convolutions, *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, IEEE Computer Society, pp. 1–9.
URL: *https://doi.org/10.1109/CVPR.2015.7298594*

Tan, C. C. and Eswaran, C.: 2008, Performance comparison of three types of autoencoder neural networks, *Modeling and Simulation, 2008. AICMS 08. Second Asia International Conference on*, pp. 213–218. ID: 12.

Tesauro, G.: 1995, Temporal difference learning and td-gammon, *Communications of the ACM* **38**(3), 58–68.

Theodoridis, S. and Koutroumbas, K.: 2009, Chapter 11 - clustering: Basic concepts, *in* S. Theodoridis and K. Koutroumbas (eds), *Pattern Recognition (Fourth Edition)*, fourth edition edn, Academic Press, Boston, pp. 595–625.

Thrun, S.: 2002, Probabilistic robotics, *Communications of the ACM* **45**(3), 52–57.

Thrun, S. et al.: 2002, Robotic mapping: A survey, *Exploring artificial intelligence in the new millennium* **1**(1-35), 1.

Tombari, F., Salti, S. and Di Stefano, L.: 2010, Unique signatures of histograms for local surface description, *Computer Vision–ECCV*, Springer, pp. 356–369.

Torr, P. H. and Zisserman, A.: 1999, Feature based methods for structure and motion estimation, *International workshop on vision algorithms*, Springer, pp. 278–294.

van Elteren, T., Shantia, A., Neculoiu, P., Oost, C., Snijders, R., van der Wal, E. and van der Zant, T.: 2013, Team description paper 2013, *Technical report*, University of Groningen.

Vaughan, R.: 2008, Massively multi-robot simulation in stage, *Swarm intelligence* **2**(2-4), 189–208.

Vecerik, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., Heess, N., Rothörl, T., Lampe, T. and Riedmiller, M.: 2017, Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards, *arXiv preprint arXiv:1707.08817* .

Veeriah, V., Oh, J. and Singh, S.: 2018, Many-goals reinforcement learning, *CoRR* **abs/1806.09605**.
URL: *http://arxiv.org/abs/1806.09605*

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A.: 2008, Extracting and composing robust features with denoising autoencoders, *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, ACM, New York, NY, USA, pp. 1096–1103.
URL: *http://doi.acm.org/10.1145/1390156.1390294*

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A.: 2010, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *Journal of machine learning research : JMLR.* **11**(2), 3371–3408.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J. and Tsing, R.: 2017, Starcraft II: A new challenge for reinforcement learning, *CoRR* **abs/1708.04782**.
URL: *http://arxiv.org/abs/1708.04782*

Vuurpijl, L. and Schomaker, L.: 1997, Finding structure in diversity: A hierarchical clustering method for the categorization of allographs in handwriting, *Proceedings of the Fourth International Conference on Document Analysis and Recognition*, Vol. 1, IEEE, pp. 387–393.

Wan, E. A. and Van Der Merwe, R.: 2000, The unscented kalman filter for nonlinear estimation, *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373)*, Ieee, pp. 153–158.

Wang, J., Sun, K., Cheng, T., Jiang, B., Deng, C., Zhao, Y., Liu, D., Mu, Y., Tan, M., Wang, X., Liu, W. and Xiao, B.: 2020, Deep high-resolution representation learning for visual recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence* pp. 1–1.

Wang, S., Clark, R., Wen, H. and Trigoni, N.: 2017, Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks, *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2043–2050.

Watkins, C. J. C. H.: 1989, *Learning from delayed rewards*, PhD thesis, King's College, Cambridge.

Watkins, C. J. C. H. and Dayan, P.: 1992, Technical note q-learning, *Mach. Learn.* **8**, 279–292.
URL: *https://doi.org/10.1007/BF00992698*

Whelan, T., Johannsson, H., Kaess, M., Leonard, J. J. and McDonald, J.: 2013, Robust real-time visual odometry for dense RGB-D mapping, *2013 IEEE International Conference on Robotics and Automation*, pp. 5724–5731.

Wiering, M. A.: 1999, *Explorations in efficient reinforcement learning*, PhD thesis, University of Amsterdam.

Wisspeintner, T., Van Der Zant, T., Iocchi, L. and Schiffer, S.: 2009, Robocup@ home: Scientific competition and benchmarking for domestic service robots, *Interaction Studies* **10**(3), 392–426.

Witten, I. H.: 1977, An adaptive optimal controller for discrete-time markov environments, *Information and control* **34**(4), 286–295.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S. and Ba, J.: 2017, Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation, *Advances in neural information processing systems*, pp. 5279–5288.

Xiao, J., Hays, J., Ehinger, K. A., Oliva, A. and Torralba, A.: 2010, Sun database: Large-scale scene recognition from abbey to zoo, *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 3485–3492. ID: 1.

Zhou, T., Brown, M., Snavely, N. and Lowe, D. G.: 2017, Unsupervised learning of depth and ego-motion from video, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1851–1858.

Publications of the Author

In this thesis

- Shantia, A., Timmers, R., Chong, Y., Kuiper, C., Bidoia, F., Schomaker, L., Wiering, M.: 2021, **Two-Stage Visual Navigation by Deep Neural Networks and Multi-Goal Reinforcement Learning**. – *Journal of Robotics and Autonomous Systems*, Elsevier, In Press.
  URL: *https://doi.org/10.1016/j.robot.2021.103731*
- Shantia, A., Bidoia, F., Schomaker, L., Wiering, M.: 2016, **Dynamic parameter update for robot navigation systems through unsupervised environmental situational analysis**. – *Symposium Series on Computational Intelligence (SSCI)*, IEEE, pp. 1-7, (**Best Paper Award**).
- Shantia, A., Timmers, R., Schomaker, L., Wiering, M.: 2015, **Indoor localization by denoising autoencoders and semi-supervised learning in 3D simulated environment**. – *The International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1-7.

Patents

- Pathak, S. , Shantia, A., Veronese, L. : 2018, **Autonomous Lane Change**. – *European Patent Office*, EP3667556.
- Shantia, A., Thorsten, W., Wedel, A. : 2019, **Method for Predicting a Change in the Direction of Travel of a Vehicle**. – *European Patent Office*, EP3543086A1.

Other Publications

- Shantia, A., Bague, E., Wiering, M.: 2011, **Connectionist reinforcement learning for intelligent unit micro management in starcraft**. – *International Joint Conference on Neural Networks (IJCNN)*, pp. 1794-1801.
- van Elteren, T., Neculoiu, P., Oost, C., Shantia, A., Snijders, R., van der Wal, E., van der Zant, T.: 2013, **BORG - The RoboCup@Home team of the University of Groningen Team Description Paper**. – *International RoboCup@Home Competitions*, Eindhoven
- Shantia, A., Mulder, A., Wolf, B., Timmers, R., van der Mark, R., Sandor, L., Knigge, L., van der Struijk, S., Vienken, G., Bidoia, F., Luneburg, N.: 2015, **BORG - The RoboCup@Home team of the University of Groningen Team Description Paper**. – *International RoboCup@Home Open Competitions*, Iran
- Jansen, S., Shantia, A., Wiering, M.: 2015, **The neural-sift feature descriptor for visual vocabulary object recognition**. – *International Joint Conference on Neural Networks (IJCNN)*, pp. 1-8.
- Cnossen, F., Sweers, N., Shantia, A.: 2016, **Supporting medication intake of the elderly with robot technology: Poster and demonstration**. – *Poster session presented at Supporting health by technology VII*.
- Bidoia, F., Sabatelli, M., Shantia, A., Wiering, M., Schomaker, L.: 2018, **A Deep Convolutional Neural Network for Location Recognition and Geometry based Information**. – *In Proceedings of the 7th International Conference on Pattern Recognition Applications and Methods*, SciTePress, pp. 27-36.
- Küppers, F., Kronenberger, J., Shantia, A., Haselhoff, A: 2020, **Multivariate Confidence Calibration for Object Detection** – *CVPR Workshop of Safe Artificial Intelligence for Automated Driving (SAIAD)*, Accepted

Summary

The work conducted in this thesis contributes to the robotic navigation field by focusing on different machine learning solutions: supervised learning with (deep) neural networks, unsupervised learning, and reinforcement learning. In the thesis, different solutions are described to solve the following essential problems in robot navigation:

- Where is the robot?
- How should the robot move?
- Where should the robot move to?

In Chapter 2, we analyze a well established and frequently used robot navigation pipeline (Lu 2014). In this system, we create a map using a grid-mapping approach with a LIDAR sensor (Grisetti et al. 2007). The robot uses this map and localizes itself through adaptive Monte Carlo localization (Fox et al. 1999). The robot calculates its general path with the Dijkstra algorithm (Misa and Frana 2010) and drives through the environment by utilizing the dynamic window approach (DWA) (Fox et al. 1997) in which the method extracts trajectories by sampling velocity and acceleration space. Trajectories that cause collisions or deviate from the path are prohibited. To have smooth behavior, we should tune many parameters of the system; Notably, the resolution of the collision grid, the number of samples, and the sampling range for velocity and acceleration in longitudinal and angular space. At each point in time, the robot updates its knowledge regarding the surrounding obstacles, and in a predictive manner, forward simulates the possible trajectories using the given parameters. However, different environments and scenarios require different settings. The robot can move faster if the immediate surrounding is clear of obstacles, but it should carefully move when there are numerous obstacles around, or it is

reaching tight quarters. We propose a semi-supervised machine learning approach that can dynamically update the parameters based on the surrounding environment of the robot, which significantly improves the performance and safety of the robot. We first record the surrounding cost map of the robot that was traversing an office space in 3D simulation. Using a customized histogram of oriented gradient (HoG) feature (Dalal and Triggs 2005), we create a feature vector that can capture the complexity of the surrounding environment during robot movement (e.g., open hallways, tight corners, doors). Then, using agglomerative clustering, we create the dendrogram of the collected data. The results show that five clusters separate the data adequately. Visualizing the average image of all the members of the clusters reveals the different scenarios that a robot faces during its navigation. Finally, we select and tune separate parameters for each of these clusters. The results show that the robot successfully identifies the change in the surrounding, which results in a thirty percent improvement in the operation of real robot experiments.

In Chapter 3, we turn our attention toward the localization problem in robotics. We would like to investigate whether a neural network is capable of encoding the topological information of the surrounding environment. To this end, we gathered images from a 3D simulated environment. A large part of the dataset contains only unlabeled images of the environment while a smaller portion includes ground truth position labels. We trained and compared the results of a traditional multilayer perceptron, a stacked denoising autoencoder (SDA) (Vincent et al. 2010), a combination of an SDA with top five SIFT feature descriptors in the scene (Lowe 2004), and an MLP which uses histograms of oriented gradients (HoG) of the captured images as its feature vector. The results show that the SDA is capable of learning the position of the robot. The SDA average error for all the locations is approximately 10cm and 4 degrees. Our comparisons show that an SDA which is pre-trained on the unlabeled dataset performed significantly better than an MLP and HoG methods. Adding the top SIFT feature descriptors increases the localization performance, but computationally is not beneficial for a real-time robot.

We continue our investigation in Chapter 4 by testing the scalability of the SDAs and comparing them to various convolutional neural network (CNN) architectures. Therefore, we add a large 3D simulated environment to the experiments. We remove the pooling layers from the CNNs to bypass scale and rotation invariance, which is detrimental to a localization system. The experiment results showed that not only CNNs perform better in a smaller environment, but their performance also does not drop in larger ones. At the same time, the SDAs suffer significantly with more complex data. However, CNNs, similar to SDAs, perform poorly if the image only contains limited texture information. The output of the network becomes the mean of all the positions with the same sparse-texture view. In addition, we observe a

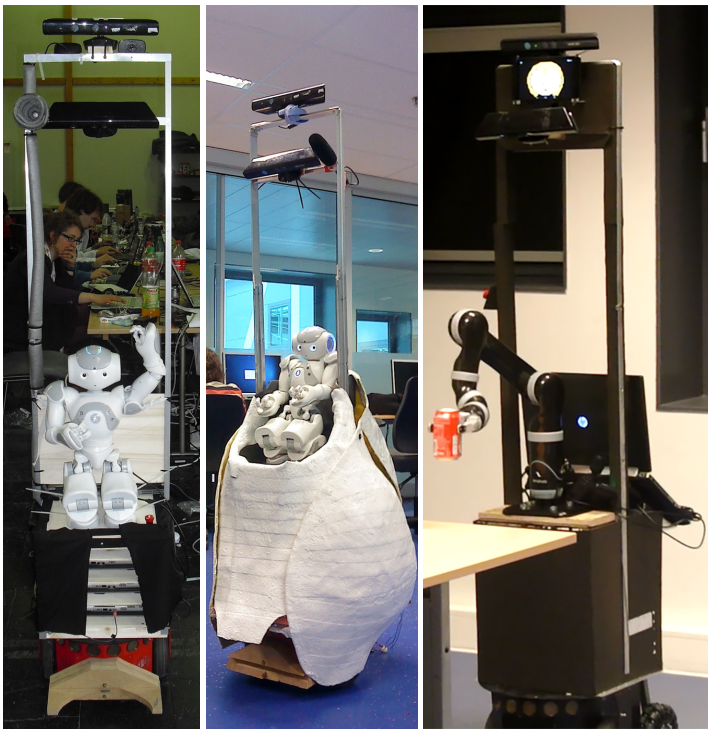


Figure 5.4: Evolution of our first robotic platform, Sudo, developed by the Borg team of the University of Groningen.

performance drop in long hallways, where the longitudinal movement of the robot does not substantially change the pixel values. On average, however, the results show that deep neural networks have strong potential in localization.

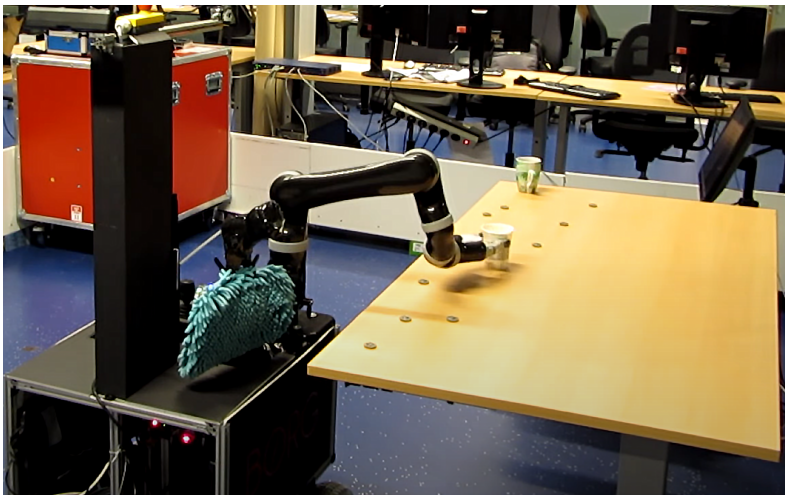
We then connected the position estimator CNN to a navigation system powered by reinforcement learning. We define a multi-goal reinforcement learning method in which the agent learns to reach multiple goals at the same time. By using the temporal-difference error of the goal states, the agent selects its next destinations based on the locations that it didn't traverse. Since learning can take a lot of time in a 3D simulated environment, we use transfer learning to speed up the convergence of the algorithm. First, the agent learns to reach all the goals in a maze created from the map of the environment. Then, we use this trained agent in a 2D physics simulator with ground-truth positioning to tackle problems that may arise from the robot control system during navigation. Finally, we test the agent in 3D simulation, in which the trained agent uses the CNN position estimator output to localize itself

in the environment. The results show that the trained agent can reach all the given goals in the 3D environment using some additional steps in comparison to the 2D simulator experiments. This deviation can be explained by the errors in the position estimation of the robot. The final system was able to learn to navigate to all goal positions using images taken in the 3D simulator. In future work it would be interesting to use the proposed system with a mobile robot to let it learn to navigate in the real world.

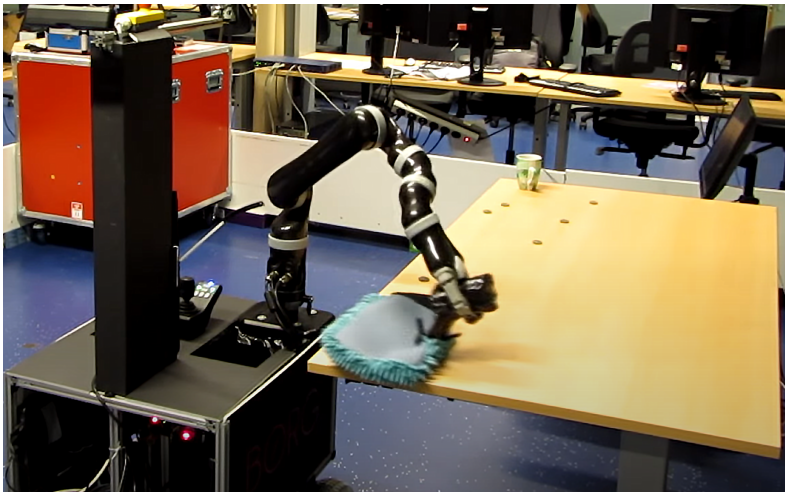
In this thesis, we proposed several methods to improve robotic navigation using a variety of machine learning approaches and demonstrated them through 3D simulation and real-world experiments. The research and development of the real-word domestic service robots were done in parallel with a team of enthusiastic students of the cognitive robotic laboratory of the University of Groningen throughout this project. The Borg team's main objective was to develop a domestic service robot that is capable of communicating with humans and carrying out complex tasks in the household, such as obstacle aware navigation, object detection and manipulation, and person recognition and tracking. We developed multiple prototypes during this project (Figure 5.4), learning from experience that we gathered during participation in RoboCup competitions (Wisspeintner et al. 2009), which assessed our robot's capabilities against standard benchmarks.

Our final prototype, Alice, can recognize humans, understand complex commands, navigate safely and efficiently in the environment, and detect and manipulate objects[1] (Figure 5.5). While a promise of a near-future with sophisticated service robots in each household is far fetched due to the complexity and price of these products, robotic solutions can be used to assist individuals in a day to day industrial and business environments. Increasing production and reducing the need for constant human monitoring and maintenance allows us to invest the saved time to develop new products and bring new ideas to the market.

[1] https://www.youtube.com/user/teamborgnl/videos



(a) Alice detecting and grasping a removable cup from the table.



(b) Alice using its cleaning tool to remove small debris from the table.

Figure 5.5: The cleaning operation of Alice.

Samenvatting

Het onderzoek in deze dissertatie draagt bij aan het werkveld van de navigatie van robots door te focussen op verschillende machine learning oplossingen: supervised learning met (diepe) neurale netwerken, unsupervised learning, en reinforcement learning. In de dissertatie worden verschillende oplossingen beschreven voor de volgende essentiële problemen in robotnavigatie:

- Waar is de robot?
- Hoe zou de robot moeten bewegen?
- Waar zou de robot naartoe moeten bewegen?

In hoofdstuk 2 analyseren we een beproefd en veelgebruikte robot navigatie pijplijn (Lu 2014). In dit systeem creëren we een rasterkaart met een LIDAR sensor (Grisetti et al. 2007). De robot gebruikt deze kaart en lokaliseert zichzelf met adaptieve Monte Carlo localisatie (Fox et al. 1999). De robot berekent zijn globale route met het Dijkstra algoritme (Misa and Frana 2010) en rijdt door de omgeving door middel van de dynamische window aanpak (DWA) (Fox et al. 1997), waarbij de methode trajecten selecteert door de snelheid en acceleratieruimte te sampelen. Trajecten die aanrijdingen veroorzaken of afwijken van het pad zijn niet toegestaan. Voor vloeiend gedrag zouden we veel parameters van het systeem moeten afstellen; voornamelijk de resolutie van het botsingsraster, de sample frequentie en het sample bereik voor snelheid en acceleratie wat betreft de hoek en afstand. Op elk moment werkt de robot zijn kennis bij over de obstakels om zich heen, en voorspelt de mogelijke trajecten met de gegeven parameters door deze te simuleren. Echter, verschillende omgevingen en scenario's behoeven verschillende instellingen. De robot kan sneller bewegen als de onmiddellijke omgeving vrij is van obstakels, maar het moet voorzichtig bewegen wanneer er veel obstakels in de buurt zijn of wanneer de

ruimte beperkt is. We stellen een semi-supervised machinaal-leren benadering voor, die de parameters dynamisch kan updaten gebaseerd op de omgeving van de robot, wat de prestaties en veiligheid van de robot significant zal verbeteren. Eerst wordt een kosten plattegrond van de omgeving opgenomen voor de robot die een kantoorruimte doorgaat in een 3D simulatie. Door een aangepast histogram of oriented gradient (HoG) te gebruiken (Dalal and Triggs 2005), wordt een vector gecreëerd die de complexiteit van de omgeving tijdens het bewegen van de robot kan vastleggen (bijvoorbeeld lege gangen, krappe hoeken, deuren). Vervolgens wordt er met agglomerative clustering een dendrogram gemaakt van de verzamelde data. De resultaten laten zien dat vijf clusters de data adequaat scheiden. Door het gemiddelde beeld van alle datapunten in de clusters te visualiseren, worden de verschillende scenario's die een robot tijdens navigatie tegenkomt onthuld. Ten slotte selecteren we voor elk van deze clusters de parameters en passen deze aan. De resultaten laten zien dat de robot de veranderingen in de omgeving succesvol identificeert, wat resulteert in een verbetering van dertig procent in het uitvoeren van echte robot experimenten.

In hoofdstuk 3 wordt aandacht geschonken aan het lokalisatie probleem van de robotica. We willen onderzoeken of een neuraal netwerk topologische informatie van de omgeving kan coderen. Hiervoor hebben we beelden van een 3D gesimuleerde omgeving verzameld. Een groot deel van de dataset bevat alleen ongelabelde beelden van de omgeving, terwijl een kleiner deel gelabelde informatie (ground truth positielabels) bevat. We trainden en vergeleken de resultaten van een traditioneel multilayer perceptron, een gestapelde ruisverwijderende autoencoder (SDA) (Vincent et al. 2010), een combinatie van een SDA met de top vijf SIFT feature descriptors in het veld (Lowe 2004), en een MLP die histograms of oriented gradients (HoG) gebruikt van de vastgelegde beelden als de feature vector. De resultaten laten zien dat de SDA in staat is om de positie van de robot te leren aan de hand van het beeld. De gemiddelde afwijking voor alle locaties is ongeveer 10 cm en 4 graden. Onze vergelijkingen laten zien dat een SDA die vooraf getraind is op de ongelabelde dataset, significant beter presteert dan de MLP met HoG methode. Door de top SIFT feature descriptors toe te voegen, verbetert de lokalisatie, maar dit is computationeel niet gunstig voor een real-time robot.

In hoofdstuk 4 zetten we ons onderzoek voort door de schaalbaarheid van de SDA's te testen en ze te vergelijken met verschillende convolutionele neurale netwerk (CNN) architecturen. Hiervoor voegen we grote 3D gesimuleerde omgevingen toe aan de experimenten. We verwijderen de "pooling" lagen van de CNN's om de schaal en rotatie invariantie te omzeilen, wat nadelig is voor een lokalisatiesysteem. De resultaten van de experimenten toonden dat niet alleen de CNN's beter presteren in een kleinere omgeving, maar dat de prestaties ook niet

verminderen in grotere omgevingen. Tegelijkertijd lijden de SDA's significant onder complexere data. Maar CNN's, net zoals SDA's, presteren slecht wanneer het beeld textuurloos is. De output van het netwerk wordt het gemiddelde van alle posities met hetzelfde textuurloze beeld. In aanvulling daarop observeerden we een prestatiedaling in lange gangen, waar de longitudinale beweging van de robot de pixelwaarden niet wezenlijk verandert. Gemiddeld echter, laten de resultaten zien dat diepe neurale netwerken grote potentie hebben in lokalisatie.

Vervolgens verbonden we de positie schattende CNN aan een navigatiesysteem dat aangedreven wordt door reinforcement learning. We definiëren een meervoudige reinforcement learning methode waarin de agent (de robot) tegelijkertijd leert om meerdere doelen te bereiken.

Door de leerfouten van de doeltoestanden te gebruiken, selecteert de agent de volgende bestemming op basis van de locaties waar het nog niet is geweest. Aangezien leren in een gesimuleerde 3D omgeving veel tijd kan kosten, gebruiken we transfer learning om de convergentie van het algoritme te versnellen. Eerst leert de agent om alle doelen te bereiken in een doolhof gemaakt van de kaart van de omgeving. Vervolgens gebruiken we deze getrainde agent in een 2D natuurgetrouwe simulator met ground-truth positionering om problemen op te lossen die zouden kunnen ontstaan door het robot controle systeem gedurende het navigeren. Ten slotte testen we de agent in een 3D simulatie, waarin de getrainde agent de CNN positie schatter output gebruikt om zichzelf te lokaliseren in de omgeving. De resultaten laten zien dat de getrainde agent alle opgegeven doelen in de 3D omgeving kan bereiken, met wat meer stappen vergeleken met de 2D simulator experimenten. Deze afwijking kan verklaard worden door de fouten in de positie inschatting van de robot. Het uiteindelijke systeem was in staat om te leren te navigeren naar alle doelposities, gebruikmakend van beelden gemaakt in de 3D simulator. In toekomstig onderzoek zou het interessant zijn om het voorgestelde systeem voor een mobiele robot te gebruiken, en het te laten leren navigeren in de echte wereld.

Acknowledgments

I clearly remember the days during my Master's studies when I worked as part of the Borg robotic team to build an automated platform that could act as a service robot. I remember the sadness when we failed to score points in our first international Robocup competition despite everyone's hard work, and I remember the joy when I received my acceptance letter from graduate school as one of the few candidates who received funding for their Ph.D. proposals. I have to admit that my Ph.D. journey took much longer than I anticipated. However, when I look back, this big project of my life was worth it with all its fluctuations. The ups and downs never stopped, nor the tremendous support of my family, friends, supervisors, and colleagues that I am ever thankful for.

Therefore, I would like to thank my supervisor and friend, Dr. Marco Wiering, for his constant support during this long journey. It was an honor being Marco's student. Our discussions during the coffee breaks or the gatherings at Noorderplantsoen were a source of inspiration for me. I would also like to thank my promoter, Prof. Lambert Schomaker, for his constant guidance and support during the project's length and his confidence in me to bring this project to an end. The valuable advice and feedback that I received in our regular meetings allowed me to plan my study and career path with a long-term vision. I am also grateful to the graduate school for funding this project and Dr. Sietse van Netten and the education department for giving me the opportunity to teach the robotics courses at our department.

However, completing this Ph.D. was not achievable without the commitment and hard work of my friends and colleagues of the Borg Robotic team. Thanks to Dr. Tijn van der Zant, we started the research on our service-robots back in 2010 from ground zero. Throughout the years, numerous enthusiastic students have joined us in this project and helped us in completing our robot prototypes, Sudo, and Alice.

My dear friends and partners, Ron, Egbert, and Christof, were part of the first group. It was a privilege to work with them as part of the robotic team and later as partners during our times at our startup, Enacer B.V. I will cherish the time that we spent together and the invaluable experiences that we gathered throughout the years. It is always comforting to know that you have friends who you can rely on.
I would also like to thank the rest of my team, Francesco, Rik, Anton, Ben, Sybren, Ayla, Noel, Marc, Yiebo, Cornel, Paul, and others for their hard work and significant impact on the development of Sudo and Alice. I would like to especially thank my good friend, Francesco, for our fruitful joint work on one of our academic papers and the great time we spent together at the university. Although I haven't managed to master the Italian Amatriciana and Carbonara recipes that he taught me, I have learned a couple of solid cocktails from him.

Although my time was mostly spent in the robot lab on the second floor, I have spent a great deal with my other department colleagues. I learned how to juggle balls from my office mate Jean-Paul and enjoyed coffee breaks with Charlotte, Harmen, Faik, Mahya, and Maruf, in addition to our fruitful discussions on research topics. I would also like to thank Elina for her constant support during my time at the university, especially the last three years, for providing me the necessary means to finish the project. I am glad that I was part of this big family, and Groningen will always remain my second home.

I would also like to take this opportunity to thank my dear friends, Mehdi Sadaghian, Elnaz, Ismaeel, Sahar, Mehdi Hatef, Parisa, Mayke, Mehdi Hamidi, Fahime, Saeedeh, Mehrsima, and Sara, who shared the happiness and hardships that I went through. I am blessed to have friends like you.

In the end, I would like to dedicate this thesis to my lovely wife Elham, who supported me every second of every day to bring this project to a finish, to my mom and dad, Aliyeh and Mehdi, to whom I owe every accomplishment of my life, and to my dear brothers and sister, Ali, Shahram, and Maryam, who I climbed on their shoulders to be able to reach this point of my life. I love you with all my heart.

Amirhossein Shantia
Karlsruhe
January 19, 2021