Data-driven generation of rules for ontology-based decision making systems in autonomous vehicles

Juha Hovi

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 31.5.2019

Supervisor

Prof. Ville Kyrki

Advisor

Prof. Ryutaro Ichise

Aalto University
School of Electrical
Engineering

Aalto University
School of Electrical
Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Juha Hovi

Title Data-driven generation of rules for ontology-based decision making systems in autonomous vehicles

Degree programme Automation and Electrical Engineering

Major Control, Robotics and Autonomous Systems **Code of major** ELEC3025

Supervisor Prof. Ville Kyrki

Advisor Prof. Ryutaro Ichise

Date 31.5.2019 **Number of pages** 70 **Language** English

Abstract

Autonomous vehicles can be controlled based on semantically abstracted knowledge of the surrounding environment of the vehicle. One such approach to knowledge based decision making is an ontology-based decision making system. This system requires a set of logical rules based on which the reasoning of correct action for each time instant is performed. Writing these rules by hand can prove challenging as covering all traffic scenarios produces a large number of rules to consider. However, these rules can also be obtained by learning them from data. In this work, creation of two datasets is covered as well as the generation of logical rules from these datasets. Two methods of learning the rules are implemented: association rule learning and a deep learning based approach. Both methods implemented produce a correct set of rules compliant with traffic rules.

Keywords Autonomous vehicles, ontology, rule-based machine learning, simulation, deep learning, association rule learning

Aalto University
School of Electrical Engineering

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi
Diplomityön tiivistelmä

Tekijä Juha Hovi

Työn nimi Tietovetoinen sääntöjen luominen ontologiaan perustuville päätöksentekojärjestelmille itseohjautuvissa ajoneuvoissa

Koulutusohjelma Automaatio- ja sähkötekniikan maisteriohjelma

Pääaine Säätötekniikka, robotiikka ja autonomiset järjestelmät **Pääaineen koodi** ELEC3025

Työn valvoja Prof. Ville Kyrki

Työn ohjaaja Prof. Ryutaro Ichise

Päivämäärä 31.5.2019 **Sivumäärä** 70 **Kieli** Englanti

Tiivistelmä

Itseohjautuvia ajoneuvoja voidaan ohjata semanttisesti abstraktoidun ympäristotiedon perusteella. Yksi tällainen tietoon perustuva lähestymistapa on ontologioihin perustuva päätöksentekojärjestelmä. Tämä järjestelmä vaatii loogisia sääntöjä, joiden perusteella oikea ajoneuvon ohjaus voidaan päätellä joka ajanhetkellä. Näiden sääntöjen luominen käsin on haastavaa, sillä liikenne toimintaympäristönä vaatii suuren määrän sääntöjä riittävän kattavuuden saavuttamiseksi. Nämä säännöt voidaan kuitenkin oppia kerätystä tiedosta. Tämä työ kattaa kahden tietoaineiston luonnin sekä loogisten sääntöjen luomisen näistä tietoaineistoista. Säännöt luodaan kahdella eri tavalla: assosiaatiosääntöjen oppimisella sekä neuroverkkoihin perustuvalla syväoppimisella. Molemmat lähestymistavat luovat sääntöjä, jotka ovat linjassa liikennesääntöjen kanssa.

Avainsanat Itseohjautuvat ajoneuvot, ontologia, säätöihin perustuva koneoppiminen, simulaatio, syväoppiminen, assosiaatiosääntöjen oppiminen

Contents

Symbols and abbreviations

Symbols

t	The set of transactions containing a specific itemset
T	The set of all transactions
X	An itemset
Y	An itemset

Operators

$supp()$	Support of an itemset
$conf()$	Confidence of a rule
$\Longrightarrow$	Logical implication
$\wedge$	Logical conjunction
$\cup$	Union of sets
$\lvert \cdot \rvert$	Number of members in a set

Abbreviations

ADAS	Advanced driving assistance systems
ANN	Artificial neural network
API	Application programming interface
A*	A-star search algorithm
CNN	Convolutional neural network
C-SPARQL	Continuous SPARQL
GPS	Global positioning system
LiDAR	Light detection and ranging
MV3D	Multi-view 3D
RBML	Rule-based machine learning
RDF	Resource description framework
ReLU	Rectified linear unit
RGB	Red, green, blue
ROS	Robot operating system
SLAM	Simultaneous localization and mapping
SPARQL	SPARQL protocol and RDF query language
SSD	Single shot multibox detector

1 Introduction

Autonomous driving and advanced driving assistance systems (ADAS) are currently one of the most topical fields of research relating to vehicles in both academia and industry. One approach for determining correct actions for the vehicle is an ontology-based decision making system [1]. The decision making system in question controls the vehicle on a high level by using a knowledge base constructed from information obtained both beforehand and in real time. The rules dictating the behavior of the system are logical rules comprised of a body in the form of a conjunction containing observations from the environment and head containing vehicle actions that are to be taken if the body evaluates to true.

In addition to ontology-based systems, other approaches to decision making have been developed [2]. One such approach is an end-to-end planning system which directly maps sensor input to actions [3]. Another method is a probabilistic approach to finding an optimal high-level policy through behavior prediction for highway ramp merging cooperation [4]. In addition to end-to-end learning, another example of a data-driven approach is a system for lane change decision making based on a support-vector machine using features such as relative position and relative velocity [5].

In the work by Zhao et. al. [1] the rules are created manually by an expert writing them by hand. The goal of this thesis is to generate these rules in an automated fashion by taking advantage of data-driven machine learning techniques. For this to be possible, suitable data must be available. Thus, this work is divided into two major parts: dataset creation and rule generation. In essence, the goal of this work is to study approaches and methods relating to the creation of these logical rules in order to:

- determine how a semantically abstracted dataset describing traffic situations can be built.
- identify suitable approaches to generate logical rules from the created dataset.

The complete decision making system is visualized in Figure 1. For the goals of this work, the important parts of the decision making system to consider are the query engine, the knowledge base, and the reasoner. The query engine is based on continuous SPARQL (C-SPARQL) [1] which is a sliding window based approach to handling queries over continuous streams of timestamped resource description framework (RDF) graphs [6]. This dictates the format of the rules generated in this work to be as described earlier. The knowledge base contains both static and dynamic information in a high level semantically abstracted format. This means that the information is in a format that describes its meaning. The static information can be, for example, map data such as streets and intersections as well as their connections. Dynamic information refers to knowledge such as vehicle and traffic light states. Additionally, the rules are stored in the knowledge base.

The rule reasoner in Figure 1 accesses the knowledge base for information on current vehicle state and surrounding enviroment as well as the logical rules for

determining the correct action. The reasoner takes advantage of semantic web rule language (SWRL). SWRL is a standardized format in which the logical rules can be displayed in [7]. Ontologies refer to graphs detailing the vocabulary for the RDF triples and the relations between the terms in the vocabulary. A specific set of ontologies created for advanced driving assistance systems are considered in this work. These ontologies are called ADAS ontologies [8].

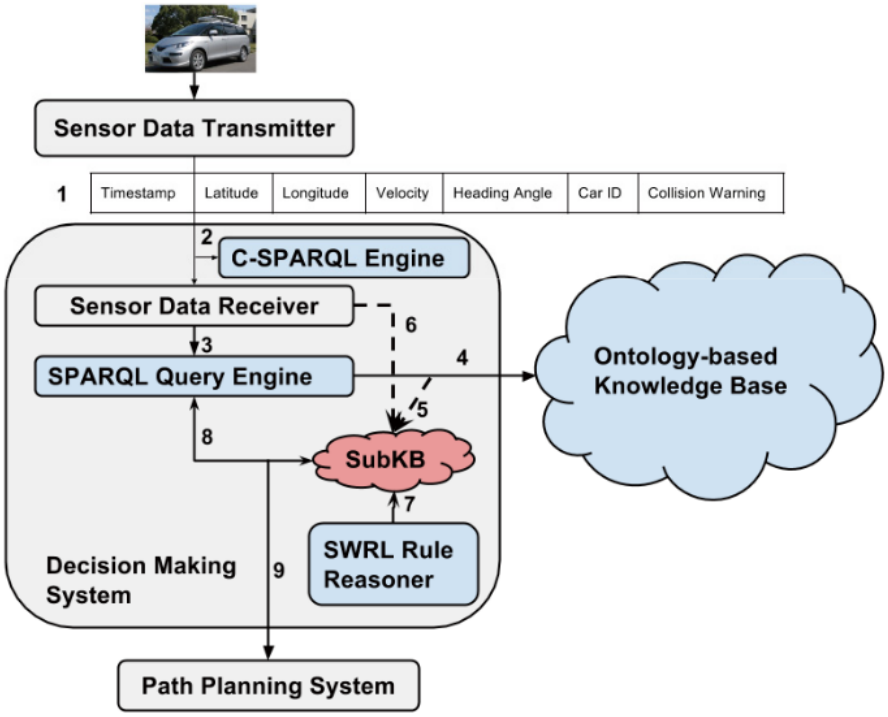

Figure 1: Diagram of the ontology-based decision making system [1].

The work of Zhao et. al. focuses on evaluating their approach in uncontrolled intersections. Similarly, the main objective for this work is to focus on scenarios taking place in uncontrolled intersections. Figure 2 visualizes an example scenario in an uncontrolled intersection. In this situation vehicle C crosses the intersection first as it is going straight. Next, vehicle A moves as the narrow road can not fit two vehicles side by side. Finally, vehicle B can turn onto the narrow road. The goal of the generated set of rules is to capture this kind of behavior for different situations to allow the reasoning system to make the correct decisions. Instead of covering the whole problem with one solution, having a set of rules allows combining multiple partial solutions into one complete solution. Uncontrolled intersection is chosen as the point of focus as it allows for more diverse rules than intersections with traffic lights where rules usually boil down to the traffic light dictating the action directly. While traffic as a whole contains a much wider field of scenarios, focusing on a specific scenario allows laying the foundation for later expansion of the findings of this work.

For the first major part of this work, the data generation, there are different

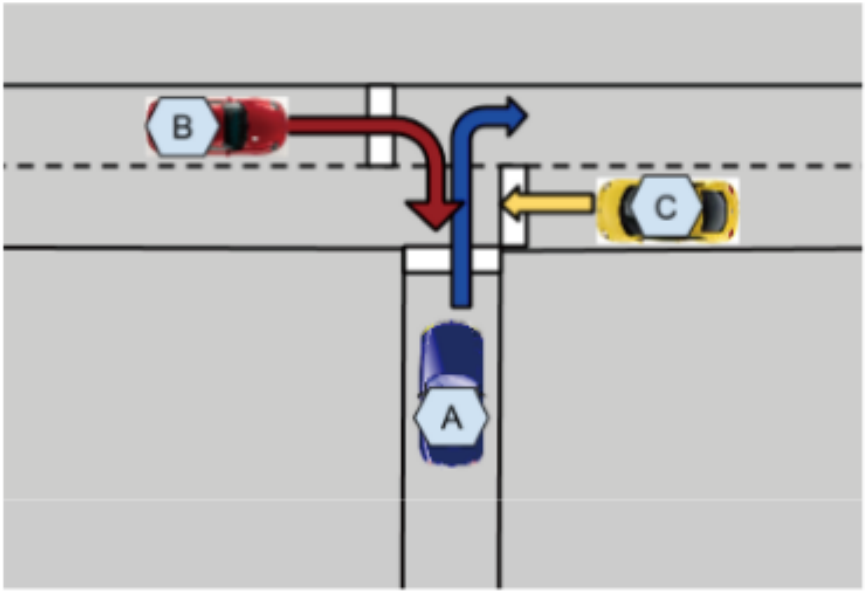

Figure 2: Uncontrolled intersection with a narrow road [1]. For safe intersection crossing, C crosses the intersection first later followed by A and finally followed by B.

approaches to generating the required data using simulated environments. A method most closely resembling a real world system is a method where data gathering originates from simulated sensors attached to the vehicle. The sensor data is then processed to detect the state of the intersection. Another approach is to retrieve higher level data directly from the simulator and bypass the sensor data processing. Due to the amount of work involved in processing the sensor data, this work focuses on retrieving the data directly from the simulator. However, through all of the data gathering process real world applications are considered in order to determine if all the data gathered would realistically be available from processing of the sensor data of a real world autonomous vehicle.

The simulator of choice is the CARLA simulator. CARLA is an Unreal Engine based simulator specifically designed for autonomous driving research [9]. CARLA was chosen due to its specific design purpose being suitable for the work as well as the promising and active development of the simulator.

The second major part of the research is the generation of rules from the generated data. Different approaches of data-driven machine learning methods are explored, considered, and compared as means to achieve reliable and correct results. The choice of methods is based on the eventual goal of generating rules from real world data. That is, a dataset gathered in real world traffic. One method used is association rule learning that is widely used in, for example, recommendation systems in online services [10]. This method lends itself well naturally to finding patterns in the data gathered in the first part of the work due to the decisions made concerning the format of the gathered data. Another method is a neural network based deep learning approach as neural networks can be trained to predict target values from given input

patterns [11]. In other words, neural networks can be used to extract underlying patterns in the data.

Section 2 introduces fields of research relevant for this work as well as discusses previous research within those fields. Section 3 introduces different methods applied in this work for each part of the work. Section 4 offers deeper insight into the experiment setup. Section 5 introduces, discusses, and evaluates the results achieved. Section 6 discusses different directions for future research aiming to expand upon the foundation laid in this work. Finally, Section 7 provides a brief summary of this work.

2 Background

This work combines knowledge and methodology from multiple fields of research. Some fields, such as data mining, have been thoroughly studied and have expansive history of implementations spanning decades. Other fields have shorter history of implementations. An example of such field is deep learning which uses artificial neural networks (ANN) with one or more hidden layers. The theory of ANN has a long history as concepts of logical neurons and neural networks were introduced in 1943 [12]. However, practical implementations of deep networks with multiple layers have more recently been enabled by increases in widely available computing power. This section provides an overall view of the fields most immediate to the methods applied in this work.

2.1 Autonomous vehicles

Autonomous vehicles are complex cyberphysical systems. As such the research relating to them is varied. Traditionally, the general components of autonomous driving can be divided into scene recognition, path planning, and vehicle control [13]. These same components can also be referred to as sensing, computing, and actuation [14] as more general descriptions of all components. These components can each be considered their own field of research within the field of autonomous vehicles and each of them consists of multiple subcomponents. Sensing deals with receiving and processing raw sensor data into a more usable form. The second step, computing, includes aspects such as further processing the sensor data, making decisions, and planning the vehicle motion. Finally, actuation deals with lower level control of actuators to control the vehicle according to the plans and decisions made in previous steps. One complete set of software containing functionality for all these steps of autonomous driving is Autoware [13]. This section introduces previous work in the relevant fields of autonomous vehicle research that this work is based on. First, usage of ontologies in autonomous vehicle applications is introduced. Then, a more traditional approach implemented in Autoware is discussed in detail. Finally, a completely different approach of end-to-end learning is introduced.

2.1.1 Ontology-based systems

Ontologies are frameworks for representing knowledge by describing concepts or classes and the relationships between those classes [15]. They provide a vocabulary to represent semantically abstracted knowledge [16]. A simple ontology, for example, can contain information on different vehicle types and their relations with each other. Such an example is shown in Figure 3 which displays a part of ADAS Car ontology [8]. Arrows are drawn from child classes to their parent classes. Thus, the arrows can be considered as a relation "*is of type*". This example contains the information that bus, truck, bicycle, and motorcycle are all types of vehicles. From these, bus and truck are specifically automobiles which is a subclass of vehicles. Ontology as a framework is general and can be used to represent many different types of information. Unlike

the example in Figure 3 ontologies in general can be complex depending on the application.

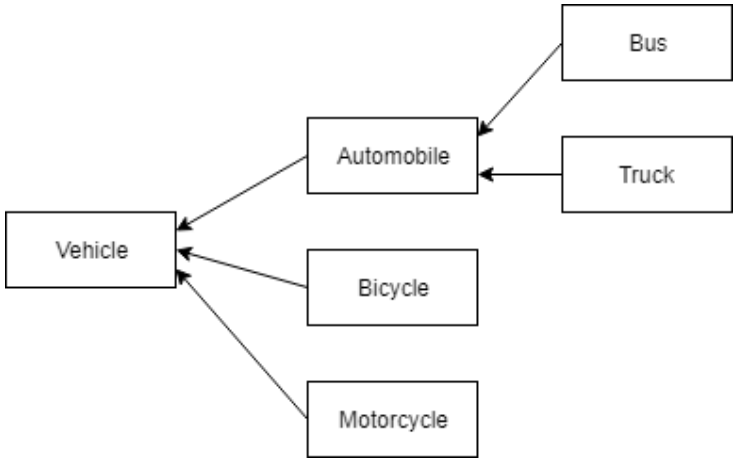

Figure 3: Part of ADAS Car ontology [8]. Relationships between different example vehicle types are shown. Rectangles represent different classes while arrows represent relations and point from child class to parent class.

The computing component of autonomous driving introduced in Section 2.1 contains a subcomponent of decision making. As with all components, multiple approaches are under active research. One of the approaches for a decision making system is a system based on ontologies introduced in 2017 by Zhao et. al. [1]. This component of autonomous driving is the most immediately related to this work as this thesis aims to better enable the use of ontology-based decision making systems. Other approaches of applying ontology-based systems to autonomous driving include, among others, using ontologies in navigation planning [17], obstacle avoidance [18], and intelligent speed adaptation systems [19].

A main contribution of ontology-based systems is the exploitation of knowledge. In this context, knowledge refers to the available information being understood in a way humans understand and process that information. The goal of such systems is to apply humanlike understanding of situations to machine systems. This, however, is problematic as human understandable formats are not inherently machine understandable. Thus, further processing of knowledge through computations still remains a challenging problem [1]. The other components of autonomous driving are still relevant for ontology-based systems, however the details within those components can differ from other approaches. Specifically sensor data processing differs significantly as the data is processed into form matching the concept of knowledge. The benefits of systems utilizing knowledge include aspects such as more straightforward information sharing, the process as a whole being more understandable by humans, and natural expandability through addition of vocabulary. These upsides do not come without their counterparts. For example, dealing with missing or partial data can pose significant problems that other approaches may be more suitable to dealing with. In addition, currently ontology-based reasoning in real time applications such as autonomous driving poses challenges that are critical to solve for any practical

real world system. Clement et. al. have made progress towards ontology-based real time reasoning by utilizing decision tables [16].

2.1.2 Autoware approach

There are myriad of other approaches to autonomous driving in addition to the ontology-based approach introduced in Section 2.1.1. One such approach is a traditional stack of software handling different components of the autonomous vehicle operation. An example of an implementation like this is Autoware. Autoware is a collection of autonomous driving software [13] taking a more traditional approach while applying state of the art techniques. While Autoware offers multiple options for most of its functions for the user to choose from, Figure 4 presents the general flow of software controlling an autonomous vehicle. Everything starts from sensing. Sensing includes receiving data from one or more sensors and preprocessing it into a form better suited for the next parts. A concrete example of this step is receiving light detection and ranging (LiDAR) and red, green, and blue (RGB) camera data and preparing calibration matrices to enable the projection of LiDAR data onto the RGB camera feed later on in the computing step.

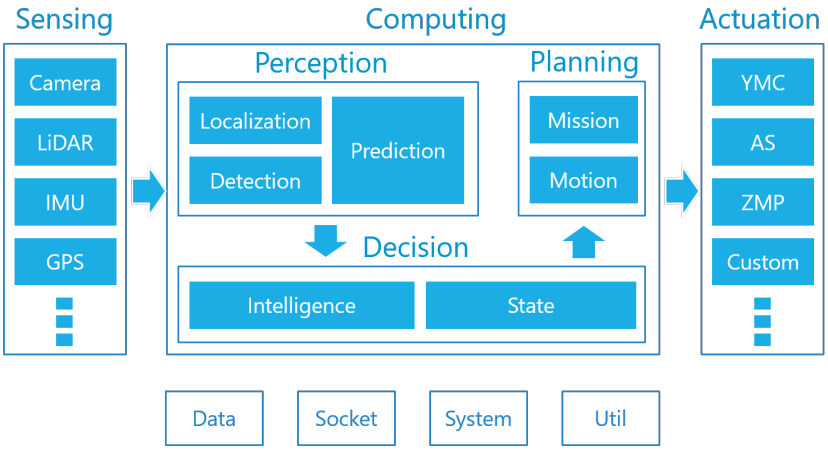

Figure 4: Overview of autonomous driving implemented with Autoware [14]. Autoware divides the process of autonomous driving into three main components: sensing, computing, and actuation.

The computing step includes three subcomponents: perception, decision making, and planning. Perception is the act of forming an understanding of the surroundings of the vehicle through processing of sensor data received from the sensing step. This includes processes such as creating a map, localizing the vehicle within a map, detecting both static and dynamic obstacles, and predicting their trajectories. For example, vehicles can be detected and localized through the following process. First, vehicles are detected within the RGB camera feed by using a single shot multibox detector (SSD). SSD is a neural network based method for detecting objects in images [20]. Through its fast performance compared to other state of the art neural network based methods, SSD is particularly suitable for real time applications such

as autonomous driving. Simultaneously, the LiDAR data is processed with the intent to find objects through Euclidean clustering. This method finds clusters of points in the LiDAR data by grouping points together based on their Euclidean distance [21]. Next, the calibration of the RGB camera and the LiDAR can be used to match the detected vehicle to a cluster in the LiDAR point cloud data through the use of MV3D algorithm. MV3D stands for Multi-View 3D networks which is a neural network based approach to fusing LiDAR pointclouds and RGB images to predict oriented bounding boxes for detected objects [22]. Through this process, vehicles in the surroundings of the autonomous vehicle can be detected as vehicles and their location relative to the autonomous vehicle can be extracted.

Next step within computing is decision making. This step uses the information gained through the previous steps to determine the current state of the vehicle and make decisions to guide the vehicle towards a desired goal state. This decision is then forwarded to the last subcomponent of computing: planning. Planning includes functionality for both global and local path planning. Global planning includes the general route the vehicle takes to reach its goal from its current position. Local planning includes aspects such as avoiding obstacles, maintaining safe operation, and planning overtaking and similar actions. One option within Autoware is to use Hybrid-State A* algorithm to generate a path plan. As its name suggests, Hybrid-State A* is a variant of the well known A* search algorithm applied to the 3D kinematic state space of the autonomous vehicle [23].

The last step of the overview in Figure 4 is the actuation. Actuation part of Autoware outputs velocity, angular velocity, wheel angle, and curvature [14]. These are target values intended to be used for controlling the vehicle. The implementation of the low level control is heavily dependent on the application and as such is not handled directly by Autoware.

The approach to autonomous driving implemented in Autoware has multiple benefits. The modular and open source nature of the software allows development and modifications of a single part without changing the whole software stack. Furthermore, this architecture is both customizable and tunable to a finely detailed level. Due to this, it is difficult to identify downsides with this approach as every module can, in theory, be replaced with a new state-of-the-art approach. Autoware currently leaves out some desirable features such as simultaneous localization and mapping (SLAM) as well as loop closure in its mapping feature. However, the software is under active development and as such these issues could addressed in the future.

2.1.3 End-to-end learning

One completely different approach to the ontology-based system introduced in Section 2.1.1 and the more traditional approach discussed in 2.1.2 is end-to-end learning. End-to-end as a term describes something that addresses all steps of a process [24].

One end-to-end approach to autonomous driving was introduced in 2016 by Bojarski et. al. from the NVIDIA Corporation [3]. This end-to-end solution covers the whole process of autonomous driving by utilizing deep learning. A convolutional neural network (CNN) was trained to map the input of raw RGB image pixels directly

to steering commands with relatively promising results [3]. Convolutional neural networks are a type of ANN well suited for image recognition applications. A deep convolutional neural network was shown to be able to map image input to output steering angles closely resembling those applied by a human driver [25]. A more general approach aimed towards utilization of uncalibrated crowdsourced datasets was introduced by Xu et. al. in 2017 [26].

End-to-end approach is interesting due to its relative simplicity. One neural network trained with data gathered in real traffic was shown to be able to control the vehicle successfully in diverse conditions [3]. However, this approach struggles with myriad of problems related to practical applications. The system replaces all the components of autonomous driving introduced in Section 2.1 with a single neural network. This results in, for example, not having a dedicated planning unit. Thus, while the system can drive successfully on a road in terms of lane following and obstacle avoidance, moving to a correct goal road in an intersection cannot be taken care of by this system in a robust way. Naturally, this approach can be augmented with supporting systems. However, in this case the end-to-end nature and the simplicity of the system is lost to some degree.

2.2 Rule-based machine learning

The goal of the rule learning is to process the dataset generated in the simulation and extract patterns from the dataset into logical rules. In other words, the goal is to identify and learn rules dictating the vehicle behavior recorded in the dataset. A field within machine learning focused on learning of this nature is called rule-based machine learning (RBML).

For a machine learning method to fall under RBML, the solution or output of the method is comprised of rules typically of form {*If: Then*}. Examples of methods for RBML include learning classifier systems (LCS), association rule learning, and others such as artificial immune systems. A common factor for these approaches is that they solve the problem by having a set of rules where each rule covers a part of the problem. In contrast, many machine learning systems aim to find a single model to cover the whole problem. Essentially, RBML learns or extracts patterns in the data. As with all learning, no rules of significant value can be learned if the data is of too random nature and contains no patterns. [27]

In this work two approaches were implemented. First, the above mentioned association rule learning was implemented and its results were evaluated. The second approach is a neural network based deep learning approach to extracting patterns in a dataset. This deep learning approach does not directly fall under any of the three categories introduced above although it has some similarities to LCS.

2.2.1 Learning classifier systems

LCS are machine learning methods for discovering rules of form {*If: Then*} within a dataset. LCS employ genetic algorithms [27] and as such are a form of genetics-based machine learning [28]. While many different algorithms have been developed, they

share a common structure and as such they can be grouped together as LCS. This framework includes aspects corresponding to two biological metaphors: evolution and learning [28]. In more practical terms, LCS employs a genetic algorithm and a learning mechanism [27]. Thus, there are two main phases to rule learning with an LCS: discovery and learning.

The goal of an LCS is to set up and evolve a set of classifiers where each classifier offers a partial solution to the problem. This way, the set of all classifiers, in optimal case, offers a solution to the problem as a whole. A classifier includes a rule and one or more parameters. Usually the parameters are used to describe the validity of the rule. The rule consists of a body containing conditions and a head containing actions. Thus, a rule states that if certain conditions are met, a specific action is to be taken. The main parameter for a classifier is the fitness of the rule. Fitness is a measurement of goodness for the rule when compared to the rest of the rules in the set. [28]

The evolution of the classifier set is done by the use of genetic algorithms [28]. As the name suggests, genetic algorithm borrow their main foundations from the theory of evolution: small changes in an organism result to better or worse performance in terms of survival. This idea can be used to achieve the above mentioned evolution of a set of classifiers. Small changes to the rules within the classifiers results to changes in the goodness of the rules. Better performing new rules are preserved and rules with low performance are eliminated from the set. One iteration of a basic genetic algorithm includes following steps [29]:

- Evaluate all rules in current set of rules.
- Select parent rules from current of rules. Better performing rules have higher probability of being selected.
- Modify parent rules by combining or slightly modifying them to generate offspring.
- Add offspring rules to the next set of rules.
- Remove rules from the next set of rules to preserve set size. Worse performing rules have higher probability of being removed.

As mentioned previously, in addition to discovering new rules LCS relies on a learning component. Learning can be defined as improving performance through gaining experience in a task [29]. This is already described from a high level perspective in the steps of the genetic algorithm. Basing the probabilities for selection of parents and selection of deletions on their performance metrics guides the evolution towards better performance. Thus, the discovery component refers to the creation of new rules while learning refers to the performance evaluation of those rules. For LCS, this learning of a goodness of an action has been done through supervised learning and reinforcement learning techniques [29]. Supervised learning refers to learning methods where correct solutions are provided to the learner. A typical example of this is the training process of an ANN where an input is given to

the network first. Then, the output of the network is compared to the known correct output and the network is adjusted accordingly. For reinforcement learning, the absolute correct solution is unknown and the learning is done based on the feedback from the environment describing the goodness of the action.

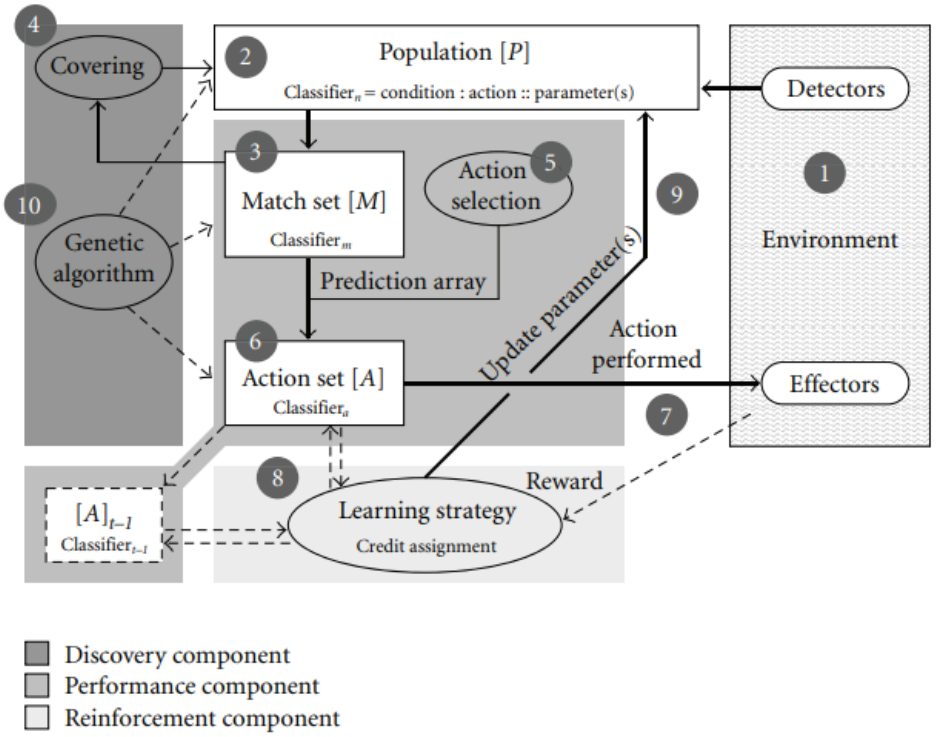

Figure 5: Visualization of a generic learning classifier system. Numbers indicate the steps executed in one iteration while thick lines correspond to the flow of information. Thin lines correspond to mechanism activations and dashed lines correspond to steps that might not be excecuted for every iteration. [29]

Figure 5 visualizes a generic LCS [29]. The figure is divided into three components: discovery, performance, and reinforcement. The discovery component depicted on the darkest background includes the genetic algorithm modifying the rules and corresponds to the discovery step introduced previously. The performance component, shown on medium darkness background, matches information received from the environment through detectors to existing rule bodies and determines an action. This action then affects the environment and informs the reinforcement component about the goodness of the effect on the environment. The reinforcement component is shown on the lightest gray background. This reinforcement component then updates the parameters of the classifiers for next iteration.

2.2.2 Association rule learning

Association rule learning is a method for finding frequently appearing patterns in a dataset [30]. Specifically, patterns consisting of two itemsets appearing frequently

together are extracted when association rule learning is used on a dataset. Common uses for association rule learning include applications such as recommendation systems in online services [10], understanding puchase behavious of a customer [31], and bioinformatics [32].

The process of association rule learning can be divided into multiple steps [33]. The first step is finding frequently appearing itemsets in the data. The second step is forming rules by combining the frequent itemsets discovered in the first step. Different statistical metrics are used to recognize frequent itemsets and statistically significant connections between those itemsets. The main metrics used are support and confidence [34]. Other metrics such as lift exist, however, basic association rule learning can be implemented by using only support and confidence. Support can be defined as

$$supp(X) = \frac{|t|}{|T|} \tag{1}$$

where X is an itemset, t is the set of transactions containing itemset X, and T is all transactions. In other words, support of itemset X is the ratio of the number of transactions containing X and the total number of transactions. Thus, support is a value between 0 and 1.

The second important metric, confidence, is a value between 0 and 1 indicating the ratio of the number of transactions containing both itemsets to the number of transactions containing the itemset that implies the other itemset. Confidence can be defined as

$$conf(X \implies Y) = \frac{supp(X \cup Y)}{supp(X)} \tag{2}$$

where X and Y are itemsets and supp(X) is as in Equation 1. Thus, confidence that the existence of X implies the existence of Y is calculated by computing the ratio between the support of the union of X and Y and the support of X. The main difference between support and confidence is that a support value contains information about a single itemset while confidence value contains information about a pair of two itemsets. Specifically, confidence describes the strength of a rule while support corresponds to statistical significance [33].

The first main step of association rule learning is finding frequently appearing itemsets in the data. This is done by inspecting different combinations of items to compute the supports of each different itemset. A support threshold is set by the user and if an itemset has a support exceeding the threshold, the itemset is considered frequent.

The second main step of association rule learning is combining the frequent itemsets found in the first step to association rules. This is done by first combining frequent itemsets into pairs. Next, confidence value of each pair is computed. Notably, $conf(X \implies Y)$ can be different from $conf(Y \implies X)$. Thus, both directions for each pair is considered. Similarly to the first step, the user defines a confidence threshold and any pairs with confidence under that threshold are discarded. This results in pairs of itemsets where, by some confidence, existence of one itemset implies the other. Thus, a rule is formed.

3 Research material and methods

The research conducted in this thesis aims to provide insight into a part of an envisioned end goal system. This system is an autonomous vehicle capable of operating in real world traffic while taking advantage of ontology-based techniques. This research is divided into two major parts: data generation and rule generation. This section discusses in detail the choices made regarding both of the parts. First, software choices and the reasoning behind those choices are discussed. Next, the data generation part of the work is discussed in detail including aspects such as prerequisites for the data gathering setup, the data gathering methodology, and creation methods for two different datasets. Finally, rule generation methods are introduced in addition to the data preprocessing necessary for application of these methods.

3.1 Software selection

The generation of data was done by simulating different scenarios. The scenarios are then sampled by detecting the state of the scenario intersection at regular time intervals. For this purpose, a suitable simulator as well as other supporting software used was chosen. This section discusses the choices made in regards to used software.

3.1.1 CARLA

CARLA is an open source Unreal Engine based simulator specifically designed for autonomous driving research [9]. In this work CARLA was used to produce the required data for both development and testing purposes. Features offered by CARLA include elements such as pre-made urban environments, different vehicle models, multiple different sensors, and a Python-based application programming interface (API) to control the simulation in multiple different ways. In addition, the API allows data gathering in real time.

CARLA was specified as the simulator to be used by the research group for which this research was conducted for. The benefits of CARLA stem from its design goal being use in research regarding different aspects of autonomous vehicles from sensor data processing to high level planning and decision making as well as low level control. Furthermore, CARLA is being actively developed and consistently receives significant feature additions. CARLA version 0.9.2 was used in this work as it was the update introducing assisting functionality for running scenarios.

3.1.2 Autoware

Autoware is a software project with a comprehensive set of modules for self-driving vehicles [35]. Autoware includes modules for different autonomous vehicle subsystems such as sensor data processing, path planning, and decision making. Autoware is a robot operating system (ROS) based software. ROS is an open source framework widely used for development of robotics applications that offers multiple ready-made packages for common tasks in robotics [36].

The envisioned autonomous vehicle system is planned to use Autoware for some of the tasks involved in autonomous driving. Thus, the functionality of Autoware was, used to evaluate if knowledge extracted from the simulator can be extracted from sensor data in real world applications.

3.2 Data generation

A dataset is required to enable data-driven machine learning. However, this work deals with concepts not included in the main research directions in the field of autonomous vehicles. A typical dataset aimed for autonomous driving research contains sensor data captured from varying traffic situations. An example of such dataset is the KITTI dataset. The KITTI dataset includes camera images, laser scans, global positioning system (GPS) measurements, acceleration measurements, and object labels [37]. Ontology-based systems deal with data that is semantically abstracted to a much higher level. Transforming the KITTI dataset to a suitable form would require a significant amount of work in itself. As a result, a decision was made to create a new dataset.

This section discusses generation of a dataset consisting of semantically abstracted high level data in detail. The goal for the content of the dataset was to cover only a specific subset of possible traffic scenarios. Additionally, the data needed to represent scenarios that progress according to real traffic rules. This allows generation of the dataset with reasonable effort while being sufficient for application of rule learning methods. Furthermore, the achieved results are easier to review by a human when the scope of the dataset is narrow.

3.2.1 Simulation setting

The envisioned data generation method was to run different scenarios in CARLA-simulator and extract the data in real time. The focus of scenarios was set to be an uncontrolled intersection. An uncontrolled intersection refers to an intersection with no traffic lights or other means of elevating some road to higher priority than others. Thus, an intersection within a ready-made town *Town03* in CARLA was chosen. At the time, it contained the only uncontrolled intersection available in the provided towns. As a result, the intersection where the scenarios were ran is a 4-way intersection with equal priority for all roads connected to it. The intersection in question is visualized in Figure 6 as seen through CARLA server view port. In the figure, two vehicles are crossing the intersection simultaneously. This type of intersection is suitable for the purpose of this work as it offers interesting scenarios with clearly defined traffic rules. The exception to this is a scenario where four vehicles approach the intersection from all sides simultaneously, which is a rare and challenging situation even for human drivers.

3.2.2 Random scenarios

The first approach to generating a dataset was to create and run random scenarios. The goal was to gain insight into both creation methods of random scenarios as well

Figure 6: The intersection where data gathering scenarios take place as seen in CARLA. Two vehicles can be seen crossing the intersection simultaneously.

as researching their applicability for rule generation. Random nature of the scenarios results in varying amount of data from each possible scenario with the possibility of some scenarios missing from the data completely. This emulates a dataset gathered from real world traffic. Multiple approaches to achieve this were considered.

CARLA offers different ways of controlling vehicles. The first available method is manual control through the Python-API by either running a software deciding the controls or using human input such as a keyboard. Second method is using a feature called agents. At the time, two different agents were available: basic and roaming. Third option similar to the roaming agent is an autopilot offered by CARLA. A fourth way for controlling vehicles was added in update 0.9.2 alongside the agent feature: behavior trees for running scenarios. Both of the agents and the autopilot were used and tested during the research.

The agents control the vehicle automatically. The difference between the two types is that the basic agent is used to navigate from one point to another while the roaming agent makes random decisions at intersections and drives around indefinitely without a goal.

The autopilot drives the vehicle around the map in the same fashion as the roaming agent. The main difference between the autopilot and the roaming agent is that the roaming agent is a newer feature built on top of the agent class enabling more flexibility for the developers while the autopilot offers no interface in addition to enabling or disabling it.

Behavior trees allow the user to define combinations of sequential and parallel behaviors with different triggers. The behavior tree feature is discussed in greater detail in Section 3.2.5 as they were used in the creation of a non-random dataset.

Out of the available methods for controlling the vehicles, the agents seemed best

suited for the random scenarios. This is due to the agents offering highly automated control of the vehicle and removing the need for lower level control implementation. As such, the options considered first were the basic agent with a random goal or the roaming agent making a random decision in an intersection. However, while testing the agent feature many problems arose. Most of the problems dealt with the agent behavior not being as intended. The basic agent is meant be able to navigate to a given goal while safely dealing with other vehicles as well as traffic lights [38]. However, in first testing the basic agent was not able to navigate to a goal, but instead veered off the road and collided into buildings. It was possible to fix this by adjusting the spawn location of the vehicle. The agents as well as the autopilot use waypoints built into the map. Choosing the spawn point of the vehicle closer to a waypoint fixed the problematic behavior and allowed the vehicle to reach its goal. However, adding a second vehicle into the scenario introduced a new problem of the agent not being able to avoid other vehicles. This problem persisted through different tests and was not fixed during this work.

The agents proved to be problematic for achieving traffic rule compliant behavior in the intersection. The autopilot feature of CARLA was the next method looked at for controlling the vehicles. The autopilot was simple to get to behave properly and it was able to negotiate with other autopilots in the same intersection to allow the scenario to progress safely without collisions. The problem arising from the use of the autopilot was that while the autopilot was able to negotiate priorities in the intersection, it was not able to follow real world traffic rules. Instead observing the autopilot behavior gave the impression of the priority being related to the timing of a vehicle arriving to the intersection. While this behavior posed clear challenges for evaluating the results of rule generation, this method of controlling the vehicles within random scenarios was chosen. This choice was made as implementing and testing knowledge extraction methods was not affected by the problems caused by the autopilot. However, evaluating rule generation methods was expected to be challenging with this dataset due to the extracted knowledge not including the pieces of knowledge dictating the behavior of the autopilot. For example, the timing of vehicles arriving to the intersection was not included in the dataset.

After the way to control the vehicles involved in the scenarios was chosen, the actual construction of the scenarios was begun. As the autopilot provided by CARLA would take care of controlling the vehicles through the intersection while choosing the goal direction randomly, defining the goal directions was not necessary. Only the starting points had to be defined manually. This involves finding suitable positions and orientations for the vehicles. There are some options for this process. One option was to drive a vehicle within the map and retrieve the coordinates through the Python-API. A second method was to spawn vehicles and finding correct coordinate values by trial and error. A third option was to access the map data within the Unreal Editor where the maps were originally built in. The scenario running feature introduced in CARLA version 0.9.2 included example scenarios. The general coordinates for the intersection were retrieved from these examples as at least one of them took place in the intersection of interest. The exact spawn locations for the vehicles were determined by trial and error by spawning vehicles and visually inspecting their

locations.

The autopilot operates by following the waypoints built into the map the same way the agents do. During the testing of the autopilot a problematic phenomenon was observed. The autopilot would not cross the intersection in all possible ways, namely turning left or right or driving straight. The reason for this was found out to be the waypoints for different directions separating from each other earlier than the chosen spawn point. To correct this unwanted behavior, two start points for each road were determined with one being on the left of the lane and the other on the right. Having two starting points per road connected to the intersection also enabled scenarios with a maximum of eight vehicles instead of the previous maximum of four. However, eight vehicle scenarios were not expected to provide any advantage over four vehicle scenarios for rule generation as traffic rules involved are known to only consider two vehicles at the time. This is because giving way to one vehicle at a time is enough. Once that vehicle has crossed the intersection, the situation is reassessed and possibly another requirement for giving way is found.

As the autopilot was deemed suitable and the starting locations were determined, the scenarios were executed. A software was created to repeatedly run random scenarios. The randomness was in form of the number of vehicles involved in the scenario being random from one to four, the spawn points of the vehicles being random, and the goals being randomly determined by the autopilot. The scenarios were executed by spawning in a number of vehicles in selected spawn locations and turning on the autopilot for each vehicle. In order to run scenarios repeatedly, an end condition for a scenario had to be defined. A scenario was considered to be over once all vehicles had exited the intersection. Detecting this required implementation of knowledge extraction, which is discussed in Section 3.2.3.

3.2.3 Knowledge extraction

Extracting high level knowledge from the simulation is a crucial part of this work. The goal is to represent the knowledge in a high level semantically abstracted form. This means the data is represented in a self-describing form understandable by humans. For example, the data needs to describe without further processing whether a vehicle is currently giving way to another vehicle or not. This kind of format for the data is used by the decision making system this work is based on and as such is a natural choice.

An assumption was made that the relevant vehicle controls in terms of the rule generation consisted only of *stop*. This is due to a default state for the vehicle being *go*. Furthermore, a path planner provides the system with the ego-vehicle goal and after a decision has been made, low level control is taken care of by a separate control module. This renders actions other than *stop* irrelevant for the purposes of this work.

The knowledge extraction process can be divided into multiple parts. First, relevant and interesting pieces of information from the perspective of generating rules were identified. Then, a concrete definition was created for each piece of data to allow the implementation of the extraction. Finally, the implementation of detecting vehicle events and states according to the aforementioned definitions took place.

Through the whole process of knowledge extraction, the goal of having a result that can be adapted for real world data acquired from driving in traffic was taken into account. Every piece of knowledge extracted and used needs to be possible to obtain from the sensor data of a real world vehicle. For some of the knowledge extracted, it is debatable whether the knowledge can be reliably acquired in real world situation while some of the knowledge is easily obtained. The functionality in Autoware was used as the basis of evaluating whether or not the information can be realistically acquired from sensor data. However, it is to be noted that the rule generating methods applied in the work are, in essence, mining patterns from the data. As a result, the methods are general enough to be applied to different data and the results will then reflect the data used. Specifically the extracted pieces of knowledge impact the final rules greatly. The decisions to extract certain data items aim to have the resulting rules reflect the format of real world traffic rules. This enables evaluation of the rules by a human reviewer rather than by exhaustive testing.

The first step of high level knowledge extraction was to decide what needs to be known in order to generate relevant rules. First, the scope of information was inspected. A real world vehicle with sensors can only see other vehicles that are directly visible to it, however, this is not an inherent limitation when dealing with data retrieved from the simulator directly. Despite this, information about vehicles that are not in the intersection of interest are not relevant for the goal of this work. As such, it was decided that the final data would only contain information about vehicles either waiting to enter the intersection or vehicles crossing the intersection. Any vehicle outside the intersection, whether it has just exited the intersection or never enters the intersection during the scenario is not relevant and was not be included in the data. Thus, a way to detect if a vehicle is in an intersection was implemented. This implementation also allowed the continuous running of scenarios as it it enables detecting when all vehicles involved in the scenario had crossed the intersection. The envisioned real world vehicle system would also require knowledge on any dynamic objects in the environment. These include actors such as pedestrians and bicycles among others. Only car-type vehicles were considered in this work.

For the detection of exiting the intersection, every vehicle was given an attribute *goal*. When this attribute was not set, the vehicle had not yet crossed the intersection. This meant that the vehicle was still relevant for the final data. Once the coordinates of the vehicles were detected to be outside the intersection and in a direction that was not its starting direction the *goal* attribute was set to be the direction the vehicle exited the intersection. This was done by checking if the current coordinates of the vehicle are within a predetermined rectangle which has corners at the corners of the intersection. Thus having a non-empty *goal* attribute implied that the vehicle had exited the intersection. However, it is arguable whether or not it is possible to access the information of vehicle goal in the real world using sensors. In the future, this might be possible relatively effortlessly if autonomous vehicles communicate with each other. Currently, this can be done to a certain degree with machine vision by detecting the turning signal of the vehicle. However, occlusion of the signal can cause problems in determining, for example, if a vehicle on the right of the ego-vehicle is

going straight or turning right.

The knowledge on the vehicle goal together with the vehicle starting direction from the perspective of ego-vehicle are both relevant information for the rule generation process. Together they determine the correct actions within the intersection. This is the result of real world traffic rules being defined as relative directions from the perspective of each vehicle. For example, usually vehicles can not enter an intersection if another vehicle is already crossing it. However, if the ego-vehicle is driving straight across the intersection while a vehicle on the opposite side is also going to drive straight across, the vehicles can cross the intersection simultaneously.

Figure 7 demonstrates two different scenarios with two vehicles each. The scenario on the left demonstrates parallel vehicle priority while the scenario on the right demonstrates sequential vehicle priority. In three vehicle scenarios this gets more complicated as they can contain both parallel and sequential vehicle priorities. The starting location of the vehicle can be extracted directly from the vehicle spawn point. The location data available from the simulator is by default in the coordinate system of the map. As such, processing the data is required to determine the relative directions, such as left, right, and opposite for both starting and goal locations. In a real world system the starting location of the vehicle is extractable from sensor data. For example, the combination of a LiDAR and a camera can be used to detect vehicles and access their coordinates relative to the ego-vehicle. These coordinates can then be used to detect from which road the vehicle is approaching the intersection. In the final ontology-based system, the vehicles would be assigned to be on a specific road that is connected to the intersection. The knowledge base would then be queried for the location of the vehicle relative to the ego-vehicle. Similarly the starting and goal locations would be defined as roads with the knowledge base containing information about the turn required to be taken to reach the goal road from the starting road through the intersection.

The implementation of *goal* attribute caused the data to be incomplete at the sampling instant. Specifically, the goal of the vehicle was not recorded in real time. The autopilot provided by CARLA can not be directly interacted with outside enabling or disabling it for each vehicle. This meant that the goal of the autopilot was not possible to extract until the vehicle had exited the intersection. This introduced a problem in saving the data as crucial information was possible to be accessed only after a scenario had already completed. To solve this problem, the data from each scenario was processed again after the scenario. Thus, the knowledge of the goal direction of each vehicle was added to the data retroactively. This is a side effect of the implementation of the autopilot within CARLA and does not interfere with the goal of having a system usable in real world applications. In addition, the data was not planned to be processed as a stream but rather as a single dataset. Thus, the retroactive addition of knowledge into the data was a viable solution. Furthermore, any data after the ego-vehicle has exited the intersection is irrelevant as it does not contribute to information about correct ego-vehicle behavior in the intersection. However, the scenario would run until every vehicle involved had exited the intersection as it was necessary to run the scenario until the end to access the knowledge of vehicle goals for every vehicle. As the post processing was already

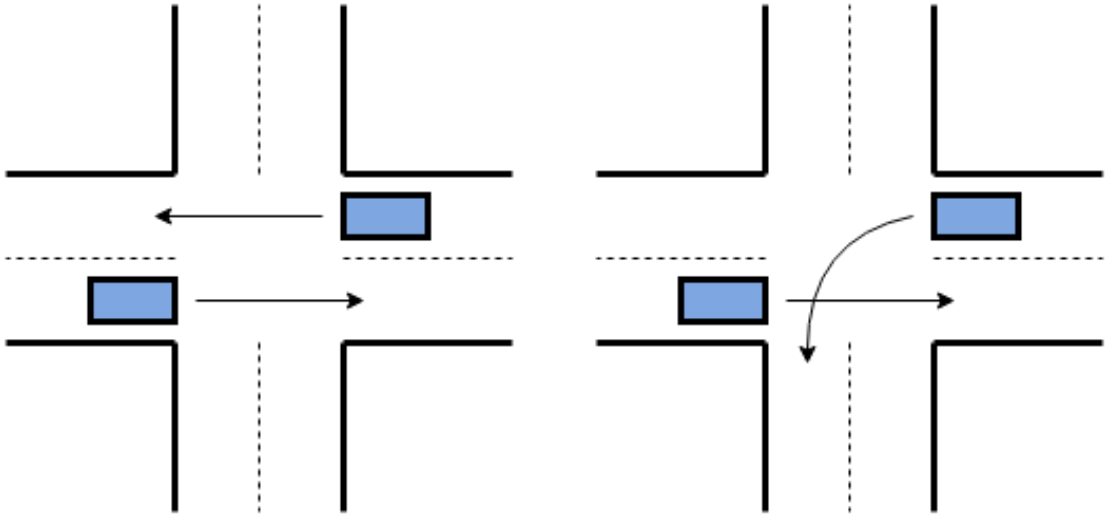

Figure 7: Examples of scenarios with parallel vehicle order and sequential vehicle order. The left image depicts a scenario where two vehicles can safely cross the intersection simultaneously. The right image shows a scenario where one vehicle must give way to another in order to avoid collision.

implemented, another post-processing step was added involving removing scenario data after the ego-vehicle reached the goal.

It was required that any vehicle stopping would be detected. This allowed the extraction of both the action of the ego-vehicle as well as the state of movement of other vehicles. A vehicle stopping was interpreted as the vehicle giving way to another vehicle in order to relate the action to traffic rules.

First, giving way was defined as the vehicle speed being zero or close to it. This definition was sufficient for generating rules from scenarios that unfold according to traffic rules. However, this definition was not able detect the vehicles starting to give way. In other words the act of starting to slow down was not detected. This would create problems when interacting with another piece of extracted knowledge: collision warning. A collision warning triggering required at least one of the vehicles moving. This meant that if a vehicle successfully avoided a collision by stopping, a collision warning and a vehicle giving way was never detected simultaneously. While this pattern can be detected by inspecting the temporal relations of different observations, temporal patterns were not addressed in this work.

The act of a vehicle giving way to another vehicle is better represented by the act of a vehicle slowing down than the vehicle being stopped. As a result, giving way was defined to cover a stopped vehicle as well as a vehicle that is losing a set portion of its speed within a set amount of time.

Detecting incoming collisions is a crucial part of autonomous vehicle safety as avoiding collisions is naturally of utmost priority. While incoming detected collision was expected to produce uninteresting rules in the next major part of the work, it is relevant when scenarios contain unexpected behavior. Thus, a way to detect collisions was implemented. The implementation was done by storing the relative locations of

Figure 8: Visualization of an issued collision warning when vehicles are simultaneously approaching the intersection. The red bounding box visualizes a detected incoming collision with the red ego-vehicle.

the other vehicles from the perspective of the ego-vehicle in sliding window buffers. This results in having access to a sampled path of a vehicle in the coordinate system of the ego-vehicle. As a result, the path takes into account both the movement of the ego-vehicle as well as the movement of the other vehicle. In addition, the resulting coordinate system only contains a single trajectory, thus removing the need to predict multiple trajectories. Fitting polynomial on this path of coordinates creates an expected trajectory of the vehicle relative to the ego-vehicle in the form of a curve. This trajectory is reliable only if the velocity of neither vehicle changes. Finally in the last step of trajectory checking, the distance between the fitted curve and the origin of the coordinate system is computed. If the trajectory is closer to the origin than a set safety distance threshold, a collision warning is issued.

The collision prediction method is visualized in Figure 9. The left side graph displays the locations of two vehicles in the coordinate system of the map at four different timesteps. Ego-vehicle locations are displayed in blue and the locations of the other vehicle are displayed in red. Assuming the vehicles continue with stable velocity, the two vehicles are on direct collision course at the collision point shown in black. The coordinate values are then transformed into the coordinate system of the ego-vehicle as shown in the graph on the right side. The locations of the other vehicle relative to the ego-vehicle are displayed in red for four different time instants. Origin of the coordinate system is the location of the ego-vehicle and is marked in black. A line fitted onto the sampled locations of the other vehicle results in a line that goes through the origin meaning the distance between the line and origin is zero, which is smaller than the specified safety distance threshold. This results in the issuance of a collision warning.

A problem with this implementation of collision warning issuance is that it cannot detect if a vehicle is moving towards or away from the ego-vehicle. Another problem is that this method issues collision warning regardless of how far the vehicles are from each other or how fast they are approaching each other. To solve these problems, the time to impact was calculated and used to avoid unnecessary collision warnings that would only take place after an unreasonably long time has passed as it is unlikely that both vehicles continue on the same trajectory for extended periods of time. The time to impact is computed by calculating the projected velocity of the other vehicle directly towards the ego-vehicle. In addition to avoiding collision warning issuances for far away collisions, tracking the projected velocity of the other vehicle directly towards the ego-vehicle enables access to information regarding whether or not the vehicles are actually approaching each other or moving away from each other. The value of the projected velocity being negative signifies that the distance between the two vehicles is growing. Optionally, the direction of the movement of the other vehicle can be solved by computing the distance of the vehicle from the ego-vehicle at subsequent time instants. However, the velocity of the other vehicle directly towards the ego-vehicle is already computed for time to impact calculations. As such, using the information known about the velocity is the natural choice. Basing the collision warning detection on time to impact in addition to estimated trajectories also removes collision warnings caused by vehicles driving on the same lane. Driving on the same lane causes the vehicles to have overlapping trajectory estimations. However, if the distance between these vehicles is not decreasing rapidly, a collision warning is not issued.

Another problem with this method of detecting incoming collisions is that fitting a line does not take into account if a vehicle is turning and always assumes it is moving straight forward. Thus, this method cannot predict a collision resulting from the vehicle turning. An example of this being relevant is frequently seen in real life traffic. A vehicle is turning to a road pedestrians are crossing. Optimally, this results in a collision warning between the pedestrians and the ego-vehicle. This problem can be approached by implementing more sophisticated trajectory prediction methods. Additionally, fitting a second degree polynomial instead of a line enables prediction of curved trajectories. However, second degree polynomials fitted to the points are likely to only be reliable up to a certain distance. This is due to the fitted curve being a parabola instead of a circular arc.

This implementation of collision warning issuance relies only on the information of the coordinates of the other vehicles relative to the ego-vehicle. As such, this information is accessible in real world applications through sensors attached to the vehicle similar to detecting starting directions of vehicles in intersection scenarios. However, this implementation was observed in simulations to only perform well when vehicles have stable velocities and are not turning.

Figure 8 shows a start of a scenario where vehicles approach the intersection at the same time. This causes collision warnings to be issued as the vehicles are detected to be on a collision course with the ego-vehicle and the time to impact is lower than the specified threshold. The red vehicle is the ego-vehicle and the red bounding boxes around other vehicles visualize an issued collision warnings between

the vehicle in question and the ego-vehicle.

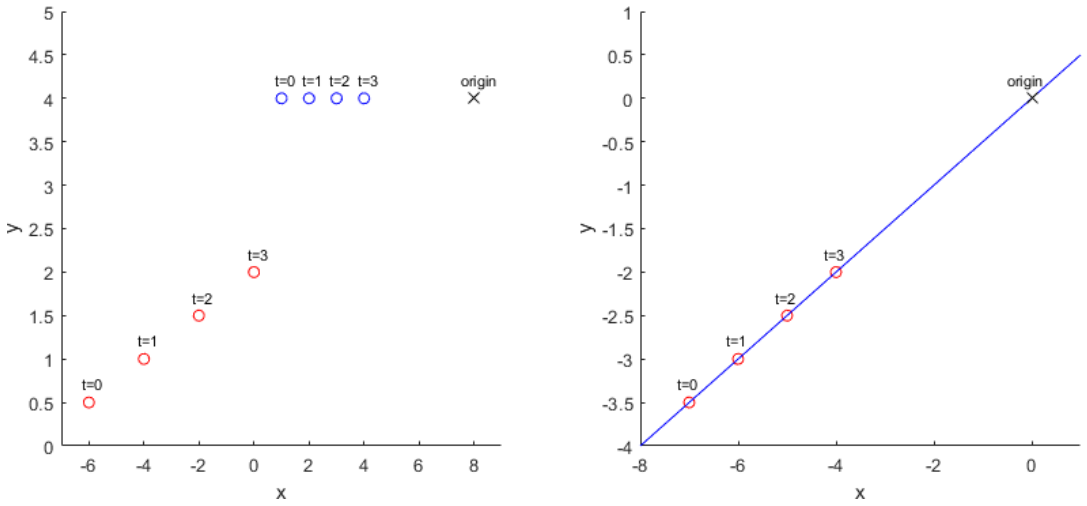

Figure 9: Incoming collision detection method visualized. Left graph shows sampled locations for two vehicles at four timesteps in map coordinate system. Ego-vehicle in blue and another vehicle in red. Collision point is shown in black. Right side graph shows the same timesteps with the coordinates transformed into the ego-vehicle coordinate system. The line fitted on the sampled points in the right figure passes through the origin and as such a collision warning is issued.

3.2.4 Knowledge data formatting

After the high level semantic knowledge was extracted from the scenarios, the next step was to store that data. The choice of data format is an important decision and it is crucial for the future applications of the software. The data consists of samples of vehicle states and information in a semantically abstracted format. As such, a strong candidate for the data format was to create an RDF graph for each sample. RDF is a widely supported and standardized format. Specifically the choice of RDF was made due to the benefits of storing data in a standard format and the the aim of this work being used with an ontology-based decision making system that is already compatible with RDF. The relationship between ontologies and RDF is that ontologies can act as vocabularies of terms and their relations while RDF is a data format where those terms are combined into triples. Triples consist of a subject, a predicate, and an object. Together a set of triples forms a graph. RDF can be used to either display an ontology in a standardized format or an ontology can be used as the vocabulary for displaying objects and relations in RDF format [39].

Figure 10 displays an example ontology for describing the control of a vehicle. From the ontology we can see that *Stop* is a *DrivingAction* which is part of the root node of the ontology. This way, an ontology can semantically describe different vocabularies. In contrast, Table 1 contains example RDF-triples from one time instant in a scenario for a single vehicle. The first column, subject, contains the ID assigned to the vehicle to allow keeping track of vehicles and connecting data from

different time instants to each other. In this case the vehicle displayed has ID *0*, and in the implementation used in this work this translates to the vehicle being the ego-vehicle. The middle column contains the predicate that connects the subject and the object. Finally, the third column contains the object of the triple. In this example it can be observed that the ego-vehicle is a car, does not have a collision warning issued, is not giving way to another vehicle, started from the same direction as ego-vehicle, and is turning right towards north.

Figure 10: Part of an example ontology containing terms concerning controlling a vehicle and the relations between those terms [8].

3.2.5 Handmade scenarios: background

As mentioned in Section 3.2.2, a notable problem is caused by the implementation of random scenarios. While the random scenarios are sufficient for testing knowledge extraction and rule generation functionality, the rules generated from those scenarios pose significant challenges in evaluating the results. This is due to the data not reflecting the underlying vehicle priority deciding factors. This makes it difficult to say if the rule generation is actually working as intended. To solve this problem, handmade predefined scenarios that unfold according to traffic rules were implemented. This ensures that the data created from the scenarios contains extractable information on the patterns described by traffic rules.

First, the interesting and relevant scenarios have to be identified. This is a consideration from the perspective of the ego-vehicle as from rule generation point

Table 1: Examples of RDF triples describing a single vehicle in a scenario at one sampled timestep. The vehicle has ID *0*, it is a *Car*, there is no detected incoming collision, it is not giving way to another vehicle, it starts from the same side of the intersection as the ego-vehicle, and it is turning right.

Subject	Predicate	Object
0	HasType	Car
0	HasCollisionWarning	False
0	IsGivingWay	False
0	HasStart	Same
0	IsTurning	Right

of view this is the only relevant perspective. The scenarios were divided into groups depending on the number of vehicles involved. Naturally, scenarios involving only one vehicle are not interesting as the vehicle can always enter the intersection. This combined with the assumption that the default state for the vehicle control is *go*, one vehicle scenarios were not implemented. The set of interesting two-vehicle scenarios from the ego-vehicle perspective were created using the following factors:

- Ego-vehicle can always start at the same position.
- The other vehicle can start in any of the other three possible starting directions.
- U-turns are not permitted.
- Ego-vehicle can have three different goals.
- The other vehicle can have three different goals per starting direction.

As a result of the above, the number of unique and interesting two vehicle scenarios is calculated by multiplying the number of other vehicle start locations with the number of ego-vehicle goal locations and the number of goals for the other vehicle. Thus, the number of relevant two vehicle scenarios is $3 \cdot 3 \cdot 3 = 27$.

In addition to scenarios with two vehicles, scenarios with three vehicles were considered. Although an assumption could be made that all regular traffic rules are derivable from two-vehicle scenarios as giving way to one vehicle at a time is enough. However, as the number of scenarios is not too high to create manually, this assumption was not made. For the three vehicle scenarios, the factors creating unique scenarios are

- Ego-vehicle can always start at the same position.
- Road with no vehicle starting there can be in three possible directions.
- U-turns are not permitted.

- Ego-vehicle can have three different goals.
- The second vehicle can have three different goals per empty direction.
- The third vehicle can have three different goals per empty direction.

Thus, the number of unique three vehicle scenarios can be calculated by multiplying number of possible empty roads with the number of ego-vehicle goals and the number of goals for the second vehicle and the number of goals for the third vehicle. As such, the number of relevant three vehicle scenarios is $3 \cdot 3 \cdot 3 \cdot 3 = 81$.

The total number of interesting scenarios is then 108. However, it can be noted that out of the 81 unique three vehicle scenarios, only 27 are truly unique and the other 54 are symmetric to the first 27 except for assigning which vehicle is the ego-vehicle. Thus, the 54 symmetric scenarios can be created as rotations of the first 27 scenarios.

As with random scenarios, there are different ways to control vehicles within the simulation. For predefined scenarios, the vehicle goal direction has to be controlled. This immediately rules out the roaming agent and the autopilot. On first glance, the basic agent seemed well suited for this purpose, however the correct order of vehicles crossing the intersection was difficult to achieve. On top of this, the earlier problems with getting the basic agent to navigate safely while considering other vehicles caused the use of basic agents to be discarded as an option. As a result, the remaining options are manual control and behavior trees. The behavior trees were introduced alongside scenario running functionality in CARLA version 0.9.2 and they are designed for exactly this purpose. As such, the behavior trees were selected as the vehicle control method for the predefined scenarios.

An example behavior tree for a single vehicle is shown in Figure 11. This behavior tree contains the behavior for a vehicle coming from south and heading east. In other words the vehicle is turning right. The direction from which the vehicle approaches the intersection is important to consider as the intersection is not square. This results in the intersection crossing involving two different left turns depending on the direction of approach. The behavior tree of a single vehicle is a sequential tree, and in the example case the sequence consists of driving straight forward for a set distance, then applying steering for a set distance, and finally driving straight forward again for a set distance. Each of the children of these sequential behaviors are parallel actions that are executed simultaneously. For example, driving straight forward for a set distance has parallel action of keeping a set velocity and driving for a set distance. Each tree has a success condition, and for the tree for driving straight forward, the overall success condition is that all of its children succeed. This way, the scenario manager knows it can move to the next behavior in the sequence contained in the parent node.

The scenario manager provided by the CARLA team is a part of their scenario runner feature. The scenario manager allows access to most of the features related to the behavior trees. The behavior trees consist of options to create sequential or parallel trees and add subtrees into existing trees. This way, for example, a two turn scenario with two vehicles driving first simultaneously followed by the third vehicle

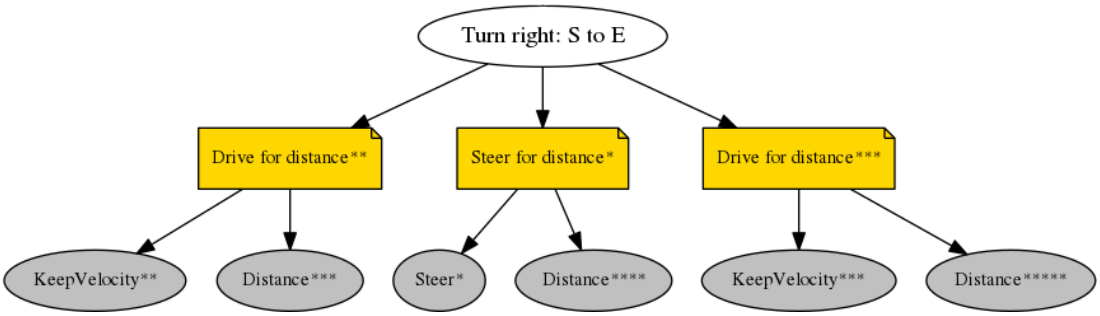

Figure 11: A behavior tree for a single vehicle. In this case, the vehicle is turning right as its starting direction is south and its goal direction is east. The turning is done by first driving straight for a set distance. Next, steer is applied to the vehicle for a set distance. Finally, the vehicle drives straight again for a set distance.

is possible. To create this kind of a scenario, a parallel tree containing the first two vehicles is added into a sequential tree followed by the addition of the behavior tree for the third vehicle. A behavior tree of this nature is shown in Figure 12. This behavior tree is inspected closer later in this section.

The behavior trees and the scenario running feature of CARLA are dependent on the benchmark mode available in the CARLA server. The benchmark mode locks the scenario framerate to a predefined value, meaning that the world is ticked a set number of times per second within the scenario time. Without benchmark mode the behavior tree vehicle control was observed to result in extremely nondeterministic behavior. Without the benchmark mode enabled, this implementation of predefined scenarios was not at an acceptable level of functionality. This nondeterministic behavior would manifest as vehicles sometimes not turning enough while sometimes they would turn 360 degrees instead of the intended 90 degrees. Enabling the benchmark mode helped the situation, however it did not completely fix the problem. According to observations from testing, the behavior seems more consistent the higher the benchmark mode framerate is set. 10 frames per second caused inconsistent behavior while 60 frames per second worked acceptably. For creating the dataset, 100 frames per second was used.

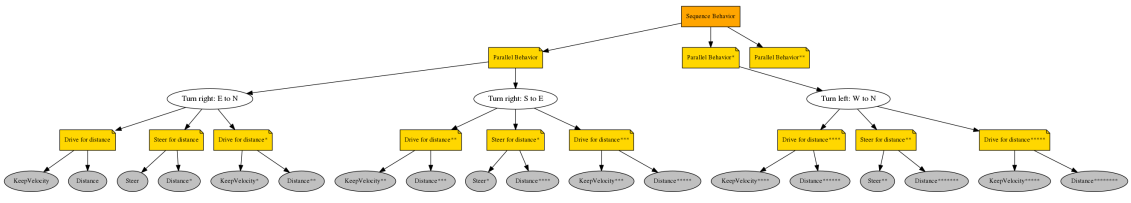

Figure 12: A behavior tree of a three vehicle scenario. Each white ellipse is a subtree containing behavior for a single vehicle. First two vehicles are in same sequential subtree as they can safely cross the intersection simultaneously while the third vehicle can only cross after the first two vehicles have crossed.

3.2.6 Handmade scenarios: implementation

The scenario runner feature of CARLA allows users to create scenarios using the behavior trees by defining a new Python class for each scenario. There are 108 scenarios to create that are highly similar to each other. Thus, the second step of creating the scenarios was to create a framework to define new scenarios with a reasonable amount of effort. Defining 108 classes for the scenarios would result in thousands of lines of unmaintainable code. As such, helpers functions were created to retrieve desired behavior. The most important function takes starting direction and goal direction as input and outputs a behavior tree for the driving sequence by computing which action is required. The available actions are turning right, going straight, and two different left turns as a result of the intersection not being a square. This allows defining the scenario driving sequences by defining the start and goal directions.

In addition to start and goal directions, one important definition is missing: the order of the vehicles to cross the intersection. The order of the vehicles was implemented by iterating over a list of lists, where each member of the primary list is a turn vehicles can drive on. Two vehicle scenarios have two available turns and three vehicle scenarios have three turns. Each turn is then represented by a list containing each vehicle to drive on that turn. This allows definitions where multiple vehicles cross the intersection simultaneously. This way, defining a scenario can be done in short and relatively effortless format. The start and goal directions are easily defined for all scenarios by combinatorics of known values. The order of the vehicles requires more thought as they need to be defined according to real traffic rules. After the 27 two vehicle scenarios and the first 27 three vehicle scenarios are defined, the rest of 54 three vehicle can be created as two simple rotations of the first 27 three vehicle scenarios.

Examples of this format are shown in Table 2 for both two and three vehicle scenarios. The first column contains the cardinal directions for the starting locations of each vehicle. Similarly, the second column contains the cardinal directions of the goal locations of the vehicles. The number of members in these columns can be used to compute the number of vehicles involved in the scenario. The final column contains the order of the vehicles. The order follows the turn format described above. Each of the brackets represents one turn, thus the number of them is equal to the number of the vehicles in the scenario. The numbers inside the brackets represent the index of the vehicle in the first two columns. For example, the first shown example is a two vehicle scenario where ego-vehicle starts in the east while the other vehicle starts in the west. We can see that both of the vehicles have their goal set as south. Thus, according to traffic rules, the vehicle starting from the west crosses the intersection first. As a result, the first turn contains number *1* representing the index of the second vehicle. The second turn contains number *0* representing the index of the ego-vehicle. Some of the examples contain scenarios where vehicles can cross the intersection simultaneously. In this case, some of the turns are empty.

The turn-based nature of the scenario definition can be seen in the behavior tree for the scenario. An example tree is shown in Figure 12. The orange root node of the

Table 2: Examples of scenarios in the compact definition format created for this work. One line describes one scenario and includes information for all vehicle starting directions, all vehicle goal directions, and the order of the vehicles. Example two and three vehicle scenarios are shown. N is north, S is south, E is east, and W is west.

Start direction	Goal direction	Order
'E', 'W'	'S', 'S'	(1), (0)
'E', 'N'	'S', 'W'	(1, 0), ()
'E', 'N', 'W'	'W', 'E', 'S'	(2, 1),(0), ()
'E', 'N', 'W'	'S', 'E', 'N'	(2), (1), (0),
'E', 'N', 'S'	'N', 'S', 'E'	(1, 2, 0), (), ()

tree is a sequential node, meaning all of its children are executed sequentially. The switch between active children nodes is done whenever the current active children reaches a success state. The sequential root node has three children. These yellow nodes represent the turns the vehicles can have in an intersection and thus the turns themselves are parallel in nature meaning all of their children are active simultaneously. Each of these turns have white children nodes that represent a single vehicle and contain the behavior tree for that single vehicle. It can be seen in the figure that the first turn contains behavior for two vehicles, the second turn contains behavior for one vehicle, and the last turn has no children. As a result, the scenario depicted has two vehicles crossing the intersection simultaneously while one vehicle is waiting for the previous two to exit the intersection.

3.3 Rule generation

This section details the process of generating the rules in automated fashion from previously acquired data. First, the rule format and data formats are discussed. Next, the implemented rule generation methods are introduced. The rule generation methods chosen are association rule learning and a deep learning based approach. These were chosen over other methods, such as LCS, due to their relative simplicity. In an effort to gain insight to viability of different methods, exploration of simple methods was determined to be a suitable starting point.

3.3.1 Rule format

The rule format is dictated by the decision system introduced in earlier work by Zhao et. al. [1] as the goal is to generate rules compatible with this system. The rules generated need to be logical rules consisting of a head implied by the body. Both the body and the head are conjunctions. In other words if each of the elements of the body evaluate to true, the body evaluates to true and as such, the head is also true. If the rule heads are the same for all rules, a set of such rules can be thought of being in disjunctive normal form. Each rule body is a conjunctive clause and these clauses

are grouped together as disjunctions. This is the case in this work. The body of the rules consists of observations from the environment, such as location and intention of ego-vehicle as well as other vehicles. In contrast, the head is made up by actions the ego-vehicle needs to take if all the observations in the body are true. In this work, the rule head can only contain *stop* to indicate that the ego-vehicle has to stop. As a result, only rules for situations where the ego-vehicle has to stop are generated. However, in case a need for more varied rule heads arises, all methods described can be extended to include other actions. Similarly, the set of knowledge items from which the rule bodies are composed of can be extended to include a greater number of observations.

3.3.2 Data formatting

The data used to generate the rules is stored in RDF-graphs consisting of triples describing high level knowledge of the state of the scenario at each time instant. This results in multiple samples from each scenario. As a result of the chosen data gathering method and data formatting, it is possible to construct a table of the data. In this table, each column is a different piece of knowledge from the scenario state including both observations and actions. Each row of the table is one sampled instant.

The data gathered contains a varying number of possible values for data items as RDF-graph as a format allows arbitrary data to be written in the triples. Some of the data is boolean while others are have multiple possibilities. For example, a vehicle at one time instant can either be giving way to another vehicle or crossing the intersection. Similarly, a collision warning with another vehicle is either issued or it is not issued. While data of this nature is naturally boolean, an example of non-boolean data would be a vehicle starting from either left, right, or opposite of the ego-vehicle. Similarly each vehicle can turn left, right, or drive straight forward.

For the purpose of storing the data as a table of boolean values and to achieve data of high usability for different methods, the non-boolean data can be transformed into multiple boolean entries. This is achieved by taking the data and dividing it into as many parts as there are possible variations of said data item. For example, a vehicle can turn left, right, or drive straight. This observation can be thought of in a different way. The vehicle can either turn right or not turn right. This way, one column of data describing the vehicle goal direction is transformed into three separate columns containing boolean data. As a result, a data table consisting fully of boolean data can be constructed. This transformation is only possible if all the possible options for the values in the table are not only finite but also known beforehand. For this purpose, the advantages of the data gathering methods used in the first part of the work are showing. In detail, the data is stored in a high level semantic form as opposed to describing the state of the scenario in other ways such as mapping observations to continuous values.

Table 3 depicts an example of partial boolean data table. The scenario being visualized is a scenario with two vehicles. Ego-vehicle is trying to cross the intersection by turning left while another vehicle, opposite of ego-vehicle, is crossing the

intersection by driving straight forward. From the data table it can be observed that while the opposite vehicle exists in the scenario, in other words before it has finished crossing the intersection, the ego-vehicle stops to give way to the other vehicle. Once the other vehicle has finished crossing the intersection it does not have any *True* entries in the table, signifying that it does not exist anymore in the data. At that point, the ego-vehicle can freely cross the intersection as can be seen from *Ego: stop* changing from *True* to *False*.

Table 3: Example part of a boolean data table. Columns correspond to observations and actions, rows correspond to a sampled time instants. Empty column and row indicate that the table contains rows and columns not shown here.

Sample	Ego: stop	Ego: left	...	Opposite: straight	Opposite: right
1	True	True		True	False
2	True	True		True	False
...					
10	False	True		False	False
11	False	True		False	False

3.3.3 Association rule learning

In order to generate rules, mining association rules from the gathered data was chosen as the starting point. Association rule learning principles are discussed in Section 2.2.2. This type of data mining is commonly used on data in the format of transactions of items. As such, the boolean data table discussed earlier can be considered to be in this format and can be utilized. For the purposes of this work transactions are represented by the sampled time instants and items are represented by the observations and actions extracted from the scenarios at each sampling time instant. Thus, this type of rule learning is a natural choice for this work due to the decisions made earlier in the process. Specifically, the decisions regarding the human understandable format of the extracted knowledge allow a relatively straightforward implementation of this data mining method.

The first step of association rule learning is the generation of all itemsets with a support value larger than a specified threshold [33]. This is not a trivial problem as the number of itemsets suffers from combinatorial explosion [40]. In other words, the generation of all possible itemsets is not feasible for large databases. The Apriori algorithm was chosen for discovering frequent itemsets in the data. The Apriori algorithm is one of the most popular approaches to itemset generation for mining association rules from transaction datasets [40]. It was originally introduced in 1994 by Agrawal and Srikant [41]. Since then, many other methods such as A-Close [42] as well as modifications of the Apriori algorithm, including Apriori-C [43] and a fast Apriori implementation [44], have been developed. Despite this, the original Apriori algorithm was chosen due to its sufficient performance for the purposes of this work

as well as the popularity of the algorithm implying thorough testing and proven reliability.

The efficiency of the Apriori algorithm is based on the concept of support and set theory. The meaning of support is discussed in Section 2.2.2. The original authors of the Apriori algorithm use terms large and candidate itemset. A large itemset is an itemset which has a support exceeding a specified support threshold. A candidate itemset is a new potentially large itemset to be inspected. New candidate itemsets are generated by only considering already found large itemsets. The basic intuition is that if an itemset is large then any of its subsets must also be large [41]. This results in the Apriori algorithm avoiding the unnecessary generation and testing of a significant number of itemsets that, based on existing knowledge, can never be large. The Apriori algorithm is a bottom-up approach that first generates short itemsets and then proceeds to generate longer ones. This significantly reduces computation requirements compared to simple combinatoric creation of all possible itemsets.

The apriori algorithm is shown in Algorithm 1. The algorithm is broken into three parts: the main loop and two steps of the *apriori-gen* function. The algorithm takes a set of large itemsets of length one as input. First, the large itemsets of length $k-1$ already found are used to generate new candidate itemsets of length k. Next, the support of each candidate is computed and all candidates that are found to be large are passed on to next iteration. This is done until no large itemsets of size k are found.

The generation of new candidate itemsets is done by the *apriori-gen* function. As an input, it takes the set of all large itemsets of length $k-1$. This function has two steps. First, in the *join* step the input set is joined with itself to create candidate itemsets of length k. Next, in the *prune* step all itemsets generated in *join* step are inspected. If any of the candidate itemsets of length k contain a subset of length $k-1$ that is not a member of the original set given as input to *apriori-gen*, that itemset of length k is deleted. This is possible due to two factors. First, the input set is known to contain all large itemsets of length k. Second, if an itemset is large then all of its subsets must also be large.

Once frequently appearing itemsets have been found, confidence is used as overall metric for rule strength to form valid rules of statistical significance. Confidence was discussed in greater detail in Section 2.2.2. Additionally, rules for an autonomous vehicle are required to be of specific format to be valid.

A valid rule is a combination of two itemsets where one of the itemsets contains only actions for the ego-vehicle and the other itemset contains only observations from both the environment and the ego-vehicle. A pair of itemsets fulfilling these requirements with a high enough confidence value can then be considered a rule. The itemset containing only observations forms the body of the rule that implies the itemset containing only actions which forms the head of the rule. Final step is to prune redundant rules that do not contribute additional information of value to the final result. This can be done by exploiting assumptions about the nature of the data or by taking advantage of the conjunction format of the rule bodies and heads. The rule pruning process is discussed in detail in Section 4.1.2 as it is relevant for all rule generation methods.

Algorithm 1 The Apriori algorithm [41].

1: $L_1 = \{\text{large 1-itemsets}\}$;

Part 1 – The main loop

2: **for** ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do**
3: C_k = apriori-gen(L_{k-1}); // New candidates
4: **for all** transactions $t \in D$ **do**
5: C_t = subset(C_k, t); // Candidates contained in t
6: **for all** candidates $c \in C_t$ **do**
7: c.count++;
8: **end for**
9: **end for**
10: $L_k = \{c \in C_k \mid c.count \geq minsup\}$
11: **end for**
12: Answer = $\bigcup_k L_k$;

Part 2 – apriori-gen: join step

13: **insert into** C_k
14: **select** $p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1}$
15: **from** L_{k-1} p, L_{k-1} q
16: **where** $p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$;

Part 3 – apriori-gen: prune step

17: **for all** itemsets $c \in C_k$ **do**
18: **for all** (k-1)-subsets s of c **do**
19: **if** ($s \notin L_{k-1}$) **then**
20: **delete** c from C_k;
21: **end if**
22: **end for**
23: **end for**

3.3.4 Deep learning

One of the main goals of this work is to deal with human understandable high level semantic data and rules. Some other approaches of representing the data call for the use of neural networks and deep learning naturally as a way to catch patterns in the data that are not directly understandable by humans. However, neural networks can also be used with the data gathered and formatted previously in this work. Admittedly, while the data and the results remain human understandable, using a neural network takes away from the understandability of the process as a whole.

Neural networks are a powerful tool for learning complex models from data. While sampling the scenarios creates a large amount of data in a sense, the data is still constrained by the number of different scenarios. Thus, the datasets used in this work are not exceedingly complex. However, specifically the handmade scenario

dataset covers all relevant scenarios that are of interest for this research. As a result, a neural network was expected to be able to extract the rule patterns from the data with high confidence.

Neural networks can be used in multiple ways. The approach taken in this research was to train the network and then generate rules by predicting outputs for different inputs. One potential way to combine neural networks with other methods is to use the network to generate new data that mimics the original data. Other methods can then be applied to the artificially created dataset. An example of such approach to using neural networks are generative adversarial nets introduced in 2014 by Goodfellow et. al. [45].

The eventual real world autonomous vehicle system is envisioned to use rules generated from data gathered in real world traffic. In this case, the data is more complex and the amount of data is much greater. However, the data might not have full coverage of all scenarios necessary for generation of rules with sufficient coverage. This can either emphasize the strong features of neural networks or prove to be a problem. For this reason, methods of generating rules by using neural networks were looked into in the scope of this work in order to lay a foundation for future work.

The boolean data table introduced earlier can be transformed into inputs as well as outputs for neural networks for training purposes. First, it is necessary to recognize and plan the structure of the network. The most important step is recognizing the inputs and the outputs. The boolean data table is one large table representing states of different observations and actions for each sampling instant. Naturally, each sample represents one set of inputs and outputs. The target rule format has bodies consisting of state observations and heads consisting of actions. This creates a division between objects in the table suitable for inputs and outputs. The observations are used as the inputs for the neural network while actions correspond to the outputs. This way, dividing the table into proper parts results in a set of 1-dimensional vectors where each vector corresponds to one sampling instant and one sampling instant produces two vectors: the input and the output vector. In this manner, a neural network can be trained and evaluated as well as used for creating predictions for different inputs. The predictions can be used to generate rules.

Generating rules from a trained network requires using the network to predict outputs for different combinations of inputs. Essentially, this can be done for the problem in question with a classification network. The goal is to have an input pattern classified as one of the possible actions for the ego-vehicle. As discussed earlier, the inputs and the outputs are binary. As such, there is a limited number of different possible inputs and outputs. The rule generation process through a neural network is in parts similar to the process of the Apriori algorithm introduced in Section 3.3.3. The rule generation requires creation of different combinations of inputs. That is, creating different combinations of observations and feeding them into the neural network. The network then produces a prediction of the output which indicates the combination of actions that are to be taken based on the observations. In other words, the network can be given different rule bodies and the network will output the rule heads. A notable difference from the Apriori algorithm process is that same assumptions do not hold for rules generated with a neural network. The

Apriori algorithm can use knowledge of previously tried combinations of observations to avoid unnecessarily checking some of the remaining combinations. This is not the case for the neural network. With a neural network, there is no guarantee that the prediction is correct. If the network is not completely accurate, testing a rule and using it as a basis for eliminating future test can lead to small inaccuracies compounding into a larger number of false rules.

The input and output encoding is visualized in Figure 13. Boolean data from one sampled time instant is encoded into a vector containing a binary value entry for each observation in the data. In the example, ego-vehicle is turning left while a vehicle to the right of the ego-vehicle is driving straight. The network then produces a classification based on the input pattern. Output is a vector with two entries indicating probabilities for each action being the correct action according to the network.

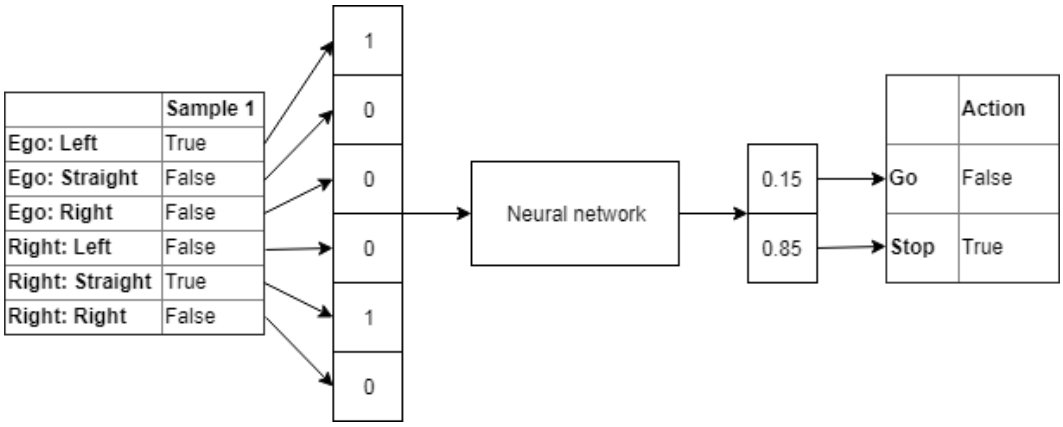

Figure 13: The encoding of the neural network input and output visualized. Observations from one sampled time instant are encoded into a vector and passed to the network. The output is a vector with probabilities for each action to be taken. Higher probability is interpreted as the correct action according to the network.

The rule bodies are generated by first generating all binary sequences of length corresponding to the length of the input vector, that is, the number of observations in the data. The number of observations in the data depends on many factors present during the data gathering as well as the methods used to store the data. Training the network on previously gathered data allows tailoring the network input layer to the data due to the existing knowledge on the dimensions of the data. In this work, for example, the whole dataset contains the number of observations required to describe the most complex scenario in the dataset. When some vehicles were not present in the scenario, all of their respective entries in the data were set to *false*.

The initial generation of all possible binary sequences generates a large number of unnecessary input vectors that can be pruned to both reduce computations and avoid false results. First, the rule bodies can be constrained to a set maximum length. This is possible as any subset of a conjunction is stronger than the superset and as such the superset is redundant. This can be seen in the conjunction elimination process, which is based on the property of conjunctions which states that if a conjunction

is true then any subsets of elements of that conjunction must also be true [46]. In addition, in the scope of this work, the correct rules are known to be relatively short. Setting a maximum length for the rules removes most of the generated rules. The resulting set of rule bodies contains invalid rules which can be removed. For example, the goal of the vehicle is represented by a combination of three binary digits. A vehicle can turn either left or right or drive straight. Only one of these can be true at one time for a single vehicle. Thus, any rules indicating more than one of these to be true can be removed as invalid.

After the input and output formats and the general applicability of the neural network has been determined, the next step is the detailed architecture of the network. While some parts of neural network architecture have to be tuned by adjusting hyperparameters, other aspects can be determined from the nature of the application. The complexity of the network structure in regards to the number of hidden layers and neurons can be adjusted to be suitable for the data used. In this work, the data amount might be large due to a high sampling rate, however the complexity of the data is relatively low. As a result, a network with small number of hidden layers and neurons was used.

The basic network structure is shown in Figure 14. The network takes input in the form of a binary vector of length corresponding to the number of observations in the data. The output vector has probability values for each element being the correct answer to the input. The output vector has a length of two to indicate the two possible outcomes. The possible outcomes correspond to the chosen action of the vehicle being *Stop* or *Go*. The network consists of two hidden layers, an input layer, and an output layer. All layers are fully connected layers and the number of neurons in the input layer and the hidden layer are subject to parameter tuning.

Neural network approach to solving this problem has its shortcomings as well as benefits. One of the aspects making neural networks a powerful tool in machine learning is their ability to predict values for missing data in a dataset. This was not taken advantage of in this work as the training set contains most, if not all, of the valid inputs the network is queried with. However, this is a potential benefit for expanded systems which have to deal with missing data. A potential benefit for using neural networks is their scalability. Changing the structure of the network is a relatively simple way to adapt to increased data complexity.

Focusing on set datapoints creates a problem with network overfitting not being detectable. Overfitting refers to a situation where the estimated model of the system corresponds to a particular set of points while being inaccurate outside those points. The goal in the scope of this work was to fit a model exactly on finite amount of data points and anything outside those points is not of interest. Thus, overfitting is not a problem. However, overfitting is to be kept in mind in case this approach is applied to other situations.

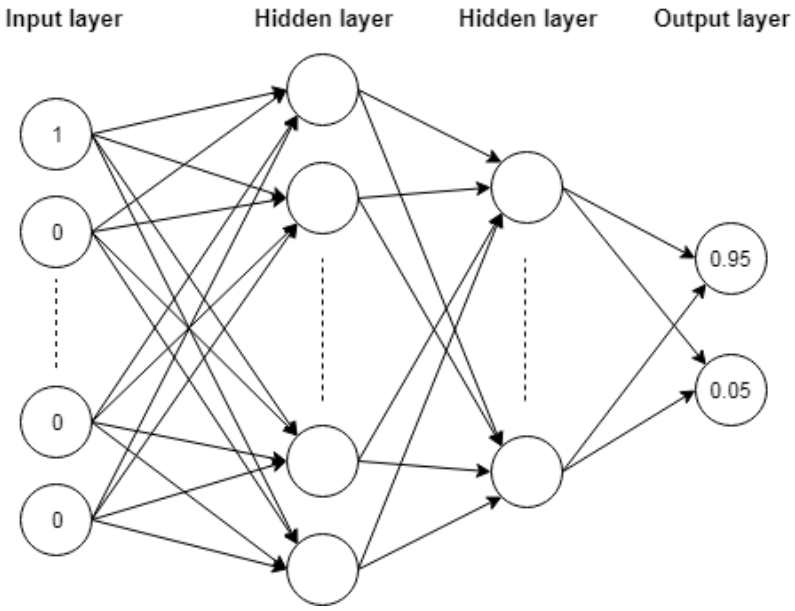

Figure 14: Visualization of the neural network architecture used. The network has two hidden layers. Input vector is a binary vector of length corresponding to the number of observations in the data. The output vector is of length two to indicate which action is taken out of *Stop* and *Go*. Values of the output vector are the probabilities of each action being correct to take according to the internal model of the network.

4 Experiments

This section details the process of experiments done during this research. First, the process of refining rules is explained. Next, the metrics used to evaluate rules and rulesets are introduced. Finally, different parameters for implemented methods and choices regarding them are discussed.

4.1 Rule structure and refinement

The rule generation process has to consider the available data to generate sensible rules. The available data can restrict the expressive power of the generated rules. However, considering the known properties of the available data can also be used to refine the results further. This section first introduces the available knowledge and the relation between the data items and the goal rule format. Then, the implications of the available data are discussed in regards to evaluating the usefulness of each data item as well as using insights into the extracted data to further refine the generated rules.

4.1.1 Extracted knowledge

The extracted knowledge items are displayed in their boolean form in in Table 4 and Table 5. These tables show the elements used to construct rule bodies and heads

respectively. For rule bodies, observation on the vehicle goal direction is included for each vehicle. In addition, the state of movement is included for non-ego-vehicles. A warning of incoming collision involving the ego-vehicle and any other vehicle is also included. In contrast, the rule heads can only contain *Stop* as the ego-vehicle action.

Table 4: The set of elements from which a rule body is constructed. To achieve the boolean format required for the applied methods all vehicles are separate and their non-boolean pieces of information are divided into mutually exclusive boolean information.

Element	Explanation
Ego-vehicle	
Ego: Left	Heading left
Ego: Right	Heading right
Ego: Straight	Heading straight across
Vehicle approaching from left of ego-vehicle	
Left: Go	Vehicle is moving
Left: Left	Heading left
Left: Right	Heading right
Left: Straight	Heading straight across
Vehicle approaching from opposite of ego-vehicle	
Opposite: Go	Vehicle is moving
Opposite: Left	Heading left
Opposite: Right	Heading right
Opposite: Straight	Heading straight scross
Vehicle approaching from right of ego-vehicle	
Right: Go	Vehicle is moving
Right: Left	Heading left
Right: Right	Heading right
Right: Straight	Heading straight across
Combined	
Collision Warning	Incoming collision between ego-vehicle and another vehicle

Table 5: The set of elements from which a rule head is constructed. In this work only *Stop* is used in the rules as *Go* is considered the default state of the vehicle.

Element	Explanation
Stop	The ego-vehicle stops

4.1.2 Rule pruning

The generated rules can contain redundant rules depending on the data used to generate the rules. The set of possible elements of rule bodies outlined in Section 4.1.1 can be pruned to contain less members by integrating assumptions. Furthermore, knowledge on conjunction properties can be used to remove redundant rules during the rule generation process. The stage of the rule generation where each of these are implemented is crucial. As discussed earlier in Section 3.3.4, the knowledge that any subset of a conjunction is stronger than its supersets can be used to remove redundant long rules. This can be implemented in any part of the process. Most importantly, the assumptions about the data have to be implemented in an early stage before generating candidate rule bodies. This is due to assumptions being used to preprocess the data to contain different set of elements to construct the rules from rather than removing redundant rules based on proven mathematical properties of logical constructs. Thus, using assumptions will remove non-redundant rules if implemented in a later stage of the process. This section discusses refining the results using these methods.

The generated rules are used to decide the behavior of the ego-vehicle in accordance to the observed environment. This sets requirements for the rules in order for them to be useful. The set of requirements was defined during the research as:

- a rule has to contain an observation on ego-vehicle state
- a rule has to contain an observation on another vehicle
- a rule must only contain information relevant to complying with traffic rules
- a ruleset must not contain rules that are covered by other rules

Out of these rules, the third rule has potential to remove rules in a way that the resulting ruleset does not have full coverage. This, in turn can lead to the vehicle taking its default action in a situation where it breaks traffic rules and results in a dangerous situation. As such, the irrelevant information removal must to be done before rules are generated. This causes the implementation of the third requirement to not modify the rules directly.

The first requirement is that the body of the rule has to contain an observation about the ego-vehicle. This is possible due to the data extracted being of such nature that one of the observed ego-vehicle states is always true at any given sampling instant. Naturally, this is not an automatically valid assumption in a more general case, for example the envisioned real world system, and has to be considered based on the use case and available knowledge. This caveat applies to each of the assumptions exploited in this work. In the case of the rule body elements presented in Table 4 the information about ego-vehicle is the goal direction the vehicle is headed. As these three possible directions are mutually exclusive, it can be assumed that a sensible rule contains only one of these. Combining this knowledge with the requirement for a rule body to contain at least one observation about ego-vehicle results in the rule having to contain exactly one observation of the ego-vehicle state.

The second requirement is that the rule has to contain information of at least one other dynamic actor, in other words the environment. Again, the validity of this assumption depends on the properties of the implementation. In the case of this work, the ego-vehicle is assumed to have the state where it is driving towards its goal. As such, a scenario without other vehicles will not show up as a rule as it is covered by the default state in the part of traffic situations being focused on. If there are no other vehicles in an uncontrolled intersection, the ego-vehicle can always proceed. As a result, the rule bodies can be assumed to be required to contain at least one observation of other vehicles.

Table 4 presents the possible elements of the rule bodies being generated. However, this work focuses on a narrow and specific part of traffic situations. Due to this, not all of the extracted knowledge is useful. The third requirement states that the rules must only contain relevant information. Specifically *Go*-observations from other vehicles are not relevant. This is the result of the inspected scenarios always unfolding according to traffic rules. In other words, all vehicles present follow traffic rules and there is no unexpected or rule breaking behavior displayed. Due to this, it is not relevant whether a vehicle is moving or not as it is merely a direct consequence of traffic rules. That is, if there exists a vehicle in the scenario to which the ego-vehicle has to give way, it does not matter whether that vehicle is moving or not as it has to be given way in either case. The knowledge of the vehicle with right of way simply existing is enough and that information is inherently included in other knowledge. If a vehicle has a goal direction it is involved in the scenario and thus exists. It is, however, possible for unexpected and traffic rule breaking behavior to be present in the scenarios used for data gathering. In this case, the knowledge on whether a specific vehicle is moving becomes useful for avoiding dangerous situations. While having useless data can be viewed as a benefit for evaluating the quality of learning and robustness of methods, this aspect was not inspected in this research.

In the scenarios dealt with in this work, *Collision Warning* is only useful for situations where unexpected vehicle behavior is present. Collision are not expected to occur in the case where scenarios unfold in a way that complies to traffic rules. However, *Collision Warning* becomes of utmost importance in case of unexpected behavior. More interestingly, a *Collision Warning* with a pedestrian occurs in completely normal situations in real world traffic. For example, a vehicle has to give way to pedestrians when a vehicle is turning left or right in a traffic light controlled intersection due to both the vehicle and the pedestrians having a green light simultaneously even though their planned paths intersect. That said, this scenario could be handled by including knowledge on pedestrians in the intersection similarly to knowledge on other vehicles instead of using *Collision Warning*. In the end, both *Collision Warning* and *Go* were be assumed to be irrelevant in the scope of this work and were pruned from the data.

Finally, the fourth requirement states that rulesets must be pruned to be a set of rules where no rule is covered by another rule. The rule bodies are conjunctions of observations. As a result, any rule that is a superset of another rule is redundant. Conjunction elimination is based on the property of conjunctions that if a conjunction is true then it follows that any subset of that conjunction is also true [46]. Thus, it

is possible to eliminate most of the generated rules and have the final set of rules be minimal in relation to the number of members in each rule body. Not only does this allow elimination of redundant rules but it also offers an additional benefit. The resulting rules are more easily understood by humans and as such easier to verify by reviewing.

4.2 Rule evaluation

While the methods applied in this work can generate rules based on the reoccurring patterns in the used dataset, evaluating these rules is important for evaluating the suitability of each method for this application. This section discusses the different ways rules were evaluated as well as the difficulties of evaluating each ruleset as a whole.

4.2.1 Evaluation factors

The evaluation of the final ruleset correctness was based on many different factors. The evaluation of a ruleset can be divided into hard and soft requirements. To elaborate, a valid ruleset has to satisfy all hard requirements to be valid. Failing to satisfy a soft requirement does not imply that the ruleset is invalid but rather that the ruleset is not in an optimal and desired format. Both the hard and soft requirements are tailored to this work and are not to be treated as general requirements. They can, however, be used as a base for evaluation of rulesets in different cases as well. All evaluation of rules and rulesets was done by review by a human knowledgeable on traffic rules. To satisfy the hard requirements, the ruleset must satisfy the following.

- Compliance with traffic rules
- Compliance with the decision making system

The first rule states that a vehicle following the rules in the ruleset must not break any traffic rules. Within the considerations of this work this requirement results in safe operation. The second rule states that the rules must be compliant with the decision making system where these rules are used. Not only does this rule state that the format of the rules must be a specific format but it also implies that the rule body must consist of observations while the rule head must consist of actions.

These two requirements stated above assume that other vehicles involved in the current scenario follow traffic rules and will not unexpectedly break rules. Exceptional scenarios such as ones containing accidents are not included within the scope of this work and are left as future work. As such, the rule of safe vehicle operation is left out of the requirement list.

Soft requirements are more focused on ruleset quality than the hard requirements which are strictly focused on ensuring the rules can be used in the first place. The soft requirements are the following.

- The ruleset must be minimal in regards to body length

- The ruleset must not include rules covered by other rules

As can be seen from the list, the above statement of a possibility for a ruleset to be valid while not fulfilling the soft requirements is true. These two requirements are intertwined. The first requirement states that the rules included in the ruleset must be as short as possible. This has both benefits and downsides. The downside is that it can result in more rules than would be included in a set that is minimal in regards to the number of rules. However, at the same time each individual rule is simpler. This makes it easier to verify the rules by human review as well as reduces computation required for the decision making system to process a single rule. The second requirement states that rules must not be covered by other rules. This rule enables the first requirement as it gives answers the question of which rules to keep and which rules to remove when redundancy is detected.

4.2.2 Rule coverage

The generated rules and rulesets are evaluated by human review. This creates the problem of rule coverage. It is easy to spot nonsensical or useless rules. However, it is difficult to say if the rules cover all required situations. This is not specific to human review as it is a common problem in many fields such as computer science and software development. While this problem is acknowledged to be a real and significant issue, it is not addressed in this work due to the inherent complexity of the problem. The final correct rulesets in this work are, however, relatively short and can be deemed to have full coverage within the scope of available knowledge, scenarios, and control options dealt with in this work. This is possible due to the rules matching known real world traffic rules.

4.3 Parameters

This sections discusses the parameters involved in the experiments with different methods. The reasons behind the choices made regarding different parameters are also discussed.

4.3.1 Association rule learning

Association rule learning mines patterns from the data in a straightforward manner without many parameters to tune. However, during candidate itemset generation a useful parameter is the maximum length of itemsets. This corresponds to the maximum number of members in a rule body or head. Setting a maximum itemset length is primarily used to reduce the amount of computation required. The problem of finding itemsets suffers from combinatorial explosion and as such is not trivial [40]. This is aided by using the Apriori algorithm as well as setting a maximum itemset length. In the cases dealt with in this work, maximum length as short as 2 was observed through trials to be enough to generate rules covering all traffic rules for the scenarios included in the datasets. However, as the computation was not a problem the maximum length was set to 5.

In addition to maximum itemset length, the itemsets and the resulting rules are accepted or rejected by setting thresholds for the statistical measures introduced in Section 3.3.3. These measures of goodness are support and confidence. It was observed that relevant itemsets have a rather low support. The low support is due to the data covering a wide selection of unique scenarios. In contrast, through testing and observations the confidence values of the correct rules were noted to be high. Confidence of 1.0 was not uncommon for a correct rule due to the clean nature of the data when generating rules from the handmade scenario data. A low support is acceptable when paired with an extremely high confidence as the incorrect rules generated were observed to have significantly lower confidence. As a result, it is possible to get correct results relatively reliably using set parameters even with some changes to the data used. The envisioned real world dataset has more varying and noisy data than the dataset generated for this work. For this reason, it might be more difficult to tune the parameters correctly in different applications.

4.3.2 Deep learning

This section discusses the used neural network structure as well as the used parameters. Structure refers to all architectural aspects of the network that are present in the network at all points in time such as training, evaluation, and predicting. Parameters are values used in training the network.

Most significant structural decisions involve the number of layers, the number of neurons in each layer, the type of the layers, and the activation function for the neurons in each layer. A basic visualization of the structure used in this work is shown in Figure 14. The network was chosen through testing to consist of two hidden layers, an input layer, and an output layer. All of the layers are fully connected. This means that each neuron is connected to all neurons in the previous and the next layer. This structure was chosen due to the simplicity of the data implying the sufficiency of a simple structure as well as observations through testing showing no significant benefits for using more complex structures. Other structures tested included larger number of hidden layers and different number of neurons for each layer. A simpler structure was chosen over more complex one if its accuracy and loss at the end of the training were similar to the ones of the more complex one.

The next structural decision regarding the network is the number of neurons in each layer. The output layer was chosen to have two neurons due to the desired output being a vector of length two. The number of neurons in the input layer was set to be the number of observations in the used dataset. The first hidden layer was set to have 32 neurons and the second hidden layer was set to 16 neurons

Similarly to the number of neurons in the output layer, the choice of activation function for the output layer was a straightforward choice of the softmax function. Softmax is often used as the output layer activation function as it causes the output layer to simply output probabilities for each possibility being true according to the network training. This is due to softmax function modifying a vector to such form that the sum of its elements is exactly one. In the case of this work, the output vector is a binary vector of length two. As such, there are two different possible

results where the sum of probabilities adds up to one. For the training data one element of the correct output vector is always one while the other one is zero. These two output possibilities correspond to *Stop* and *Go* actions for the ego-vehicle. The input layer and the hidden layers were chosen to have rectified linear unit activation (ReLU) function. ReLU is the most popular activation function in deep networks [47].

In addition to the structure, the neural network has multiple parameters that can be tuned in the training stage of the network. These parameters affect the learning speed, the amount of computation required, and the final accuracy and output of the network. There are multiple choices to be made with the most significant choices being the loss function, the optimizer, the learning rate, batch size, and the number of epochs. There are other parameters, however these are the most significant ones and as such these are discussed in this section.

The loss function is the function that is being optimized while training the network through modifying the network weights through propagation. The loss function was chosen to be categorical cross-entropy. This enables the network to be a classification network that classifies the inputs into two different classes: *Stop* and *Go*.

As stated, the loss function is the function being optimized, typically minimized. This optimization can be performed by different methods called the optimizers. The optimizer of choice for this work is stochastic gradient descent without momentum. Stochastic gradient descent optimizes the loss function by, in essence, moving along the loss function towards a minimum by computing the gradient of the loss function and then taking a step towards lower loss. Specifically, stochastic gradient descent pics a random training example at each iteration as the basis of parameter update [48].

Learning rate is the parameter for deciding how much the network is adjusted in each iteration. In other words, a large learning rate initially causes a network to adjust rapidly. However, a large learning rate can have difficulties reaching the optimal values as it can overshoot them. In contrast, a low learning rate causes the network to learn slowly with higher probability of convergence. Learning rate can be adjusted during training through, for example, decaying the value to be smaller and smaller as the training progresses. Decay was not used in this work as it was not deemed necessary. Learning rate was chosen to be 0.01.

The batch size and epochs are related to the way the data is fed to the network during training as well as how many iterations are done. Batch size refers to feeding the data to the network in batches and adjusting the network for each batch instead of the whole data. Number of epochs dictates the number of times all of the data is fed to the network.

5 Results

This sections covers the rules generated from the gathered data using methods described in Section 3. For association rule learning, rules generated from both handmade scenarios as well as random scenarios are covered to showcase the reasons

behind the need to create the hand made scenario dataset. For deep learning, only the handmade scenario dataset is used.

5.1 Association rule learning

The rules generated through association rule learning and process of the rule refinement procedure is detailed in this section. First, the process of rule generation using the data extracted from random scenarios is detailed. Next, the rule generation using the data from handmade scenarios is discussed.

5.1.1 Handmade scenarios

Rules generated from the handmade set of scenarios created for this work are expected to result in easily verified and correct rules if the method is valid. The dataset captured from these scenarios is highly clean with no noise or exceptions. As such, this handmade scenario dataset gives the applied methods their best chance of succeeding and is, thus, used as a baseline in this work. This section details the rule generation and refinement process by showcasing and discussing the ruleset at each step of the process. First step of the rule generation process was to find the underlying patterns in the data by using the Apriori algorithm. First, association rule learning was applied to the dataset using support and confidence thresholds close to zero. No rule pruning was done. It was observed that each generated rule has low support. By inspecting the generated rules, a suitable support threshold value of 0.01 was found to result in reliably keeping the correct rules in the ruleset. Confidence values of correct rules were observed to be as high as 1.0 or close to it. Conversely, incorrect or redundant rules were observed to have significantly lower confidence values. Thus, a suitable confidence threshold was found to be 0.9. As mentioned earlier, the maximum itemset length was set to 5, which was observed to be longer than necessary.

The initially generated rules can contain nonsensical or redundant rules and the resulting ruleset is relatively large compared to the set of correct rules. For example, a rule containing two mutually exclusive observations is nonsensical. The steps of refining the initial results into the final correct rules was discussed in greater details in Section 4.1.2. First step of this refining process is to remove any rules with information not relevant in regards to the desired result. In the case of this work, these rules are the ones that contain information about vehicles other than the ego-vehicle moving or information about incoming collisions. These correspond to *Collision Warning* and ** Go* in Table 4 where * is any vehicle that is not the ego-vehicle.

The ruleset generated from the handmade scenario data after removing rules with irrelevant information is shown in Table 6. Rules in bold are part of the desired correct ruleset. The set contains all seven correct rules as well as six unwanted rules. Thus, 54% of the rules are correct. Additionally, two rules are acceptable but not optimal due to not being minimal in regards to body length due to an irrelevant third member. Additionally, these two rules are covered by the optimal rules.

Table 6: The ruleset generated with association rule learning from handmade scenarios. Rules with irrelevant information have been removed as the first step of the pruning process. The rules in bold are desired correct rules. Underlined rules are acceptable but not optimal. $Support \geq 0.01$, $confidence \geq 0.9$, and $length \leq 5$.

Supp	Conf	Rule
0.15	0.97	$Right : Left \implies Stop$
0.02	1.00	$Left : Left \wedge Right : Left \implies Stop$
0.02	1.00	$Left : Right \wedge Right : Left \implies Stop$
0.02	1.00	$Opposite : Straight \wedge Right : Left \implies Stop$
0.01	1.00	$\underline{Ego : Straight \wedge Left : Left \wedge Right : Left \implies Stop}$
0.01	1.00	$\underline{Ego : Straight \wedge Right : Left \wedge Left : Straight \implies Stop}$
0.04	1.00	$\boldsymbol{Ego : Left \wedge Opposite : Straight \implies Stop}$
0.05	1.00	$\boldsymbol{Ego : Left \wedge Opposite : Right \implies Stop}$
0.09	1.00	$\boldsymbol{Ego : Left \wedge Right : Left \implies Stop}$
0.08	1.00	$\boldsymbol{Ego : Left \wedge Right : Straight \implies Stop}$
0.06	1.00	$\boldsymbol{Ego : Straight \wedge Right : Left \implies Stop}$
0.02	1.00	$\boldsymbol{Ego : Straight \wedge Right : Straight \implies Stop}$
0.01	1.00	$\boldsymbol{Ego : Straight \wedge Right : Right \implies Stop}$

It can be seen in the ruleset shown in Table 6 that some of the rules are clearly irrelevant or incorrect. For example, the first rule states that if there is a vehicle on the right of the ego-vehicle heading left, the ego-vehicle needs to stop. While this is correct in most situations, the ego-vehicle is not required stop in this situation if it is turning right. Moreover, this rule contains no information about the state of the ego-vehicle and as such is needlessly general and does not take advantage of there always being information available on ego-vehicle state. The next step is to exploit assumptions about the data and the environment where this data was gathered. As stated, there is always knowledge available on the state of the ego-vehicle. Similarly, if there is a need to stop, there is always knowledge available concerning at least one other vehicle. Again, the ego-vehicle proceeding according to planned route is the default state and as such, if there is a need for a rule, there is a need to stop. The ruleset resulting from requiring the rule body to contain observation on both ego-vehicle and one other vehicle is shown in Table 7. Additionally, this ruleset has been required to contain an ego-vehicle action, *Stop* in this case, in the rule head. However, this requirement did not change the actual rules with these parameters as all rules already satisfied this constraint. As this constraint requires *Stop* to be in the rule head, it inherently removes rules where *Stop* is in the rule body. In the case of more than one action possibility being included, the rules containing any actions in the body are removed.

The ruleset shown in Table 7 has had several rules removed from the previous step. It contains all seven correct rules and two unwanted rules resulting in 78% of the rules being correct. The two unwanted rules are acceptable due to being

Table 7: The ruleset generated with association rule learning from handmade scenarios refined further by applying assumptions about the availability of knowledge. The rules in bold are desired correct rules. Underlined rules are acceptable but not optimal. $Support \geq 0.01$, $confidence \geq 0.9$, and $length \leq 5$.

Supp	Conf	Rule
0.01	1.00	$\underline{Ego : Straight \wedge Left : Left \wedge Right : Left \implies Stop}$
0.01	1.00	$\underline{Ego : Straight \wedge Right : Left \wedge Left : Straight \implies Stop}$
0.04	1.00	$\mathbf{Ego : Left \wedge Opposite : Straight \implies Stop}$
0.05	1.00	$\mathbf{Ego : Left \wedge Opposite : Right \implies Stop}$
0.09	1.00	$\mathbf{Ego : Left \wedge Right : Left \implies Stop}$
0.08	1.00	$\mathbf{Ego : Left \wedge Right : Straight \implies Stop}$
0.06	1.00	$\mathbf{Ego : Straight \wedge Right : Left \implies Stop}$
0.02	1.00	$\mathbf{Ego : Straight \wedge Right : Straight \implies Stop}$
0.01	1.00	$\mathbf{Ego : Straight \wedge Right : Right \implies Stop}$

compliant with traffic rules, however they are covered by the correct rules and thus redundant. Additionally, the third member in both of the two underlined rules is irrelevant.

Notably the first rule in Table 6 has been removed and as such an incorrect rule has been eliminated. Closer inspection of the rules reveals redundant rules. For example, the last rule in the ruleset states that if ego-vehicle is heading straight, a vehicle on the right is heading left, and a vehicle on the left is heading straight, the ego-vehicle needs to stop. This case is, however, covered by the fifth rule stating that if ego-vehicle is heading straight and a vehicle on the right is heading left, the ego-vehicle needs to stop. This is due to the earlier discussed knowledge on conjunction properties stating that if a conjunction evaluates to true, any if its subsets must also evaluate to true. As such, the final step of refining the ruleset is to remove redundancy. This can be done by checking if rules are subsets of each other and removing any found supersets. This results in a ruleset with no redundancy that is minimal in regards to the body length. The final ruleset is shown in Table 8.

The ruleset in Table 8 is exactly the desired result with 100% of the rules being correct. The ruleset is minimal in regards to the body length thus containing no redundancy. All rules exploit knowledge on data availability and imply an action. In addition, all seven rules in the ruleset can be deemed correct by human review. The correctness is evaluated by correspondence to real world traffic rules for right side traffic. This ruleset completely describes traffic rules for all scenarios in 3-way and 4-way intersection for two and three vehicles when a U-turn is not allowed. The support of all seven rules adds up to 0.35. If the rules are of full coverage, then the default action taken whenever one of these rules is not applied must cover all of the other situations.

While these rules could be written manually with little effort, the contribution of this work is that these rules have been generated automatically using general

Table 8: The final minimal ruleset with no redundancy generated with association rule learning from handmade scenarios. The rules in bold are desired correct rules. $Support \geq 0.01$, $confidence \geq 0.9$, and $length \leq 5$.

Supp	Conf	Rule
0.04	1.00	$\boldsymbol{Ego : Left \wedge Opposite : Straight \implies Stop}$
0.05	1.00	$\boldsymbol{Ego : Left \wedge Opposite : Right \implies Stop}$
0.09	1.00	$\boldsymbol{Ego : Left \wedge Right : Left \implies Stop}$
0.08	1.00	$\boldsymbol{Ego : Left \wedge Right : Straight \implies Stop}$
0.06	1.00	$\boldsymbol{Ego : Straight \wedge Right : Left \implies Stop}$
0.02	1.00	$\boldsymbol{Ego : Straight \wedge Right : Straight \implies Stop}$
0.01	1.00	$\boldsymbol{Ego : Straight \wedge Right : Right \implies Stop}$

methods not specific to this particular application. As such, these methods can be applied to generate rules for other, more comprehensive, applications. However, the scalability of these methods is not researched in this work. These results only serve as proof of concept and require further research and expansion to different situations including both traffic rule following as well as traffic rule breaking scenarios.

5.1.2 Random scenarios

The results for the handmade scenarios are promising and suggest that mining association rules is a valid method to generate rules from the type of data dealt with in this work. However, handmade scenarios result in unrealistically clean data. This is evident by the confidence values of 1.00 in Tables 6, 7, and 8. The first approach to data generation in this work was generating data from random scenarios with more dynamic vehicle behavior. This section covers the rules generated from the data gathered from random scenarios and inspects the suitability of the dataset for creation of rules that describe real world traffic rules.

Similarly to the handmade scenario case, the first step was to find the patterns in the dataset with the useless information removed. Due to the nature of the data, the parameters used are different from the handmade scenario case. Support threshold used was 0.01, confidence threshold was 0.75 and maximum itemset length was 5. This generated dozens of rules where most of the rules are not valid. In contrast to the handmade scenario case, the ruleset now contains rules where the rule head consists of an observation instead of an action. According to evaluation factors discussed in Section 4.2 rules of this nature are not valid rules as they are not compliant with the decision making system. Again, the next step is to constrain rule bodies to contain an observation of both ego-vehicle and at least one other vehicle as it is known that this information is always available when a rule is needed. The resulting set of rules is shown in Table 9.

The ruleset contains six rules that are clearly traffic rule compliant. Thus, 32% of the rules are acceptable but not optimal. Additionally, the set contains seven

Table 9: The set of rules generated from random scenarios through association rule learning. Useless knowledge has been removed. Body is required to have observation on ego-vehicle and at least one other vehicle. Head is required to consist of only actions. Rules in bold are compliant with traffic rules. Underlined rules are acceptable. $Support \geq 0.01$, $confidence \geq 0.75$, and $length \leq 5$.

Supp	Conf	Rule
0.01	0.76	$Left : Right \wedge Opposite : Left \wedge Ego : Straight \implies Stop$
0.03	0.76	$Opposite : Left \wedge Ego : Straight \implies Stop$
0.20	0.76	$Opposite : Straight \wedge Ego : Straight \implies Stop$
0.02	0.79	$Opposite : Straight \wedge Ego : Right \implies Stop$
0.04	0.76	$Opposite : Straight \wedge Ego : Straight \wedge Left : Left \implies Stop$
0.07	0.77	$Opposite : Straight \wedge Ego : Straight \wedge Left : Straight \implies Stop$
0.01	0.79	$Ego : Left \wedge Opposite : Straight \wedge Right : Straight$ $\wedge Left : Straight \implies Stop$
0.01	0.79	$Opposite : Straight \wedge Ego : Straight \wedge Left : Left$ $\wedge Right : Left \implies Stop$
0.02	0.77	$Opposite : Straight \wedge Ego : Straight \wedge Left : Left$ $\wedge Right : Straight \implies Stop$
0.02	0.79	$Opposite : Straight \wedge Ego : Straight \wedge Right : Left$ $\wedge Left : Straight \implies Stop$
0.03	0.79	$Opposite : Straight \wedge Ego : Straight \wedge Right : Straight$ $\wedge Left : Straight \implies Stop$
0.01	0.77	$\underline{Left : Right \wedge Opposite : Straight \wedge Ego : Straight}$ $\underline{\wedge Right : Left \implies Stop}$
0.01	0.78	$\underline{Left : Right \wedge Opposite : Straight \wedge Ego : Straight}$ $\underline{\wedge Right : Straight \implies Stop}$
0.02	0.80	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Left : Left \implies Stop}$
0.02	0.79	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Right : Left \implies Stop}$
0.02	0.79	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Right : Straight \implies Stop}$
0.05	0.79	$\boldsymbol{Ego : Straight \wedge Opposite : Straight \wedge Right : Left \implies Stop}$
0.07	0.78	$\boldsymbol{Ego : Straight \wedge Opposite : Straight \wedge Right : Straight \implies Stop}$
0.02	0.79	$\boldsymbol{Ego : Straight \wedge Opposite : Straight \wedge Right : Right \implies Stop}$

rules with body length of four from four vehicle scenarios. Out of these, two are underlined. The underlined rules describe situations that are solvable through traffic rules as the vehicle to the left of the ego-vehicle does not need to give way to anyone. The ego-vehicle stopping in this situation is the correct response and as such, these rules can be considered acceptable. This bring the percentage of acceptable rules to 42%.

The next step was to check whether any of the rules are subsets of other rules and then remove the longer rules in order to comply with the requirement of being minimal in regards to the rule body length. The final minimal set of rules with no redundancy is shown in Table 10. The rules compliant with real world traffic rules are in bold.

Table 10: The final set of rules generated from the random scenario dataset through association rule learning. Ruleset is minimal in regards to the rule body length and there is no redundancy within the set. Rules in bold are compliant with traffic rules. $Support \geq 0.01$, $confidence \geq 0.75$, and $length \leq 5$.

Supp	Conf	Rule
0.03	0.76	$Opposite : Left \wedge Ego : Straight \implies Stop$
0.20	0.76	$Opposite : Straight \wedge Ego : Straight \implies Stop$
0.02	0.79	$Opposite : Straight \wedge Ego : Right \implies Stop$
0.02	0.80	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Left : Left \implies Stop}$
0.02	0.79	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Right : Left \implies Stop}$
0.02	0.79	$\boldsymbol{Ego : Left \wedge Opposite : Straight \wedge Right : Straight \implies Stop}$

The ruleset contains three rules that are compliant with traffic rules as well as three rules that are not. Thus, 50% of the rules are acceptable but not optimal. Each of the acceptable rules have three members in their body out of which one is not needed to determine an action in the described scenario.

The ruleset as a whole can be quickly determined to be invalid due to it failing the requirement for compliance with traffic rules. The very first rule breaks traffic rules by indicating that an opposite vehicle heading left has right of way over the ego-vehicle heading straight. In reality, the ego-vehicle has right of way. However, this ruleset might be true to the scenarios where the data was gathered from.

The confidence values dealt with are significantly lower than in the handmade scenario case and as such the threshold has to be lower. Finding an optimal threshold was difficult due to the difference of confidence for correct and incorrect rules being small. In addition, it was difficult to determine which rules are correct and which are not as the data comes from scenarios that do not unfold according to any set of rules related to the knowledge included in the dataset. Instead, the order the vehicles cross the intersection seems to be tied to the order they arrive to the intersection. However,

even after observations this is difficult to confirm outside ruling out first-in-first-out. As such, the full logic of the vehicle order is not known from observation nor the generated rules.

This showcases the difficulty of evaluating anything other than absolute basic functionality of the rule generation methods using the dataset of random scenarios. As a result, it is clear why the handmade set of scenarios was made to make it possible to evaluate the rule generation methods further than simply confirming that they find patterns in the data.

The finding that the patterns in the dataset are dictated by factors other than the ones extracted raises questions relating to eventual real world implementations. Inspecting temporal patterns might be important in some real world situations just as they appeared to be important in describing the autopilot behavior. Additionally, boolean data might not be sufficient to represent complex scenarios in traffic.

5.2 Deep learning

This section discusses the results achieved with deep learning. The rules generated through deep learning for the handmade scenario dataset are presented. The random scenario dataset is not used. This is due to the results of Section 5.1 suggesting that the extracted knowledge is not sufficient to determine the patterns dictating the vehicle behavior in the random scenario dataset. In addition to presenting the rulesets, this section explains and showcases the process of refining the results to their final form. Finally, the generated rulesets are evaluated according to the metrics introduced in Section 4.2.1.

5.2.1 Handmade scenarios

This section covers the process of generating a set of rules from the dataset gathered from handmade scenarios. First, the network has to be trained using the dataset. Then, rules can be generated through network predictions.

Section 3.3.4 introduces the process of generating rules through deep learning. First step is to train the network. Training involves feeding the network all observations in the dataset and checking whether the network outputs the correct action. The network is then adjusted through propagation towards a more correct model. Training was done in batches of 8 with 10 epochs. The loss during training as well as network accuracy are shown in Figure 15 with the loss value for each epoch depicted in blue and the accuracy of the network depicted in orange. It can be seen from the figure that the network does learn the patterns in the data. This is evident from steadily decreasing loss as well as steadily increasing accuracy. The number of epochs can be said to be sufficient as the loss and accuracy values have reached a stable state and no significant further learning can be expected by increasing the number of epochs.

Once the network was trained, rules were generated by querying the network for predictions for given inputs. In general, this was done by first generating a set of rule bodies and giving these as inputs for the network. As a result, the network outputs

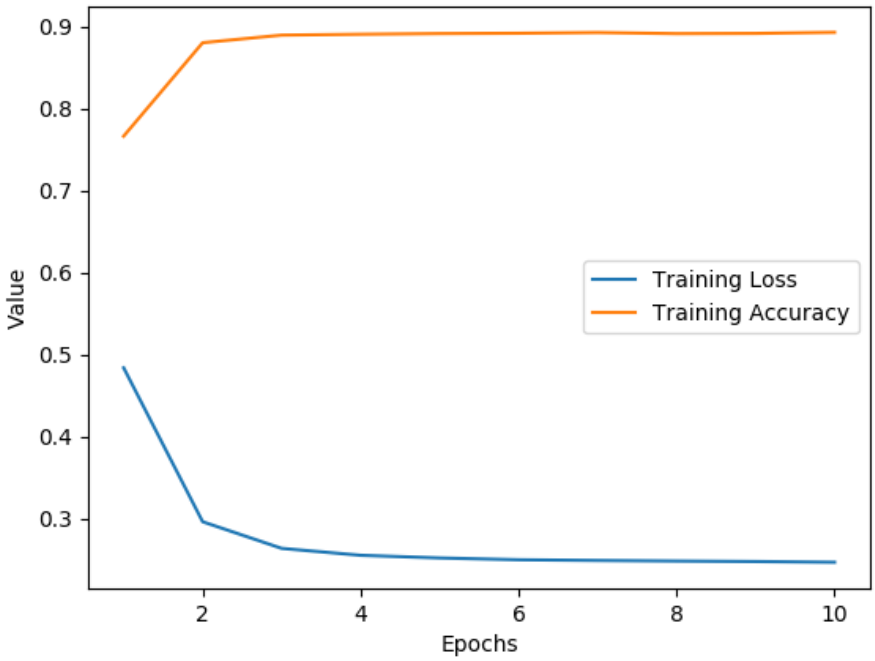

Figure 15: The training process of the network using the dataset gathered from handmade scenarios. Batch size 8, number of epochs 10. Yellow line corresponds to the accuracy during training while blue line indicates the loss function value during training.

the rule head. Through the softmax activation in the output layer, it is possible to read the probability with which the network evaluates the possible actions to be correct. The sum of these values always add up to exactly one. This allowed the extraction of one action predicted to be most likely correct as well as setting a threshold for the confidence value of the result. A suitable confidence threshold was observed to be 0.995 and that was used throughout the rule generation process for this dataset. The confidence threshold was determined by observing the set of all produced rules and determining a suitable value.

Initially, all possible rule bodies were generated. In this initial set, however, most rules were either redundant or nonsensical. Thus, the set of rule bodies was pruned before querying the network for the rule heads to avoid unnecessary computation. First, the rule bodies were restricted to a maximum length. Similarly to Section 5.1, the maximum length for a rule body was set to 5. This was known to be longer than necessary to extract correct rules in this work.

The initial ruleset generated with these restrictions is shown in Table 11. It contains the seven correct rules in bold, 11 incorrect rules, and two rules that can be interpreted as correct indicated by underlining. This results in 35% of the rules being optimal and 45% of the rules being correct but not optimal. The two rules that can be interpreted as correct describe a situation where ego-vehicle is driving

Table 11: The initial ruleset generated through deep learning from the handmade scenario dataset. The rules in bold are desired correct rules. Underlined rules can be interpreted as being correct. $Length \leq 5, confidence \geq 0.995$.

Confidence	Rule
1.00	$Right : Right \wedge Right : Straight \implies Stop$
1.00	$Right : Left \implies Stop$
1.00	$Right : Right \wedge Opposite : Right \implies Stop$
1.00	$Opposite : Right \wedge Right : Straight \implies Stop$
1.00	$Right : Right \wedge Opposite : Straight \implies Stop$
1.00	$Opposite : Straight \wedge Right : Straight \implies Stop$
1.00	$Opposite : Straight \wedge Opposite : Right \implies Stop$
1.00	$Left : Right \wedge Right : Right \implies Stop$
1.00	$Left : Right \wedge Right : Straight \implies Stop$
1.00	$Left : Right \wedge Opposite : Straight \implies Stop$
1.00	$Ego : Left \wedge Ego : Straight \implies Stop$
1.00	$\underline{Left : Right \wedge Ego : Straight \wedge Right : Go \implies Stop}$
1.00	$\underline{Ego : Straight \wedge Opposite : Right \wedge Right : Go \implies Stop}$
1.00	$\boldsymbol{Ego : Straight \wedge Right : Right \implies Stop}$
1.00	$\boldsymbol{Ego : Straight \wedge Right : Straight \implies Stop}$
1.00	$\boldsymbol{Ego : Straight \wedge Right : Left \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Right : Straight \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Right : Left \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Opposite : Right \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Opposite : Straight \implies Stop}$

straight across the intersection while a vehicle on the right is moving. This implies that a vehicle on the right exists. In this case, it does not matter where the vehicle on the right is turning as ego-vehicle has to always give way to it. These two rules are redundant in that this situation is already described in the optimal correct rules, the two rules both describe the same situation, and the third member in both rules is irrelevant.

The ruleset can be observed to contain rules that do not take advantage of all available information, similarly to the case in Section 5.1.1. It is known that knowledge about both the ego-vehicle and at least one other vehicle is available whenever a rule is necessary. Thus, the rule bodies can be required to contain at least one observation of ego-vehicle state as well as one observation of the state of at least one other vehicle. Due to the rule bodies being generated through generating all possible binary sequences of certain length, there are more factors to consider that can make a rule invalid. The set of rule bodies can be generated directly to include only valid rules. However, it was decided that this top-down approach was more straightforward than a bottom-up approach where vectors representing rule bodies are constructed by concatenating valid vectors in different combinations.

An example of an invalid rule is the first rule of the ruleset shown in Table 11. This rule Introduces a scenario where a vehicle on the right of the ego-vehicle is turning right as well as driving straight. Clearly these two observations are contradictory and the rule is invalid. Thus, the rules were be restricted to not only contain at least one observation of ego-vehicle and another vehicle but they were restricted to contain exactly one observation per vehicle. This is dependent on the extracted knowledge and is only valid in this general form if all observations of a vehicle are mutually exclusive. This is the case for this work only if the observation for a vehicle moving is not used. As stated in Section 5.1.1 this information is not useful and can be safely disregarded. If this was not the case, the observations must be inspected in terms of mutual exclusivity and rule bodies must be flagged valid or invalid according to the found mutually exclusive properties. Similarly, information for a detected collision warning can be discarded in this case. The set of rules with these restrictions is shown in Table 12. Finally, redundant longer rules which have subsets as separate rules were be removed. These restrictions include all necessary restrictions and result in valid rules.

Table 12: The final minimal ruleset with no redundancy generated with deep learning from handmade scenarios. The rules in bold are desired correct rules. $Length \leq 5, confidence \geq 0.995$.

Confidence	Rule
1.00	$\boldsymbol{Ego : Straight \wedge Right : Right \implies Stop}$
1.00	$\boldsymbol{Ego : Straight \wedge Right : Straight \implies Stop}$
1.00	$\boldsymbol{Ego : Straight \wedge Right : Left \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Right : Straight \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Right : Left \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Opposite : Right \implies Stop}$
1.00	$\boldsymbol{Ego : Left \wedge Opposite : Straight \implies Stop}$

The final set of rules shown in Table 12. 100% of the rules are correct. The ruleset is the exact same set as in Table 8. These rules are correct and of full coverage to the extent it is possible with this data as stated in Section 5.1.1. A notable difference being that from training to training the resulting rules can differ slightly by additional rules appearing rarely. As the rules, however, are almost always correct and during testing additional rules only appeared in addition to the desired ones, a solution could be to do multiple training passes and inspect the average or minimal results. This was not looked into further.

The resulting rules all have confidence of 1.00 similarly to the results in Table 8. In the case of Table 8 this could be explained with the clean and simple nature of the data. A data of this nature allows the network to accurately adjust its internal model to match the datapoints. This could be seen as overfitting. The implications of overfitting in regards to this application are discussed in Section 3.3.4 with the conclusion that it is not harmful due to the network not being used to predict missing

values in the data. This could be different in the envisioned real world application and must be examined on an application to application basis. Additionally, Figure 15 indicates that the network never reaches absolute accuracy during training. This indicates that the neural network never learned the scenarios completely. This knowledge combined with the confidence values of 1.00 in Table 12 suggests that the network is inaccurate in some other cases. Possible cases not learned completely are situations where ego-vehicle is moving as well as situations with observations of more than one vehicle in addition to ego-vehicle.

6 Future work

The creation of the datasets explained in Section 3 as well as the resulting rules shown in Sections 5.1 and 5.2 showcase the validity of these methods for the application in question. However, on the way towards the envisioned real world system there remains key points to be addressed. This section introduces some future directions the results in this work can be taken towards.

6.1 Integrating ontologies

The rules generated in this work are aimed to be used in an ontology-based decision making system. However, currently ontologies are not properly integrated into the rule generation process. This integration can be done in the knowledge extraction phase of the dataset creation while keeping the actual rule learning phase intact. For example, the map within which the scenarios take place should be described in the knowledge base in a semantic fashion using a suitable ontology as a vocabulary. An example of a part of an intersection described as RDF-triples is shown in Table 13. In this example, two roads are described to be connected to same intersection. Additionally, it is know that turning right from *Road-1* in *Int-1* takes the vehicle to *Road-2*. This can be integrated into the process during the detection and knowledge extraction phases. Knowing that the ego-vehicle is entering *Int-1* from *Road-1* allows a vehicle detected to the right of the ego-vehicle to be placed on *Road-2* within the knowledge base by creating temporary new knowledge that refers to the static map knowledge. Now, the knowledge extraction phase can access the road names the vehicles are on and determine that there are two vehicles entering the intersection and there is another vehicle to the right of the ego-vehicle. If the integration of ontologies is done in the knowledge extraction, the data can be processed to be in the same form as used in Section 3.3. Thus, the rule generation methods can be applied as described in this work.

6.2 Expanding knowledge extraction

The knowledge extracted from the scenarios into the datasets is detailed in Tables 4 and 5. This knowledge captures key information about a specific traffic scenarios. However, it is not sufficient to describe many other scenarios. As such, a more

Table 13: Part of a description of an intersection as RDF-triples. Each line is a single triple containing a subject, a predicate, and an object. Shown information contains an intersection with two roads attached to it.

Subject	Predicate	Object
Road-1	type	Road
Road-2	type	Road
Int-1	type	Intersection
Road-1	isConnectedTo	Int-1
Road-2	isConnectedTo	Int-1
Road-1	turnRightTo	Road-2
Road-2	turnLeftTo	Road-1

comprehensive list of knowledge items should be defined and methods to extract them correctly and efficiently should be looked into. In addition, *Collision Warning* can be improved further as is elaborated on in Section 3.2.3.

The integration with ontologies discussed in Section 6.1 should be considered when expanding the extracted knowledge. For example, on multilane roads the knowledge base should contain knowledge on nearby vehicle locations relative to the ego-vehicle as well as knowledge on which lane vehicles are on. This knowledge is crucial in lane change situations.

Modifying the knowledge format to contain aspects not considered in this research could prove useful. The semantic format of the data displayed advantages such as enabling application of association rule learning and making human review of results possible. However, the data format used in this work can be expanded and should be evaluated against other possibilities. For example, temporal properties of the data can be used to extract patterns containing important information that is not extractable from single samples. Additionally, the sufficiency of boolean data to represent traffic situations is a subject for further research.

6.3 Expanded dataset

Expansion of the datasets from which rules are generated is crucial and should eventually be done with real world data. Data gathered in real world traffic contains varied situations and presents unique challenges for rule learning. Intermediate steps can be taken with simulations. Specifically, simulation environment is well suited to explore unexpected and dangerous situations.

The datasets used for rule generation in this work assume having access to data that is difficult to obtain using current sensors. An example of such data is the goal direction of other vehicles in an intersection. This can result in the data being either incomplete or noisy.

Expanded datasets allow testing of rule generation methods in terms of their scaling and tolerance for uncertainty in the form of noisy, incomplete, or varied data.

The datasets generated during this work are not sufficient to answer these questions on the scalability and robustness of different methods including the ones used in this work.

6.4 Refining rule generation

This work introduces two different approaches to extracting rules from data. However, data mining and machine learning is a widely researched field and contains myriad of methods potentially applicable for the purpose in question. As a result, future directions for refining rule generation methods include testing the methods introduced in this work with more comprehensive datasets, refining these methods, exploring different approaches, and combining different methods. For example, learning classifier systems introduced in Section 2.2.1 present an established rule learning framework not included in this work. Similarly, automated rule correctness and coverage evaluation are not addressed in this work.

7 Conclusion

This work set out to answer questions on how a semantically abstracted dataset describing traffic situations can be built and how to generate logical rules from that dataset. As such, this work was divided into two major parts: the data generation and rule generation.

A three step process was proposed for the generation of a dataset. In this process, traffic scenarios were simulated first. Next, semantically abstracted knowledge was extracted from these scenarios. Finally, the knowledge was stored in a standardized format to form a dataset. For scenario simulation, two different approaches were introduced. One used randomness in specific aspects of the scenarios to produce randomized scenarios. The other approach involved defining scenarios by hand to achieve a clean dataset with little abnormalities and the desired content.

For the second part of the work, rule generation, two different approaches to learning patterns from data were developed. The first approach was based on a well known method of association rule learning while the second utilized neural networks and deep learning. A process for further refining results was developed. This process involved exploiting properties of logical constructs as well as applying assumptions about the data content and the application. Both rule learning methods successfully produced the same correct sets of rules describing the traffic rules involved in traffic scenarios taking place in an uncontrolled four way intersection. However, more research is required on the way towards real world systems.

The research conducted in this work acts as a proof of concept and is not complete. The scenarios inspected in this work cover only a narrow subset of all traffic scenarios. Additionally, many assumptions on data availability and scenario properties were made. Many aspects of expanding both the data generation as well as rule generation are left as open research questions. These include topics such as taking advantage of ontologies during knowledge extraction, the scalability and robustness of the rule generation methods, and automated evaluation of rule correctness and coverage.

References

[1] L. Zhao, R. Ichise, Z. Liu, S. Mita, and Y. Sasaki, "Ontology-based driving decision making: A feasibility study at uncontrolled intersections," *IEICE Transactions on Information and Systems*, vol. E100.D, pp. 1425–1439, 07 2017.

[2] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018. [Online]. Available: https://doi.org/10.1146/annurev-control-060117-105157

[3] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[4] J. Wei, J. M. Dolan, and B. Litkouhi, "Autonomous vehicle social behavior for highway entrance ramp management," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, June 2013, pp. 201–207.

[5] C. Vallon, Z. Ercan, A. Carvalho, and F. Borrelli, "A machine learning approach for personalized autonomous lane change initiation and control," in *2017 IEEE Intelligent Vehicles Symposium (IV)*, June 2017, pp. 1590–1595.

[6] E. Della Valle. Continuous sparql (c-sparql). [Online]. Available: http://streamreasoning.org/resources/c-sparql

[7] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosofand, and M. Dean. (2004, May) SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Submission. [Online]. Available: http://www.w3.org/Submission/SWRL/

[8] Ichise Laboratory. Advanced driving assistant system ontology. [Online]. Available: http://ri-www.nii.ac.jp/ADAS/index.html

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[10] O. R. Zaiane, "Building a recommender agent for e-learning systems," in *International Conference on Computers in Education, 2002. Proceedings.*, Dec 2002, pp. 55–59 vol.1.

[11] L. Francis, "The basics of neural networks demystified," *Contingencies*, vol. 11, no. 12, pp. 56–61, 2001.

[12] J. Pospíchal and V. Kvasnička, "70th anniversary of publication: Warren mcculloch & walter pitts - a logical calculus of the ideas immanent in nervous activity," in *Emergent Trends in Robotics and Intelligent Systems*, P. Sinčák, P. Hartono,

M. Virčíková, J. Vaščák, and R. Jakša, Eds. Cham: Springer International Publishing, 2015, pp. 1–10.

[13] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," *IEEE Micro*, vol. 35, no. 6, pp. 60–68, Nov 2015.

[14] Autoware Foundation. Autoware. [Online]. Available: https://github.com/autowarefoundation/autoware

[15] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki, "Core ontologies for safe autonomous driving." in *International Semantic Web Conference (Posters & Demos)*, 2015.

[16] M. Clement and R. Ichise, "Faster ontology reasoning with typed propositionalization," *Proceedings of the Annual Conference of JSAI*, vol. JSAI2018, pp. 1F102–1F102, 2018.

[17] C. Schlenoff, S. Balakirsky, M. Uschold, R. Provine, and S. Smith, "Using ontologies to aid navigation planning in autonomous vehicles," *The knowledge engineering review*, vol. 18, no. 3, pp. 243–255, 2003.

[18] C. Schlenoff, "Linking sensed images to an ontology of obstacles to aid in autonomous driving," in *Proceedings of the 18th National Conference on Artificial Intelligence: Workshop on Ontologies for the Semantic Web*, 2002.

[19] L. Zhao, R. Ichise, S. Mita, and Y. Sasaki, "An ontology-based intelligent speed adaptation system for autonomous cars," in *Semantic Technology*, T. Supnithi, T. Yamaguchi, J. Z. Pan, V. Wuwongse, and M. Buranarach, Eds. Cham: Springer International Publishing, 2015, pp. 397–413.

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015. [Online]. Available: http://arxiv.org/abs/1512.02325

[21] R. B. Rusu, "Semantic 3d object maps for everyday manipulation in human living environments," Ph.D. dissertation, Computer Science department, Technische Universitaet Muenchen, Germany, October 2009.

[22] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3d object detection network for autonomous driving," *CoRR*, vol. abs/1611.07759, 2016. [Online]. Available: http://arxiv.org/abs/1611.07759

[23] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, "Practical search techniques in path planning for autonomous driving," in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*. Chicago, USA: AAAI, June 2008.

[24] Cambridge University Press, "Cambridge dictionary: English dictionary," 2019. [Online]. Available: https://dictionary.cambridge.org/dictionary/english/

[25] V. Rausch, A. Hansen, E. Solowjow, C. Liu, E. Kreuzer, and J. K. Hedrick, "Learning a deep neural net policy for end-to-end control of autonomous vehicles," in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 4914–4919.

[26] H. Xu, Y. Gao, F. Yu, and T. Darrell, "End-to-end learning of driving models from large-scale video datasets," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[27] R. J. Urbanowicz, "Introducing rule-based machine learning: Capturing complexity," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, ser. GECCO '17. New York, NY, USA: ACM, 2017, pp. 576–604. [Online]. Available: http://doi.acm.org/10.1145/3067695.3067719

[28] H. H. Dam, H. A. Abbass, C. Lokan, and X. Yao, "Neural-based learning classifier systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 1, pp. 26–39, Jan 2008.

[29] R. J. Urbanowicz and J. H. Moore, "Learning classifier systems: A complete introduction, review, and roadmap," *Journal of Artificial Evolution and Applications*, vol. 2009, 2009. [Online]. Available: https://doi.org/10.1155/2009/736398

[30] M. Versichele, L. de Groote, M. C. Bouuaert, T. Neutens, I. Moerman, and N. V. de Weghe, "Pattern mining in tourist attraction visits through association rule learning on bluetooth tracking data: A case study of ghent, belgium," *Tourism Management*, vol. 44, pp. 67 – 81, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0261517714000417

[31] M. Kaur and S. Kang, "Market basket analysis: Identify the changing trends of market data using association rule mining," *Procedia computer science*, vol. 85, pp. 78–85, 2016.

[32] H. Kashyap, H. Ahmed, N. Hoque, S. Roy, and D. K. Bhattacharyya, "Big data analytics in bioinformatics: Architectures, techniques, tools and issues," *Network Modeling Analysis in Health Informatics and Bioinformatics*, vol. 5, 06 2015.

[33] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," *SIGMOD Rec.*, vol. 22, no. 2, pp. 207–216, Jun. 1993. [Online]. Available: http://doi.acm.org/10.1145/170036.170072

[34] Guoqing Chen, De Liu, and Jiexun Li, "Influence and conditional influence-new interestingness measures in association rule mining," in *10th IEEE International Conference on Fuzzy Systems. (Cat. No.01CH37297)*, vol. 3, Dec 2001, pp. 1440–1443 vol.2.

[35] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling

autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems*, April 2018, pp. 287–296.

[36] M. Quigley, K. Conley, B. P Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3, 01 2009.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[38] CARLA Team. Carla 0.9.2: Upgraded ros-bridge, traffic scenario engine and a new agent class to perform driving navigation from client side! [Online]. Available: http://carla.org/2018/12/24/release-0.9.2/

[39] W3C. Semantic web: Vocabularies. [Online]. Available: https://www.w3.org/standards/semanticweb/ontology

[40] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, Jan 2008. [Online]. Available: https://doi.org/10.1007/s10115-007-0114-2

[41] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB'94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Morgan Kaufmann, 1994, pp. 487–499. [Online]. Available: http://www.vldb.org/conf/1994/P487.PDF

[42] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal, "Discovering frequent closed itemsets for association rules," in *Database Theory — ICDT'99*, C. Beeri and P. Buneman, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 398–416.

[43] V. Jovanoski and N. Lavrač, "Classification rule learning with apriori-c," in *Progress in Artificial Intelligence*, P. Brazdil and A. Jorge, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 44–51.

[44] F. Bodon, "A fast apriori implementation," in *Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations*, 2003.

[45] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[46] M. Genesereth and E. Kao, "Introduction to logic, second edition," *Synthesis Lectures on Computer Science*, vol. 4, no. 2, pp. 1–163, 2013. [Online]. Available: https://doi.org/10.2200/S00518ED2V01Y201306CSL006

[47] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–44, 05 2015.

[48] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," in *Advances in Neural Information Processing Systems 20 (NIPS 2007)*, J. Platt, D. Koller, Y. Singer, and S. Roweis, Eds. NIPS Foundation (http://books.nips.cc), 2008, pp. 161–168. [Online]. Available: http://leon.bottou.org/papers/bottou-bousquet-2008