Meaningful Control of Autonomous Systems

MCAS Project - Final report

Several contributors from TNO, UvA and CWI:
e.g. Jan-Pieter Paardekooper, Arnoud Visser and Eric Pauwels

First version: November 12, 2020 — Current version: February 10, 2021

Contents

Chapter 1

Introduction and Motivation

The **Meaningful Control of Autonomous Systems** (MCAS) initiative is a collaboration between the *Universiteit van Amsterdam* (UvA), the *Centrum Wiskunde & Informatica* (CWI) and *Nederlandse organisatie voor toegepast-natuurwetenschappelijk onderzoek* (TNO). The ultimate goal is to strengthen the mutual knowledge acquisition on Artificial Intelligence (AI) and explore the possibilities for a Joint Innovation Centre on this topic. For the first year, a sub-goal is determining how a fruitful collaboration can be established on the topic of self-driving vehicles. This will be established by tackling a use case that utilizes the strengths of the different partners to showcase the added value of combining the existing expertise.

In order to improve the current state of the art in automated vehicles, improvements in at least 3 AI areas could be made:

- **Sensing:** improved recognition of the environment, both static (e.g. lines, highway entree) and dynamic (e.g. other road users).
- **Thinking:** assessment of the current and future situation, including predicting the most likely future traffic state.
- **Acting:** taking the right action given all above information.

The result of this cooperation could be a demonstration of a system with three AI components working together to correctly navigate the above scenario, for example on a simulation platform. The partners need to investigate what kind of demonstration is realistic, and what options may already exist for demonstration purpose. The proposed AI system should be scalable to other scenarios, so that in future work it can be shown to make traffic safer.

For CWI the initial focus lay on the use of (deep) reinforcement learning (RL) to improve the selection of safe and effective **actions** based on the sensing input. Currently, deep RL doesn't play a significant role in designing the control algorithms for self-driving cars. This is partly due to the fact that training is computation- and data-intensive. However, another reason is that standard RL methodologies are mostly geared towards numeric feedback and it is difficult to incorporate symbolic reasoning (e.g knowledge graphs and representations generated in the thinking part). CWI therefore proposes to focus on the use of graph neural networks (GNN) to combine the sensing input with (symbolic) information gleaned from the thinking component. Graph neural networks are more readily combined with knowledge graphs to make use of domain knowledge.

TNO has the focus on the **thinking** aspect: given information from sensing as input, create a representation of the current traffic state and extrapolate to the future. This information will subsequently be sent to acting component. For the knowledge representation of the traffic state, TNO will build on in-house developments of a knowledge graph for automated vehicles. The knowledge graph will be used for *competence assessment* of a deep neural network that predicts the intention of other road users. This approach will be validated using in-house data from passenger vehicles and truck platooning tests.

The UvA proposes to focus on the **sensing** aspect: given the input stream of the different sensors on board of the vehicle (e.g. vision, lidar, radar), reconstruct the 3D scene as good as possible. The objects in this scene detected, classified, identified and tracked with panoptic segmentation. The result is a representation of the current traffic state which could be further used in the thinking component. To build such perception algorithms, the UvA experience with building virtual datasets (with realism based on game engines) will be used. A virtual dataset will provide the ground-truth needed to use a supervised learning of a convolutional neural network. The realism of this models will be tested in a small experimental setup: a miniature highway with mini-vehicles in the Intelligent Robotics Lab.

This technical report likes to highlight the progress made in the year 2020 and the plans for the continuation of cooperations on the subject of *Meaningful Control of Autonomous Systems*.

Chapter 2

Initial ideas

In September 2019 the partners TNO, CWI and UvA signed a Letter of Intend on a collaboration to enforce the knowledge development on the subject of *Meaningful Control of Autonomous Systems*. On December 4, 2019 the group on *Automated and Cooperative Driving* had their Kick-off meeting, where they studied the following scenario:

The *Onderzoeksraad voor de Veiligheid* published a report 'Wie stuurt?' on the safe introduction of automated systems on the Dutch roads (20). This report describes six accidents with automated systems on the Dutch roads as real-life examples of problems with the current state of technology. During the Kick-off meeting the partners decided to use one of these accident scenarios to demonstrate how the partners can work together on improving AI in automated systems to prevent such accidents from happening in the future.

The selected accident scenario is described on page 33 of the 'Wie stuurt?' report. Near Bathmen a vehicle in autopilot mode crashed onto a truck that breaks out of a platoon to make space for a truck entering the highway.

Figure 2.1: The situation after the accident for the proposed scenario. Courtesy: Hof van Twente & Onderzoeksraad voor Veiligheid (20)

This scenario is a perfect showcase for possible improvements in automated driving:

- According to the report, the vehicle did not detect the lane-changing truck in time because the autopilot system can only adequately detect other vehicles in the same lane.
- The automated system could benefit from taking into account information from the complete traffic state, including the truck entering the highway, to anticipate the lane change of the truck.
- According to the report, the automated system decided too late that a braking action was necessary to prevent a collision.

Accidents like this scenario can only be prevented when the three AI components sensing, thinking and acting in harmony and support each other in their decisions.

Chapter 3

Learning to Drive - End-to-End Learning

To recreate this scenario, several simulation platforms have been studied. Initially we looked at Autoware ecosystem[1], which is a Gazebo based simulation environment. Later in the project we shifted our attention to the Carla simulator[2], which is based on the Unreal Engine.

For this Carla simulator a Amazon Elastic Compute Cloud (EC2) server was configured, where the researchers from TNO, CWI and UvA can work together in their developments. The ports of the EC2 server can be forwarded to your local machine, so that you visualize the scenario.

The Carla simulator is already used to generate data for two autonomous driving tasks: vehicle detection and lane following. The approach on vehicle detection is described in Section 4.1, which summarizes UvA's student team's participation in the Waymo challenge. The approach on lane following is described in two papers submitted to the Intelligent Autonomous Systems conference (21; 28).

In the first paper the Carla simulator was used to train the vehicle to follow a lane. Such task can be accomplished model-based by first detecting the lane-markings and than command the vehicle to stay in the middle of the lane-markings, but this approach is prone to fail in situations where lane-markings are missing. A more advanced approach is to train the vehicle to stay in the lane using the full camera feed (end-to-end learning), as described in (21).

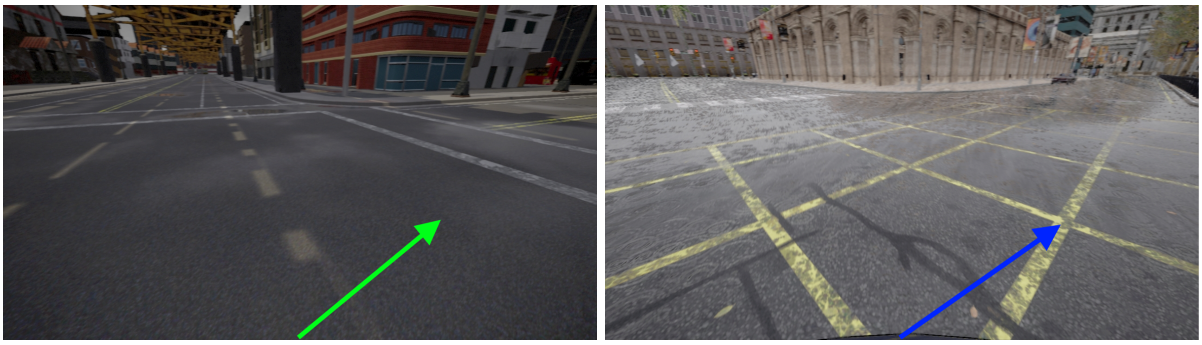


Figure 3.1: Situation encountered in the Carla Simulator for Town 3 (left) and Town 10HD (right). The green and blue vector are the learned steering angles for this situation.

The model was trained in Carla simulator based on imitation learning, by providing examples of the vehicle driving around through the simulated Town 3 environment based on the waypoints available in

[1] https://www.autoware.ai/

[2] https://carla.readthedocs.io/en/latest/start_introduction/

this world. This town is in one of the most complex environments available in the simulation environment CARLA. From the training in this environment it is clear that complex situations remain difficult to learn without high-level commands.

After the training, the vehicle can follow the road reliably in the training map, a behavior that can be transferred to a non-complex map with circumstances it has not seen before. The learned behavior was then validated in two town not seen during training: the complex Town 10HD environment (See Fig. 3.1) and a less complex Town 6 environment. The result was a success-rate of respectively 62% and 90%.

The behavior trained in simulation can also be transferred from simulation to a real environment. In the Intelligent Robotics lab this was done on the DuckieTown highway (see Fig. 3.2). The lane following behavior of this vehicle was trained in OpenAI's Gym framework. While the simulator fidelity of OpenAI's Gym is quite low, the real world DuckieTown environment is also rather simple, featuring large road markings in an indoor environment. It does not often feature visual effects such as shadows, water reflections or sun flare effects, reducing the need for a photo-realistic simulator, such as the Carla simulator. Yet, it is quite impressive to see transfer learning demonstrated. More details can be found in (28).

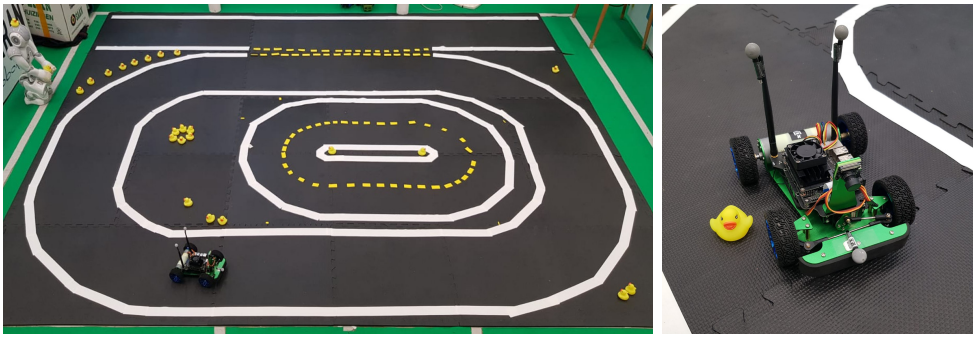


Figure 3.2: The DuckieTown highway used for real world evaluation. The outer track circle is used for the JetRacer. The JetRacer vehicle is shown at the right.

The code of this study is published on two private github pages; the installation instructions for the code are added as an appendix to this document.

Chapter 4

Intermezzo: challenges

4.1 Waymo challenge

On March 19th, 2020, the company Waymo launched a challenge[1] to improve the detection rates on their Waymo Open Dataset (27). The challenge run until May 31th and was a great opportunity to bootstrap the sensing part of our cooperation. To train detection algorithms a lot of data is needed, which was provided in this challenge, based on "the largest and most diverse multimodal autonomous driving dataset to date" (27).

The Waymo Open Dataset is unique in the sense that it has extensive high-quality LiDAR data with high quality annotation. Yet, our UvA team concentrated on the 2D Detection challenge, which restricts the input data to camera images, excluding LiDAR data. Still, it is an impressive dataset, with 9.9M 2D Boxes, recorded at 1000 different scenes for training and validation, and 150 scenes for testing, where each scene spans 20s. The scenes were all recorded in the USA, although at three different locations (San Francisco, Mountain View, and Phoenix).

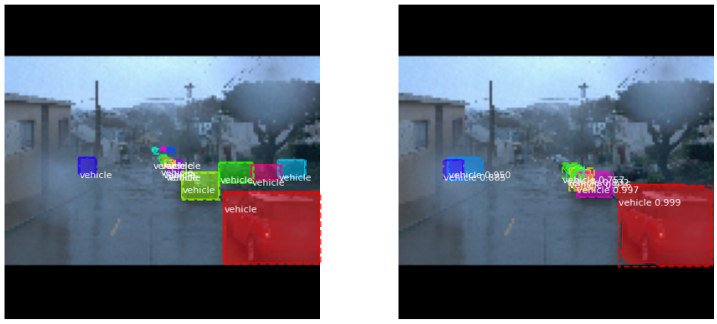


Figure 4.1: First results of the Mask R-CNN model using pre-trained COCO weights (at the left the ground-truth, at the right the prediction)

The vehicles driving around for Waymo were equipped with five LiDAR sensors and five high-resolution cameras. The Waymo Open Dataset is also accompanied with a baseline implementation

[1] https://waymo.com/open/challenges

for the 2D Detection challenge, as described by Sun *et al* (27). They used the Faster R-CNN object detection architecture (25), with ResNet-101 (8) as the feature extractor and pre-trained the model on the COCO Dataset (17) before fine-tuning the model on the Waymo Open Dataset. With a comparable approach, based on the Mask R-CNN model (7) (an extension of Faster R-CNN), the UvA team was able to reproduce those results, although it is clear from Fig. 4.1 that a multi-scale approach is needed to also detect the vehicles further away.

During the competition several approaches (i.e. Yolo (24), several variants of Masked RCNN) were tried by our team, but the score based on submissions of subsets of the validation-set (10K images) remained low. Only when the results for the full validation set (200K images) were submitted the results improved. With a processing speed of 4FPS it took more than 8h to process the whole validation-set and to bundle it in submission bins. The result was an average score over all object types is 0.3075/L1 and 0.2513/L2[2], which should be interpreted as an high precision (above 80%) combined with a recall of 40%. This performance resulted in a top-30 qualification. The champions of the competition (12) showed that multi-scale training and full resolution training were essential for their success, which resulted for an high precision combined with a recall of nearly 80%.

4.2 Lyft challenge

On August 24th, 2020, the company Lyft launched a challenge[3] to predict the future motion of traffic agent, if their previous trajectory is already detected based on 3D tracking of objects by fusing of camera and LiDAR measurements (16). The challenge run until November 25th and was highly applicable for the future of the cooperation. The dataset of this challenge is still available for training (11).

In order to fulfill the prediction task one needs significantly more detailed information than the positions of the vehicles. One also need context information about the environment including, such as semantic maps that encode possible driving behaviour to reason about future behaviours, as illustrated in Fig. 4.2.

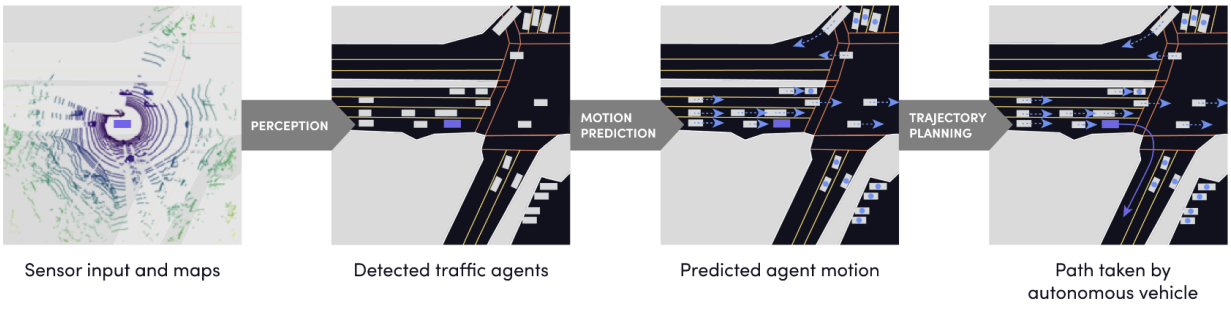


Figure 4.2: An example of a state-of-the-art self-driving pipeline as given by (11)

The leading solutions for this task apply deep learning techniques on the birds-eye-view of the scene. An example of such approach is the submission of the Epoch team[4], a student team of the University of Amsterdam & Technical University Delft, who started with a ResNet (9) with 101 layers to predict

[2] https://waymo.com/open/challenges/2d-detection/
[3] https://www.kaggle.com/c/lyft-motion-prediction-autonomous-vehicles/
[4] https://www.teamepoch.net/

multiple possible trajectories. They quickly found out that there was not enough training data for such a deep network, so instead they trained a ResNet with 34 layers with the same loss-function as used in the challenge. This resulted in a top-100 qualification.

Currently this dataset is used for a graduation study (5) with a more in-depth analysis on how to improve the predictions of the future paths.

4.3 AI Driving Olympics

On September 15h, 2020 the DuckieTown organisation launched the 5th edition of the AI Driving competitions, as competition track of the NeurIPS 2020 conference[5]. The competition consisted of a simulated DuckieTown circuit, which could be used for training. Another DuckieTown circuit was used for evaluation. The challenge consisted of a lane-following task, augmented with other obstacles on the road.

For this challenge, the Duckiebot should to drive on the right-hand side of the street within Duckietown without a specific goal point. Duckiebots will will be judged on how fast they drive, how well they follow the rules (see Fig. 4.3) and how smooth or "comfortable" their driving is.

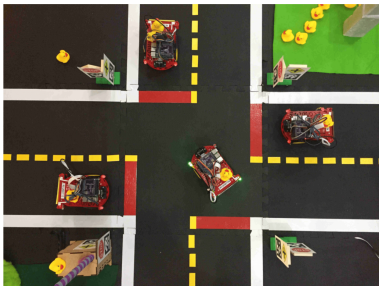


Figure 4.3: A complex Duckietown crossing.

This scenario was very comparable with the end-to-end learning approach taken in this project, so same the Proximal Policy Optimization method (26) for this competition. This resulted in a top-10 qualification in all three challenges.

[5] https://neurips.cc/Conferences/2020/CompetitionTrack

Chapter 5

Learning to Drive Safely - Combining Data-Driven AI with knowledge

5.1 Improved situational awareness

The safety of automated vehicles (AVs) could be enhanced by introducing the concept of situational awareness of the AI. The goal of situational awareness is to have an assessment of how well the current situation and the situation in the near future reflect the training of the AI, or in other words, how known they are. This will give a level of competence of the AI. Fig. 5.1 shows the proposed architecture of the system.

5.2 A Hybrid-AI Approach to Situational Awareness

The tremendous success of Deep Neural Networks (DNNs) in the recent years (15) has lead to many applications in automated driving, ranging from perception (3) and trajectory prediction (4) to decision making (1). The strength of DNNs is the capability to deal with complex problems, but one important drawback for their application in safety-critical systems is how they deal with new situations (10; 18). DNNs learn a (possibly very complex) mapping from input data to output, but they lack an understanding of the deeper causes of this output. Hence, these algorithms cannot reason about whether they are competent to produce reliable output based on the input data. To safely apply DNNs (or any learning algorithm) in automated vehicles, we need to add situational awareness: the comprehension whether the system understands the current environment and is capable of producing reliable output.

Here we describe a hybrid-AI approach (6; 19) to situational awareness. In this approach, a data-driven AI is coupled to a knowledge graph with reasoning capabilities. The current application is a DNN that predicts the intention of other road users to merge into the lane of the ego vehicle (cut-in maneuver). This is combined with a knowledge graph of the traffic state that relates the current situation to what the predictor has learned from the training data. The knowledge graph reasoner returns an estimate on the reliability of the predictor, which it forecasts into the immediate future (2 seconds ahead) to be able to warn the driver or safety system in advance that takeover of control is imminent in the near future. A more detailed description of the approach can be found in (22).

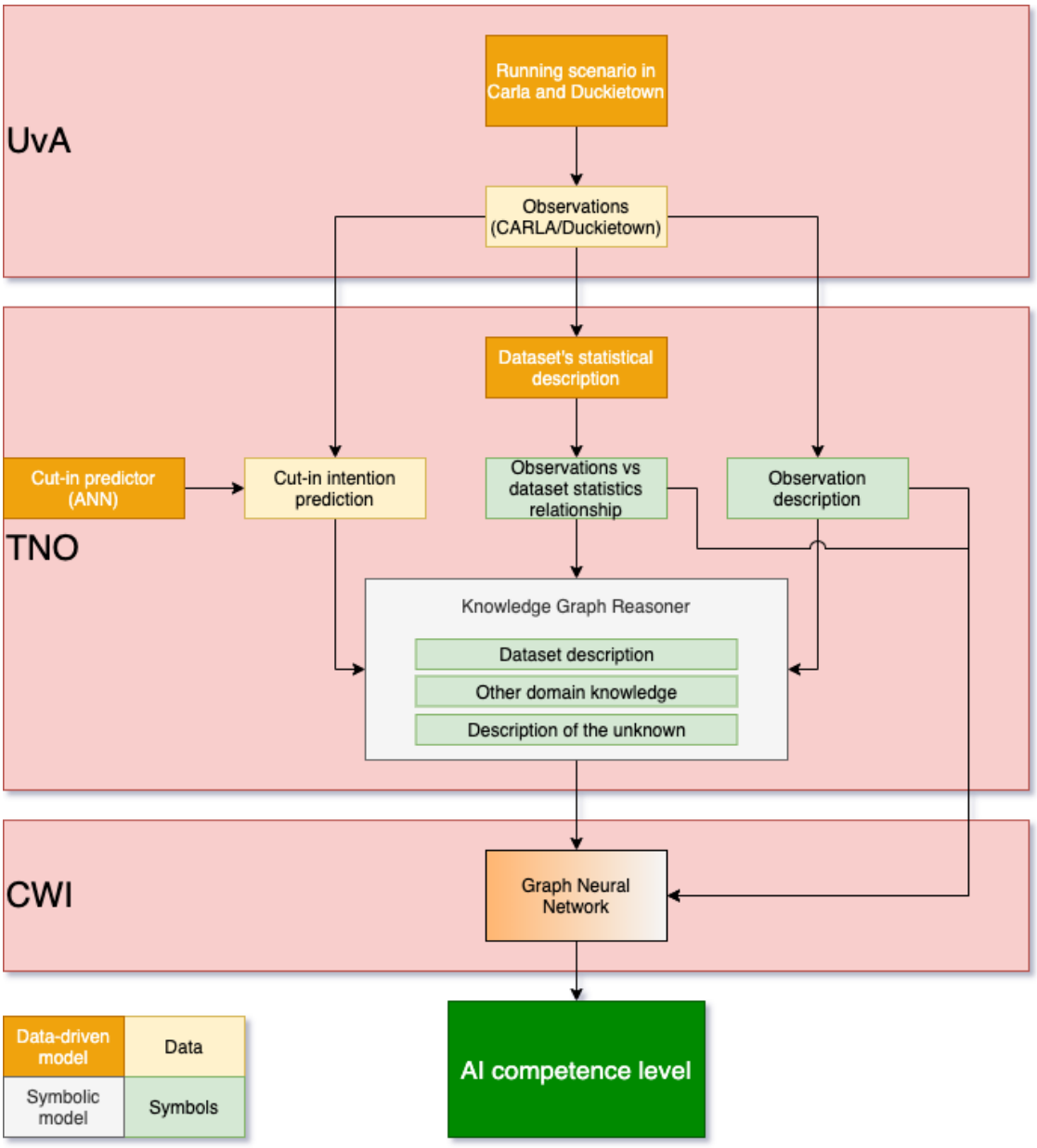


Figure 5.1: Proposed architecture of the system.

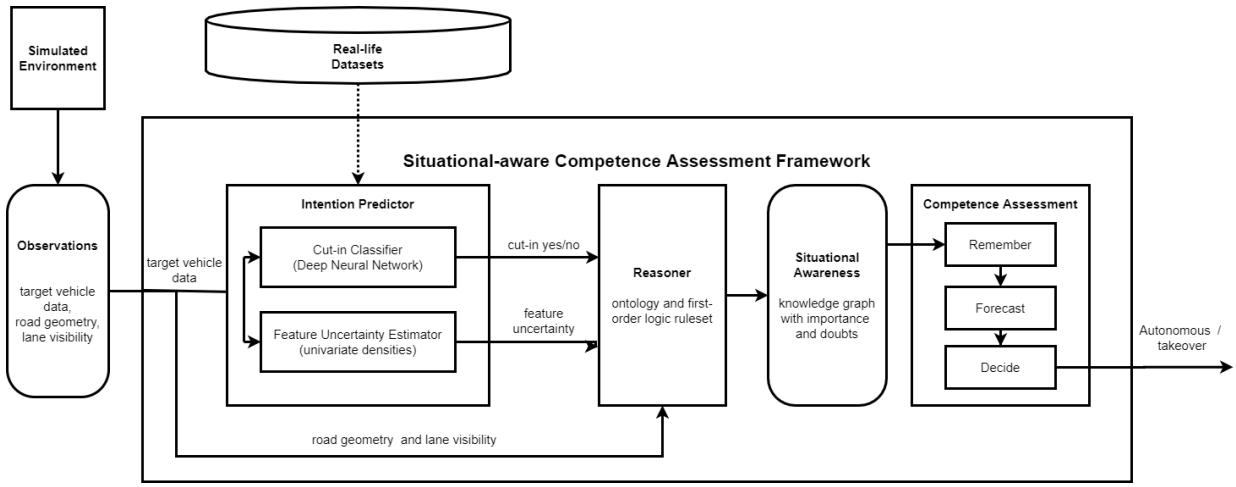


Figure 5.2: The overall architecture of our situation-aware competence assessment framework. The dotted arrow from Real-life Datasets to Intention Predictor is not part of the online information flow.

5.2.1 Architecture

To assess the competence of data-driven-AI automated-driving capabilities we developed a pipelined framework, depicted in Figure 5.2. The framework receives as input the observations of the current road situation and, via a pipelined information flow, outputs the decision on whether the driving mode should remain autonomous or should be handed over to the human driver or backup safety system. The framework's internal structure is divided into three modules: Intention Predictor, Reasoner and Competence Assessment.

Raw observations related to each target vehicle, such as their speed and acceleration, are fed to the Intention Predictor. This module processes the information via two sub-modules. The first one is a deep neural network trained to output (predict) whether a given target vehicle will perform a cut-in maneuver (Cut-in Classifier). The second sub-module is a Feature Uncertainty Estimator. It holds univariate densities of the classifier's training set input features and provides information on the in-distribution likelihood of the network's input data.

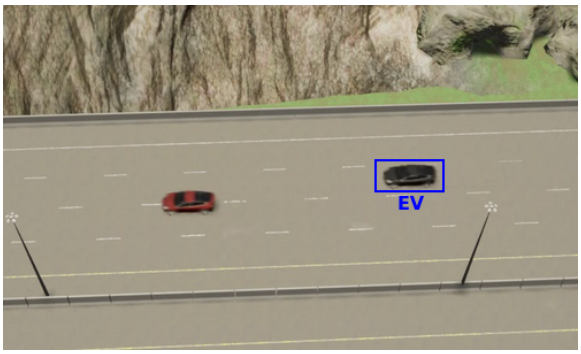
The Intention Predictor's output, the observations related to road geometry (e.g. presence of entry lanes) and lane visibility are fed to the framework's second module, the Reasoner. The reasoner, characterised by an ontology and first-order logic rules, fuses the input observations with domain knowledge (encoded in the ontology and in the rules) into a knowledge graph. The graph realises the framework's situational awareness, as it holds a unified representation of the current situation and is aware of what entities are important and doubtful.

The graph is then fed to the last module of our framework: Competence Assessment. The module first organises its present and past situation-aware knowledge. Then it projects such knowledge into the future. Finally, it decides whether such forecast is outside the autonomous system's competence level.

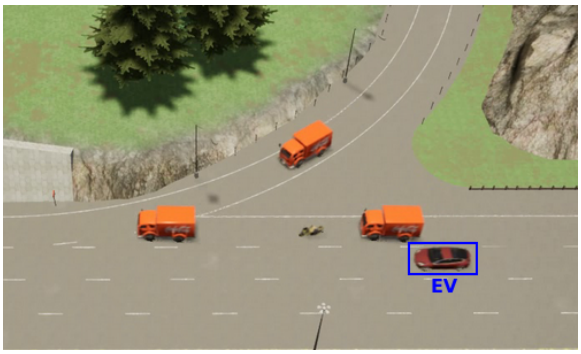
5.2.2 Results

We have assessed the competence of the intention predictor in two cut-in scenarios. The first scenario describes a cut-in by a target vehicle (TV) on an otherwise empty road (Fig. 5.3a). The velocity, distance and driving profile of the TV was designed not to pose any risk to the ego vehicle (EV). In

addition, every vehicle present in the scenario was known to the knowledge graph.



(a) The cut-in scenario is *within* the operational design domain.



(b) The lane entrance scenario which is *outside* the operational design domain.

Figure 5.3: Snapshots from the two different scenarios as shown in the CARLA simulator.

The second scenario (Fig. 5.3b) describes multiple vehicles (two trucks and a motorcycle) on the first right-most lane and a truck approaching from the entrance lane. The EV is in the left lane and cannot see the approaching truck as it is occluded by the vehicles on its right. Note that his scenario is inspired by the real-life accident scenario decribed in Chapter 2.

The features in this scenario are out-of-distribution: only two features lie within the training set domain. Moreover, the scenario includes an unknown entity in the form of a motorcycle that was not part of the training set. The rationale for this is that a type of vehicle not present in the training set might display a driving profile that the intention prediction does not expect. In other words, the output of the predictor might be incorrect since it relies on the detection of spatio-temporal patterns in the vehicle's driving behaviour. The visibility on the road was reduced by the traffic on the first lane; this lane was considered of high importance due to the road entrance. This scenario was designed to pose potential risk to the autonomous system, due to the out-of-distribution features and unknown entities. The two scenarios were evaluated at the moment that one of the TVs performs a cut-in.

We found that the reasoner correctly assigns high competence to the Intention Predictor in the situation in which some features of the DNN are uncertain, but the TV poses no safety threat to the EV due to the large distance and high lane visibility. The added value of the Reasoner is also shown in a situation that contains a vehicle (in this case a motorcycle) that has never been seen before by the predictor, in an environment with important entities that require attention (in this case an entrance lane). The predictor output is unreliable in this case, potentially leading to erratic and dangerous behaviour of the EV if taken at face value. Here, the Reasoner correctly assigns a low competence to the predictor based on the presence of the motorcycle (high doubt) and the presence of the entrance lane (high importance).

5.2.3 Limitations

While we have successfully shown how competence assessment can be applied for increased safety of a vehicle, there are still several limitations of the approach that need to be addressed. For example:

- To infer the system's competence, we currently rely on a simple embedding procedure: we compute the weighted average over the importance of all the entities and attributes in the knowledge graph.
- The future competence of the system is based on a linear regression over the last N values computed for the system's competence.
- The binary decision on whether the system is competent or not is made on a simple thresholding function on the inferred competence.

The introduction of **Graph Neural Networks** might be a solution to these issues.

5.3 Learning an appropriate degree of caution

5.3.1 Motivation

Since it will be impossible to predict every possible eventuality it is a prudent strategy to design a learnable level of computational **caution**, somewhat akin to *attention*, as this is also something that humans do. Some simple examples might clarify this notion:

1. Suppose the AV is driving on a relatively narrow and empty road and approaching a cyclist ahead. The *sensor* system will detect this cyclist, and the *attention* system might decide to focus more resources on the cyclist. The *intention prediction* system will predict that the cyclist will continue along the same straight path, but the *caution system* should dictate that the over-take manoeuvre should leave a generous amount of space between the vehicle and the biker (especially is the latter is a youngster or a frail elderly). There is no guarantee that this cautious behaviour will be learned through training as there there is likely no situation in which a biker did make an unexpected manoeuvre.
2. In some situations, (e.g. the sun low above horizon), from your perspective you might be able to see everything clearly (back-lit by the sun) but the approaching vehicle might be temporarily blinded – again this is a state-of-mind which requires anticipation and should raise the level of caution.

The question then becomes: can we train a system that outputs a reliable and operable level of caution (to be used as a parameter in other decision making and control tasks), based on:

- accurate sensor input (UvA),
- prior symbolic knowledge (e.g. knowledge graphs - TNO)
- learning from experience in a reliable fashion (e.g. using graph neural networks: CWI)

A reliable estimation of the level of computational **caution** could be used to give an indication of the AI competence level, as indicated in the architecture illustrated in Fig. 5.1.

5.3.2 Graph Neural Networks for estimation of caution

To estimate the level of computational **caution**, the knowledge graphs which are based on prior knowledge, could be augmented with sub-symbolic information which indicate the confidence in the information and the contradictions found in the symbolic knowledge. We propose to draw on some recent developments in Graph Neural Networks to combine symbolic and numerical information. Using graphs has several benefits:

- Graphs provide a flexible way of dealing with abstract concepts like relationships and interactions as specified in knowledge graphs.
- Graphs can model heterogeneous data and relationships. These relationships are especially important to improve the explainability of the results.
- Prior (domain) knowledge can be encoded in the structure or choice of attributes of graph. This will be important when it comes to explaining the way the networks arrive at their conclusions.
- Graphs suggest natural ways to simplify problems (e.g. by indicating appropriate aggregation levels). This will be an important asset when it comes to concisely explaining the solutions that have been obtained.
- A huge body of research (and software) is available (see for a recent overview (29)).

The general structure of graphs in GNN is depicted in Fig 5.4. Specifically, graphs comprise nodes and (possibly directed) edges. The nodes, edges and the graph as a whole have attributes that are updated during the learning phase.

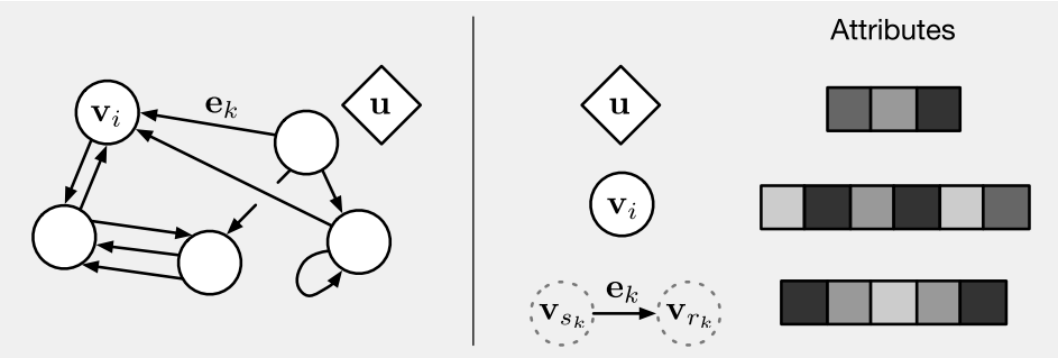


Figure 5.4: Graph structure used in GNN: graphs comprise nodes and (possibly directed) edges. The nodes, edges and the graph as a whole have attributes that are updated during the learning phase. See main text for more information.

In the case at hand, nodes will present objects of interest such as vehicles or infrastructure elements (roads, traffic signs, etc). The corresponding attributes are either assigned during initialisation (e.g. vehicle make) or read out from sensors (e.g. speed).

Updating attributes in GNN Loosely speaking, the learning in GNN proceeds through several stages in which the GNN attributes are iteratively updated as follows (also see Fig 5.5):

- First, the attributes of edges are updated, using the current values for the whole spectrum of attributes (i.e. nodes, edges and global).
- Next, node attributes are updated using the updated values of the edges feeding into the node, as well as the current node and graph attributes.
- Finally, the updated edge and node information is combined with the current

The update and aggregation functions are usually implemented as neural networks, but using tailor-made or knowledge-based functions is also possible. This opens the possibility to include existing (high level) knowledge in the learning process.

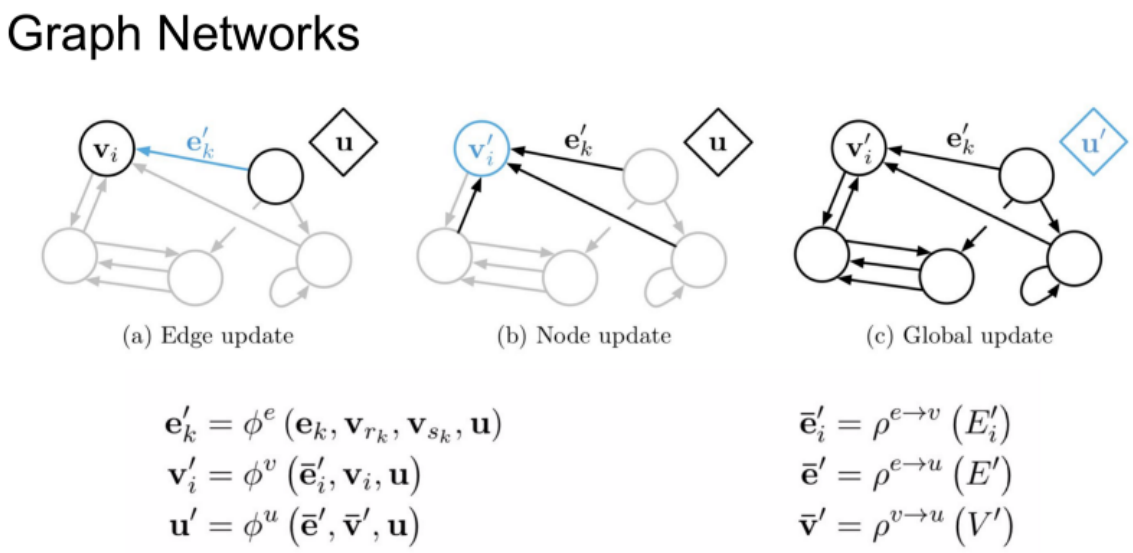


Figure 5.5: Updating graph attributes in GNNs. In a first stage the attributes of edges are updated, next those of nodes and finally the graph attribute. See main text for more information.

Algorithm 1 Steps of computation in a full GN block.

function GRAPHNETWORK(E, V, $\mathbf{u}$)
  for $k \in \{1 \dots N^e\}$ **do**
    $\mathbf{e}'_k \leftarrow \phi^e\left(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}\right)$ ▷ 1. Compute updated edge attributes
  end for
  for $i \in \{1 \dots N^n\}$ **do**
    let $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i,\, k=1:N^e}$
    $\bar{\mathbf{e}}'_i \leftarrow \rho^{e\to v}\left(E'_i\right)$ ▷ 2. Aggregate edge attributes per node
    $\mathbf{v}'_i \leftarrow \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$ ▷ 3. Compute updated node attributes
  end for
  let $V' = \{\mathbf{v}'\}_{i=1:N^v}$
  let $E' = \{(\mathbf{e}'_k, r_k, s_k)\}_{k=1:N^e}$
  $\bar{\mathbf{e}}' \leftarrow \rho^{e\to u}\left(E'\right)$ ▷ 4. Aggregate edge attributes globally
  $\bar{\mathbf{v}}' \leftarrow \rho^{v\to u}\left(V'\right)$ ▷ 5. Aggregate node attributes globally
  $\mathbf{u}' \leftarrow \phi^u\left(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}\right)$ ▷ 6. Compute updated global attribute
  return $(E', V', \mathbf{u}')$
end function

Figure 5.6: Update cycle for GNN (Source: Battaglia et al. Arxiv 1806.01261v3 (2018) (2).

Computational caution as a graph attribute We propose to implement *computational caution* as a global graph attribute, the value of which depends on the node and edge attributes. More precisely we proceed along the following steps:

- The TNO domain knowledge that is relevant for autonomous driving is encoded in a GRAKN database (see grakn.io). Information in this database is organised according to well-defined schemata that specify entities and their relationships. In addition, there are deduction rules that allow the reasoning system to deduce additional facts or conclusions (for an example, see Fig. 5.7). These deduction rules are examples of manually encoded domain knowledge and are triggered whenever fresh relevant information becomes available.

```
invisible-road-uncertainty-1 sub rule,
    when {
        $e isa ego;
        $do (any-vehicle:$e, type-of-road:$t) isa drives-on;
        $rc (type-of-road:$t, part-of-road:$p) isa road-is-composed-of;
        $p has visibility-front $front;
        $front == 0.1;
        $r (reference-ego:$e, reference-road-part:$p) isa observation-relation;
    }, then {
        $r has uncertainty 0.9;
    };
```

Figure 5.7: Example of GRAKN encoded deduction rule that – roughly speaking – concludes that uncertainty on an observation is high when forward visibility is low.

- The GRAKN database is queried for up-to-date sensor information regarding the situational parameters (position, motion, etc) of both the ego-vehicle and other nearby road occupants.
- The reasoning rules implemented in GRAKN are triggered to deduce derived quantities;
- These current data are fed into a pre-defined GNN that has been trained to produce the level of caution as a graph attribute. In this respect, it is important to point out that some of the GNN update rules can be hard-coded rather than learned. This speeds up the learning considerably as it is inefficient to learn simple rules from scratch. As an example, motorcycles tend to have higher speed than cars, and hence observing a motorcycle behind the ego-vehicle should result in a higher level of caution. While it is straightforward to encode, learning this from simulations would be cumbersome and inefficient.
- In summary, the GNN will be an hybrid graph network in which some of the update functions are learned through simulated interaction, while others are hard-coded, depending on the amount of available domain knowledge.

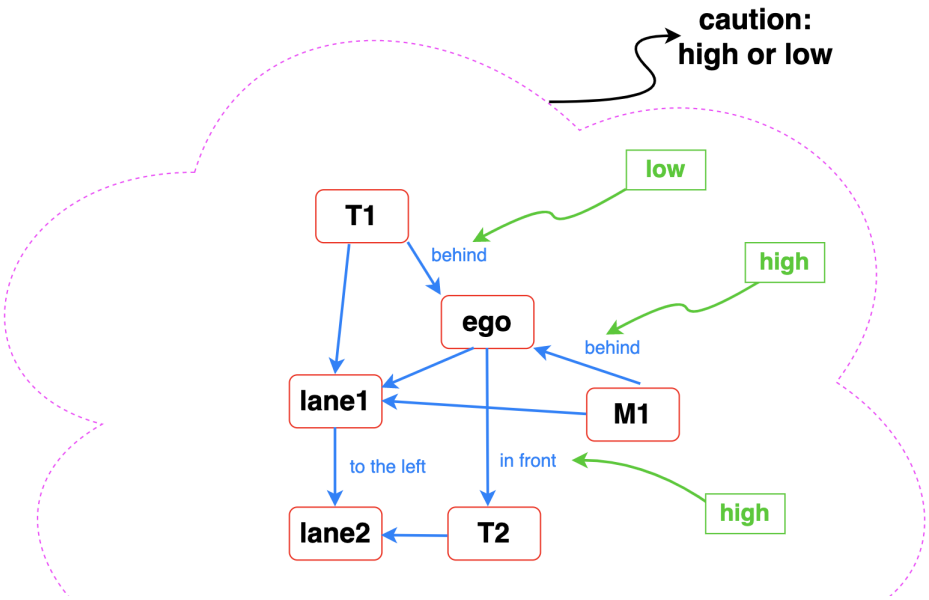


Figure 5.8: Caution as a graph attribute

Chapter 6

Future plans

6.1 Human-interpretable features

In December 2020 researchers of TNO and UvA collaborated in phase 2 of the NWO Perspective round, with a proposal on *AI for Peace, Justice and Safety*. Although the call concentrated on trustworthy infrastructures, the proposal concentrated on the Collaborative, Coordinated, Automated and autonomous Mobility domain. The expertise of the members of the *Automated and Cooperative Driving* group was mainly used for the Safety aspect of the call, by providing situational-aware AI with could be combined with normative AI.

For situational-aware AI perception is the starting point. Perception for autonomous driving is a challenging and fast developing field (13). It is not only important that the vehicle makes the right decisions, but that the driver also understands the decisions, so that trust can be build (14). To make the right decisions, the vehicle has to have situational awareness. This complex task requires 3D scene understanding, which includes sub-tasks such as depth estimation, scene categorization, object detection and tracking, event categorization, and more.

Most state-of-the-art algorithms which use a generic model such as a deep neural network to combine several aspects visible in the scene jointly in order to exploit the complementary nature of the different cues and to obtain a more holistic understanding. In this way they circumvent human-engineered features based on heuristics or intuitions, which may be inaccurate or not generic applicable. Yet, those intermediate features have the advantage of interpretability, and ease of introducing prior knowledge, such as traffic rules. Bridging this gap between human-engineered and machine-learned features remains an unsolved problem to date.

In the proposal we suggest that the pipeline of deep neural networks can be tapped at its intermediate levels, to generate visualizations which have a clear correlation with human-interpretable features, including a probabilistic estimate of its confidence (23). This modular approach allows to build interfaces to the percepts needed in the planning & control software of a self-driving stack.

6.2 Knowledge-graph approach

The knowledge-graph approach is not only essential for autonomous driving, but also for other autonomous control applications. One example of the use the use a more generic application of the knowledge-graph approach is the TNO SNOW-project, where a rescue robot explores a (devastated) house, and increases the situation awareness by classifying the objects that can be seen and inference possible relations to the objects. Cooperation between the MCAS- and SNOW-projects on situation awareness and knowledge-graphs can lead to interesting research.

Another challenge is the incorporation of information encoded in knowledge graphs in graph neural networks. The information from knowledge graphs will have an impact at several levels: the topological structure of GNNs, the update rules as well as the learning. This has only been tested for relatively easy problems but needs a substantial effort for up-scaling to realistic driving scenarios.

6.3 Unstructured environments

Although autonomous driving is already hard on on-road situations, the road provides a relatively well-structured and highly predictable environments with limited obstacle classes such as vehicles, traffic signals and pedestrians.

Compared with this an off-road environment is more challenging and complex, with three dimensional surfaces, compliant soils and vegetation, hundreds of obstacle classes, lower fidelity or limited mapping data, unique platform-surface interactions, continuous motion planning, and no defined road networks or driving rules. This is the environment which one could encounter in the DARPA Racer challenge; a challenge where TNO could participate. Once a vehicle can drive under such unstructured circumstances, it can start to think about driving in the inner city of Amsterdam.

It would be interesting to see how a multi-spectral camera could help with classifying three dimensional surfaces as compliant soils and vegetation. The former could even be interesting for planetary rovers, where estimating the traversability of soils is essential for autonomous driving.

Chapter 7

Conclusion

This report shows the synergy that can be gained in cooperation between the autonomous driving researchers from the UvA, CWI and TNO. In the future we hope to integrate the modules from the proposed architecture into a system that interprets sensor input and transforms this into a learnable level of computational **caution**, based on knowledge graphs, which allows to estimate the uncertainty in the system, alert the human driver in time that the system has difficulties with asserting the situation and make the current assessment explainable based on the symbolic information collected in the knowledge graphs and graph neural networks.

References

[1] M. Bansal, A. Krizhevsky and A. Ogale, "ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst", in "Robotics: Science and Systems", 2019.

[2] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li and R. Pascanu, "Relational inductive biases, deep learning, and graph networks", 2018.

[3] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding", in "Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", 2016.

[4] N. Deo and M. M. Trivedi, "Multi-Modal Trajectory Prediction of Surrounding Vehicles with Maneuver based LSTMs", in "IEEE Intelligent Vehicles Symposium, Proceedings", pp. 1179–1184, University of California, San Diego, San Diego, United States, IEEE, October 2018.

[5] I. Fodi, "Building motion-prediction models for self-driving vehicles", Bachelor thesis, Universiteit van Amsterdam, February 2021.

[6] F. van Harmelen and A. ten Teije, "A Boxology of Design Patterns for Hybrid Learning and Reasoning Systems", *arXiv.org*, May 2019.

[7] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask r-cnn", in "Proceedings of the IEEE international conference on computer vision", pp. 2961–2969, 2017.

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

[9] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", June 2016.

[10] D. Hendrycks and K. Gimpel, "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks", *Proceedings of International Conference on Learning Representations*, 2017.

[11] J. Houston, G. Zuidhof, L. Bergamini, Y. Ye, L. Chen, A. Jain, S. Omari, V. Iglovikov and P. Ondruska, "One Thousand and One Hours: Self-driving Motion Prediction Dataset", in "Proceedings of the Conference on Robot Learning (CoRL)", November 2020.

[12] Z. Huang, Z. Chen, Q. Li, H. Zhang and N. Wang, "1st Place Solutions for Waymo Open Dataset Challenges – 2D Detection", CVPR Workshop on Scalability in Autonomous driving, 2020.

[13] J. Janai, F. Güney, A. Behl, A. Geiger *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art", *Foundations and Trends® in Computer Graphics and Vision*, volume 12(1–3):pp. 1–308, 2020.

[14] J. Koo, J. Kwac, W. Ju, M. Steinert, L. Leifer and C. Nass, "Why did my car just do that? Explaining semi-autonomous driving actions to improve driver understanding, trust, and performance", *International Journal on Interactive Design and Manufacturing (IJIDeM)*, volume 9(4):pp. 269–275, 2015.

[15] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning", *Nature*, volume 521(7553):pp. 436–444, May 2015.

[16] M. Liang, B. Yang, Y. Chen, R. Hu and R. Urtasun, "Multi-Task Multi-Sensor Fusion for 3D Object Detection", in "Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)", June 2019.

[17] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, "Microsoft COCO: Common objects in context", in "European conference on computer vision", pp. 740–755, Springer, 2014.

[18] R. McAllister, Y. Gal, A. Kendall, M. van der Wilk, A. Shah, R. Cipolla and A. Weller, "Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning", in "IJCAI International Joint Conference on Artificial Intelligence", pp. 4745–4753, University of Cambridge, Cambridge, United Kingdom, January 2017.

[19] A. Meyer-Vitali, R. Bakker, M. van Bekkum, M. d. Boer, G. Burghouts, J. v. Diggelen, J. Dijk, C. Grappiolo, J. d. Greeff, A. Huizing *et al.*, "Hybrid ai: white paper", Technical report, TNO, 2019.

[20] Onderzoeksraad voor Veiligheid, "Wie stuurt? Verkeersveiligheid en automatisering in het wegverkeer", Nov 2019.

[21] T. van Orden and A. Visser, "End-to-end Imitation Learning for Autonomous Vehicle Steering on a Single Camera Stream", in "Proceedings of the 16th Intelligent Autonomous Systems conference", Jun 2021, submitted.

[22] J.-P. Paardekooper, M. Comi, C. Grappiolo, R. Snijders, W. van Vught and R. Beekelaar, "A Hybrid-AI Approach for Competence Assessment of Automated Driving Functions", in "AAAI's Workshop on Artificial Intelligence Safety", February 2021.

[23] P. Palasek, N. Lavie and L. Palmer, "Attentional demand estimation with attentive driving models", in "British Machine Vision Conference", The British Machine Vision Association, 2019.

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement", preprint arXiv:1804.02767, 2018.

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

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, "Proximal policy optimization algorithms", preprint arXiv:1707.06347, 2017.

[27] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset", in "The IEEE Conference on Conference on Computer Vision and Pattern Recognition", pp. 2446–2454, 5 2020.

[28] T. Wiggers and A. Visser, "Learning to drive fast on a DuckieTown highway", in "Proceedings of the 16th Intelligent Autonomous Systems conference", Jun 2021, submitted.

[29] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang and S. Y. Philip, "A comprehensive survey on graph neural networks", *IEEE transactions on neural networks and learning systems*, 2020.

Appendix: Code installation instructions

MCAS Real world scenario

This are the instructions from the private github repository, as published by Thomas Wiggers at the end of the project (December 2020).

Introduction

The Meaningful Control of Automated Systems (MCAS) initiative aims to accelerate research on the safety of self driving systems. To do so both simulated and real world experiments are required. Reproducibility is an important part of research, and therefore this repo contains all steps necessary to repeat the physical experiments performed as part of the MCAS initiative.

This repository focuses on the fysical experiments which make use of the duckietown environment and jetracer vehicles.

Track layout

All experiments are performed on the Duckietown Highway. This is an adapted version of the conventional duckietown towns, featuring higher speed corners and on- and off-ramp merging. The road width is identical to the standard duckietown, and the corner radius has been increased to TODO.

TODO include image

Hardware setup

The vehicle used in the experiments is the high speed, front wheel steering JetRacer by waveshare

The assembly manual for the JetRacer can be found here.

Steering calibration

The steering wheel calibration step is best performed in these steps:

- Disconnect the steering column.
- Set the servo to zero.
- Connect the steering column.
- Adjust the steering column such that the wheels are straight.

Software setup

The instructions for installing Ubuntu on the Jetson nano can be found here.

The drivers for the JetRacer can be cloned from this repository. This is used to control the motors of the car.

The instructions for installing these drivers can be found here

Testing

With the jetracer drivers installed, the hardware can now be tested using the notebook `jetracer/notebooks/basic_motion.ipynb` .

Confirm that the rear wheels rotate in the correct direction. If this is incorrect, the direction can be reversed by switching the connectors of the two rear motors. Also test that the steering is unbiased, and does not interfere with the chassis at the steering limits.

TODO link to repo with our code

For troubleshooting, please see the section below.

Usage

To use the jetracer copy a trained parameters (such as `ppo_net_params.pkl`) to the directory `robocar_jetson/models/`. Then in a separate terminal (or tmux session) start the motor control server with `python3 movement/racingcar.py`.

To manually control the racecar, use `python3 movement/racingcar.py --control`

To use lanefollowing, use `python3 use_imitation_jetracer.py --speed SPEED`. A sensible speed on the jetracer is 0.75.

To use Optitrack for navigation use the command `python3 -m routefinding.drive.py --speed SPEED --target TARGET`. Target can be used as parameter to select which coordinates from the built in list to drive to. TODO this is for testing not for user friendlyness. This makes use of the Reeds-Shepp algorithm for the shortest path between the current location and the target location.

Simulation and training

Details on training can be found in training.md on the private github repository

Running real world experiments

How to run it? How to collect datasets?

OptiTrack

Details on the setup and usage of Optitrack for GPS like information can be found in optitrack.md on the private github repository

TODO ROS

Troubleshooting

Hardware troubleshooting

No video output

Verify that there is an SD card in the jetson, and that it has been flashed with an operating system.

JetRacer rear motor direction reversed

Swap the left and right motor connectors.

Motor control does not work

Ensure that the waveshare github is used, and not the nvidia jetracer github.

Final note

Always ensure that the connections are solid! Improperly soldered connections can often be a source of unstable behaviour.

Software troubleshooting

Running in the background or without display

`xvfb-run -a cmd` is useful to run without X. The -a option is used to auto select a free display, which allows parallel use of the command.

MCAS CARLA simulation scenario

This are the instructions from the private github repository, as published by Thomas van Orden at the end of the project (December 2020).

Introduction

This README gives a brief tutorial of all the proposed solution's aspects: from dataset generation till validating the neural network. Many code has been adapted from CARLA tutorials. Training and testing the neural network is done via Tensorflow. Please read the requirements section carefully to prevent cross-package dependency bugs.

Requirements

The repo is tested on a system with the following specifications:

- Ubuntu 18.04.4 LTS (`GNU/Linux 4.15.0-128-generic x86_64`)
- NVIDIA Driver Version: 450.57
- CUDA Version: 11.0
- 2 GPUs
  - NVIDIA GeForce RTX 2080 Ti
  - NVIDIA TITAN V
- Packages/Software
  - CARLA 0.9.9.4
  - CARLA Additional Assets 0.9.9.4
  - Conda (Anaconda)
  - All packages included in `MCAS.yml`

If you not have CARLA installed yet, please refer to the *Installation instructions below*. The code in from the private github repository should be cloned in the parent folder of the CARLA installation folder. For example:

```
AutonomousDriving/
    CARLA/
        CarlaUE4/
        CarlaUE4.sh
        CHANGELOG
        ...
    MCAS/
        README.md
        train.py
        main.py
        ...
```

Conda environment

The file MCAS.yml is a conda environment export with all the packages needed to use this repository. Please note that this environment is tested on a machine with the above mentioned specifications. If you not yet have conda installed, please follow this official instructions to do so.

Create a new conda environment:

- Open a terminal
- Make sure conda is working with `conda env list`. If the error `conda: command not found` is raised, please `run source ~/.bashrc` or restart your terminal.
- Exit the current environment, if there is one activated, with conda deactivate
- Create a new environment from the MCAS.yml file with `conda env create -f MCAS.yml` The environment can now be activated with `conda activate MCAS` and deactivated with `conda deactivate`.

Throughout this documentation all commands should be run inside this environment, **unless explicitly mentioned differently**.

Troubleshooting

Your machine might have an other configuration that needs other packages or package versions than provided in the conda environment. This section discusses the most common packages that causes failures.

cuDNN

Since we rely on Tensorflow and Keras to train the models, the right match between your CUDA version and the cuDNN package is very important. In a terminal, check your CUDA version with `nvidia-smi`. At the top right the CUDA version is listed. Conda has different versions of cuDNN available. Visit this page to see which version matches your CUDA version. If multiple cuDNN version match your CUDA version, a rule of thumb is to always take the most recent cuDNN version. If you have found the right version:

- Open a terminal
- Activate the conda environment with `conda activate MCAS`
- Uninstall the old cuDNN version with `conda remove cuDNN`
- Install the right cuDNN version with `conda install cuDNN={YOURVERSION}`

KERAS and Tensorflow

This repository requires the Tensorflow-GPU 2.3 version. However, if your CUDA version does not support this version of Tensorflow older Tensorflow version might work too. Please note that we do not provide support or documentation for this, but this page has a listed of tested configurations for Tensorflow-GPU. Probably the most important difference with older Tensorflow version is that EfficientNet was not yet included. In `cnn_steering_discrete.py`, please remove EfficientNet from the import. The import should look something like this:

```
from tensorflow.keras.applications import ResNet50, ResNet101V2, DenseNet121,
      Xception, EfficientNetB7.
```

CARLA Installation

CARLA offers multiple options for installation. I recommend using the option described under section B of the Quick Start. In short, this comes down to:

- From this page download `CARLA_0.9.9.4.tar.gz` and `AdditionalMaps_0.9.9.4.tar.gz`
- Move `CARLA_0.9.9.4.tar.gz` into the `AutonomousDriving/CARLA folder`.
- Unpack the file with for example `tar -xzf CARLA_0.9.9.4.tar.gz`.
- Move `AdditionalMaps_0.9.9.4.tar.gz` into `AutonomousDriving/CARLA/Import`, with for example
  `mv AdditionalMaps_0.9.9.4.tar.gz AutonomousDriving/CARLA/Import`.
- In the main folder CARLA, run ./ImportAssets.sh to import the additional assets.
- `CARLA_0.9.9.4.tar.gz` could be removed now to save disk space, with for example `rm CARLA_0.9.9.4.tar.gz`.

CARLA is now installed and ready to be started.

Starting

The CARLA server can be started with a number of options. The default way to start the server simulation is to run `./CarlaUE4.sh` inside the installation folder. To set the quality level of the simulation, use `--quality-level={Low,Epic}`. You can specify the RCP port for client connections with `-carla-rpc-port=N`. Streaming port is set to `N+1` by default, or can be set with `-carla-streaming-port=N`. A full list of command line options can be found here.

After starting the script, you should see the following output in the terminal:

```
$ ./CarlaUE4.sh
4.24.3-0+++UE4+Release-4.24 518 0
Disabling core dumps.
```

CARLA terminal

It is important to not close this terminal, since the simulation will be terminated then. If you would like to run CARLA in the background, I recommend using a terminal multiplexer such as tmux.

Inspection

Changing the map, setting a fixed FPS and other configurations of the simulation can easily be adjusted through the `config.py` script. The script is located in `AutonomousDriving/CARLA/PythonAPI/util/`. To make sure that the dependencies for this script are correct, make sure to activate the conda environment. Open a new terminal and start the environment with `conda activate MCAS`. A short summary of the script's interesting features:

- List available maps and weather conditions with `-l, --list`.
- Change the map of the simulation with `-m, --map {Town10HD}`
- Change the weather condition with `--weather {ClearNoon}`
- Set a fixed FPS with `--fps N`
- Enable or disable rendering of the simulation with `--rendering` and `--no-rendering`.

Any of the above options will take affect in the simulation. If there are clients connected with the simulation, a time-out error might occur. Reconnecting should resolve this issue.

Troubleshooting

CARLA might fail to start. A common error code is raised signal 11. Please try and verify the following steps before starting CARLA again. The main reason for CARLA crashing is another CARLA ghost process running on the machine. This is probably due to CARLA exiting abnormally which causes processes to be kept alive.

- Make sure a display is attached to the machine. A virtual display is sufficient too.
- If applicable, make sure the virtual display is attached with `ps -aux  grep Xvfb—`.
- Make sure there are no other CARLA instances running with `ps -aux  grep Carla—`.
- Make sure the configured ports, default 2000 and 2001, are listening with `sudo lsof -i -P -n`.
- If none of the above helps, a reboot of the machine might fix the problem.

If you encounter an error similar to the following from your CARLA client:

```
WARNING: Version mismatch detected: You are trying to connect to a simulator
that might be incompatible with this API
WARNING: Client API version = 0.9.9.2
WARNING: Simulator API version  = 4dc4cb81
Segmentation fault (core dumped)
```

You probably have used an older version of Python than the distributed .egg files inside the `CARLA/PythonAPI/carla/dist` folder. Make sure you choose one of the Python versions that is provided here.

Dataset generation

To create a large dataset, the `dataset_generator.py` can be used to automatically create one. The world running at the server is restarted by default and an ego vehicle is spawned. By default this is a Tesla Model 3. See the command line arguments for more configuration options.

The ego vehicle will be spawned to waypoints covering the whole map and capturing images at every time step. The town running at the server is used as default map. In order to change this, one could use the `config.py` script inside `CARLA/PythonAPI/util/config.py`. By default, only a RGB camera is attached to the ego vehicle. However, a segmentation camera with segmentation color pallet can be added as well (see `dataset_generator.py -h`).

By default, the script will run for 500 000 frames. At every tick, a random waypoint from a set of possible waypoints is chosen. Multiple possible waypoints may exist at one vehicle's location due to for example junctions. Gaussian (0,0.8) noise is added to both the x and y location of the waypoint.

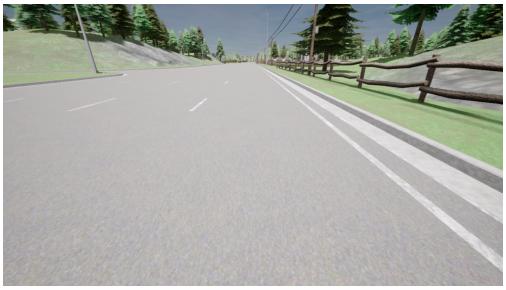
To enhance a consistent dataset, the ego vehicle has a fixed speed. This may be adjusted by setting a different constant speed (speed), or by adjusting the acceleration parameter (acceleration).

All images will be stored in a folder dataset inside the current directory. The default filename is in the format: `dataset/{PREFIX}_{DATE}_{IMG.FRAME}_CONTROL={CONTROL}.jpg`. PREFIX is set to "RGB" and "SEG" for the RGB camera and segmentation camera accordingly.

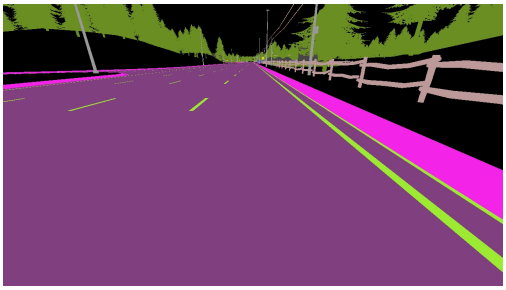
An example to create a dataset with RGB images and segmentation images, using a speed of 15 km/h:

```
python3 dataset_generator.py --host 127.0.0.1 --port 2000
        --filter vehicle.tesla.model3 --speed 15 --segmentation_camera
```

An example of an RGB sample:



An example of the according segmentation sample:



Training

Our approach to learn the steering angle is to learn a steering angle class from a discrete action space. We provide multiple convolutional neural networks (CNN) to choose from. To be able to train a network, the script expects a dataset as created by `dataset_generator.py` where every picture is 1080 (width) by 720 (height) pixels. Also, make sure a folder weights exists in the same directory as where you run the script from!

The discrete action space classes are [-10, -5, 0, 5, 10]. This represents the steering angles in degrees. The steering angles from the dataset are mapped to its closest discrete class.

To train a model open a terminal and activate the conda environment:

```
python3 train.py -N 1000 --split 0.8 --model Xception --epochs 20
  --batchsize 32 --scale 20 --info TEST_RUN1 dataset/20201201/rgb
```

A short summary of the command line options:

- To train on a subset of the whole dataset, use `-N NUM` to select the first NUM images from the dataset.
- The validation split can be adjusted by `--split`. The given value corresponds to the training proportion. For example, `--split 0.6` will use 60% for training and 40% for validation.
- The number of epochs and the batchsize can be easily set by `--epochs and --batchsize`, respectively.
- To limit the memory usage and possibly speed up the training one can rescale the input images. Please note that this might cause lower results. The `--scale` option represents the scaling percentage. For example, if an input image is 1080 by 720 and `--scale 20` is used, the images will be resized on the fly to 216 by 144.
- To store some extra information about the training, one can use the `--info` argument. This might be useful to distinguish training runs later.

All provided model options are:

- NVIDIA, this architecture is inspired by NVIDIA's paper.
- ResNet50, Keras' implementation of ResNet50 with additional Flatten and Dense layer.
- ResNet101V2, Keras' implementation of ResNet101V2 with additional Flatten and Dense layer.
- DenseNet121, Keras' implementation of DenseNet121 with additional Flatten and Dense layer.
- Xception, Keras' implementation of Xception with additional Flatten and Dense layer.
- EfficientNetB7, Keras' implementation of EfficientNetB7 with additional Flatten and Dense layer (only available in Tensorflow 2.3+).

Please note that some models might require extensive memory. Limiting the batchsize and reducing the scale could help to reduce memory usage.

Default configurations

By default the Adam optimizer with learning rate 0.0001 is used in combination with `categorical_crossentropy` loss. The accuracy, precision, recall and F1 score are used as metrics. Class weights are automatically calculated and applied to the training. Especially for unbalanced datasets, this might be helpful. Those options are not adjustable through command line arguments, but can easily be changed in the code itself.

Callbacks

The model weights are saved by the ModelCheckpoint callback. It only saves the best weights by default to reduce disk space usage. The weights are stored in the weights folder. Also, the name of the weight files is based on the current date and all hyperparameters.

For example, a weight filename might be:

```
20201025-143743__dataset_dir_..MCAS_internalMCASdataset20200621__N_131118__
split_0.8__model_ResNet101V2__epochs_50__batchsize_4__optimizer_Adam__scale_
20__info_RESNET101V2_RUN1_NEW_.h5
```

During training, the logs are saved to logs/ and logged to Tensorboard by the Tensorboard callback. The same filename formatting is used. The logs are updated every batch.

To prevent training getting stuck, the learning rate of the optimizer is reduced automatically by the ReduceLROnPlateau callback. By default, if after 7.5% of the total number of epochs (at any given time) there is no improvement in the validation loss, the learning rate is reduced by a factor 10.

Validation and Benchmark

To see your model in action `main.py` can be used. This section will discuss the different options of `main.py` and elaborate on the included benchmark option.

Please note: there is a problem with CARLA and possibly Tensorflow-GPU that causes CARLA and/or Tensorflow to crash if they run on the same GPU. Therefore, we recommend to use a system with two dedicated GPUs such that model inference can be done on one GPU and the CARLA simulator can run on the other GPU. This possible solution is purely based on experience, so other solutions might fix this issue as well.

First of all, to use `main.py`, we need the CARLA simulator to be running. For example, in a tmux session. However, it is not mandatory to activate the MCAS conda environment here. Inside CARLA folder, start the simulator with

```
./CarlaUE4.sh
```

Do not close this terminal/tmux session, since the simulation will be terminated then.

We use an Agent class to interact with the CARLA simulation. Please note that this class has currently only the basic functions implemented. Any additions can be made to the Agent class in `agent.py`. The most important function is `run_step()`, this should return a CARLA VehicleControl instance with the predicted steering angle as steer attribute value.

Basic validation To test a single agent (read: model) main.py can be used in combination with the –agent option. The path to the model's weights should be the value of this argument. To specify the Town to validate the agent in, give the town name as positional argument. An example:

```
python3 main.py --agent Xception_Agent.h5 Town02
```

This launches a new pygame window and spawns our agent with a fixed speed of 5 km/h. The agent's autopilot is disabled and all steering is therefore be done by model inference (`agent.run_step()`). The pygame window is rendered at a variable or given frames per second in which you should be able to see

the agent's performance. In the terminal output the frame number is shown and the model's steering prediction. If the agent has crashed the script will continue to run, to stop the script use Ctrl-C.

Benchmark validation

To compare different models we provide a basic benchmark. Official benchmarks that support CARLA are CoRL2017 and NoCrash. Please refer to their documentation and publication for more details.

The basic idea of our benchmark is to let an agent perform multiple runs while resetting the environment to average out any non-determinism in CARLA. The benchmark can be configured to some extend by `modifying benchmark_config_v1.json`. In addition, a comprehensive Jupyter notebook and visualize script is provided to gain insight in the model's benchmark performance. The benchmark can be activated by running

```
python3 main.py --benchmark --config config_file.json --weights weights_folder
        Town06
```

Benchmark details

The benchmark consists of a number of runs that consist of a number of episodes, for every agent/model. A Metrics class is attached to the Benchmark class to read and write the agent's performance to the benchmark. The benchmark is saved as a Pandas DataFrame. The column names are explained below.

The benchmark logs the following attributes:

- `Run` Run number
- `Episode` Episode number
- `Model index` Model index to distinguish different models
- `Model` Possible model information as given by –info during training
- `Config` The benchmark configuration as specified in the JSON file
- `Agent speed` The agent's speed log: the speed of the agent at every timestep
- `Agent steering` The agent's steering log: the steering angle of the agent at every timestep
- `Start_pos` The start position of the agent
- `End_pos` The end position of the agent
- `Trajectory` The trajectory the agent drove: agent's position at every timestep
- `Duration` The elapsed simulation time
- `Distance` The driven distance in meters
- `Collisions` All collisions; collided object name and impulse value
- The lane infractions logs

The latter is specified per lane marking type. Those lane marking types correspond to CARLA's implementation of the ASAM OpenDrive standard. The following lane marking types are currently included in the benchmark: NONE, Other, Broken, Solid, SolidSolid, SolidBroken, BrokenSolid, BrokenBroken, BottsDots, Grass and Curb.

Benchmark configuration

The benchmark can be configured by a JSON file. An example:

```
{
  "benchmark": {
    "runs" : 5,
    "episodes" : 25,
    "max_distance" : 200,
    "speed" : 10
  }
}
```

Every run starts by restarting the CARLA environment. This comes down to the random initialization as implemented in the World class in `manual_control.py` by `world.restart()`. A lane, collision and rgb sensor are attached to the agent.

Every episode during one run starts by restarting the agent to ensure all metric values are set to their default. The agent is spawned at a random spawn point provided by CARLA.

At every timestep, the agent has to control the steering of the vehicle which is done with `agent.run_step()`. Also, the metrics are updated and terminal output shows the elapsed simulation time and distance traveled.

The episode is finished if the conditions of the benchmark are met. This comes down to three options:

- The agent has collided
- The agent has driven more than or equal to `max_distance` meters
- The elapsed time is more than or equal to `round(max_distance / (speed / 3.6)) * 1.1`
- At the end of every episode the benchmark, and possibly a video (see Video), is saved.

Recommendations

To be able to create reproducible results, it is recommended to use a fixed frames per second and set the simulation in synchronous mode. An example:

```
python3 main.py --benchmark --config config_file.json --weights weights_folder/
--synchronous --fps 25 Town05
```

Benchmark results

To see the benchmark statistics one can use the notebook `BenchmarkResults.ipynb`. As indicated by the comment `# Insert path to benchmark file here`, you can load the benchmark file that is saved as a pickled Pandas DataFrame.

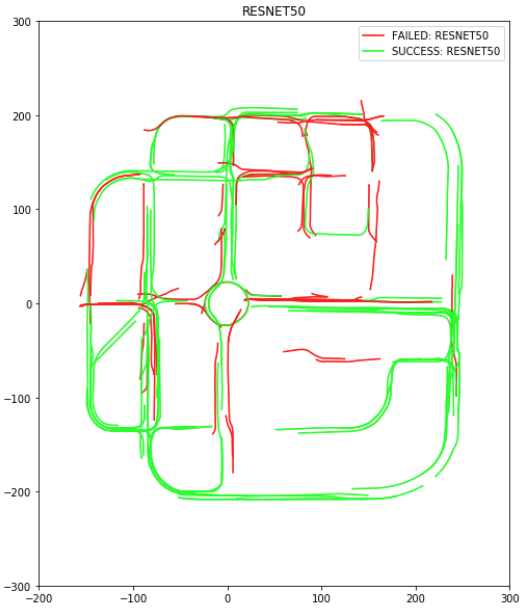
By default for every model, the success rate is calculated per run, the trajectories are plot, and the lane marking collisions statistics are given.

Success rates example per model:

```
------------------------------ RESNET50 ------------------------------
*Run 1 success rate 0.68
*Run 2 success rate 0.52
*Run 3 success rate 0.64
*Run 4 success rate 0.6
*Run 5 success rate 0.52
**Overall runs success rate 0.592 +- 0.064
----------------------------------------------------------------------

------------------------------ XCEPTION ------------------------------
*Run 1 success rate 0.6
*Run 2 success rate 0.48
*Run 3 success rate 0.68
*Run 4 success rate 0.72
*Run 5 success rate 0.64
**Overall runs success rate 0.624 +- 0.0824
----------------------------------------------------------------------
```

Trajectories driven in the benchmark example:



Lane marking type infractions example:

```
------------------------------ RESNET50 ------------------------------
----------------------------------------------------------------------
Total simulation distance: 20228.726115414873
Total simulation time: 6932.959845036268
----------------------------------------------------------------------
*NONE 0.0846 +- 0.1502 (1.712 +- 3.0379)
*Other 0.0 +- 0.0 (0.0 +- 0.0)
*Broken 0.0906 +- 0.1296 (1.832 +- 2.6222)
*Solid 0.2444 +- 0.2811 (4.944 +- 5.6869)
*SolidSolid 0.0656 +- 0.1184 (1.328 +- 2.3959)
*SolidBroken 0.0929 +- 0.1858 (1.88 +- 3.7579)
*BrokenSolid 0.0759 +- 0.1451 (1.536 +- 2.9354)
*BrokenBroken 0.0 +- 0.0 (0.0 +- 0.0)
*BottsDots 0.0 +- 0.0 (0.0 +- 0.0)
*Grass 0.0 +- 0.0 (0.0 +- 0.0)
*Curb 0.0036 +- 0.0251 (0.072 +- 0.5087)
**Overall infractions 0.6577 +- 0.4085 (13.304 +- 8.264)
----------------------------------------------------------------------
```

Segmentation validation

Video
With all above mentioned options, one can use the –video argument as well. In short, this saves a video of the agent's performance as mp4 file. Please note: the current video implementation is memory-extensive. If the script runs for a long period of time, it might crash due to out of memory issues.

TODO: Show –video option from `main.py` and include example video.

TODO: mention different behaviour of `--video` with and without segmentation.

Extra

Segmentation
TODO: How to use Keras `Segmentation.ipynb`. Might need to use another Tensorflow-GPU version.

Spectator
TODO: How to use `spectator.py` in multi computer set-up.