Driving Context Detection and Validation using Knowledge-based Reasoning

Abderraouf Khezaz[1,2], Manolo Dulva Hina[1], Hongyu Guan[2] and Amar Ramdane-Cherif[2]
[1]*ECE Paris, 37 quai de Grenelle, 75015, Paris, France*
[2]*Université Paris-Saclay, UVSQ, Laboratoire d'Ingénierie des Systèmes de Versailles, 78140, Vélizy-Villacoublay, France*

Keywords: Ontology, Knowledge Base, ADAS, Unity, Simulator.

Abstract: The intensive research on artificial intelligence and internet of things is speeding up the rise of smart cities and autonomous vehicles. In order to ensure the safety of the drivers and pedestrians, the transportation network needs to be connected to its surroundings and consider every valuable piece of information it can gather. Knowledge bases have proven themselves to be efficient in the storage and processing of structured data, making them interesting solutions for the management of transportation networks. This study focuses on the building of a driving simulator allowing the gathering of practical data that can be processed by an ontology and a set of rules, and can quickly and continuously infer a result to suggest the driver on an optimal choice to make. The accuracy results are encouraging, yet giving us extra room for improvement.

1 INTRODUCTION

The field of intelligent vehicles has progressed with an important advancement in the last decades. Nowadays, the modern transportation environment has become a dynamic and complex network made up of vehicles, infrastructure and pedestrians, and road users need to evolve in rapidly changing environment.

Real-life use of autonomous vehicles is becoming more popular, popularized by scientific developments such as the US DARPA Grand challenge (Defense Advanced Research Projects Agency), which is a race of autonomous cars organized by the US Army (Veres et al., 2011). The concept is also finding its way in civilian use, with cities like Dubai planning to deploy a fleet of self-driving taxi, and already leading tests with autonomous vehicles on their roads (Tesorero, 2019). However there are still ethical and technical questions pending on the security of road users and the responsibility of the driver in case of an accident: In 2018, a Tesla vehicle crashed and killed the driver due to an arguably wrong decision, and it took almost 2 years for the US Safety Administration to investigate the case (Chokshi, 2020). If anything, those elements are the proof that testing and validation need to be made in a more rigorous and controlled environment, and that it is also natural to assert that autopilot algorithms and ADAS (Advanced driver-assistance systems) systems still need to be improved. Most of those algorithms are based on machine learning and neural networks features, and although those techniques show great results in decision-making, there is still room for improvement in their ability to store and use data (Neoklis Polyzotis, 2017). Those vehicles need to deal with a significant amount of information, and those data are frequently interlinked between them. On the other hand, ontologies and semantic web have proven themselves to be efficient when dealing with organised and structured data. In addition, there are modern tools that can be used to apply rules and "intelligent" reasoning to those data, and generate logical outputs from the assertion input information.

Considering the previous statements, it would be interesting to have a simulated environment that could be used for experimenting new models in a practical way. Hence the goal of this study will focus on the setting up of a knowledge-base able to quickly react to unexpected events and advise the driver on the optimal action to take. It will be evaluated on reaction time and performance, and the experiment will be validated by a realistic driving simulator specifically built for this study.

Structure of the paper: Section 2 is dedicated to the related works. Section 3 describes the general concept of a knowledge-base and the building of the one used for this study. The simulator used for the experiment is introduced in Section 4, and the testing

Khezaz, A., Hina, M., Guan, H. and Ramdane-Cherif, A.
Driving Context Detection and Validation using Knowledge-based Reasoning.
DOI: 10.5220/0010135802190226
In *Proceedings of the 12th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management (IC3K 2020) - Volume 2: KEOD*, pages 219-226
ISBN: 978-989-758-474-9

and results are described in Section 5. This paper is concluded in the last section, which also contains our future works.

2 RELATED WORKS

Semantic networks have been studied as early as the 1960's(Collins, 1969), and has seen an increase in popularity with the rise of the semantic web and ontologies.

As such, there has been some studies attempting to use semantic web with vehicles. One of the first was made by (Tönnis et al., 2008) with the development of SCORE (Spatial Context Ontology Reasoning Environment), a distributed system that collects data and deduces the relative position of the road users. The data gathering is made of static RSU (Road Side Units), while the reasoning is directly embedded in the vehicles. The reasoner and rules embedded in the database allow the understanding of basic situation such as "Is there a spatial obstacle on the road?" or "Is vehicle A overtaking vehicle B?". A specific HUD (Heads-Up Display) then alerts the driver on the distance and direction of the other vehicles surrounding them. While functional and proving the usability of ontologies in a vehicular environment, the SCORE system is solely focused on space and does not consider speed, an important component of vehicles. It also has a limited set of reasoning rules and does not actively advises the driver on the optimal decision to take. Besides, the gathering units (called "Federation units") are static and have a limited range, making them powerless if a spontaneous and unexpected event is happening out of their reach.

Another study (Fuchs et al., 2008) worked on a smaller scale and developed an ontology-based context-model able to analyze a scene and quickly present a decision recommendation. It also relied on fuzzy logic in case of uncertainty about a situation. The system showed good performances and included different variables such as speed and brake intensity, but the perception was limited to the surrounding of the vehicle and did not include information about the other road users. There could also have been an interest in logging the generated recommendations and compare them to the action taken by the driver, to have the system "learn" from previous outputs.

(Kannan et al., 2010) proposed an ontology modelling approach for ADAS based on a richer context, considering different elements such as weather and the vehicle's dimensions. The inputs for the simulation were statically generated in XML files directly manipulated by the team, but they proved the feasibility of the concept. In another study in 2016, (Hina et al., 2016) proposed a more complete ADAS system embedded on a smartphone application and tested in a state-of-the-art simulator built by European automotive constructors. Being on a smartphone, the model had none to little interaction with the other road users and had to rely on the hosting vehicle's sensors to gather data.

Trying to have a model inspired from human behaviour was also studied by (Morignot and Nashashibi, 2013). They made the point that the vehicular context was not a strictly regulated one, and that a driver might find itself in a situation where they need to bypass traffic regulation,invoking the example of being stuck behind a defective car with an engine problem. A human driver would probably overstep the continuous line, while an autonomous vehicle could be stuck for hours. Their proposed solution was to use a complex set of rules that would let the vehicle detect that there is an "illegal space" (the lane over the continuous line) that could be used to bypass the problematic zone. The inferred decision took around 350ms, an acceptable time in a non-dangerous situation. However, and as noted by the author, "a drawback of an ontology-based approach is that a vehicle and its environment are represented in discrete, symbolic terms: things are true or false but there is no way to represent something intermediate". Those kind of challenges have been improved in the last few years through the use of computer intelligence, such as deep learning algorithms(LeCun et al., 2015). Another point would be the vehicle's ability to perceive the illegal space: one of their premises was for the cars to be able to communicate, but being behind an opaque object still limits the sight potential, and having other means of gathering data could be necessary in some situations.

In 2018, (Chen and Kloul, 2018), proposed a 3-layer ontology for an automatic generation of use cases in a highway insertion context. They describe a use case as "one or several scenarios applied to functional ranges and behaviors to simulate an ADAS. A scenario describes the temporal development between several scenes in a sequence of scenes" *[sic]*, meaning that a use case is the overall situation where the event takes place, and scenarios are the different possible outcomes that can happen in said situation. They use specific ontologies dedicated to the highway and the weather to describe the contextual environment, and a third one dedicated to the vehicle for the management of possible actions. Their study showed great results, but the testing was not done in real-time conditions, and therefore could not evaluate the speed of the process.

This study aims to build a knowledge-base able to correctly and quickly perceive the surroundings of the vehicle. For validation purpose, a simulator mimicking real-time behaviour of a vehicle has also been built and used for the gathering and processing of data.

3 KNOWLEDGE BASE

The "Principles of Modeling" (Starfield, 2005) defines a model as a simplification of a real-world problem. Modelising a problem presents many advantages, such as a formalization and logical description of the problem, a better understanding of the affected data, and a simplification of the testing procedure.

There are different ways to implement a knowledge-based model, such as logic programming (Jaffar and Maher, 1994), a knowledge-graph (Paulheim, 2016) or an expert system (Balci and Smith, 1986). This article focuses on the use of **ontologies**. An ontology usually serves as a hierarchical data structure containing all the entities of a specific context and the different rules, axioms and properties regulating them. In addition to the technical interest, the ontological approach shows some functional advantages that make it an interesting choice :

- **Scalability:** Once the classes, properties and rules are defined, the instantiation is managed by a Java API and it is easy to populate the ontology with new elements
- **Exportability:** The knowledge-base and its actors are ultimately independent from the application and can be used for another operation set in a vehicular environment

Being more specific, the Stanford 101 Guide on creating an ontology (Noy and Mcguinness, 2001) defines an ontology as "a formal explicit description of concepts in a domain of discourse, properties of each concept describing various features and attributes of the concept, and restrictions on slots". An ontology basically defines the main actors within a domain of discourses and the different interactions between them.

- Classes: Describe the concepts in the domain, whether they are abstract ideas or physical actors. Classes can be hierarchized by levels, for example having a Vehicle top-class containing Car, Bus and Bike sub-classes
- Properties: The specific information relative to classes. They can be intrinsic to an object, or extrinsic, representing the interconnections between different concepts
- Individuals: Real instances representing the elements of the ontology.

An ontology which is completed and filled with a full set of individuals, rules and properties is refered to as a knowledge base. In technical terms, the knowledge base is composed of the **Tbox** and **Abox**, respectively Terminological Box and Assertion Box. The former represents the ontology where the informations are stored, and the latter encompass the rules and properties. In addition to being able to represent all the elements of a situation, it is possible to add a layer of intelligence and reflection through the use of reasoners. A reasoner is a tool that can infer logical conclusion from a set of given facts, making the classification of an ontology easier. For example, if we declare an instance V as a Car, and the class Car is a sub-class of vehicle, then the reasoner infers that V is a vehicle(Hina et al., 2018). For a more complex situation, some reasoners can be supplied by SWRL rules (Semantic Web Rule Language) (O'Connor et al., 2008). It is a language of logic description that enables the combination of different rules to build a more complex axiom. The official documentation gives the following basic example to define the syntax :*hasParent(?x1,?x2) ^ hasBrother(?x2,?x3) -> hasUncle(?x1,?x3)*. By joining the two axioms *hasParent* and *hasBrother*, it is possible to apply the *hasUncle* relation to the individuals, hence making the individual X1 the child of X2 and the nephew of X3.

3.1 Structure of the Model

In the context of this study, the ontology revolves around a driving environment and was built using the Protege software(Musen, 2015). It is an open-source ontology editor developed by the Stanford University and that has been established as an important tool for knowledge management. Each actor of the environment is associated to a specific Class, including the Vehicles and the Roads, but also physical concepts such as overspeeding or the distance between objects. The data are received in real-time from the sensors (or the simulator) and transmitted once every element is classified. The reasoner is then called upon to infer an Action, which is sent back and executed by the vehicle. The interfacing is managed by a Java application built around the Jena API(Carroll et al., 2004), an Apache framework dedicated to the Semantic Web, and has been reinforced with the Java OWL API (Horridge and Bechhofer, 2011). In addition to those two tools, the SWRLAPI(O'Connor et al., 2008) was also used for the management of the SWRL rules. Here is an example of a basic SWRL rule for managing over-

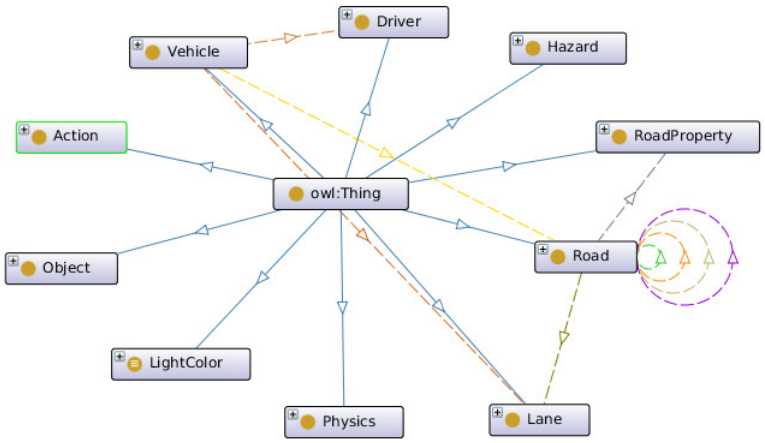

Figure 1: List of the main classes.

speeding. The pre-processing of the data and conversion from numerical value to a specific class (ie. a speed of 100km/h correspond to an overspeed situation) is done during the data gathering process.

```
Vehicle(?X) ^ hasSpeed(?X, Overspeed) ^
hasDriver(?X, MainDriver) ^ Action(A)
            -> Brake(A)
```

The ontology used for this study focuses on both the vehicle and its surroundings. As shown in Figure 1, there are many different interlinked classes, but only a few of them make up the core of the application

- *Vehicle* representing the different vehicles detected in the environment. The class is not limited to cars only and has different sub-classes such as Trucks and Bikes.
- *Road* and *RoadProperties* lists all the road segments and their potential properties, such as *SpeedLimit* or *WeatherCondition*. Roads are also linked between them with the ”isConnectedTo” object property, allowing a global mapping of the area by creating a virtual link between connected roads, similar to the concept of doubly linked lists
- *Action* is the master class of all the possible actions that can be inferred by the reasoner displayed to the driver. They can be directive orders such as ”Accelerate” or ”Turn Left” or informative like ”Fire Hazard Detected”.

Naturally, the main class for this application is the *Vehicle* one, which is the one with the most relations with the others. As shown in Figure 2, experiments focus on a single car named *MainVehicle* and is the one equipped with the ADAS feature.

- Each *Vehicle* has a *Driver*. The *MainVehicle* is identified by having the *MainDriver*
- A *Vehicle* is geographically defined as being on a specific *Road*. As stated above, every *Road* has *RoadProperties*
- The class *Physics* contains the subclasses *Speed* and *DistanceFromVehicle*, which is the distance between any object and the *MainVehicle*
- The other entities of the road are labelled under the *Object* class, which is itself split between *DynamicObject* and *StaticObject*

The ontology is pre-loaded with all the necessary classes and rules, and it is progressively populated by individuals detected in the vehicle’s surrounding. Considering that sensors have a detection range, only the elements at a certain distance can be added to the knowledge-base.

3.2 Data Fusion

Multimodal data fusion is defined by (Lahat et al., 2015) as ”the analysis of several datasets such that different datasets can interact and inform each other”, effectively meaning that information from different sources can be compared and cross-referenced to offer a better understanding of the situation where an intelligent agent is evolving. The implementation of this process requires an architecture capable of efficiently classifying data, and a necessary processing power (Hall and Llinas, 1997). In a real driving situation, an important quantity of data needs to be considered, and most of the time they are of different type. Being able to manage and join seemingly unrelated information might be critical in this context. For this study, the data fusion is made on two-levels, as illustrated on the Figure 3.

- The raw data are preprocessed and labelled by the sensors of the vehicle (i.e. the simulator). This is

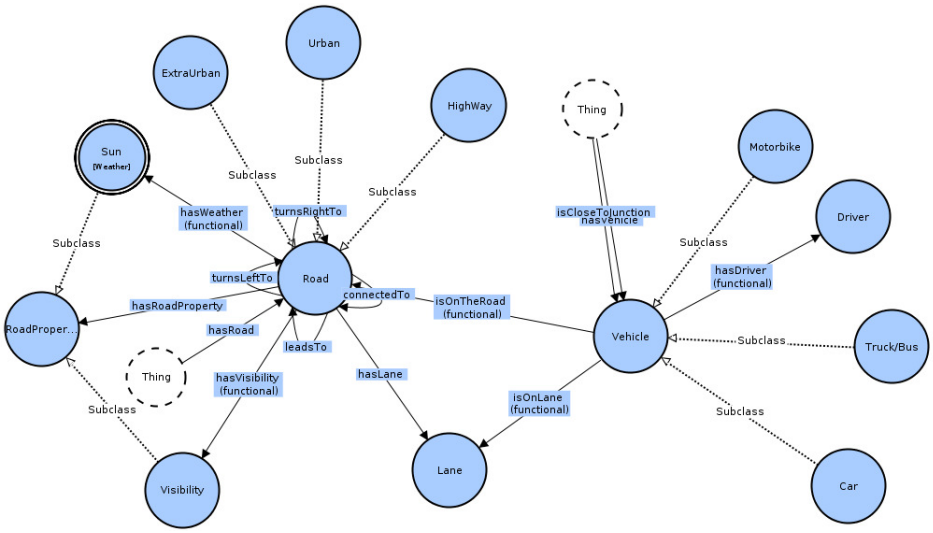

Figure 2: The Vehicle and Road Classes.

done in order to ease the processing time of the reasoner, and this part deals with light classification of information : If a car is going at 100km/h on a road limited to 80km/h, then the *MainVehicle* has an object property *hasSpeed* of *Overspeed*

- Through the use of SWRL rules, the ontology can then make logical processing and fuse all the available pre-treated information to infer a result

This study emphasizes on a model's ability to quickly react to random and unexpected events, hence the focus on trying to infer with the highest precision and the fastest time. Having the vehicle's more powerful computing units process a part of the information can help reach an optimal result.

4 SIMULATOR BUILDING

4.1 Environment Description

The first step in building the model is to set-up a reliable testing environment. The interface was based on the Udacity(Udacity, 2017) project, a car simulator built with the Unity engine(Haas, 2014). It allows the building of driving surroundings (Roads, obstacles) and both manual and automatic driving of vehicle.

The simulation file is then read by a Java program that will interface the simulator and the ontology. Most of the libraries used come from the Protege application project. Data about the driving simulation can then be recorded in order to be processed. They are formatted into JSON to make their interpretation easier by the ontology.

An ontology containing the information related to the driving context is then setup. It includes the different classes of road users: Vehicles, Pedestrians... and their numerous properties, such as their Speed or their distance from the main vehicle.

The ontology reads the simulation data in near real-time and based upon the set of SWRL rules it contains, the reasoner can infer an "Action" class into one of the possible outcomes : *Brake, Accelerate, Turn Left, Turn Right, Remain in Lane, Change Lane, Bad Weather and Fire Hazard.*

Based to the inferred action, basic instructions are then generated and written in another text file that will be accessed back by the simulator used to direct the vehicle's movements. This is used for the automatic-driving option. In the manual mode, the informations are directly displayed on the screen for the driver to see.

4.2 Unity Environment

The simulation environment was built on Unity, based on the Udacity open-source driver simulator. The Udacity project shows mathematically-accurate driving physics and comes with a set of pre-existing maps and the necessary tools for building one's own environment.

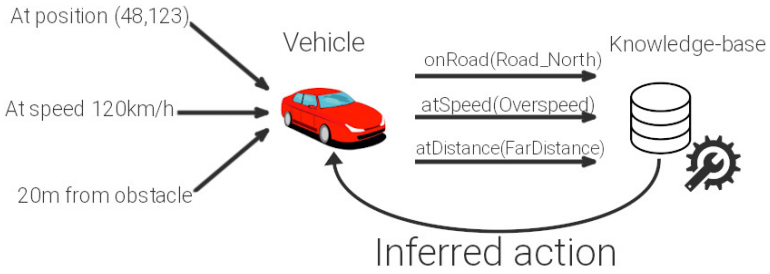

Figure 3: Illustration of the data fusion process.

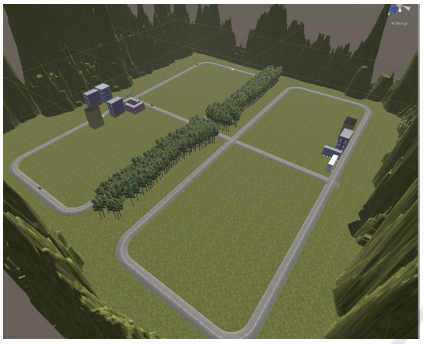

Figure 4: Global view of the simulated environment.

Table 1: Data relative to the road object.

Type of the data
Name of the road
Speed limit
Weather situation
Is it connected to another road?
Is another object present on it?

5 MODEL TESTING

5.1 Use Case Presentation

The simulator allows quick development of testing scenarios. The one considered in this study consist of 2 circular roads linked by a middle path. There are also natural environment objects like buildings or traffic signs and the environment is populated by dynamic actors such as moving and stopped vehicles and scripted pedestrians that randomly cross the road. In order to bring more realism to the situation, unexpected events are also prone to happen: Rain can start pouring on a road and a building can catch fire. All those elements have their own dedicated classes and instances in the knowledge base.

The simulation has for objective to validate that the knowledge base functions properly. The vehicle gathers raw data and convert them into a machine-readable format (Brioschi, 2016). They are then processed by the SWRL rules and inform the driver in real-time of the surrounding situation. There can be multiple information inferred at a given time. If the vehicle is overspeeding and a fire happens, they will be informed of both events. Speed control is the most controlled variable, continuously informing the driver of what they should do.

5.2 Tests and Results

The tests were conducted by having a human operator manually drive the vehicle around the map while having the logging and processing API running in parallel. The reason for choosing manual driving instead of automatic is because the accuracy of a human is superior and less prone to deadlock, and this mode makes it easier to forcibly generate events that otherwise depends on luck, like raining event. The driving recommendations are still displayed on the screen and monitored by the driver. The experiment requires to have both the Java API and the Unity project running at the same time: The simulator gathers the data and the Java program process them. In order to validate the behaviour in the concept, two scenarios are considered : An easy situation right at the beginning of the experiment, and another one in a more complex situation.

5.2.1 First Scenario

The first testing is done right at the launch of the simulator. The vehicle is stopped in the middle of a rainy road. The vehicle is stopped, prompting the ontology to detect a speed value of *NoSpeed* and inferring the "Accelerate" action. The spawning spot also presents heavy rain, which is detected by the "Bad Weather" inferred class. The following rules are applied:

```
Road(?Y)^ Vehicle(?X) ^ hasWeather(?Y, Rain)^
    isOnTheRoad(?X, ?Y) ^ Action(?Z) ->
          BadWeather(?Z)

  Vehicle(?X) ^ hasSpeed(?X, NoSpeed) ^
    hasDriver(?X, MainDriver) ^ Action(Z) ->
        Accelerate(Z)
```

5.2.2 Second Scenario

In order to test the model in a stressful situation, the second experiment is led in a more stressful scenario illustrated in Figure 5. The vehicle is on a road limited to 80km/h and going at almost 90km/h, hence over-speeding. In addition to that, one of the buildings in the detection range is on fire, prompting the following rule:

```
Building(?B) ^ isOnFire(?B, Fire) ^
   Action(?Z) -> FireHazard(?Z)
```

Figure 5: A stressful situation in the simulation.

Both those elements are gathered by the sensors and logged in the knowledge base. The Java application output those results in real time, as shown in Figure 6.

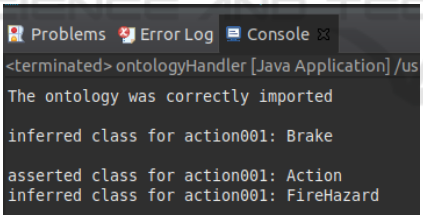

Figure 6: Output from the Java API.

5.3 Discussion

The inferring process showed great results, almost always inferring the correct situation. The few mistaken cases were either due to a delay in the logging and computation or a corruption of the knowledge base. On the other hand, the execution time proved to greatly depend on the situation. As shown in Figure 7, the first execution takes an important amount of time of around 3300ms, which makes sense considering the required time to load the knowledge base in the application and start the reasoner. The execution time then stabilizes at around 750ms and can decrease to 560ms when the car is stopped in an isolated area, meaning the newly gathered data are similar to the previous state. On the opposite, in a stressful situation where the car is over-speeding in an urban area and a fire hazard is detected (ie. Figure 5), the processing time can reach up to 2200ms.

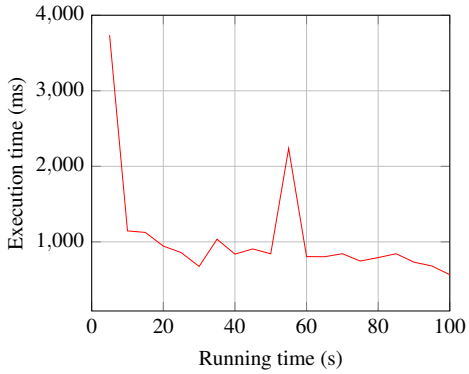

Figure 7: Computed inferring time.

The tests were run on an Ubuntu OS with an *Intel i7-8550* CPU and 16GB of RAM memory. The computation time between every inference might seem high, with an average of 1s, but it is interesting to note that no optimizations were made : The data were renewed between each frame of the simulator, and the knowledge base was rebuilt anew each time. This constitutes a stressful situation for the system, and in rare instances the fast reading/writing rhythm caused a corruption of the knowledge base. Those issues can be fixed by optimizing both the gathering process and the Java code.

6 CONCLUSION AND FUTURE WORKS

This work presented a knowledge-base model able to accurately infer the situation surrounding the vehicle. The model was validated through the use of a simulator where mathematically realistic data were gathered and processed. The different elements were managed by a Java application. The results were highly encouraging, however some axes of improvement can already be identified: *Improving the processing time, making the vehicle fully autonomous and expand the set of rules*.

In addition to the previous points, this work falls under the scope of a larger project involving drone/vehicle interactions and wireless communication. Many steps are still necessary before the completion of the project, and both the simulator and knowledge base are bound to evolve in the future.

ACKNOWLEDGEMENT

This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health.

REFERENCES

Balci, O. and Smith, E. P. (1986). Validation of expert system performance. Technical report, Department of Computer Science, Virginia Polytechnic Institute & State ....

Brioschi, G. (2016). Autonomous smart secured interactive automobile (assia). a knowledge based system for driving assistance. *Politecnico di Milano*.

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., and Wilkinson, K. (2004). Jena: Implementing the semantic web recommendations. In *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, WWW Alt. '04, page 74–83, New York, NY, USA. Association for Computing Machinery.

Chen, W. and Kloul, L. (2018). An Ontology-based Approach to Generate the Advanced Driver Assistance Use Cases of Highway Traffic:. In *Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, pages 75–83, Seville, Spain. SCITEPRESS - Science and Technology Publications.

Chokshi, N. (2020). Tesla autopilot system found probably at fault in 2018 crash. *New York Times*.

Collins, A. M., . Q. M. R. (1969). Retrieval time from semantic memory. journal of verbal learning and verbal behavior.

Fuchs, S., Rass, S., and Kyamakya, K. (2008). Integration of Ontological Scene Representation and Logic-Based Reasoning for Context-Aware Driver Assistance Systems. *Electronic Communications of the EASST*, page Volume 11: Contextaware Adaption Mechanisms for Pervasive and Ubiquitous Services. Publisher: European Association of Software Science and Technology.

Haas, J. K. (2014). A history of the unity game engine.

Hall, D. L. and Llinas, J. (1997). An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23.

Hina, M. D., Hongyu Guan, Ramdane-Cherif, A., and Nan Deng (2016). Secured data processing, notification and transmission in a human-vehicle interaction system. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1277–1284, Rio de Janeiro, Brazil. IEEE.

Hina, M. D., Thierry, C., Soukane, A., and Ramdane-Cherif, A. (2018). Cognition of Driving Context for Driving Assistance. 12(2):11.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semantic Web*, 2(1):11–21.

Jaffar, J. and Maher, M. J. (1994). Constraint logic programming: a survey. *The Journal of Logic Programming*, 19-20:503–581.

Kannan, S., Thangavelu, A., and Kalivaradhan, R. (2010). An Intelligent Driver Assistance System (I-DAS) for Vehicle Safety Modelling using Ontology Approach. *IJU*, 1(3):15–29.

Lahat, D., Adali, T., and Jutten, C. (2015). Multimodal data fusion: an overview of methods, challenges, and prospects. *Proceedings of the IEEE*, 103(9):1449–1477.

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

Morignot, P. and Nashashibi, F. (2013). An Ontology-based Approach to Relax Traffic Regulation for Autonomous Vehicle Assistance. In *Artificial Intelligence and Applications / 794: Modelling, Identification and Control / 795: Parallel and Distributed Computing and Networks / 796: Software Engineering / 792: Web-based Education*, Innsbruck, Austria. ACTAPRESS.

Musen, M. A. (2015). The protégé project: a look back and a look forward. *AI Matters*, 1(4):4–12.

Neoklis Polyzotis, Sudip Roy, S. W. M. Z. (2017). Data management challenges in production machine learning.

Noy, N. and Mcguinness, D. (2001). Ontology development 101: A guide to creating your first ontology. *Knowledge Systems Laboratory*, 32.

O'Connor, M., Nyulas, C., Shankar, R., Das, A., and Musen, M. (2008). The SWRLAPI: A Development Environment for Working with SWRL Rules. page 5.

Paulheim, H. (2016). Knowledge graph refinement: A survey of approaches and evaluation methods. *SW*, 8(3):489–508.

Starfield, T. (2005). Principles of modeling: Real world - model world. *University of Vermont Lectures*.

Tesorero, A. (2019). Dubai's driverless cab now on trial run. *Khaleej Times*.

Tönnis, M., Fischer, J.-G., and Klinker, G. (2008). From Sensors to Assisted Driving – Bridging the Gap. *JSW*, 3(3):71–82.

Udacity (2017). Udacity self-driving car project. https://github.com/udacity/self-driving-car-sim.

Veres, S. M., Molnar, L., Lincoln, N. K., and Morice, C. P. (2011). Autonomous vehicle control systems — a review of decision making. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*.