DEGREE PROJECT IN COMPUTER ENGINEERING,
FIRST CYCLE 15.0 CREDITS
STOCKHOLM, SWEDEN 2022

Autonomous agents in Industry 4.0

A self-optimizing approach for automated guided vehicles in Industry 4.0 environments

Leo Hjulström

Abstract

Automated guided vehicles are an integral part of industrial production today. They are moving products to and from shelves in storage warehouses and fetching tools between different workstations in factories. These robots usually follow strict pre-determined paths and are not good at adapting to changes in the environment. Technologies like artificial intelligence and machine learning are currently being implemented in industrial production, a part of what is called Industry 4.0, with the aim of increasing efficiency and automation. Industry 4.0 is also characterized by more connected factory environments, where objects communicate their status, location, and other relevant information to their surroundings. Automated guided vehicles can take advantage of these technologies and can benefit from self-optimizing approaches for better navigation and increased flexibility.

Reinforcement learning is used in this project to teach automated guided vehicles to move objects around in an Industry 4.0 warehouse environment. A 10x10 grid world with numerous object destinations, charging stations and agents is created for evaluation purposes. The results show that the agents are able to learn to take efficient routes by balancing the need to finish tasks as fast as possible and recharge their batteries when needed. The agents successfully complete all tasks without running out of battery or colliding with objects in the environment. The result is a demonstration of how reinforcement learning can be applied to automated guided vehicles in Industry 4.0 environments.

Keywords Automated guided vehicles, Industry 4.0, Agents, Reinforcement learning

Abstrakt

Automatiserade styrda fordon är en integrerad del av dagens industriproduktion. De flyttar produkter till och från hyllor i lagerlokaler och hämtar verktyg mellan olika arbetsstationer i fabriker. Dessa robotar följer vanligtvis strikta förutbestämda vägar och är inte bra på att anpassa sig till förändringar i miljön. Teknik som artificiell intelligens och maskininlärning implementeras just nu i industriproduktion, en del av det som kallas Industri 4.0, i syfte om ökad effektivitet och automatisering. Industri 4.0 kännetecknas också av mer uppkopplade fabriksmiljöer, där objekt kommunicerar sin status, plats och annan relevant information till sin omgivning. Automatiserade styrda fordon kan utnyttja de här teknikerna och kan dra nytta av självoptimerande metoder för bättre navigering och ökad flexibilitet.

Förstärkningsinlärning används i detta projekt för att lära automatiserade styrda fordon att flytta runt föremål i en Industri 4.0 lagermiljö. En 10x10 stor rut-värld med flertalet destinationer, laddningsstationer och agenter skapas i utvärderingssyfte. Resultaten visar att agenterna kan lära sig att ta effektiva vägar genom att balansera behovet av att slutföra sina uppgifter så fort som möjligt och ladda upp sina batterier när det behövs. Agenterna slutför framgångsrikt sina uppgifter utan att få slut på batteri eller att kollidera med föremål i miljön. Resultatet är en demonstration av hur förstärkningsinlärning kan tillämpas på automatiserade styrda fordon i Industri 4.0-miljöer.

Nyckelord Automatiserade guidade fordon, Industri 4.0, Agenter, Förstärkningsinlärning

Table of Contents

1 Introduction 3
1.1 Background 3
1.2 Problem 4
1.3 Purpose 5
1.4 Goal 5
1.5 Benefits, Ethics and Sustainability 5
1.6 Methodology / Methods 6
1.7 Stakeholders 6
1.8 Delimitations 6
1.9 Outline 7
2 Industry 4.0 and Multi-Agent Systems 9
2.1 Industry 4.0 9
2.2 Multi-Agent Systems 10
2.3 Automated guided vehicles 11
3 Reinforcement learning 13
3.1 Description 13
3.2 Theory 13
3.2 Neural Networks 14
3.4 Q-learning 16
3.5 Related work 17
4 Methodologies and Methods 19
4.1 Data collection 19
4.2 Data analysis 19
4.3 Quality assurance 20
4.4 Software engineering methods 20
4.5 Software 21
5 Requirements and System design 23
5.1 Interviews with stakeholders 23
52 Requirements 23
5.3 System design 24
6 Implementation 27
6.1 The Multi-Agent System 27
6.2 Environment 27
6.3 System Architecture 28
7 Results 31
7.1 Evaluation methods 31
7.2 Presentation of data 32
8 Conclusions and Future work 35
8.1 Discussion 35
8.2 Future work 36
References 38

1 Introduction

The industry today is currently in the process of implementing the fourth industrial revolution [1]–[3]. The economic impact of this fourth industrial revolution, or Industry 4.0, is expected to be huge, and it is currently a top priority for many companies, research centres, and universities [4]. Industry 4.0 is characterized as both the merging of virtual and physical production systems, known as Cyber-Physical Systems, and more intelligent automation. Achieved in part by the use of Cyber-Physical Systems and the implementation of technologies such as Artificial Intelligence, Industry 4.0 is leading to a new paradigm shift in how companies manufacture, produce, and distribute their products [1], [4], [5].

Within the scope of Industry 4.0, usage of Automated Guided Vehicles has led to increased efficiency and flexibility in the logistics sector of production [6], [7]. Still, the current approaches lack the flexibility required by Industry 4.0 standards [6], [8]. Usage of physical pre-defined paths for navigation does not meet the requirements of a flexible, decentralized, and scalable industry environment [6], [8]. Automated guided vehicles need to become “smarter” to handle the dynamic and data driven nature of Industry 4.0 factories [7], [9]. This means a more automated and self-sufficient approach. It includes using self-optimization, greater use of sensors and communication with the environment, and navigation without any artificial markers [5], [10].

1.1 Background

Automated guided vehicles (AGVs) are unmanned mobile robots used for transporting goods within a factory environment [9]. They most often rely on guided paths in the environment for navigation and use batteries as a source of power [11]. The batteries are managed at charging stations in the environment where a depleted battery is either swapped for a new one or is recharged while the AGV remains at the charging station [11]. Automated guided vehicles are a vital part of industrial production [8]. However, they need to adapt to the changes brought about by Industry 4.0 which demand more decentralized, flexible, and autonomous approaches to decision making and navigation in production environments [7], [8].

Industry 4.0 is the integration of new technologies such as Cyber-Physical Systems and cloud computing in industrial production [1], [3]. Cyber-Physical Systems are systems where software and hardware components are seamlessly integrated. They are also connected to and communicate with each other which is a key concept of Industry 4.0 [1]. The enormous flow of data from the communication between Cyber-Physical Systems, and advancements in technology like self-optimization, opens up new possibilities for flexible and autonomous solutions to be applied to automated guided vehicles [6], [9].

AGVs can be seen as agents, a term borrowed from the domain of artificial intelligence [12], and which are anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators (devices for control or movement of a system) [13]. A key aspect of agents is that they are autonomous, meaning that they act independently and can make their own decisions [12].

Intelligent agents are a specific type of agent. They are capable of flexible autonomous actions to meet their objective [14], which is crucial for AGVs in Industry 4.0 environments [7]. Another type of agent is a learning agent. These agents can operate in unknown environments and can become more competent than their initial starting knowledge was [14]. This is achieved by having the agents interacting with their environment and receiving rewards based on their actions [15]. The goal of these learning agents is to become self-optimizing - meaning being capable of adapting themselves to a task in order to operate with the greatest efficiency possible. In a dynamic environment such as a warehouse, this method, called reinforcement learning, is a good way of making agents learn how to carry out their tasks [16]. Reinforcement learning, which is an area of artificial intelligence, works by having agents learn from a series of rewards and punishments [13], [17]. For example, an agent performing tasks in a warehouse might receive rewards when tasks are completed, and punishments if it crashes or runs out of battery.

When a number of agents work together in an environment, it is called a Multi-Agent System. A Multi-Agent System is one way to automate the manufacturing process in an efficient and decentralized way [4]. It improves flexibility by enabling intelligent and decentralized decision making [6]. Multi-Agent Systems are currently being applied in many areas of industrial production [9], one of which is automated guided vehicles. As decentralized control of an Autonomous Guided Vehicle System is expected to lead to high flexibility and robustness [6], a Multi-Agent System approach is a suitable option for organizing and managing AGVs in Industry 4.0 environments [4].

1.2 Problem

Automated guided vehicles in industrial production today most commonly use pre-determined paths relying on artificial markers on the floor for navigation and pathfinding [6], [8]. Maintenance of the markers make this a costly approach. Layout changes and adjustments add to the costs. Furthermore, AGVs are out of service when markers are cleaned and/ or changed [6], [8], [9]. The movement/ motion-planning of automated guided vehicles are today also usually done using centralized methods [7], [8]. These methods are not suitable for future automated guided vehicle systems in Industry 4.0 environments, which demand flexible and decentralized approaches [6]–[8].

This research project focuses on a self-optimizing, trial-and-error approach, using reinforcement learning for decentralized control of AGVs. The problem statement is “How can a self-optimizing Multi-Agent System be implemented for automated guided vehicles transporting objects in an Industry 4.0 environment?”.

1.3 Purpose

The purpose of the thesis is to present a self-optimizing system for automated guided vehicles in Industry 4.0. The main task is to describe how a Multi-Agent System can use reinforcement learning to learn how to transport objects to different locations while not running out of battery.

1.4 Goal

The goal of the degree project is a 2D simulation of agents tasked with moving objects to different destinations; and verifying that the tasks are performed correctly and without any agent running out of battery. The result of the project is a self-optimizing Multi-Agent System using reinforcement learning to perform tasks of automated guided vehicles.

1.5 Benefits, Ethics and Sustainability

The main beneficiary of a more automated manufacturing process is the industry itself. The benefits of automation are many and include higher production rates and increased productivity, more efficient use of materials, and improved safety [18]. This increased productivity may be beneficial to society if essential products can be produced more efficiently. Other benefits are a shorter workweek for labour and worker safety, as automation can safeguard workers from the hazards of the factory environment

Sustainability is often defined as “development that meets the needs of the present without compromising the ability of future generations to meet their own needs” [19]. From this perspective, automation may have a positive impact as it can lead to more efficient use of materials and energy. On the other hand, it may also have a negative impact if the lowered cost- and increased efficiency of production means more goods can be produced, requiring more materials to be used.

Sustainability is also often discussed in terms of the three pillars of sustainability – economic, social, and ecological sustainability [20], [21]. Economic sustainability is either defined as economic development that does not come at the expense of natural and social capital, or simply as equated to economic growth. Ecological sustainability is when the ecosystem has time to regenerate its used resources. Social sustainability concerns the well-being, justice, and needs of individuals. This project may have a negative effect on social sustainability because of worker displacement. It is estimated that around 47 percent of all jobs in the USA are at risk of being automated away [22] and the negative effects on workers whose jobs have been automated away may range from undergoing a period of emotional stress, having to relocate to find work, to not finding new work at [18]

1.6 Methodology / Methods

The research methods and methodologies used in this project are explained by the *portal of research methods and methodologies* [23], which is a tool to support choosing the best-suited methods for a selected project. It works by starting at the outermost layer and working inwards, choosing a method/ strategy at every level of the portal. The first step is a choice between using a *Quantitative* or a *Qualitative research method*, or possibly a combination of both (triangulation). A *Quantitative research method* usually has large datasets and uses statistics to verify or falsify theories or computer systems' functionalities, while the *Qualitative research method* concerns understanding meanings and behaviors to reach theories or develop computer systems. As one moves through the portal, a choice of a research method is made. Examples of research methods are descriptive research, which studies phenomenon and describes characteristics of a situation (not the causes or occurrences of the situation), and analytical research, which tests and validates hypothesis using facts and collected information [23].

This degree project is a *Case study*, where a Multi-Agent System tasked with moving objects is developed using reinforcement learning and analyzed to answer the given problem statement. It is thus a *Qualitative research method*, and it belongs in the framework of *Applied research* methods since this framework involves developing technologies for answering specific questions and practical problems [23].

1.7 Stakeholders

This thesis is a part of a research project by the Center of Artificial Intelligence at the Arctic University of Norway. Their research focuses artificial intelligence and machine learning and how it can be applied to areas such as manufacturing (Industry 4.0), health, biology, and the energy sector [24]. The research project is about Robust systems, Robust decision-making in autonomous cyber-physical systems [25].

This thesis was suggested by a professor of the research center. The proposed degree-project was about using reinforcement learning for agents in Industry 4.0.

1.8 Delimitations

The Multi-Agent System developed is only tried in a simulation under ideal conditions. There is no evaluation of how it performs in the real word. How the agents' pickup, carry, and drop-off the objects they are tasked to move around is also not considered in this project.

Another delimitation is the size of the environment, including the number of agents, charging stations, and task-related destinations used. The Multi-Agent System is also only trained and evaluated in one specific setup of the environment, i.e., the layout of walls, charging stations, and task-related destinations.

An additional delimitation is the behavior of the agents' batteries. The batteries always use up the exact same amount of energy. Their performance does not decrease over time, which more realistic batteries would do, depending on how often and how fully they are recharged. How the charging stations work (i.e., if the depleted battery is swapped or recharged) is also not taken into consideration in this project.

1.9 Outline

In Chapter 2, on overview of Industry 4.0, Multi-Agents Systems, and automated guided vehicles is given. Reinforcement learning is discussed in Chapter 3, combined with related work and software. Chapter 4 gives a brief description of research methods and methodologies. Chapter 5 describes the requirements and the design of the system. The implementation of the system is described in Chapter 6. The last two chapters, Chapter 7 and 8, discusses the results and conclusions of the project respectively. Chapter 8 also discusses future work.

2 Industry 4.0 and Multi-Agent Systems

This chapter gives an overview of Industry 4.0, Multi-Agent Systems, and automated guided vehicles. Industry 4.0 is described in section 2.1. Multi-Agent Systems are presented in section 2.2, and automated guided vehicles in section 2.3.

2.1 Industry 4.0

The term Industry 4.0 is used to describe the fourth industrial revolution. The three prior industrial revolutions have been the invention of the steam engine and introduction of mechanical production facilities (1st industrial revolution), the discovery and usage of electricity (2nd industrial revolution), and digitalization and usage of IT systems and the Internet (3rd industrial revolution) [1], [4], [5].

The term “Industry 4.0” first appeared in 2011, at an industry fair in Hanover Germany, in an approach to strengthen the competitiveness of the German manufacturing industry [1], [4]. It was later adopted by the German government as one its key initiatives of its high-tech strategy [4]. The term also gained popularity in 2016 when the World Economic Forum (held in Davos) called the changes, that took place on the world industrial and digital scene, the fourth industrial revolution [1].

This fourth industrial revolution is about the integration of technologies such as the Cyber-Physical Systems and artificial intelligence in industrial production [1], [5]. Industry 4.0 can thus be seen as a merge of advanced technologies to further improve the manufacturing process [9]. Cyber-Physical Systems is a key concept of Industry 4.0. Cyber-Physical Systems are physical objects that are connected to- and communicate information to each other over networks [1]. The foundation of Industry 4.0 relies on the communication between these systems [1], [4]. Cyber-Physical Systems are with the help of smart sensors, actuators, and network connectivity, enabling real-time interaction and data sharing among different smart devices/objects [3]. The communication between Cyber-Physical Systems enables a paradigm shift from centrally controlled to decentralized production processes [4], which is another major concept of Industry 4.0 [5].

A form of Cyber-Physical Systems are intelligent agents. Agents, are computer systems that can decide for themselves what to do in order to meet an objective, and intelligent agents are agents that can operate in rapidly changing and unpredictable environments [26]. Intelligent agents are commonly referred to as autonomous entities with both reactive and proactive abilities [14]. Reactive means to perceive the environment, react to changes and act accordingly. Proactive means to look ahead into the future and take actions to produce meaningful results, or in other words, goal-directed behaviour [12]. Both of these abilities are crucial for intelligent agents, such as automated guided vehicles, to be able to effectively automate the manufacturing process.

A key enabler for the effective use of intelligent agents is the connected nature of Industry 4.0 factories. In an Industry 4.0 environment, smart products and devices will be able to process data and communicate. They will be able to identify themselves, share their status, history, and other information relevant to the environment [9]. The rapid and vast exchange of information enables intelligent agents to quickly and accurately react to changes in the Industry 4.0 environment. Connected goods and products that communicate information to the agents will lead to a more flexible manufacturing process and increased productivity [2], [9].

2.2 Multi-Agent Systems

An environment that contains multiple agents is called a Multi-Agent System [13]. A Multi-Agent System can be configured in numerous ways and the degree of collaboration and cooperation amongst the agents can vary. One possibility for Multi-Agent Systems is to only have one decision making agent [13]. In such a system, the decision-making agent makes plans for the other agents and tells them what to do. The other possibility is to have multiple decision-making agents in the environment. In such cases, the agents can either all share a common goal, or each pursue their own personal goals [13]. In the latter case, the goals can be non-aligned (as in a non-cooperative environment) or even diametrically opposed, as in chess.

When agents pursue a common goal, such as increasing productivity in a warehouse, they often face what is referred to as the coordination problem [13]; the agents need to ensure that they are pulling in the same direction and not accidently obstructing each other’s plans. This is often problematic when it comes to latency in communication, especially in environments where interaction between agents is about recognizing each other at the right time [14]. An example is physical agents (intelligent, autonomous agents with physical bodies), that need to recognize other agents in the environment using cameras and/ or sensors. If an agent’s computation takes too long and the communication with other agents is slow, collisions can occur. Timing errors and miscommunication can also result in agents carrying out tasks in the wrong order.

The usage of Multi-Agent Systems is popular in many areas today [14]. One area that is highly interesting and gaining popularity is Cyber-Physical Systems [14]. Autonomous, mobile agents can be developed as intelligent Cyber-Physical Systems capable of making their own decisions. They are useful in real-time, real-world environments like factory warehouses [14]. An advantage of Multi-Agent Systems is computational efficiency; agents can solve tasks in parallel and thus increase efficiency. Another advantage is that agents can cover for each other and help each other out if needed. This increases the reliability of the system and is especially true for decentralized systems which prevents “single points of failure” [14]. An automated guided vehicle system comprised of several intelligent agents can continue to operate even if one agent breaks.

2.3 Automated guided vehicles

An automated guided vehicle (AGV) is a driverless robot used for transporting products and materials [27]. They are most often used in industrial settings like manufacturing facilities or warehouses [4]. A number of these robots in the same environment, in control of transporting objects from given departures to given destinations, are referred to as an Automated Guided Vehicle System [27]. These Automated Guided Vehicle Systems were invented in the 1950s and are today utilized in almost all industries to automate and optimize material flow [10].

Many different approaches have been taken for navigation and pathfinding of automated guided vehicles. Approaches range from methods where the entire environment is first mapped and routes are set, to dynamic methods were the robots plan their paths based on information from the environment such as the robots' current locations and data from sensors. Planning using set routes is often favourable in more static (unchanging) environments. Dynamic planning, which is planning in real time based on information from the environment, is usually better for tasks and environments that change often. Industries today often take advantage of both methods; AGVs follow markers on the ground for simple navigation and use sensors to watch out for unexpected objects and obstacles in the way [28]. Commercial AGVs used in the industry today rely extensively on physical markers for guidance in the form of magnets, tapes or QR codes on the floor for pathfinding and navigation [6], [28].

Industry 4.0 have opened possibilities for automated guided vehicle systems to become more autonomous and more connected with their environment. Cyber-Physical Systems, low cost and intelligent sensor systems, and software developments run via the internet that can analyse these sensor systems is the technological basis for these new possibilities [10]. More extensive and efficient communication between smart systems and sensors within industry environments can be utilized by automated guided vehicles for more efficient navigation and decentralized decision-making [6], [7], [28]. Communication between Cyber-Physical Systems in the environment creates a huge amount of data. This data combined with increased processing power provided by Industry 4.0 provides an opportunity for AGVs to use reinforcement learning approaches, which is learning from experiences to make a sequence of decisions [17], to become more flexible and autonomous.

AGVs need to become "smarter" and more flexible to meet the requirements of future work in Industry 4.0 [4]-[6]. More flexibility means that they cannot be fixed onto guided paths using physical markers but instead be able to move more freely in the environment [8]. They need to be able to perceive the local environment and act properly to changes at a moment-to-moment basis [9]. AGVSs will take advantage of Industry 4.0 technologies like Cyber-Physical Systems for better communication with the environment, and artificial intelligence for decision making. Future AGVSs will become more decentralized and work in environments without physical markers [7], [8].

3 Reinforcement learning

This chapter gives an overview of reinforcement learning and Q-learning. It also discusses related work and the software that was used in the project. Section 3.1 and 3.2 introduces the area of reinforcement learning. Q-learning is presented in section 3.3. Section 3.4 discusses related work and the software and libraries used are presented in section 3.5.

3.1 Description

Reinforcement learning is an area of machine learning that deals with training for making a sequence of decisions [14]. It is a class of algorithms that aims at allowing an agent to learn how to behave in an environment. The agent receives rewards based on its actions in the environment. The goal of the agent is to perform actions that maximizes its rewards in the long run [15], [29]. Besides the agent and the environment, there are four main elements of a reinforcement learning system: a policy, a reward signal, a value function, and optionally a model of the environment [17].

A policy defines the behaviour of the agent. It is the core of a reinforcement learning agent in the sense that it is sufficient to determine the agent's actions. Policies may be simple functions or lookup tables or involve an extensive search process to find the right action to take. Policies may specify probabilities for each action [17].

After every action an agent takes, the environment sends a scalar value to the agent, called a reward [17]. The reward signal indicates what is good for the agent in an immediate sense. The value function specifies what is good for the agent in the long run, by determining the value of being in the current state. The value is the expected accumulated future rewards from a specific state, where state simply means the current available information to the agent about its environment. One can think of rewards as pleasure or pain and values as levels of satisfaction in the future. Rewards are given directly from the environment, but values must be estimated from sequences of observations by the agent.

Models of the environment are used by agents for planning. Models can predict future states and rewards and they can be used by agents to decide on actions by considering possible future situations before they are experienced. There exists reinforcement learning methods that use models, called model-based methods, and simpler methods that do not use any models, called model-free reinforcement learning methods [17]. Model-free methods do not involve any planning and are explicitly trial-and-error learners.

3.2 Theory

The problem of sequential decision making can be formally defined as a Markov decision process. It is a mathematical idealized form of the reinforcement learning problem [17]. It includes all its essential elements <S, A, P, R>: S is the state of the environment, A is the action taken by the agent, P is the state transition probability, and R is the reward received by the agent [17], [29], [30]. The state transition probability is the probability of the environment transitioning to a certain state by taking a certain action [30], e.g., the action to throw a die gives a probability of reaching a certain state.

The action an agent takes in a state is determined by its policy, π, which can either be stochastic or deterministic. A stochastic policy, π(a|s), has a probability of selecting each action a in a given state s [17]. A deterministic policy, π(a), defines an action for each state [29]. The goal for a Markov decision process is to find the optimal policy π∗ that maximizes the value function for all states [29]. The action-value function, $Q_\pi(s,a)$, gives the expected return/ value, of taking action a in a state s and following policy π thereafter. With the discount factor γ ∈ [0, 1], which trades off the importance of immediate and future rewards, the action value function is defined as [17], [31], [32]:

$$Q_\pi(s,a) \equiv \mathrm{E}[R_t + \gamma R_{t+1} + \ldots \mid S_0 = s,\ A_0 = a, \pi], \tag{1}$$

where E denotes the expected value and R_t is the reward at time step t. The expected return E is used since the transitions between states may not be deterministic but instead depend on a state transition probability P [17]. In such cases, following policy π will lead to an expected sum of rewards based on the random state transitions. The Q function expresses how "good" action a in state s is. Such a state-action pair is the total rewards obtained by taking action a, and then continuing taking actions in future states according to policy π [17].

A fundamental property of value functions is that they can be expressed in the form of the Bellman equation [29], [30], [32]:

$$Q_\pi(s,a) = \mathrm{E}[R_1 + \gamma Q_\pi(s', a')] \tag{2}$$

The Bellman equation expresses a relationship between the expected return of a state and the expected return of its successor states [17]. Many reinforcement learning algorithms iteratively update the Bellman equation (2) to estimate the action-value function [33].

3.3 Neural Networks

Neural networks are sets of connected artificial neurons formed as a network [14]. These networks are inspired by the structure of the human brain, even though the resemblance to real neural cells and structures is superficial [13]. An artificial neuron is a computational structure with a set of inputs and usually a single output. Each input has a parameter associated with it that controls the influence of that input in the network. Training a neural network is accomplished by modifying the input parameters [14].

Neural networks are a form of function approximators [17]. This means that they can infer functions from observations, or in other words, take examples from a function such as a value function, and attempt to generalize from the examples to construct an approximation of the desired function [17]. This is useful for difficult problems with complex data where a design by hand is impractical [14]. The objective of a neural network is to approximate a function for a data set as good as possible [17]. This means that the error between the approximated output value from the network and the expected value should be as small as possible. The error is usually calculated as the square of the difference between the two values [17]. A function that calculates the error is called a loss function and one of the most common loss functions is the mean squared error [14]. The mean squared error calculates the squared sum of the difference between the output value x_i and the expected value y_i and divides it by the number of observations n [14].

$$Mean\ squared\ error = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{n} \tag{3}$$

Artificial neurons are structured in layers where the neurons in each layer are connected to the neurons in the preceding and succeeding layers. A single-layer network consists of a single layer whereas more advanced neural networks that have several layers are called multi-layered neural networks [14]. When a network consists of more than one hidden layer (layers between the input and output layers), it can be called a deep neural network and is currently the most widely used approach for applications such as machine translation, visual object recognition, image synthesis, and speech recognition, as well as in reinforcement learning applications such as game playing and robot control [13].

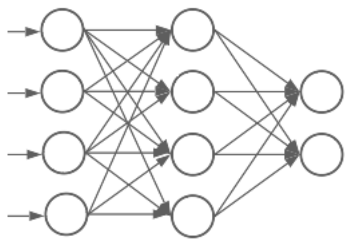
Figure 1. A neural network with one hidden layer.

An optimizer is an algorithm that is used to help to tweak neural networks by calculating how the weights of the neurons and the learning rate should be updated [34]. The learning rate is how quickly the network adapts to changing data [34]. If the learning rate is high, then the network will rapidly adapt to new data but tend to quickly forget old data. If the learning rate is small than the network will learn slower but be less sensitive to noise in the data. There exist multiple different optimizers. One example is the Adam optimizer, which is a popular state-of-the art optimizer algorithm. It is an adaptive learning rate algorithm [34]. This means that it reduces the learning rate by itself, for faster learning. An adaptive learning rate algorithm adapts the learning rate to help point the resulting updates to the network parameters in the optimal direction [34].

3.4 Q-learning

Q-learning is a popular and widely used reinforcement learning method [29], [31]. It combines different theories including Markov decision processes and Bellman equations to learn the action-value function $Q_\pi(s,a)$ [30], [35]. The learned action-value function directly approximates the optimal action-value function [17]:

$$Q_*(s,a) = \max_\pi Q_\pi(s,a) \tag{3}$$

Q-learning uses the action-value function $Q_\pi(s,a)$ combined with an ε-greedy policy to select actions [31]. An ε-greedy policy selects the action with the greatest estimated value in each state with a probability of ε. Otherwise a non-greedy action is chosen. Although choosing a non-greedy action reduces the expected reward, it helps to improve the estimates of the non-greedy actions [17]. Q-learning is one of the most effective reinforcement learning methods [31].

In 2015, a deep Q network (DQN), which is a deep neural network representing the Q function [13], surpassed the performance of all previous algorithms on a set of Atari 2600 games (classic games such as Breakout and Space Invaders) [33]. To address the insecurities and unstable learning of using a neural network to represent the action-value function the authors [33] proposed two key ideas: (1) – The use of an experience replay buffer. It removes correlations between samples, for a smoother and more independent distribution when training, by randomly selecting samples from the buffer. (2) - A target network. The target network uses the same parameters as the current Q function, but its parameters are only updated periodically - every n steps. It reduces the correlation with the target and prevents instability of changes made during training. The use of the replay buffer dramatically increased performance of the algorithm by reducing the correlations between the sample data [33].

In 2010, an algorithm called Double Q Network was proposed [36]. The Double Q Network algorithm learns two different value functions; one for selection and one for evaluation [36]. The idea behind this algorithm is that Q-learning uses the same values both to select and evaluate an action. This makes the algorithm choose overestimated values more often, leading to poor performance because of overestimated action values. The proposed solution was to decouple the selection from the evaluation by learning two different value functions.

In a paper from 2015, it was shown that the DQN algorithm suffers from severe overestimations in some Atari 2600 games [35]. The same paper also showed that the ideas behind the Double Q-learning algorithm can be generalized for large-scale function approximators. The proposed Double DQN (DDQN) algorithm is an algorithm that combines Double Q-learning with DQN and uses a dual network structure in the target Q-function. By combining Double Q-learning with DQN, the DDQN algorithm reduced the overoptimism of the action-values which led to more stable and reliable training. It significantly outperformed previous algorithms on several Atari 2600 games [29].

3.5 Related work

Tai et al. [37] used a deep reinforcement learning algorithm, which is a reinforcement learning algorithm that uses a deep neural network [13], called ADDPG (Asynchronous DDPG) for navigation of mobile robots. The algorithm is an extension of the DDPG (Deep Deterministic Policy Gradient) algorithm, which is a model-free deep Q learning algorithm [32]. It combines the use of replay buffers from DQN and deep neural networks with another type of reinforcement learning algorithm, called Actor-Critic methods [32]. They trained a mapless motion planner for the robots from scratch using this algorithm. Their findings show that a deep reinforcement learning algorithm for path planning can be applied in unseen real-world environments after being trained only in a virtual environment.

An algorithm based on Double Q-learning Network (DDQN) was used for collision avoidance and autonomous navigation of mobile robots in a paper by Xue et al. [38]. The system's input was the target position, state of the robot, and information about the obstacles such as other robots. The output was the direction of movement of the robot. The authors [38] showed that their method can be applied in practical applications with good results. The training time using the DDQN algorithm was also less than 10 % of using a DDPG method.

Lei et al. [39] also investigated the use of the DDQN algorithm for path planning and navigation in unknown environments. The results show that a DDQN method is flexible and capable of handling navigation in a new dynamic environment after training in different environments. This suggests that AGVs trained using reinforcement learning can generalize their knowledge to many different factory layouts. The results were verified by simulations and tests in the real world using a mobile robot with lidar technology.

Deng et al. [40] investigated the use of deep reinforcement learning for battery management of AGV systems. The battery management problem was formulated as a Markov decision process, and a novel continuous control reinforcement learning algorithm was presented for solving it. Extensive evaluation showed increased performance compared to rule-based policies, which are systems based on hand-crafted rules [40]. The results [40] indicates that AGVs can increase their efficiency by using self-optimizing approaches compared to rule-based policies for battery recharging planning.

The DDQN algorithm is used in this project. The DDQN algorithm is chosen as it has been shown to work well for automated guided vehicles for similar tasks [39]. Another advantage is that the DDQN algorithm has been shown to be faster at training compared to the DDPG algorithm [38].

4 Methodologies and Methods

This chapter gives a brief description of research methodologies and methods and explains which ones are used in this project. It also discusses the software development methods and contains a section describing the software used in the project. Section 4.1 describes data collection, section 4.2 data analysis, 4.3 quality assurance, 4.4 software engineering methods, and 4.5 presents the software.

4.1 Data collection

There exist different data collection methods. These methods, are as the name suggests, used to collect data for research [23]. Depending on if a qualitative or quantitative research method has been chosen, there are different methods to choose from. Data collection methods for quantitative research involves Observations, Experiments, and Questionnaire Case Studies [20]. Common data collection methods for qualitative research are Case Studies, Observations, Questionnaires, Language and Text, and Interviews [20]. The case study data collection method is used with the case study research method, and it is an in depth analysis of a single or small amount of participants [23]. Interviews are used to get a deeper understanding of a problem and capture participants' point of view [23]. There are three main types of interviews: structured - using predefined questions, semi-structured - predefined questions combined with follow-up questions, and unstructured - which is more like discussions [20].

Since this project uses the case study strategy, which is qualitative research, the data collection case study method is used. Data is collected from a test run of the system where important aspects of an automated guided vehicle system are measured. The project also involves some interviews with the stakeholders, which are explained more in chapter 5.

4.2 Data analysis

Data analysis methods are used to analyze and draw conclusions from collected data. Data analysis can involve processes such as cleaning, transforming, and modelling data [20]. Data cleaning is the process of detecting and removing errors, duplicates, and inconsistencies to improve the quality of the data [41]. Data transformation is done for the ease of comparison and interpretation. A common data transformation is to convert data to a normal distribution [42]. Data modelling is a way of describing, structuring and organizing data [43]. For qualitative research, the most used data analysis methods are Coding, Narrative Analysis, Hermeneutic, Semiotic, Grounded Theory, and Analytic Induction [20]. Hermeneutic and Semiotic data analysis methods are methods that analyses texts and documents [23]. Grounded Theory and Analytic Induction are iterative data analysis methods, that alternate between collections and analysis [20]. Coding is a data analysis method that names and labels concepts and strategies and applies statistics. Coding turns qualitative data into quantitative data [23]. Narrative Analysis is a data analysis method that concerns literary analysis and discussion [23].

In this project, the self-optimizing Multi-Agent System is evaluated, and data regarding its performance is modelled using coding. The data is analyzed to draw conclusions about if the developed system can be applied to real world automated guided vehicles in Industry 4.0.

4.3 Quality assurance

Quality assurance is the verification and validation of the research material [20]. Within qualitative research, both the quality of the data and the quality of the analysis should be considered [44]. Qualitative research should also be analyzed and discussed in terms of its validity, dependability, confirmability, transferability, and ethics [20]:

- Validity is the trustworthiness of the research. That it has been conducted according to existing rules, and that the results are correctly understood [20].
- Dependability is the reliability of the research. That the research will give similar results if conducted again by other researchers [45].
- Confirmability confirms that the research has been performed without the researcher's bias, own interests, or perspectives having affected the results [20], [37].
- Transferability is how applicable the results are in other contexts [37]. That the findings can be used by other researchers.
- Ethics is the moral principles when planning, conducting, and reporting the result of the research [20].

Quality assurance of the thesis is done through discussions of the research's validity, dependability, and transferability in Chapter 8 Conclusions and Future Work.

4.4 Software engineering methods

Software development consists of the following basic steps [46]: 1) Requirement analysis - understanding the problem, 2) Design - deciding a plan for the solution, 3) Coding - coding the planned solution, and 4) Testing the program. There is also a fifth step, maintenance, which consists of maintaining the program after deployment. Even though a software program does not "wear out" over time, newly discovered bugs and errors in the program needs to be resolved [46].

There are two main different software engineering methods, sequential approaches and incremental approaches [46]. The sequential approach, or waterfall model, is structured in one linear monolithic cycle. A project is structured in steps that are completed one after another. The incremental approach instead uses a series of mini-projects, or increments, to finish the work. An incremental model uses multiple small sequential cycles [45].

This project is of such sort that it requires an incremental approach. Small and simple task environments are created and tested and are then iteratively expanded upon adding more complexity.

4.5 Software

This section presents the software that is used in the project. The code for the system is written in Python [47] and it uses the TensorFlow [48] and TF-Agents [49] libraries.

Python is one of the most popular programming languages in the world and has been so for a long time [50]. It is a general purpose, high-level language with an easy-to-read syntax. Its ease of use and compact style speeds up development time compared to other languages [50]. Also, many high-level libraries and code platforms can be used in Python, including TensorFlow and TF-Agents.

TensorFlow is the most popular reinforcement learning library [51]. It is an open-source library originally created by researchers at Google. TensorFlow enables users to effectively program, train, and deploy neural networks and other machine learning models. The core of TensorFlow is the usage of tensors (vectors of higher dimensions) for fast and efficient numerical computations [51]. TensorFlow provides basic building blocks such as activation functions, different types of layers, and loss functions.

TF-Agents is a reinforcement learning library based on TensorFlow [34]. Its purpose is to make designing, implementing and testing reinforcement learning algorithms easier by providing well tested and flexible modular components [49]. TF-Agents implements many advanced state-of-the art reinforcement learning algorithms. The reinforcement learning algorithm is represented as an agent and TF-Agents provides standard implementations for a variety of agents, including DDPG, DQN, and DDQN agents [34].

In TF-Agents, an agent has two policies; one main policy used for evaluation and deployment, and one “collect policy” used for collecting data. This is because a TF-Agents system is usually split into two parts that run in parallel; one part that explores the environment and collects experiences, and one part that learns and updates the agent [34]. This architecture provides modular components and flexibility. It allows for exploration of multiple environments in parallel, taking advantage of multiple CPU cores. It also provides less correlated experiences by having them saved in a buffer and then sampled by the agent. The replay buffer also allows for an experience to be sampled multiple times, increasing the sample efficiency.

The TF-Agents library is customizable, and one can easily create one’s own neural networks or environments – which is done in this project. The most common workflow in TF-Agents is to code the environment in Python (by subclassing the PyEnvironment class) and then use an environment wrapper to convert it into a TensorFlow environment. This is done because implementing an environment in Python is easier, and the Tensorflow environment is more computationally efficient. The PyEnvironment class has two key methods that needs to be overwritten – the reset() method which resets the environment to its starting state, and the step(action) method that applies an action to the environment and returns an observation of the next state and a reward. The class also needs to define the shape, datatype, and ranges of the actions and observations for the environment.

5 Requirements and System design

This chapter describes the requirements given by the stakeholders and the design of the system chosen to meet the requirements. Section 5.1 describes the interviews with the stakeholders, section 5.2 the requirements, and section 5.3 explains the system design.

5.1 Interviews with stakeholders

During the course of the project there have been meetings with the stakeholders every two to three weeks, about ten meetings in total. During these meetings there have been discussions about the progress of the project and feedback has been given on both the thesis and the work. These meetings have also been used as semi-structured interviews where questions regarding the project have been prepared. These questions regard suggestions about what specific areas the work should focus on, to more detailed questions about the setup of the task environment.

The requirements on the system have also been discussed during the meetings. Some of the requirements were already set at the start of the project by the stakeholders. Those included the requirement to use a Multi-Agent System and the requirement of it operating in an Industry 4.0 environment. Other requirements were discussed and decided along the way as the focus of the project became clearer.

5.2 Requirements

The following requirements have been set by the stakeholders:

- The system should be a Multi-Agent System. It should be comprised of multiple agents working in the same environment to complete a set of tasks.
- The agents should be autonomous. They should be able to act independently and make their own decisions based on input from the environment.
- The agents should be able to move around freely. There should be no restrictions on their movement. Instead, they should learn for themselves how to move around appropriately in the environment.
- The system should operate in an Industry 4.0 environment. This means that the system should perform production related tasks, in this case moving tools/ products from one place to another. It also involves making use of the easy exchange of information between Cyber-Physical systems in the environment.
- Reinforcement learning should be used. The agents should learn how to complete their tasks via a trial-and-error approach using reinforcement learning.
- The agents should have a battery level. They should need to go to charging stations in the environment to recharge their batteries when needed.

5.3 System design

The first step in designing a system is to look for what is not stated as a requirement. This way no unnecessary work is spent on designing aspects of the system that are not needed. Firstly, there is no requirement for any visual processing of the environment. This means that there is no need to implement any graphical elements. In fact, the environment does not need to be a 3D environment at all. The requirements can be met by using a 2D environment where the space is divided up into squares, i.e., a grid world.

Secondly, even though the task of the agents is to move objects around, there are no requirements of how the pickup, carry, or drop-off of the objects are handled. The agents must simply move to the correct locations, or squares, and the handling of the objects is done automatically.

The requirements on the environment are that it should be an Industry 4.0 environment and that it should contain charging stations. The charging stations are represented as special "charging station squares" in the grid world. The Industry 4.0 aspect is met by having the task objects themselves communicate to the agents that they should be picked up, their current location, and where they should be dropped off. How this exchange of information is done is not relevant for the design of the system; the information is simply an input to the agents. The environment includes special "task squares", where the task objects are either dropped off or picked up by the agents.

The agents can move around freely in the environment. They can move to a new square either up, down, left, or right directly adjacent to their current square. The observational input from the environment to the agents is the status of these four squares. The status of a square can either be empty, a charging station, a task square, or a wall/ obstacle. The agents also receive input information about their current location, the location of a "task square" to either drop off or pick up an object at, and their current battery level.

All agents take one step in the environment together. No agent takes two steps while another only takes one. However, the agents do not move all at the same time. Instead, they take one step in the environment after each other. As the focus of the project is not on collision avoidance between objects in motion this design was chosen as it is easier to implement. The problem of collision avoidance with objects in motion requires the agents to either have more input information of the environment, rather than only the immediately adjacent squares, or involve communication with the other agents about their planned motion. The agents are perceiving objects that are moving around (changing location) in the environment but are not interacting with objects that are moving at the same time as they are.

The agents are trained using reinforcement learning, as stated by the requirements. They start off knowing nothing. By only using the input they are given they perform actions and explore the environment. Based on the rewards they receive they learn to operate in the environment. A task is represented to the agent as a location of a task square. When an agent arrives at its task square, that task is considered done and the agent receives a new task square as input.

The battery level of an agent is a value that decreases a certain amount each time the agent performs an action. When an agent arrives at a charging station, its battery is recharged. If an agent runs out of battery before arriving at a charging station in time, it will receive a negative reward and the environment will be reset. The agent will then continue to explore and learn from the environment, now hopefully being a little bit better at arriving at a charging station in time and not running out of battery.

There is no communication between agents in the environment explicitly coded in the project. As task scheduling is not a focus in this project, the communication between agents and objects about which agent should do which task, is not implemented. Also, the only information an agent gets about other agents is if one of the four observational input squares represents an agent occupying that square. In a real-world implementation of the system, this data will come from cameras or proximity sensors on the automated guided vehicle. There is also no communication between an agent and its battery implemented in the system. In a real-world implementation in an Industry 4.0 environment, a battery might be a connected system that communicates with the agent and other systems in the environment such as charging stations. In this project, the exchange of information to an agent from its battery is only represented as the integer value of the battery level given as input to the agent.

The final design has the following inputs to each agent: 1) The coordinates of the agent, 2) The coordinates of the target task square, 3) The agent's current battery level, and 4) An observation of the environment – the status of the four adjacent squares.

The output is one of the four possible actions the agent can take - either move up, down, left, or right.

The environment is a 10x10 grid world with three agents, three task squares, and two charging stations.

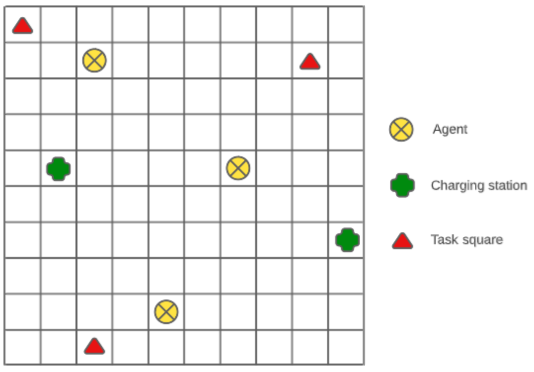

Figure 2. The environment

Figure 2 depicts the grid world environment. The three yellow circles are the agents, the three red triangles are the task squares, and the two green pluses are the charging stations. The agents are moving around in the environment from task square to task square, avoiding other agents, and moving to charging stations when they get low on battery. After recharging, the agents continue their tasks, moving to their target task destinations. In figure 3, the arrows indicate an example of agents moving in the environment.

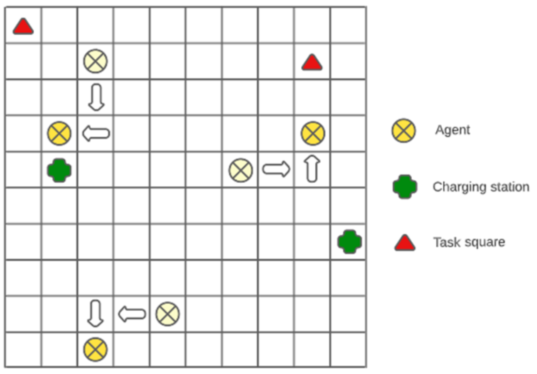

Figure 3. Agents moving in the environment

Figure 3 depicts the agents taking three steps each in the environment. The agents start at the more faintly colored yellow circles, the arrows indicate the steps that each agent takes, and the more colored yellow circles are the agents' final positions, after taking three steps. This is an example of how the agents can move around in the environment.

6 Implementation

This chapter explains how the system was implemented. Section 6.1 describes the system, section 6.2 how the environment was created, and section 6.3 the architecture of the system.

6.1 The Multi-Agent System

The system is a Multi-Agent System of intelligent agents. It uses reinforcement learning to teach the agents how to operate in the environment. Rather than trying to implement, debug, and fine-tune reinforcement learning algorithms and environments from scratch, it is often better and more efficient to use well-tested libraries like TensorFlow and TF-Agents. As Python supports both libraries and is an easy and time efficient language to code in, it is used in this project. Both the system and the environment are coded in Python.

The environment is created as a subclass of the PyEnvironment class. It is then turned into a TensorFlow environment, which allows for more efficient computations. This makes interacting with the environment faster which speeds up the training of the system. The reinforcement learning algorithm chosen for the system is the DDQN algorithm, as it has been shown to yield good results in complex and similar tasks [35], [38], [39]. The TF-Agents DDQN Agent, which implements the DDQN algorithm, is used for the system. In the project, the DDQN agent interacts with the environment and receives rewards. The rewards are used by the DDQN agent to calculate how to update the parameters of the DDQN algorithm. Updating the neural network input parameters and the Q-values is how the agent learns and becomes better at operating in the environment.

6.2 Environment

The PyEnvironment class needs to have the shape, datatype, and ranges of the actions and observations for the environment specified. The action is an integer value between zero and three representing one of the four actions an agent can take. The observation is a nine long integer vector containing the x- and y- coordinates of the agent and the target task square, the battery level, and a value representing the status of each of the four observation squares. An agent's battery level is reduced by one at every time step. When an agent arrives at a charging station its battery level is recharged fully.

The agents receive the following rewards from the environment:

- When an agent arrives at its target task square it receives a reward of +100.
- If an agent crash into a wall or another agent, it receives a negative reward of -70.
- If an agent runs out of battery it receives a negative reward of -100.
- When an agent arrives at a charging station it receives a reward based on how much battery it has left. If it has less than a third of its battery left it receives a reward of +50, and it has more than two thirds of its

battery left it receives a negative reward of -10, discouraging the agent from recharging when it is not needed.

- Otherwise, the agent receives a constant negative reward of -1. This is to encourage the agent to finish its tasks as fast as possible and not to wander around aimlessly.

The agents receive a certain number of tasks and interact with the environment until either all the tasks or done or an agent runs out of battery. This is referred to as an episode, and after each episode has ended the environment is reset, and a new episode begins. An instance of the grid world environment is wrapped in a time limit wrapper that resets the environment after a fixed number of time steps. This is to avoid an episode running for too long.

6.3 System Architecture

The system uses the DDQN agent provided by the TF-Agents library. The Q Network is configured with three hidden layers of a hundred neurons each. The choice of number of layers and neurons is one of the difficulties of using a neural network, as there is no standard way of arriving at a good configuration. Generally, two or three hidden layers is often enough for many problems [52]. Previously it was common to form the hidden layers like a pyramid with fewer and fewer neurons in each layer but nowadays it is more common to use the same number of neurons in each hidden layer. This is because it seems to perform just as good, or even better than using a pyramid structure, plus the fact that it is only one parameter to tune compared to one for each layer [34]. This exact configuration was decided upon by some trial-and-error on a small 5x5 environment. Some smaller networks with fewer neurons, including networks with only two hidden layers, were experimented with. The network seemed to learn quicker using three hidden layers of a hundred neurons each. There was also no noticeable increase in speed of learning when using more neurons. More than three hidden layers were not tried as this configuration performed well.

The DDQN agent is initialized with the Q Network and an optimizer. The Adam optimizer with a learning rate of 10^{-5} is chosen for the agent. It is used since it is a popular state-of-the art algorithm. It also requires little tuning of the learning rate as it is an adaptive learning rate algorithm. The Q Network is initialized with the environment's observation and action specifications, and the number of layers and neurons per layer.

In the beginning of training, an episode ends when an agent has completed two tasks, i.e., successfully reached its target task square twice. Every 30 thousand episodes, one more task is to be completed before an episode ends. After 120 thousand episodes, the number of tasks stop increasing and stays at six, for the remaining 80 thousand episodes of training. This gradual increase in number of tasks is implemented as it has a noticeable increase in the speed of which the agents learn, compared to when the number of tasks is set to six immediately. The number of tasks for each episode should not be too low since the agents must learn to take their battery levels into consideration. If the number of tasks is high, each episode will run for longer and the system will take a longer time to train.

During training, the average episode length and the rewards are monitored to gauge how well the system is learning. If the current configuration seems to be performing poorly, training is stopped and values such as neural network parameters and rewards are adjusted, using a trial-and-error approach.

An agent’s battery is coded as an integer value representing the current battery level. This value is given as an input to the agent.

7 Results

This chapter describes the results. Section 7.1 discusses evaluation methods and section 7.2 presents the results.

7.1 Evaluation methods

It is important to evaluate the performance of a system. The evaluation is used to compare it with other systems or to see if it passes some requirement criteria. For this thesis there were no requirements on the performance of the system. This means that it is harder to say whether the system has failed or accomplished its objective, but it can still be evaluated.

The most common way to evaluate the performance of an AI or machine learning system is to use a performance measure. There are many different performance measures depending on the type of problem. Some of the most common performance measures are the root mean square error for regression problems or a confusion matrix for classification problems. The mean squared error is also often used. It leads to the same result as using the root mean squared error but is faster to compute since it does not involve the square root operation. For a deep Q Network, the mean squared error can be calculated between the target and predicted Q values for the experienced state-action pairs.

However, it turns out that the mean squared error is not such a good indicator of a reinforcement learning model's performance. The agent might get stuck in a small region of the environment where the deep Q network starts overfitting (i.e., excessively memorizing at the expense of knowledge generalization). This means the error might go down, but the model will perform worse overall [34]. Instead, a better evaluation of a reinforcement learning model is the average return. The return is the sum of all rewards in an episode, and the average return is calculated over several episodes. This metric is used to evaluate the model's performance and track progress during training.

There is a problem with using the average return as the only final evaluation of the system. This is because the return is dependent on the rewards, which might not accurately represent the true goals of the system. If the rewards are not set correctly, the agents are not trying to learn their true objective. Therefore, it is not possible to determine if the system functions as intended based on the average returns alone.

The final evaluation of the system is a test run where the following metrics are evaluated; 1) the number of tasks completed, 2) the number of steps taken, 3) the number of crashes, and 4) the number of times an agent runs out of battery. The test run will have each agent complete 100 tasks each. The three agents all start with a battery level of ten and when an agent arrives at its target task square it is randomly assigned one of the two remaining task squares as its new target.

7.2 Presentation of data

The following figure depicts the average sum of rewards during training. The average sum of rewards is calculated over 100 episodes every 1000 episodes.

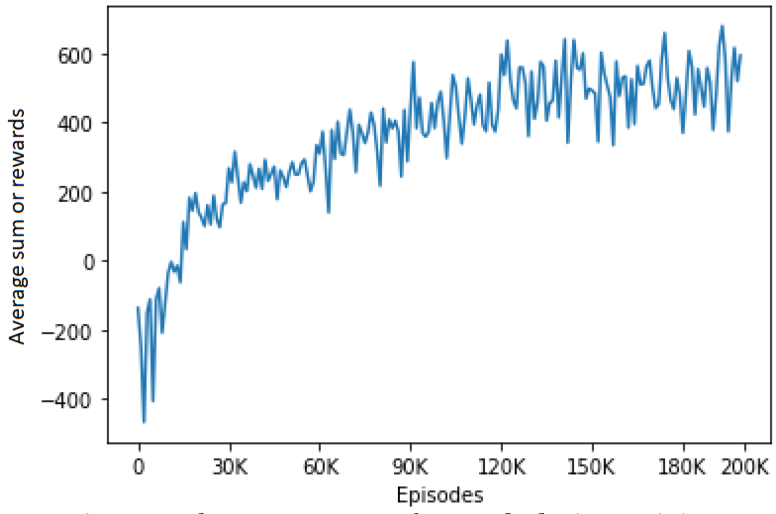

Figure 4. The average sum of rewards during training

The following figure depicts the average episode length (i.e., the average number of steps taken in an episode) during training. The average is calculated over every 200 episodes.

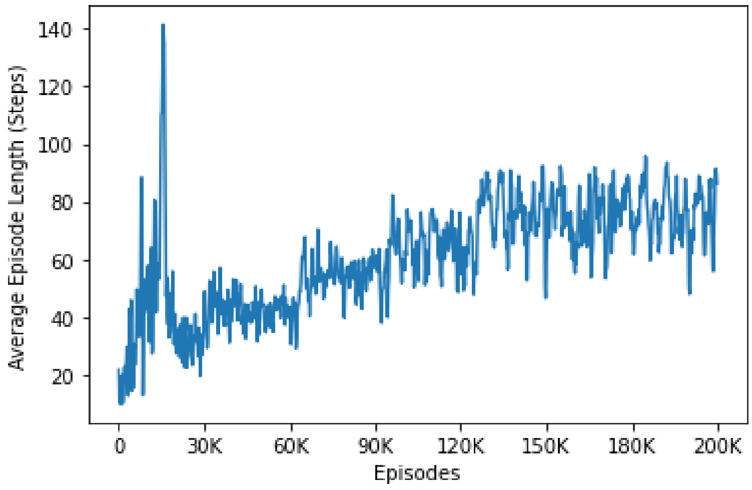

Figure 5. The average episode length during training

It is evident from figure 4 and figure 5 that the system has learned to optimize and improve itself during training. This is evident by the fact that both graphs seem to level out after 120 thousand episodes, which is when the number of tasks to be completed for each episode stops increasing. That the graphs level out indicates that the system has learned a good policy that is difficult to improve upon. That the graphs are oscillating up and down can most likely be attributed to the known problem of unstable learning, which virtually every reinforcement learning algorithm faces [34]. When the algorithm learns something new it may break what it has previously learned and thus lead to unstable learning.

The graph for the average number of steps during training in figure 5 follows the pattern one should expect. In the beginning, the agents do not know that they need to recharge their batteries, so the average episode length is low as an episode ends when an agent runs out of battery. The average then shots up, indicating that the agents have learned to go and recharge their batteries when needed. As the agents learn to complete their tasks and navigate more effectively, the average drops to around 30 steps right before the 30 thousand episode mark. At this point, the number of tasks to complete is still two, and by multiplying two with the average distance between tasks, which is eleven, plus the steps needed to go to a charging station – an average of 30 appears to be around the optimal average. The graph then makes stepwise jumps every 30 thousand episodes, corresponding well with the increase in number of tasks to be completed.

Table 1 Test run results

Collisions	0
Battery depletions	0
Total nr of steps	1 392
Tasks completed	300/300

The agents completed all 300 tasks of the test run successfully. There were no collisions with any objects (walls or other agents) and no agent ran out of battery. It took 1 392 steps per agent to complete all tasks, i.e., each agent moved 1 392 steps before all tasks were completed. The time it took to run the test is not relevant as the environment is very simplistic. The time it takes is no more than a few seconds at most.

The graphs for the average return and average episode length seems to indicate that the system is working as intended. The results from the test run also supports this conclusion. The test run results show that the agents were able to complete a hundred tasks each without ever colliding into anything or running out of battery. It is more difficult to determine how efficient the agents are navigating based on the results of the number of steps taken. By dividing the total number of steps taken with the number of tasks, then subtracting with the average distance between tasks, we get a value of 2.59 which is the average number of extra steps taken for each task. The facts that the agents need to recharge their batteries and take a few extra steps now and again to avoid collisions – around two and a half extra steps for each task suggest the agents are navigating relatively efficiently.

The results of the test run indicate that the system has learned to optimize itself to fulfil its goal, that is, to operate successfully as an automated guided vehicle system. The agents' behaviour has been optimized, using a trial-and-error/ reinforcement learning method, and they are completing their tasks without running out of battery or colliding, as the test run results show.

8 Conclusions and Future work

This chapter discusses the results of the project, states the conclusions, and discusses ideas for future work. Section 8.1 states the conclusions and discusses the project. Section 8.2 discusses future work.

8.1 Discussion

The purpose of this degree project is to present a self-optimizing method for automated guided vehicles in an Industry 4.0 environment. The Multi-Agent System developed demonstrates that agents can learn to move objects from one place to another by using reinforcement learning. As the results show, the agents could successfully perform the tasks both efficiently and without running out of battery. The results clearly show that reinforcement learning is a viable option for automated guided vehicles.

Industry 4.0 is incorporated in the project in two main ways. Firstly, the automated guided vehicles are Cyber-Physical Systems. They are communicating with the environment about the location of objects and where they should be dropped off. Furthermore, the idea is that the objects that the AGVs are moving are Cyber-Physical Systems as well. Even though this is not explicitly implemented in the project, the idea is that the objects are directly communicating with the automated guided vehicles in the environment, and it is how the AGVs get information about where to pick up and drop off objects. Secondly, the Multi-Agent System uses artificial intelligence (reinforcement learning) which is a part of Industry 4.0.

Regarding the validity of the thesis, it is clear from the results that a Multi-Agent System using reinforcement learning can be used for automated guided vehicles in an ideal environment such as a 2D grid world. The results of the test run measure the most important aspects of the automated guided vehicle system; its ability to complete tasks, not running out of battery, and not crashing into anything. The results measure what they should and are easy to understand which raises the validity. The fact that the system is not tested in the real world but only in a simulation impacts the validity negatively.

The dependability, or reliability, of the project is high. There is no obvious reason for why the results should not easily be reproducible. The project uses well-tested and reliable frameworks and libraries which is positive for the dependability. That the system is only tested on one environment configuration with a set number of agents, charging stations, and task squares negatively affects the reliability of the results.

The transferability of the results is limited to the demonstration of the possibility of using a Multi-Agent System using reinforcement learning for automated guided vehicles. The results might encourage more research into this topic but there is little more transferability than that.

The goal of the project is reached. A self-optimizing Multi-Agent System is designed and tested in a 2D simulated environment. The performance of the system is verified in a test run and the results show that the agents can perform the tasks successfully while not running out of battery.

It is unlikely that the results of the system would differ depending on the language and reinforcement learning libraries that are used. If the network and algorithms used are coded correctly and without any bugs, it should not matter what language or libraries the system is programmed with. The development might take more time and the code might be more prone to bugs depending on these choices, but the end result should be the same.

The size of the environment and the choice of number of agents, charging stations, and target destinations do impact the results. This is immediately obvious when imagining an environment of the smallest and simplest configuration possible. For example, it is of course easier for a single agent in a three-by-three grid world environment with only two task destinations and one charging station to operate successfully than in a bigger and more complicated setup. It is thus unlikely that the system would yield the same results in a bigger environment with more agents and target destinations.

The results suggests that an automated guided vehicle system can use reinforcement learning to learn how to move objects around in a factory environment. Although it is a significant difference between a simulated environment and the real world, an automated guided vehicle system can still be trained using a simulation of the environment. Ideally, automated guided vehicles would train directly in their real-world factory, but this is obviously unfeasible; the training would be way too slow, and it would involve a lot of expensive crashes. Instead, training can take place in a simulated environment using the exact layout of the real-world factory. The automated guided vehicles can then directly apply their learned knowledge to the real-world [37]. With this approach, automated guided vehicle systems can train on factory environments that have not yet been finished building, or at the same time as reconstruction of a factory takes place.

Some difficulties during development were the time it took to setup, configure, and get familiarized with the frameworks and libraries used, and the long time it took to train the system. The time it took to setup the programming environment for the libraries took a lot of time in the beginning of the project even though in the end it turned out to be relatively easy to get everything up and going. Despite this, the use of high-level and well-tested libraries most certainly saved huge amounts of development time overall. The time it took to train the system was another thing that slowed down the project. For future and more complex machine learning projects, cloud computing services aimed at speeding up training for machine learning problems might be a good idea to use.

8.2 Future work

More incorporation of the Industry 4.0 environment is suggested for future work. The charging stations can be implemented as intelligent software agents. They can communicate with agents about if they are occupied or not and get information about if agents are waiting in line to recharge. In the developed system the agents are fully recharged immediately which is not realistic. In a more realistic environment, where it takes time to recharge, the charging stations can make intelligent decisions about how long an agent should recharge for (if there are agents waiting), to make the system run as efficiently as possible. Communication between charging stations and agents, about occupied status and agents' plans to go and recharge, might enable charging stations to communicate amongst themselves and then inform agents of the most optimal charging stations to go to.

To give more validity to the results, tests of reinforcement learning methods for automated guided vehicles in real world environments is also suggested for future work.

References

[1] I. Karabegović, E. Karabegović, M. Mahmić, and E. Husak, ‘Implementation of Industry 4.0 and Industrial Robots in the Manufacturing Processes’, in *New Technologies, Development and Application II*, Cham, 2020, pp. 3–14. doi: 10.1007/978-3-030-18072-0_1.

[2] X. Liu-Henke, S. Jacobitz, M. Göllner, J. Zhang, S. Scherler, and O. A. Yarom, ‘Cyber-physical Industry 4.0 laboratory test field to simulate self-optimizing intralogistics’, in *2020 19th International Conference on Mechatronics - Mechatronika (ME)*, Dec. 2020, pp. 1–6. doi: 10.1109/ME49197.2020.9286614.

[3] R. Zhong, X. Xu, and L. Wang, *IoT-enabled Smart Factory Visibility and Traceability Using Laser-scanners*, vol. 10. 2017, p. 14. doi: 10.1016/j.promfg.2017.07.103.

[4] M. Hermann, T. Pentek, and B. Otto, ‘Design Principles for Industrie 4.0 Scenarios’, in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Jan. 2016, pp. 3928–3937. doi: 10.1109/HICSS.2016.488.

[5] H. Lasi, P. Fettke, H.-G. Kemper, T. Feld, and M. Hoffmann, ‘Industry 4.0’, *Bus. Inf. Syst. Eng.*, vol. 6, no. 4, pp. 239–242, Aug. 2014, doi: 10.1007/s12599-014-0334-4.

[6] Y. C. Dundar, ‘Dynamic path finding method and obstacle avoidance for automated guided vehicle navigation in Industry 4.0’, *Procedia Comput. Sci.*, vol. 192, pp. 3945–3954, Jan. 2021, doi: 10.1016/j.procs.2021.09.169.

[7] W. Xia, J. Goh, C. A. Cortes, Y. Lu, and X. Xu, ‘Decentralized coordination of autonomous AGVs for flexible factory automation in the context of Industry 4.0’, in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, Aug. 2020, pp. 488–493. doi: 10.1109/CASE48305.2020.9216961.

[8] ‘Automated guided vehicle systems, state-of-the-art control algorithms and techniques | Elsevier Enhanced Reader’. https://reader.elsevier.com/reader/sd/pii/S0278612519301177?token=5B6F017046579C568BABE6CC2777D3A522DEE4B3C60EA093AACF00CA7E61C2C75AAD77BF1C397F84B40E61D39FEC6F6C&originRegion=eu-west-1&originCreation=20220331164605 (accessed Mar. 31, 2022).

[9] J. Mehami, M. Nawi, and R. Y. Zhong, ‘Smart automated guided vehicles for manufacturing in the context of Industry 4.0’, *Procedia Manuf.*, vol. 26, pp. 1077–1086, Jan. 2018, doi: 10.1016/j.promfg.2018.07.144.

[10] G. Ullrich, *Automated Guided Vehicle Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015. doi: 10.1007/978-3-662-44814-4.

[11] ‘Increasing manufacturing flexibility through battery management of automated guided vehicles | Elsevier Enhanced Reader’. https://reader.elsevier.com/reader/sd/pii/S0360835218300330?token=425CF1AC2AC7B2AB610226A0956615743ADE0FE7DF4FAF1D714E1F24ADE5452948BC36E3D3C7F2A805A5216D58375BC3&originRegion=eu-west-1&originCreation=20220405173216 (accessed Apr. 05, 2022).

[12] S. Karnouskos, P. Leitao, L. Ribeiro, and A. W. Colombo, ‘Industrial Agents as a Key Enabler for Realizing Industrial Cyber-Physical Systems: Multiagent Systems Entering Industry 4.0’, *IEEE Ind. Electron. Mag.*, vol. 14, no. 3, pp. 18–32, Sep. 2020, doi: 10.1109/MIE.2019.2962225.

[13] S. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, 4th ed. Pearson Education, 2021.
[14] A. Håkansson and R. Hartung, *Artificial Intelligence Concepts, Areas, Techniques and Applications*. Studentlitteratur, 2020.
[15] M. Han, 'Reinforcement Learning Approaches in Dynamic Environments', p. 125.
[16] P. Thombre, 'Multi-objective Path Finding Using Reinforcement Learning', Master of Science, San Jose State University, San Jose, CA, USA, 2018. doi: 10.31979/etd.2ntb-4j8q.
[17] R. Sutton and A. Barto, *Reinforcement Learning An Introduction*, Second. The MIT Press, 2018.
[18] 'automation - Advantages and disadvantages of automation | Britannica'. https://www.britannica.com/technology/automation/Advantages-and-disadvantages-of-automation (accessed Mar. 28, 2022).
[19] 'What is Sustainability?', *UCLA Sustainability*. https://www.sustain.ucla.edu/what-is-sustainability/ (accessed Mar. 28, 2022).
[20] B. Purvis, Y. Mao, and D. Robinson, 'Three pillars of sustainability: in search of conceptual origins', *Sustain. Sci.*, vol. 14, no. 3, pp. 681–695, May 2019, doi: 10.1007/s11625-018-0627-5.
[21] 'Sustainable Development', *KTH*. https://www.kth.se/en/om/miljo-hallbar-utveckling/utbildning-miljo-hallbar-utveckling/verktygslada/sustainable-development/hallbar-utveckling-1.350579 (accessed Mar. 31, 2022).
[22] 'The future of employment: How susceptible are jobs to computerisation? | Elsevier Enhanced Reader'. https://reader.elsevier.com/reader/sd/pii/S0040162516302244?token=F3F8D231511B6FC13A45FFC95118B3C64977BFB5D36E0CB700BD4A2F9E955889FCCD75A4473B3D28AF16D1B42F9F127B&originRegion=eu-west-1&originCreation=20220505233014 (accessed May 06, 2022).
[23] A. Håkansson, 'Portal of Research Methods and Methodologies for Research Projects and Degree Projects', *Comput. Eng.*, p. 7, 2013.
[24] 'Center for Artificial Intelligence (CAI) | UiT'. https://en.uit.no/forskning/forskningsgrupper/gruppe?p_document_id=504980 (accessed Apr. 28, 2022).
[25] A. Håkansson, A. Saad, A. Sadanandan Anand, V. B. Gjærum, H. Robinson, and K. Seel, 'Robust Reasoning for Autonomous Cyber-Physical Systems in Dynamic Environments', *3966-3978*, 2021, doi: 10.1016/j.procs.2021.09.171.
[26] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.
[27] M. Rhazzaf and T. Masrour, 'Deep Learning Approach for Automated Guided Vehicle System', in *Artificial Intelligence and Industrial Applications*, Cham, 2021, pp. 227–237. doi: 10.1007/978-3-030-51186-9_16.
[28] M. Shneier and R. Bostelman, 'Literature Review of Mobile Robots for Manufacturing', National Institute of Standards and Technology, NIST IR 8022, May 2015. doi: 10.6028/NIST.IR.8022.

[29] M. Wiering and M. van Otterlo, Eds., *Reinforcement Learning*, vol. 12. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. doi: 10.1007/978-3-642-27645-3.

[30] K. Zhu and T. Zhang, 'Deep reinforcement learning based mobile robot navigation: A review', *Tsinghua Sci. Technol.*, vol. 26, no. 5, pp. 674–691, Oct. 2021, doi: 10.26599/TST.2021.9010012.

[31] H. Sun, W. Zhang, R. Yu, and Y. Zhang, 'Motion Planning for Mobile Robots—Focusing on Deep Reinforcement Learning: A Systematic Review', *IEEE Access*, vol. 9, pp. 69061–69081, 2021, doi: 10.1109/ACCESS.2021.3076530.

[32] T. P. Lillicrap *et al.*, 'Continuous control with deep reinforcement learning', *ArXiv150902971 Cs Stat*, Jul. 2019, Accessed: Apr. 05, 2022. [Online]. Available: http://arxiv.org/abs/1509.02971

[33] V. Mnih *et al.*, 'Human-level control through deep reinforcement learning', *Nature*, vol. 518, no. 7540, Art. no. 7540, Feb. 2015, doi: 10.1038/nature14236.

[34] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and Tensorflow*, Second Edition. O'reilly.

[35] H. van Hasselt, A. Guez, and D. Silver, 'Deep Reinforcement Learning with Double Q-learning', *ArXiv150906461 Cs*, Dec. 2015, Accessed: Apr. 05, 2022. [Online]. Available: http://arxiv.org/abs/1509.06461

[36] H. Van Hasselt, *Double Q-learning*. 2010, p. 2621.

[37] L. Tai, G. Paolo, and M. Liu, 'Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation', in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 31–36. doi: 10.1109/IROS.2017.8202134.

[38] X. Xue, Z. Li, D. Zhang, and Y. Yan, 'A Deep Reinforcement Learning Method for Mobile Robot Collision Avoidance based on Double DQN', in *2019 IEEE 28th International Symposium on Industrial Electronics (ISIE)*, Jun. 2019, pp. 2131–2136. doi: 10.1109/ISIE.2019.8781522.

[39] X. Lei, Z. Zhang, and P. Dong, 'Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning', *J. Robot.*, vol. 2018, pp. 1–10, Sep. 2018, doi: 10.1155/2018/5781591.

[40] Y. Deng, B. An, Z. Qiu, L. Li, Y. Wang, and Y. Xu, 'Battery Management for Automated Warehouses via Deep Reinforcement Learning', in *Distributed Artificial Intelligence*, Cham, 2020, pp. 126–139. doi: 10.1007/978-3-030-64096-5_9.

[41] E. Rahm and H. Hai Do, 'Data Cleaning: Problems and Current Approaches', *2000*, vol. 23, no. 4.

[42] M. S, 'Data transformation', *J. Pharmacol. Pharmacother.*, vol. 1, no. 2, pp. 126–127, Dec. 2010, doi: 10.4103/0976-500X.72373.

[43] H. K. Klein and R. A. Hirschheim, 'A Comparative Framework of Data Modelling Paradigms and Approaches', *Comput. J.*, vol. Volume 30, no. No. 1, 1987.

[44] J. Sargeant, 'Qualitative Research Part II: Participants, Analysis, and Quality Assurance', *J. Grad. Med. Educ.*, vol. 4, no. 1, pp. 1–3, Mar. 2012, doi: 10.4300/JGME-D-11-00307.1.

[45] G. Treharne and D. Riggs, 'Ensuring Quality in Qualitative Research', 2015, pp. 57–73. doi: 10.1007/978-1-137-29105-9_5.

[46] A. Mujumdar, G. Masiwal, and P. M. Chawan, ‘Analysis of various Software Process Models’. International Journal of Engineering Research and Applications (IJERA), 2021.
[47] ‘Welcome to Python.org’, *Python.org*. https://www.python.org/ (accessed Jun. 14, 2022).
[48] ‘TensorFlow’, *TensorFlow*. https://www.tensorflow.org/ (accessed Jun. 14, 2022).
[49] ‘TensorFlow Agents’. https://www.tensorflow.org/agents (accessed May 02, 2022).
[50] B. Lubanovic, *Introducing Python: Modern Computing in Simple Packages*. O’Reilly Media, Inc., 2014.
[51] B. Pang, E. Nijkamp, and Y. N. Wu, ‘Deep Learning With TensorFlow: A Review’, *J. Educ. Behav. Stat.*, vol. 45, no. 2, pp. 227–248, Apr. 2020, doi: 10.3102/1076998619872761.
[52] A. Burkov, *The hundred-page machine learning book*. Andriy Burkov, 2019.

TRITA - EECS-EX-2022:288