Dream-like simulation abilities for automated cars

DREAMS4CARS

Grant Agreement No. 731593

Deliverable:	D5.3. – Evolved Agent
Dissemination level:	PU - Public
Delivery date:	3/December/2019
Status:	Final

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731593

Deliverable Title	Evolved Agent		
WP number and title	WP5 Agent evolution, evaluation of ability levels and final assessment of the technology		
Lead Editor	Henrik Svensson, HIS		
Contributors	Mauro Da Lio, UNITN		
Creation Date	14/April/2019	**Version number**	0.9
Deliverable Due Date	31/October/2019	**Actual Delivery Date**	3/December/2019
Nature of deliverable	**X**	R - Report	
	X	O – Other – Software, technical diagram	
Dissemination Level/ Audience	**X**	PU – Public, fully open	

Version	**Date**	**Modified by**	**Comments**
0.1	14/April/2019	Mauro Da Lio	Initial draft (initial document structure).
0.2	31/August/2019	Mauro Da Lio	Section 5 dealing with steering actuator inverse model control.
0.3	23/September/2019	Mauro Da Lio	Sections on Jeep Renegade forward models.
0.4	24/September/2019	Mauro Da Lio	Sections on Jeep Renegade lateral inverse model control.
0.5	19/November/2019	Mauro Da Lio	The document content has been restructured by splitting the description of the agent development (which remains in this document) and the quantitative assessment of the system achieved abilities (which has been moved to D5.4). This was done to better adhere the DoA and, also, because the contents of D5.4 are mostly confidential (including materials that cannot be anticipated because it is going to be used in future publications).

			Excerpt of D5.4 which can be made public are given in appendix. As a consequence, in D3.2/3.3 where examples pointing to D5.3 were mentioned, they have to be interpreted as pointing to D5.4.
0.6	19/November/2019	Mauro Da Lio	Section 1 (new) and Appendix 1.
0.7	28/November/2019	Mauro Da Lio	Sections 2.1 and 2.2.
0.8	2/December/2019	Mauro Da Lio	Sections 2.3-5 and section 3.
0.9	3/December/2019	Mauro Da Lio	Final version after internal revision.

Executive Summary

This deliverable describes the process followed for the evolution of the agent.

We first provide an overall assessment of the tests carried out and passed in the quality assurance program: there are 13 Automotive grade tests that have been progressively implemented and passed with codriver versions beginning with 9.5.

In the second part of the deliverable the developmental workflow is described and contrasted with the traditional workflow in order to increase the confidence of potential adopters of the technologies.

The public version of the agent and simulation system is mentioned in section 3, which points to D5.5 (the open data pilot).

Table of Contents

List of Diagrams

1 Evolution of the Agent sensorimotor system

This deliverable describes the evolution of the codriver agent. The document explains the process for the development of the agent and how this workflow differs from the traditional engineering one. The quantification of the system abilities is given separately in D5.4.

1.1 Agent implementations

In Dreams4Cars there are 4 different implementations of the agent:

1) The OpenDS simulator agent,
2) The CarMaker simulator agent,
3) The Jeep Renegade car agent,
4) The MIA car agent.

They differ mostly in the lowest-level control, because the dynamics of the 2 real vehicles (Jeep/MIA) and the 2 virtual vehicles (Carmaker model, OpenDS model) are different.

OpenDS. The OpenDS vehicle dynamics is relatively simple and so there was no need for learning forward and inverse models. The OpenDS agent and the OpenDS environment are public and part of the Open Data Pilot (see D5.5). Hence this deliverable also describes the public agent of D5.5.

CarMaker. The CarMaker vehicle model should be a virtual version of the Jeep Renegade (it was carefully parametrized with state-of-the-art vehicle dynamics identifications methods by CRF). However, it turned out being slightly different than the real vehicle (the neural network forward models could clearly detect the difference).

Jeep Renegade. The Jeep Renegade is the CRF test vehicle. It was used for a variety of evaluation tests on the CRF Safety Centre at relatively high speed (70 km/h).

MIA car. The MIA car is the DFKI test vehicle. It is endowed with a richer set of sensors and a better positioning system and was used for developments as a platform complementary to the Renegade. The MIA car has different engine, steering actuator and overall dynamics than the Renegade. The maximum speed of this vehicle is relatively low and hence its dynamics is to some extent simpler (except for the non-linearities in the actuator subsystems, see D5.5).

1.2 Overview of the agent evolution

The initial codriver agent in Dreams4Cars was version 7.8, corresponding to the final version of the Adaptive project. It was an entirely hand-coded agent that was intended to be used as a benchmark.

The final version of the codriver is 9.7. The most relevant changes from 7.8 to 9.7 involved the inclusion of modules to enable learning methods (D3.2) and the functional architectural improvements described in D2.2. The codriver 9.7 uses an algorithmic scaffold hosting neural network modules (it is not a black box monolithic neural network). In addition to architectural improvements, between 7.8 and 9.7 there were also software modifications involving bug fixes, an updated interface with the host environment (v12.04) and several other improvements in the hand-coded scaffold of the agent.

The assessment and monitoring of the agent progress were carried within WP1.4 (see D1.4). To shortly recap, a progressively increasing number of “standardized” tests have been arranged in the CarMaker environment and used to quantify the agent performance.

The suite of tests is the following:

1. **Speed Adaptation in Highway Scenario (Oval L1).** The system tested at SAE Automation Level 1 (L1). The system has to adapt speed according to speed limits and road curvature. The initial speed is 120 km/h.
2. **Speed Adaptation in Test Track (Safety Centre L1).** Same as above but in the (virtual) Safety Centre Test Track, where narrower curves are present. The initial speed is 70 km/h.

3. **Lane Following in Highway Scenario (Oval).** The lane keeping/following ability of the system is tested at SAE Automation Level 2 (L2), including testing level transitions. The initial speed is 120 km/h.
4. **Lane Following in Test Track (Safety Centre).** Lane following abilities are also verified in Safety Centre Test Track, with narrower curves (as low as 60 metres curvature radius).
5. **Lane Change.** The lane change ability is verified with standard Lane Change manoeuvre. Two stationary vehicles are located one on the left lane and the other on the right lane 50 m behind in a way that the agent must change at the gap between the two and then returns into the right lane. The initial speed is 90 km/h.
6. **Overtake slow Vehicle.** The slow vehicle scenario is used to verify the ability of the system to perform an overtake manoeuvre. The vehicle is approaching a slower vehicle and is expected to overtake. The initial speed is 120 km/h. The slow vehicle speed is 30 km/h.
7. **Euro NCAP - Car Following (CCRm).** According to test plan, also some Safety Assist scenarios related to AEB (Autonomous Emergency Braking) C2C (Car-to-Car) system have been implemented as defined in the Euro NCAP protocol. In the Car-to-Car Rear Moving (CCRm) scenario the host vehicle is approaching a slower vehicle ahead and has to slow down in order to avoid collision. Between the different configuration of the CCRm scenario, the worst case has been implemented with the vehicle arriving at 80 km/h and the obstacle travelling at 20 km/h.
8. **Euro NCAP - Stationary Obstacle (CCRs).** In the Car-to-Car Rear Stationary (CCRs) Euro NCAP scenario the vehicle has to stop in order to avoid collision with a front obstacle standing in the lane. The worst case has been considered, with the vehicle entering the scenario at 80 km/h.
9. **Euro NCAP - Front Vehicle Braking (CCRb).** In Car-to-Car Rear Braking (CCRb) Euro NCAP scenario our vehicle is following a front obstacle at constant speed of 50 km/h, then the front obstacle decelerates with constant deceleration until it stops. This scenario has been implemented with -2 m/s^2 constant deceleration of the front obstacle.
10. **Euro NCAP - Walking Pedestrian (CPLA).** Euro NCAP defines tests for interactions with Vulnerable Road Users (VRU), where the vehicle has to react with an Autonomous Emergency Braking (AEB) manoeuvre. In the CPLA (Car-to-Pedestrian Longitudinal Adult) test a pedestrian is walking along centre of the lane at 5 km/h (in same direction of the car). The incoming vehicle has to slow down in order to avoid collision or reduce impact speed. This test has been implemented with the vehicle arriving at 50 km/h.
11. **Euro NCAP - Crossing Pedestrian (CPNA).** In the CPNA (Car-to-Pedestrian Nearside Adult) test a pedestrian *unexpectedly* crosses the road, at 5 km/h speed when the vehicle arrives. The vehicle must brake in order to avoid collision or reduce impact speed. This test has been implemented with the vehicle arriving at 50 km/h and 75% offset (side position of the pedestrian with respect to the vehicle axis).
12. **Euro NCAP - Running Pedestrian (CPFA).** In the CPFA (Car-to-Pedestrian Farside Adult) test a pedestrian is crossing the road running at 8 km/h speed when the vehicle is arriving. The vehicle has to brake in order to avoid collision or reduce impact speed. This test has been implemented with the vehicle arriving at 50 km/h
13. **Complex Highway Scenario.** A highway scenario has been implemented in which the vehicle has to travel for 10 km in the traffic flow on two lanes alternating car following, lane changes and overtakes in order to travel safely as fast as possible. The traffic is generated and moved with stochastic characteristics, and each traffic vehicle reacts to the other vehicles. In this way different situations are encountered by the system that have not been specifically programmed.

The 13 tests above formed the test set used for monitoring the quality of the developed agent in D1.4. In addition, a 14th scenario was developed in the final part of the project, focusing on complex urban situations with traffic generated in co-simulation with PVT-Vissim (a traffic simulator: https://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/) as follows:

14. **Urban Scenario.** A complementary urban scenario has been implemented. The host vehicle travels on road with priority with two driving directions and crossings; speed limit is 50 km/h. Realistic traffic simulation is created with Vissim (in co-simulation with CarMaker).

Figure 1 gives a global assessment of the number of tests that were carried out and passed with successive versions of the agent. One remark that is evident from the picture is the fact that, beginning with version 9.5 every new test that was introduced was passed by the agent (we do not count here a few failures that were due to the ancillary environment).

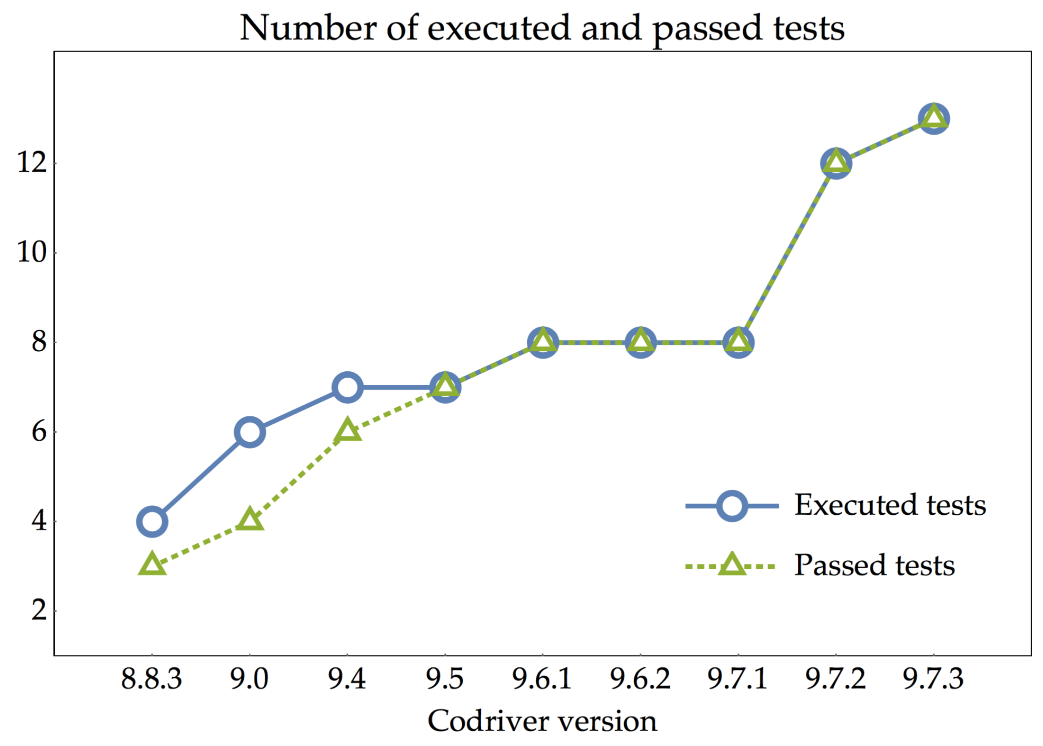

Figure 1: Overview of agent evolution. The chart shows the number of QA tests versus the number of passed test.

Commenting the progress in more depth we can make the following additional remarks. Version 8.8.3 was the codriver version that, following a number of software modifications to adapt the original agent of AdaptIVe, was the first that entered the assessment/monitoring programme. Between versions 8.8.3 and 9.0 the Carmaker environment was also upgraded with the realistic parametrization of the Renegade test vehicle.

With version 9.0 we began testing the agent in realistic conditions (albeit in the simulation environment) in terms of vehicle dynamics, actuator models, electronic horizon model, sensor models and the final interfaces. The following versions marked the progressive implementation of neural network control, which is described in following section 2.

2 Developmental workflow

This section describes the workflow used for development of the agent and compares it to more traditional workflows with the goal of *clarifying how the development process could be organized in comparison with more traditional processes* (and also which parts of the development process can be adopted individually without needing the entire agent architecture).

The workflow follows the scheme given in D3.3 Figure 1, which is here reported for readers' convenience (Figure 2).

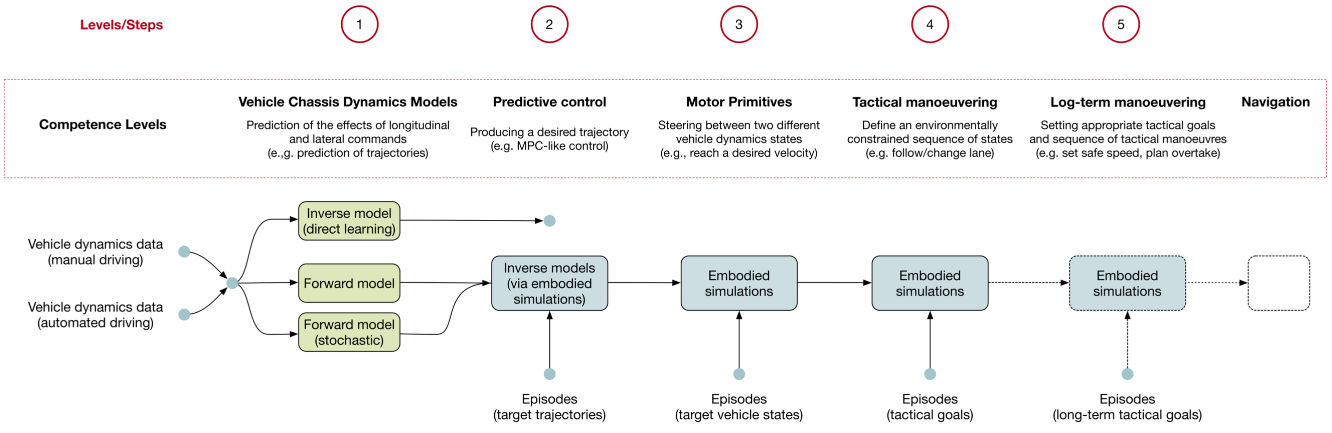
Levels/Steps
1
2
3
4
5
Competence Levels
Vehicle Chassis Dynamics Models
Prediction of the effects of longitudinal and lateral commands (e.,g. prediction of trajectories)
Predictive control
Producing a desired trajectory (e.g. MPC-like control)
Motor Primitives
Steering between two different vehicle dynamics states (e.g., reach a desired velocity)
Tactical manoeuvering
Define an environmentally constrained sequence of states (e.g. follow/change lane)
Log-term manoeuvering
Setting appropriate tactical goals and sequence of tactical manoeuvres (e.g. set safe speed, plan overtake)
Navigation
Vehicle dynamics data (manual driving)
Vehicle dynamics data (automated driving)
Inverse model (direct learning)
Forward model
Forward model (stochastic)
Inverse models (via embodied simulations)
Embodied simulations
Embodied simulations
Embodied simulations
Episodes (target trajectories)
Episodes (target vehicle states)
Episodes (tactical goals)
Episodes (long-term tactical goals)

Figure 2: Development workflow.

2.1 Step 1 – Vehicle dynamics models

The process starts with the collection of data for learning the models of the vehicle dynamics. This step corresponds to two steps in the traditional workflow:

a) Developing (an equation based) mathematical model of the vehicle dynamics;
b) Parametrizing the mathematical model.

These two steps are, instead, just one single step with the neural network approach, where the model and its parametrization are learned simultaneously.

One important idea here is using neural networks with a structure biased toward the superposition of effects that physical insight can give [1]. Hence while the role of a human modeller in the traditional case is describing in minute details all important phenomena (and simplifying and deciding which one to neglect), with a physically inspired neural network topology the role of a human designer is less critical and limited to specifying which are the effects (forces) occurring in the system and which are the causes for these forces. This approach leads to producing neural networks that are biased toward efficient learning of a particular class of models and still flexible enough to accommodate unexpected/unthought effects.

Another important remark is that, when using physically biased neural network structures, the learning and parametrization of a vehicle model is very efficient and can be carried out with little data (a neural network with generic structure like, e.g., a multilayer perceptron would not be sample efficient and would require more training data; e.g., [2], D5.4).

Conversely parametrization at point b) often requires costly and long experiments, especially for tires. Further discussion on these ideas can be found in papers [1]–[3] as well as in D3.3 section 3.1 in the appendix and in D5.4.

Examples of the learning of forward models for the MIA car (longitudinal and lateral models) are given in the Open Data pilot (D5.5).

Further development of the notion of forward model learning is constitued by a technique that allows learning *stochastic* forward models, which is explained in D3.2 section 3.1.6[1]. This makes the basis for robust adaptive control at following developmental steps.

[1] At the current time this is not explained in the public version (D3.3) of the methods in order not to anticipate the contents of possible upcoming publications and also to reserve some IPR.

We can observe that at step 1, inverse dynamics of the vehicle can also be learned directly from the collected data (for example following the principles explained in [4]). This however turned out being far less efficient than via episodic simulations (D3.2 section 3.2).

2.2 Step 2 – Predictive control

Step 2 is concerned with the synthesis of inverse models of the vehicle dynamics. Such inverse neural networks receive future desired trajectories in input and produce the current (instantaneous) control commands.

Iteratively using the inverse network within a receding horizon scheme drives the plant onto that trajectory. An inverse neural model is hence *functionally equivalent to Model Predictive Control* (MPC) in the traditional work-flow.

However, there are substantial differences in the performance that can be expected.

MPC is usually based on linearized (linear time variant) vehicle dynamics models. This simplification is (almost always) necessary to obtain a quadratic programming problem, which can be formally guaranteed to be convergent. Non-linear MPC exists, but is much more computational expensive and its convergence is not universally granted.

Conversely the inverse neural network is trained offline and works as a large lookup table inline with no issues related to real time computation. Furthermore, the network can be trained on a generic forward model that does not need to be linear. The network can be tested offline and, if given a particular explainable architecture, some formal properties can also be guaranteed (e.g. BIBO stability can be assured with particular network architectures). One can also seek for simplified networks (for example linear parametric controllers) that best invert nonlinear plants.

Another aspect where inverse neural network go beyond the traditional approach is robust control. D3.2 section 3.2.3 explains the synthesis of robust inverse models exploiting the knowledge of the stochastic forward models learned at step 1. It has to be remarked that robust MPC is still a research topic with several complex implications [5]. Further discussion is given in D3.2 section 3.3.6.

Concerning the method for training the inverse models two options are available: a) from data (D3.2 section 3.2.1) or via episodic simulations (D3.2 section 3.2.2). Episodes can be conveniently generated via crossover and mutation operator from recorded example trajectories. We have shown several advantages of episodic training versus training on data. These include the generation of an inverse model that better cancels the forward dynamics as well as the ability to smooth the (controller) dynamics.

2.2.1 Example: learning and compensating the steering actuator slow dynamics

With the realistic simulation environment introduced with Codriver version 9.0, several problems arose (also because of some software glitches that followed the conversion from the AdaptIVe agent). Many tests (about 50%) initially failed, as shown in Figure 1 for version 8.8.3 and 9.0.

Among these the major problem found was related to the steering actuator. In particular, the actuator used in the Jeep Renegade was an adaptation of the device used in the standard production vehicles for lane keeping assistance (LKA). It was not a device specifically designed to be fast and accurate (no resources were allocated for the development of an ad-hoc device which would have been expensive on one side and not the central focus of Dreams4Cars on the other side). The device has several limitations, which are functional for LKA but not desirable for autonomous driving, among which a lowpass band (about 1Hz) and a dead zone in the straight position.

As a consequence, the original control schemes that worked fine with faster vehicle dynamics (and still works fine with the OpenDS and the MIA car) performed poorly and could not be accepted.

In order to collect a complete picture several delay compensation schemes were tested but did not work greatly because the steering actuator cannot be modelled as a simple delay. As a collateral test we verified that, if the actuator could be passed (i.e., if it had infinitely fast dynamics) the quality of the vehicle lateral control would have been restored.

Hence, with version 9.4 a neural network model of the steering actuator was trained (both the inverse and the forward model). The inverse model was trained from data with the methods of D3.2 section 3.2.1 (i.e., the “simple way” without using episodic simulations).

The inverse model was used to rectify the actuator dynamics and worked fine, restoring the original controllability of the vehicle. This success story means that (albeit actuators with good dynamics are required for Automated Driving) there may be a trade-off where instead of using very precise actuators one can relax somewhat the specifications as long as the actuator has dynamics that can be compensated with a neural network.

Appendix 1 - CRF steering actuator compensation excerpted from D5.4 gives more details on this case study (in particular see Figure 5).

2.3 Step 3 – Motor primitives

This step corresponds to tactical-level trajectory planning where, given a short-term goal (for example change the lateral position to the centre of the next lane) a trajectory that reaches the target position is produced (which can be used for level 2 predictive control).

In traditional approaches this step is carried out with a variety of trajectory planning methods (for example, with Optimal Control a trajectory that terminates in a desired state while satisfying pre-defined optimality criteria can be produced).

In the approach of Dreams4Cars a neural network generating a trajectory given the target endpoint is trained via episodic simulations exploiting the inverse/forward models of step 2 (D3.2 section 3.3). This network is also referred as a generator of motor primitives, because it provides an atomic trajectory for any given goal.

The motor primitive network functionally works like any other trajectory planning methods (it returns a trajectory given a goal position). It uses the learned models of the plant and its low-level controller just like, for example, Optimal Control uses the mathematical model of the vehicle dynamics. However, as in step 2, the network is trained offline, can be tested offline and works with no computational and convergence issues inline (see D3.2 section 3.3.6 for a comparison with Optimal Control). A further possibility is that the trajectory generator network can be trained to be robust, i.e., to generate trajectories that minimize the mathematical expectation of the final position error or any other metric (D3.2 section 3.3.2).

Despite the above features may look interesting, the most important feature of using a neural network for trajectory generation is that it can be trained to predict the trajectory cost (or value) without actually predicting the trajectory in details. In other words, networks that predict the values of actions without actually simulating the actions can be obtained. These form the excitatory circuits necessary to compute the “salience” function (D2.2 section 2.1.2). With some adaptation the same approach can be used for creating the inhibitory circuits.

Conversely, with a traditional approach, evaluating the value of one potential action necessarily requires to first compute a trajectory. In other words, in order to evaluate the particular cost of one possible action (say change lane) methods like e.g., Optimal Control need to solve the motion problem at the lowest level, with the computation of a detailed trajectory. Combining this with the fact that inline trajectory planning is computational expensive, limits the number of possible alternatives that can be evaluated. Neural networks permit *dense* action priming, i.e., with almost continuous spectrum of possible actions among which to choose (the only way to achieve such dense priming is with analytical approximate solutions of the trajectory planning problem that provide analytical expressions for the trajectory costs). In turn dense action priming significantly contributes to adaptive behaviours (for example finding a way to move in a partially occluded lane, or fitting in between lanes in emergency situations).

The relation with the value functions computed in this way, and Reinforcement Learning is discussed in D3.2 section 3.3.7.2. We here remark that while in RL the value of actions is obtained by trial and error, here it is obtained with a process that more closely is a synthesis process that build on the offline manipulation of learned models.

2.4 Step 4 – Action sequences

This step corresponds to longer-term trajectory planning. The Dreams4Cars agent (except for some demo features) did not need to be trained for longer term sequences. Instead, a biasing mechanism implementing the traffic rules (namely the conditions for a lane change) has been implemented following D2.2 section 2.2 (this has allowed, for example to inhibit lane change in curves).

2.5 Step 5 – Reinforcement Learning

We used reinforcement learning in a way different from the mainstream examples that can be found in the literature. This means that, instead of learning motor control from scratch (e.g., [6]) we focused on learning some high-level behavioural parameters that influence the agent control. In particular we have demonstrated the learning of safe speed choice in pedestrian scenarios.

The method that we use exploits the fine-tuned low-mod level behavioural stack trained in steps 1-4. In this way the RL problem dimensionality is reduced (e.g., define a safe speed given the car and pedestrian states) and the policy trained acts on a sandbox constituted by the codriver basic behaviours (which can veto any dangerous requests). Learning some behavioural parameters for an agent that is already able to operate in the real world also greatly contribute to solve the issue of transferring learned behaviours from simulations to the real world.

3 Released public agent

A public version of the Codriver agent is released within the Open Data pilot (D5.5). This version is the final Codriver agent (9.7) in the OpenDS environment.

4 Biblioraphical References

[1] M. Da Lio, D. Bortoluzzi, e G. P. Rosati Papini, «Modeling Longitudinal Vehicle Dynamics with Neural Networks», *Veh. Syst. Dyn.*, vol. in press.

[2] S. James, A. Sean, e D. L. Mauro, «Longitudinal Vehicle Dynamics: A Comparison of Physical and Data-Driven Models Under Large-Scale Real-World Driving Conditions», *Veh. Syst. Dyn.*, vol. (submitted).

[3] S. James e S. R. Anderson, «Linear System Identification of Longitudinal Vehicle Dynamics Versus Nonlinear Physical Modelling», in *2018 UKACC 12th International Conference on Control (CONTROL)*, Sheffield, 2018, pagg. 146–151.

[4] J. Porrill, P. Dean, e S. R. Anderson, «Adaptive filters and internal models: Multilevel description of cerebellar function», *Neural Netw.*, vol. 47, pagg. 134–149, 2012.

[5] A. Bemporad e M. Morari, «Robust model predictive control: A survey», in *Robustness in identification and control*, vol. 245, A. Garulli e A. Tesi, A c. di London: Springer London, 1999, pagg. 207–226.

[6] D. Ha e J. Schmidhuber, «World Models», *ArXiv180310122 Cs Stat*, mar. 2018.

[7] P. Falcone, F. Borrelli, H. E. Tseng, J. Asgari, e D. Hrovat, «Linear time-varying model predictive control and its application to active steering systems: Stability analysis and experimental validation», *Int. J. Robust Nonlinear Control*, vol. 18, n. 8, pagg. 862–875, mag. 2008.

[8] E. Kim, J. Kim, e M. Sunwoo, «Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics», *Int. J. Automot. Technol.*, vol. 15, n. 7, pagg. 1155–1164, dic. 2014.

[9] R. Donà, G. P. Rosati Papini, M. Da Lio, e L. Zaccarian, «On the Stability and Robustness of HierarchicalVehicle Lateral Control with Inverse/ForwardDynamics Quasi-Cancellation», *IEEE Trans. Veh. Technol.*, vol. Accepted.

5 Appendix 1 - CRF steering actuator compensation

Among the first challenges faced by Dreams4Cars the most urgent was related to the steering actuator of the Jeep Renegade.

Often, traditional approaches to lateral vehicle control are based on models of the vehicle dynamics that assume the steering wheel angle to be the control input. For example MPC typically assume bicycle-like lateral dynamics with steering angle input [7]. However, the dynamics of the steering actuator may not be negligible [8]; and, on the other hand, developing a precise steering actuator with negligible latency may be a costly endeavour.

In the particular case of Dreams4Cars, the actuator used for steering the Jeep Renegade was an adaptation of the device used in the standard production vehicles for lane keeping assistance (LKA). It was not a device specifically designed to be fast and accurate (no resources were allocated for the development of an ad-hoc device which would have been expensive on one side and not the central focus of Dreams4Cars on the other side).

The device has several limitations, which are functional for LKA but not desirable for autonomous driving, among which a lowpass band (about 1Hz) and a dead zone in the straight position. They have been already discussed in D3.2, section 3.1.7. We recall here the degradation of the impulse response that the actuator causes on the lateral vehicle dynamics: Figure 3, which is the same of D3.2 Figure 4, shows that the delay between steering wheel angle and yaw rate, which would be approximately 100 ms without the steering actuator, becomes ~300-350 ms when the actuator dynamics is counted. Furthermore, the lateral impulse response with the actuator is significantly spread across a large interval of times.

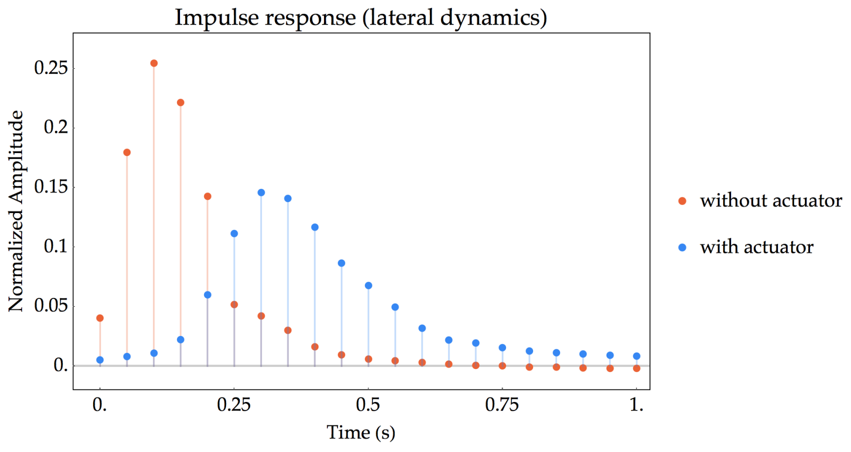

Figure 3: Impulse response of the Renegade lateral dynamics with and without including the steering actuator dynamics.

The schemes used for vehicle control have been introduced in D2.2 and better discussed in [9].

5.1 Baseline control

The baseline control scheme is based on the motor primitives generated by the codriver agent as follows. First the motor primitive representing the rate of the trajectory curvature is computed at the current time, compensating perception, computation and other delays. Second, the rate of curvature of the trajectory is transformed into the steering rate via a speed dependent gain that accounts for the understeering gradient. Third, the steering rate is integrated to produce the requested steering angle sent to the actuator. In short: *the vehicle is controlled in steering rate with compensation of the codriver loop lags*.

In paper [9] we have demonstrated that this schemes works fine as long as the gain and delay values used for compensation are not very different from the real one (the paper establishes stability margins).

In fact, this control scheme works fine with the OpenDS implementation (that uses a simple vehicle dynamic model) and worked also fine with CarMaker as long as the steering actuator was assumed to be ideal (and the vehicle dynamics was modelled realistically).

5.2 Control with inverse model of the steering actuator

The CarMaker implementation, stated with a realistic model of the vehicle, was upgraded with a realistic model of the actuator later, at the end of 2018. When the model of the actuator was introduced (Figure 3), adjusting for the increased lag in the baseline control scheme did not work satisfactorily, primarily because the delay of the loop with the actuator is spread across a large interval of times and it is hence significantly different than a simple lag.

Turning this problem into a case study, an inverse model for the steering actuator was thus developed. The control scheme with the inverse model is shown in Figure 4. Instead of the curvature rate, it stars from the trajectory curvature primitive $\kappa(t)$, which is sampled at $t_i = i\,\Delta T$ to generate the input vector of future curvature values κ_i necessary for the neural network such as, e.g., D3.2 Figure 14. In this process it is possible to account for, and compensate, the time elapsed from the perception of the data used for computing the motor primitives (time t_0 start of the motor primitive) and the current time (t) where the control action is going to be executed (i.e., compensating or the total delay of the loop of Figure 4.

The control pipeline is now as follows: First the curvature of the trajectory is transformed into the desired steering angle via the same speed dependent gain accounting for the understeering gradient. Second, the desired steering angle is converted into a steering angle to be requested to the steering actuator via the steering actuator inverse model, with the purpose of cancelling the steering actuator dynamics. This ideally restores the situation of the previous Carmaker model with realistic vehicle dynamics and ideal steering.

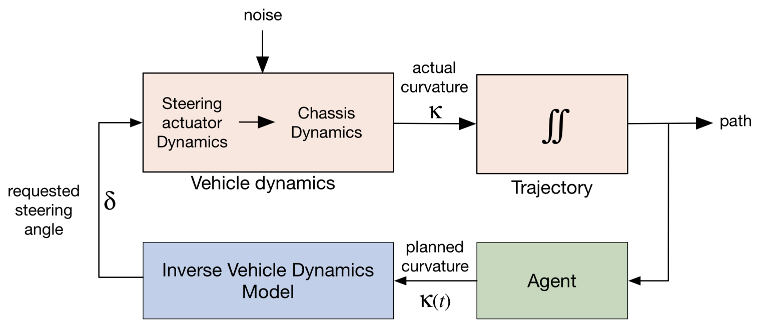

Figure 4: Receding horizon control based on inverse models (adapted from D2.3 Figure 12).

5.3 Evaluation

Figure 5 compares the baseline control with the control scheme using the inverse model of the steering actuator (note that with reference to Figure 4 this cancels only the steering actuator dynamics, not the dynamics of the complete plant including the vehicle).

The charts refer to the CarMaker Renegade complete dynamics model (vehicle with actuator). The simulation refers to the CRF Safety Centre, for which the entrance to the first curve is shown: the lane curvature profile is shown at the bottom.

The baseline used a lag compensation that was manually tuned for the case. Nonetheless the baseline (steering rate with lag compensation) control scheme reveals oscillations that are due to the lags and spread response of the actuator. These oscillations are triggered by entering the curve and are very low dampened (Figure 4, centre). They are more evident in the lateral position (top) that is an integrated effect. The vehicle wheels also trespass the inner edge of the lane.

The control with the steering actuator inverse model is definitely better. There are no oscillations and the trajectory is smoother. The car moves towards the outer art of the lane before entering the curve and then towards the inner part of the lane but without trespassing the edge. These lateral displacements are planned by the codriver agent intentionally to smooth the curve. The steering wheel angle also follows much better the curvature profile (centre and bottom). The agent behaviour is still not perfect (no inverse model of the vehicle is used for example) but no oscillation due to the steering actuator delays are present.

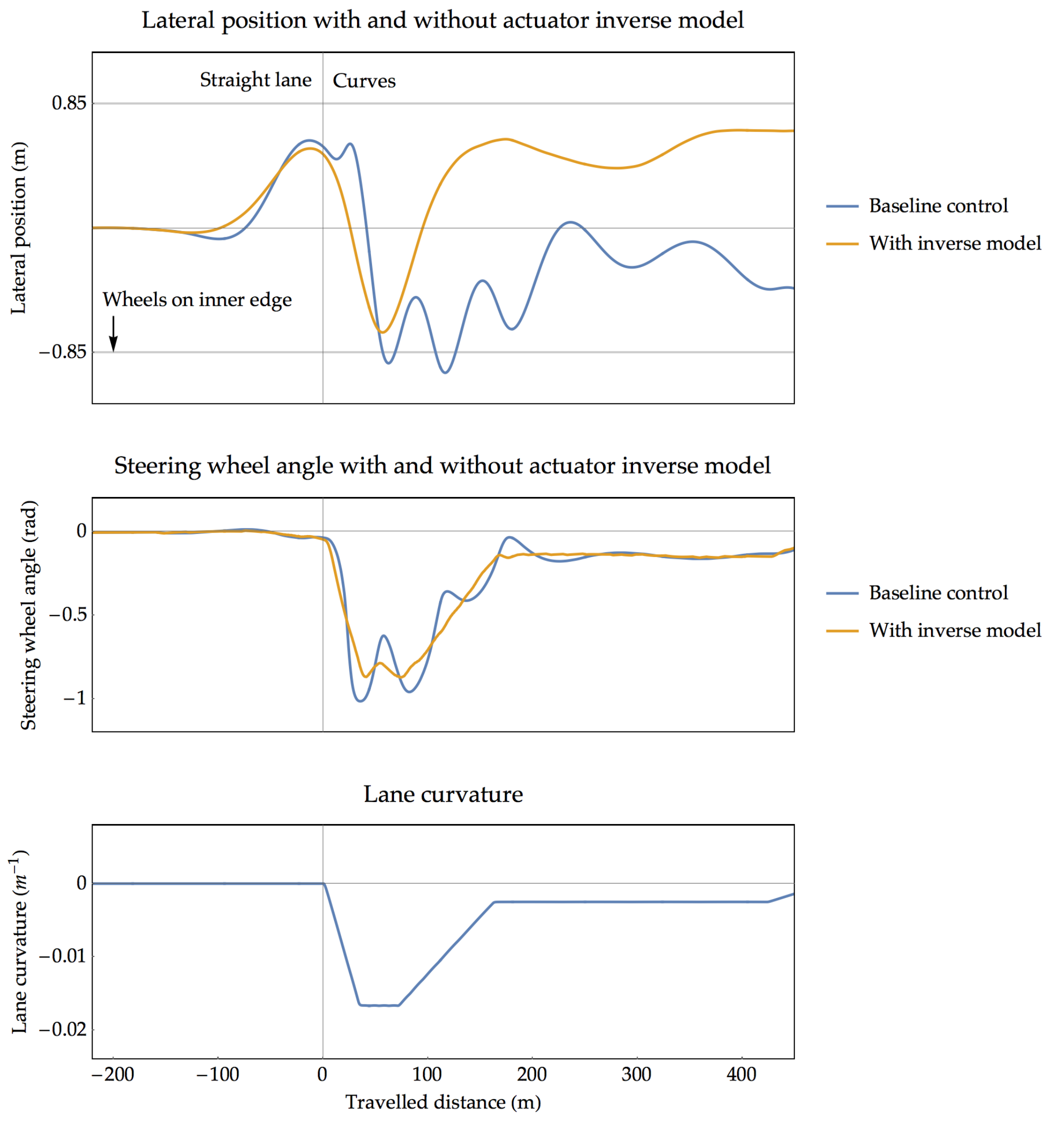

Figure 5: Comparison of baseline control to control with inverse model of the steering actuator (see text)