Dream-like simulation abilities for automated cars

DREAMS4CARS

Grant Agreement No. 731593

Deliverable:	D2.3 – Report on the Runtime system (public version)
Dissemination level:	PU - Public
Delivery date:	27 December 2018
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	Report on the Runtime system (public version)		
WP number and title	WP2 Runtime system		
Lead Editor	Mauro Da Lio, UNITN		
Contributors	David Windridge, MU		
	Andrea Saroldi, CRF		
Creation Date	24 December 2018	**Version number**	0.5
Deliverable Due Date	31 December 2018	**Actual Delivery Date**	27 December 2018
Nature of deliverable	x	R - Report	
		DEM – Demonstrator, pilot, prototype, plan designs	
		DEC – Websites, patents filing, press&media actions	
		O – Other – Software, technical diagram	
Dissemination Level/ Audience	x	PU – Public, fully open	
		CO - Confidential, restricted under conditions set out in MGA	
		CI – Classified, information as referred to in Commission Decision 2001/844/EC	

Version	Date	Modified by	Comments
0.1	24 December 2018	Mauro Da Lio	Conversion of D2.2 and D2.1 into D2.3
0.2	27 December 2018	Mauro Da Lio	Further work on conversion.
0.3	27 December 2018	Andrea Saroldi	Minor comments.
0.4	27 December 2018	David Windridge	Review/Minor text amendment.
0.5	27 December 2018	Mauro Da Lio	Final version approved.

Executive Summary

This document is the public description of the Dreams4Cars Agent.

The document describes the multiloop agent architecture and how the individual loops contribute to the cognitive abilities of the agent, focusing on the runtime part of the agent (the dreaming mechanisms will be described in D3.3).

The document complements D1.3 (System Architecture, release 2) and provides details, references and guidelines for the implementation of an agent able to produce the cognition abilities targeted by Dreams4Cars.

Table of Contents

List of Diagrams

List of Tables

1 Objectives of this deliverable

This deliverable is the public description of the implementation of the Artificial Driving Agent (the Codriver) of Dreams4Cars.

1.1 System-level architecture

The system architecture is presented in section 1 of deliverable D1.3 System Architecture (Release 2). For readers' convenience and ease of reference, Figure 1 shows the system-level architecture schematically. The Codriver is the agent that drives the vehicles at runtime (Figure 1, "Real Driving" box) and which learns new optimized behaviours in the offline simulation environment (Figure 1, "Simulation" box).

In the following, the agent implementation is described. Another public deliverable (D3.3, due in month 30) will describe the dreaming mechanisms.

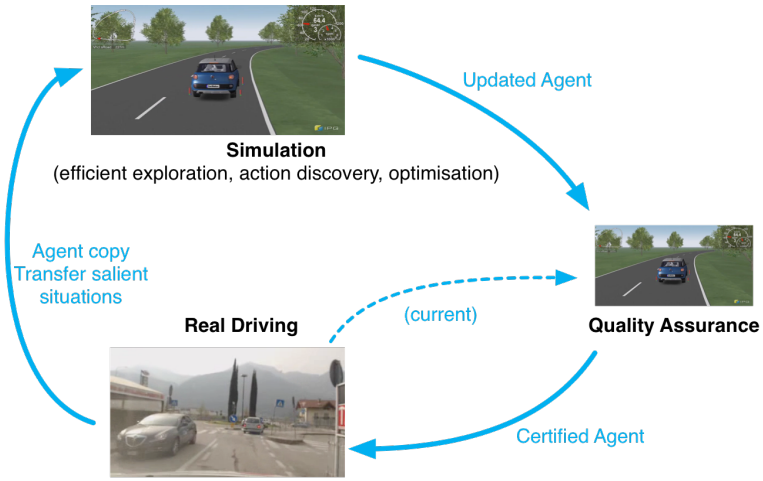

Figure 1: System architecture (from D1.3, Figure 1 – see D1.3 for explanations).

1.2 Agent architecture

The agent architecture is presented in section 2 of deliverable D1.3 System Architecture (Release 2). Figure 2 (from D1.3) is shown here. The architecture has been inspired by several studies concerning the large scale organization of the human brain, among which [1]–[3].

1.2.1 Sensorimotor loops

The agent is made of 5 loops:

1. The action priming loop (dorsal stream), which converts sensory data into activation patterns of the motor cortex, encoding the many instantaneous action possibilities in parallel (this deliverable section 2.1);
2. The frontal cortex loop, which implements a logical reasoning system that can steer the agent behaviour via the biasing of action selection (this deliverable section 2.2);
3. The action selection loop (basal ganglia), which implements robust and bias-able selection of one action (this deliverable section 2.3);
4. The cerebellar loop, which learns forward and inverse models of the vehicle dynamics (this deliverable section 2.4);
5. The motor output loop, which converts the selected actions into vehicle specific commands via inverse vehicle dynamics models and enables adaptation and portability to different vehicle types (this deliverable section 3).

1.2.2 Simulation and learning mechanisms

In the agent architecture, simulation and learning occur in different places. They are summarized here in order to frame the agent context and the purposes of the loops, but their actual description will be given in D3.3.

a) The cerebellar loop learns forward and inverse models of the vehicle dynamics. These are used to instantiate embodied simulations for low-level and tactical level controls, and for the synthesis of the dorsal stream loops that compute the salience function.
b) The frontal loop (at the action-selection step) learns biasing of actions to achieve longer term rewards; hence learning strategic-level action sequences.
c) The convergence-divergence zones (CDZ) organization of the dorsal stream learns compact representations of events that may be later used offline to instantiate a first form of episodic simulations.
d) The logical reasoning module in the frontal loop records semantic annotated events that are used to instantiate a second form of episodic simulations based on recombination of episodes via Genetic Algorithms.

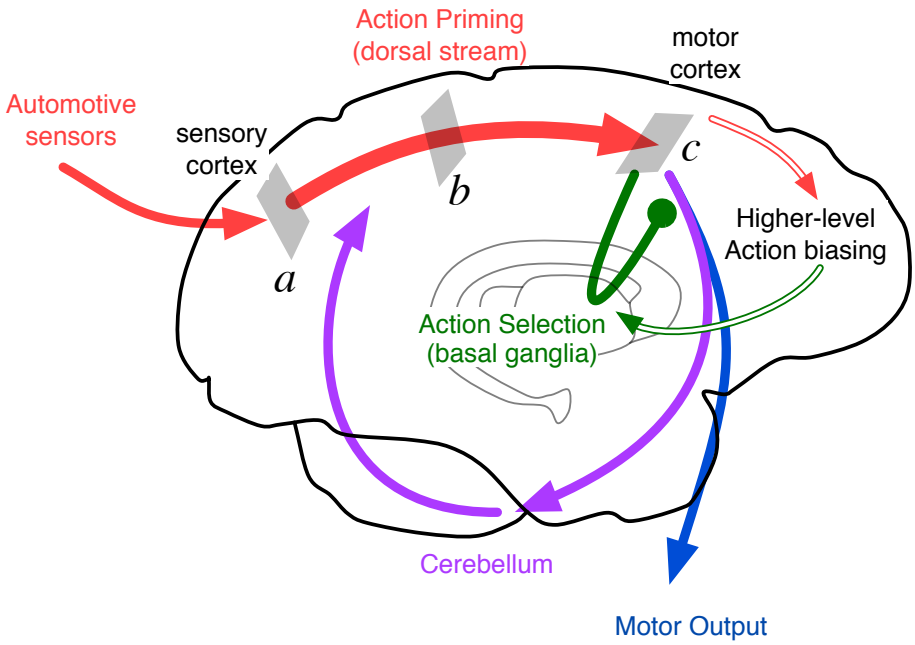

Figure 2: Agent architecture (from D1.3, Figure 2).

2 Implementation

2.1 Dorsal stream

2.1.1 Working principles

The role of the dorsal stream is to recognize the actions latent in the environment and to prepare motor plans for them. These are encoded with humps of activity in a two-dimensional tensor –corresponding to the motor cortex in the biological brains [1]– as shown in the example of Figure 3.

The intensity of each hump indicates the strength/urgency of each action request, which is called "salience". The position of each hump encodes the lateral and longitudinal control required to *initiate* the particular action (the action plan in its entirety is distributed across the dorsal stream, and in the frontal cortex Logical Reasoning Module (LRM) for long-term action sequences).

To further clarify, let us consider the example of Figure 3. At the legal lane-manoeuvring level the agent has 3 possible lane choices $\{a, b, c\}$. Hence there will be 3 discrete humps in the motor cortex, corresponding to the control choices of the 3 lanes. At the physical-manoeuvring level, the agent has however much more choices that still permit to remain (alive) in the road: for example, squeezing between lanes such as, e.g., trajectory a'. Hence a broader set of control choices, albeit of lesser value/priority, are also available which are shown with the lighter shaded area in Figure 3 (trajectory a', in which the vehicle travels partly over the lane markings, is of this type).

When obstacles are present, some of the trajectories determined so far must be excluded. Hence, just like lanes/road map into *active* regions of the motor cortex, obstacles map into *inhibited* regions of the motor cortex (see Figure 3, right).

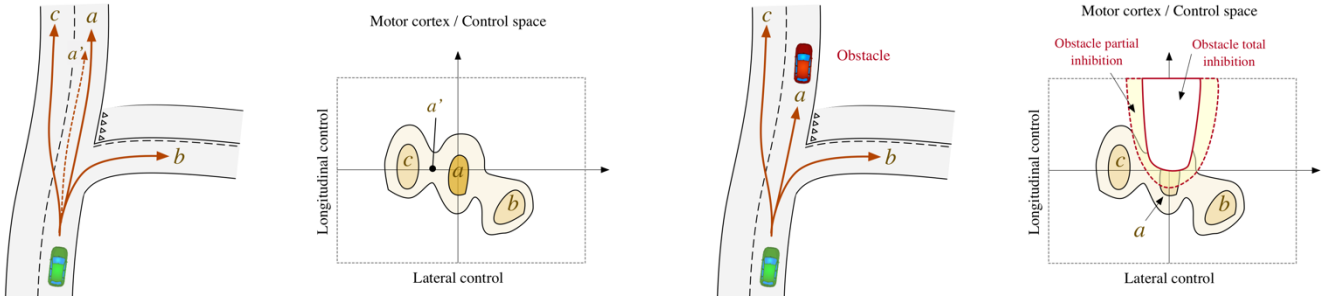

Figure 3: The dorsal stream computes activation patterns in the artificial agent motor cortex. The intensity of activation encodes the strength/urgency of corresponding actions, the location encodes longitudinal and lateral control to select that particular action. The dorsal stream is implemented with streams of two types. a) Left: parallel streams that compute active regions in the motor cortex, encoding affordable lanes/roads; b) Right: parallel streams that compute inhibited regions, encoding obstacle and mandatory traffic directives (traffic lights, stop, etc.).

2.1.2 The salience function

To implement the dorsal stream, one needs thus to implement the computation of active and inhibited regions (Figure 3). This may be done with either *analytical approaches* (that however require several simplifications to be tractable [4]) or by means of *deep neural networks* that can be trained with different and absolutely general vehicle and environment models.

In either case the dorsal stream is ultimately a function that, given the vehicle state, vehicle dynamics and environment, returns the salience function:

$$s = f(r_0, j_0) \tag{1}$$

which tells the value of a particular instantaneous choice of lateral/longitudinal controls (r_0, j_0).

The salience function may be discretized with a two-dimensional tensor (a neural layer), where columns and rows represent discrete values of the lateral and longitudinal controls (r_0, j_0).

2.1.3 Formal definition of salience and methods for computation

We give a formal definition of salience, which allows *unsupervised learning of the salience function* for any vehicle and environment.

Let $\gamma(t)$ be a trajectory originating from the current vehicle configuration. Let $r(t)$ and $j(t)$ be respectively the lateral and the longitudinal control[1] that produce $\gamma(t)$. Let $V(\gamma)$ be a functional operator that returns a scalar quantity representing the "value" of trajectory $\gamma(t)$.

There are many possible definitions for V that may work, but in general a proper choice must account for the magnitude and complexity of the control sequence $\{r(t), j(t)\}$, for physical limits (e.g., tire adherence), for human driving criteria (e.g., the two-thirds law [5]–[7] and comfortable accelerations, smooth control, etc.) and for goal related trajectory constraints (e.g., to remain inside a target lane).

- For analytical formulations $V(\gamma)$ is inversely related to the cost function of the Optimal Control Problem used to determine $\gamma(t)$: namely the minimum energy criterion for $\{r(t), j(t)\}$ subject to inequality constraints that express the physical limits and human driving criteria above. In [4] an example of simplified linearized minimum square jerk Optimal Control Problem is given, where the time-to-lane crossing is used as a surrogate of $V(\gamma)$.
- For neural network implementations (our main focus) $V(\gamma)$ is inversely related to a purposely built loss function equivalent to the optimal control cost above. The training of the neural network is carried out offline, via examples produced with the solution of (direct or indirect) Optimal Control Problems that use the learned vehicle dynamics. This way neural networks that maximize V can be trained on top of learned forward models. Embodied simulations that minimize the loss function may also be used as an alternative approach. This is equivalent to the solution of a direct Optimal Control problem (which in turn is equivalent to a Reinforcement Learning problem [8]).

The formal definition of salience can thus be given.

Given one legal lane goal g (e.g., $g = a, b, c$, in Figure 3) the salience related to achieving goal g may be defined as follows:

$$s_g(r_0, j_0) = Sup(V(\gamma) | r(0) = r_0, j(0) = j_0, \gamma \in g) \quad (2)$$

This definition means that the salience s of goal g is the supremum[2] of the values (V) of all trajectories beginning with control $\{r(0) = r_0, j(0) = j_0\}$ and satisfying g ($\gamma \in g$).

Implicit in this definition is the fact that $\gamma(t)$ is subject to the dynamics of the system ($\gamma(t)$ is produced by $\{r(t), j(t)\}$) and subject to the initial vehicle configuration ($\gamma(t)$ originates for the current vehicle configuration).

The same definition may be re-used for the computation of low-priority actions (physically feasible but undesirable actions such as when g means to remain in the road but with violation of lanes, as discussed in section 2.1.1). It is sufficient to replace the notion of lanes with the notion of drivable road surface.

[1] In Dreams4Cars $r(t)$ and $j(t)$ are kinematic abstractions of the lateral and longitudinal control; namely the curvature rate (r) and the longitudinal jerk (j).

[2] Other norms might be used instead for Sup. One should also note that the choice of any norm implicitly means that part of the decision is anticipated in the dorsal stream as, in this way, the motor cortex encodes the value of the optimal trajectory originating at r_0, j_0. This pre-processing is nonetheless necessary to reduce the dimensionality of the motor cortex.

2.1.4 Modular architecture

With the above definition, salience may be computed for different goals independently, and then composed as follows.

2.1.4.1 Excitatory circuits

A first level of modularity happens because the values of lanes and roads may be computed independently. So, given a learned vehicle dynamics model –so that $\gamma(t)$ may be computed as a function of $\{r(t), j(t)\}$– and given an initial state and an environment, the salience of goal g, $s_g(r_0, j_0)$, is computed by a neural network trained as in (2). The aggregate salience when several possible goals (individual lanes a, b, c or the entire $road$) exist can thus be defined as:

$$s(r_0, j_0) = Max(s_g(r_0, j_0), g = a, b, c..road) \quad (3)$$

In a deep learning implementation, hence, a network that can compute the salience for a generic road strip (be it a lane or an entire road) can thus be used for the computation of the individual s_g, and the aggregated salience s, in practice creating the humps of activities of the motor cortex in *parallel*[3].

2.1.4.2 Inhibitory circuits

A second level of modularity happens because the inhibitions of obstacles may also be computed independently, and independently from the lane/road activations too. Furthermore, there is a third level of modularity, because inhibitions caused by one obstacle can be computed by repeatedly evaluating inhibitions of discrete future time predicted positions of obstacles[4], as shown exemplified in Figure 4 (see also [9]).

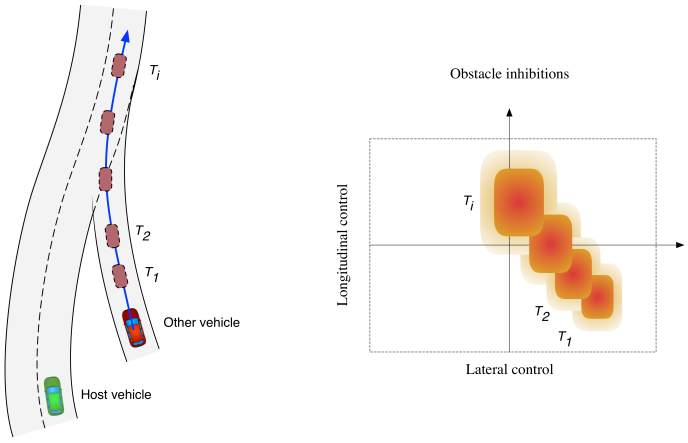

Figure 4: Inhibitions caused by obstacles are evaluated as inhibitions for space-time future positions predicted for obstacle motion [9].

[3] Note that the Max operator in (3) is just another form of decision. However, unlike equation (2) this time the individual s_g may be passed to the action selection algorithm as they are (without composition into one single s) because they are a finite number (whereas for (2) curves are ideally infinite in number and reduction of dimensionality is definitely necessary). This is indeed the way it is implemented in Dreams4Cars.

[4] The predicted trajectory is necessary and may be obtained via simple mirroring processes or via episodic simulations (see D1.3, section 3.2.1 or this deliverable section 2.1.5).

In order to compute inhibited regions, the value function V must be adapted to express collision avoidance merits of control choices. One possible solution is to compute the value V on the complement space-time sets (space time locations not occupied by obstacle). Another, somehow simplified possibility, is computing optimal collision trajectories and use the initial controls of them to define a region to be inhibited in the motor cortex. An example implementation of this method is given in [9]. Inhibited regions are used as multiplicative factors to reduce to zero the salience as computed by (3). In terms of embodied simulations that means dreaming actions that cause collision for the purpose of not implementing those actions.

Prediction of obstacle trajectories may be created via episodic simulations with the Dorsal Stream CDZ architecture as demonstrated in D3.1 section 3 and discussed here in next section.

2.1.5 Convergent-divergent network structure for episodic simulation

Besides the "wake" time use described so far, the dorsal stream has also a role in conceptualizing episodes. For this, the dorsal stream must have a convergence-divergence architecture (Figure 5, following [3]) as explained with more details D1.3, section 3.2.1.

Encoding of episodes does not need to be carried out inline. Rather, it may happen via post processing of raw log data offline. If decoding of episodes is carried out inline, this may be used for "wake" time imagery of events, such as, for example, predicting the behaviour of other road users (this is not the only way to predict other road users' intentions; mirroring process are in general more effective [10]–[12]).

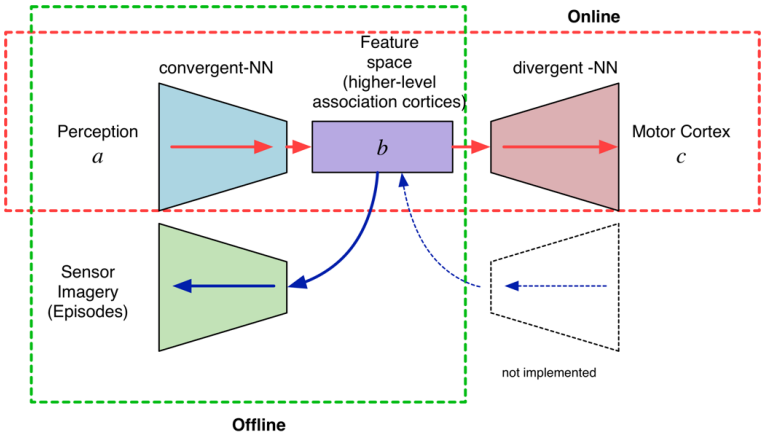

Figure 5: Convergence-divergence zones implementation of the dorsal stream for episodic simulations (see D1.3, section 3.2.1).

2.2 Biasing loop (frontal cortex loop)

While section 2.1 deals with the generation of safe mid/low-level behaviours (lanes/roads trajectories and obstacle avoidance), to implement more complex symbolic rule-based behaviours such as legal action sequence-planning (e.g. overtaking), further layers are constructed on top of the dorsal stream that *steer* the agent low-level behaviours to produce legal action sequences for longer-term goals.

This high-level loop is specified as a hierarchical Perception-Action (PA) subsumption architecture. As such, it provides a unified framework for:

a) Semantic annotated event logging.
b) Generation of legal priors for action selection via the basal ganglia (BG) loop.
c) High-level motor babbling/top-down dream instantiation (for the offline system).

Dreams4Cars has thus implemented a unified architecture (the Logical Reasoning Module) with a common symbolic/sub-symbolic PA interface that operates across the three distinct symbolic/sub-symbolic information-flow modalities (bottom-up semantic annotation, top-down legal intention biasing, top-down dream instantiation).

This deliverable is focused on a) and b) functionalities; the dream-instantiation process itself is treated in deliverables D3.3 and only discussed here in so far as it influences the design of the common run-time system and interfaces.

2.2.1 Biasing principle

The way in which the Logical Reasoning Module (LRM) affects the low-level motor control is by biasing action selection (see also D1.3, section 2.2) via increasing the weight of the humps of activities that correspond to the actions of a desirable sequence (biological inspiration in [2]). In addition, the weights of humps of activities that correspond of undesired actions are conversely reduced, hence opposing illegal/undesired (but physically possible) actions.

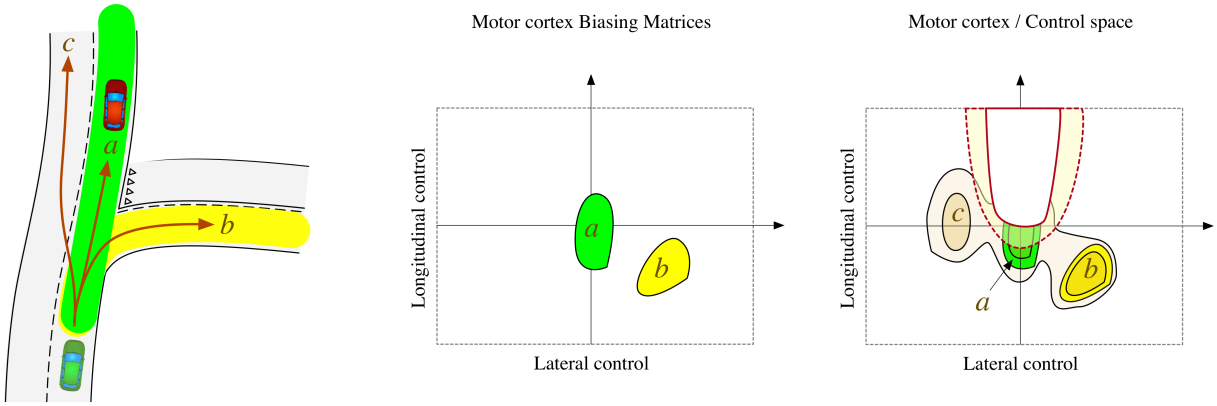

Figure 6: Biasing principle. Suppose the long-term goal is to remain in the lane (*a*, green shading). Suppose also that taking exit *b* is not desired (which means that a secondary priority is to stay in the main road and, if necessary *c* has to be preferred to *b*). Centre: the salience function of goals *a* and *b* are computed (*c* is neutral and not shown), normalized and used via variable weightings to artificially increase the strength of hump *a* and decrease the strength of hump *b* in the motor cortex (right).

As an example (see Figure 6), suppose that long term behaviours such as remain in the lane (*a*), overtake (*c*) and take the exit (*b*) have been formulated by the LRM and that, after the high-level action selection (see section 2.3) *a* is marked as preferred action and *b* as undesired action. The salience function of goals *a* and *b* and *c* are computed (*c* is neutral and not shown) and used to define regions of the original motor cortex to be strengthened/weakened (Figure 6, right, see section 2.3 for the algorithm). The weights for actions *a* and *b* may be changed ($w_a > 1, 0 < w_b < 1$), and may be learned for optimizing long-term strategical behaviours.

Note, at this point, that the LRM can only make recommendations (as per the subsumptive `principle of lower-level veto'), such that the final choice is in charge of the lowest level motor control (dorsal stream). This way, for example, should the obstacle be so close to completely cancel hump *a*, the bias a would be ineffective and the agent would change lane (to avoid the obstacle) whatever the strength of the recommendation from the LRM.

The final authority is thus always in the responsibility of the dorsal stream physical loops and, when they are proved to be safe, there is no means by which errors from the LRM can induce collisions because the LRM can only use the safe low-level building blocks.

Next section 2.2.2 will describe the Logical Reasoning Module and in particular how it instantiates action priors such as those shown in Figure 6. A further section will then deal with action selection that occurs at two levels: first at the level of action priors to define weights for different long-term actions, and then at the bottommost level of fine motor control.

2.2.2 The Logical Reasoning Module

The Logical Reasoning Module (LRM) is concerned with the hypothetical reasoning component of the system architecture in relation to the legal road rules, i.e. the Highway code (HWC). It is implemented so that the agent conforms to a perception-action subsumption architecture, which requires that a principle of *perception-action bijectivity* is enforced within the hierarchy (e.g., see [13]).

The subsumptive Perception Action hierarchy embodied within the LRM consequently implements the symbolic (i.e. high-level representational) component of the Codriver architecture, being responsible for high-level scene interpretation/annotation and for introducing legal biasing in intention (note that the highway code itself does not necessarily identify unique actions within a given road context, but rather gives rise to a degenerate equi-legal set of action possibilities).

The LRM acts via a mixture of theorem proving-via-resolution and functional extrapolation in order to apply the HWC in unfamiliar scenarios, with the former constituting the highest level of PA subsumption. The road configuration is thus represented within the LRM as instantiated logical variables, irrespective of the LRM's operational modality (as indicated, the LRM subsumption framework is constrained to have the capability to act reversibly, that is to say, in a *generative manner* via reverse PA logical-variable instantiation, such that hallucinated high-level legal road configurations are spontaneously generated alongside the corresponding legal intentionality in order to instantiate the offline dreaming process. The latter is an instance of top down exploratory PA motor babbling, in which theorem proving-via-resolution is applied to random instantiations of logical variables in order to establish self-consistent Herbrand (i.e. logically-self consistent) interpretations, i.e. scenarios consistent with the legal road protocols).

Thus, while the offline dreaming process is one of top-down symbolic grounding through the full PA subsumption architecture, it is conversely the case that the run-time high-level scene-description and annotation (function a) may be seen as a process of bottom-up symbolic abstraction. The two processes are hence *the precise inverse of each other* in the LRM's design.

The design specification for the PA subsumption framework embodied by the LRM is therefore as follows.

2.2.2.1 PA Subsumption Design Principle Adopted by the LRM

The criteria for the number of levels in the hierarchy is defined by the twin notions of *subsumption* and *Percept-Action bijection.*

Application of the PA bijectivity criterion means that we should, as far as possible, represent only those percepts that distinguish intentional actions on a given layer. This means that each intention must bring about a perceivable change in such a way that the total set of percepts is minimised with respect to the available actions (affordances), *consistent with the highway code representation of a priori meaningful perceptual objects.* In practical terms, application of this principle means that, for example, it is not possible to have two consecutive legal gaps within a lane, since a 'legal gap' in order to exist as a high level percept must be *distinguishable by a correspondingly legally-definable intention* (a legal gap is defined as a potential legal place of relative occupation for the Ego car within a given lane, and as such is not sub-divisible at the highest level of legal intentionality).

The notion of *Subsumption* in the LRM is thus related to the legal sub-structuring of higher-level intentionality; in particular, where perceptual targets are fine-grained by sub-intentions, for which the same PA bijectivity condition also applies.

This principle also extends to levels below that indicated by the HWC; however, the lowest intentional level indicated by the HWC is that of the *linearized road metric*; this therefore dictates the *interface point* of the LRM with the rest of the system (or, equally, this is the symbolic/sub-symbol cut off in the run-time system) as shown in Figure 7.

From the hierarchical PA perspective, there are thus *two* distinct symbolic reasoning layers implicit in the Highway Code (because the HWC explicitly excludes both navigational considerations and motor processes

from its remit, which would respectively extend the higher and lower levels of the hierarchy if present). The two levels are: the *discrete symbolic* level and the *logico-linear metric* level, as shown in Figure 7.

Consequently, legal-intention related configurations can only be defined in the above terms; they collectively represent the high-level semantic annotation (or equivalently, the high-level scene understanding) brought about by hierarchical PA considerations.

The LRM is therefore architected on two distinct layers (see Figure 7), with a perception/action interface specified between each level at the appropriate level of symbolic abstraction.

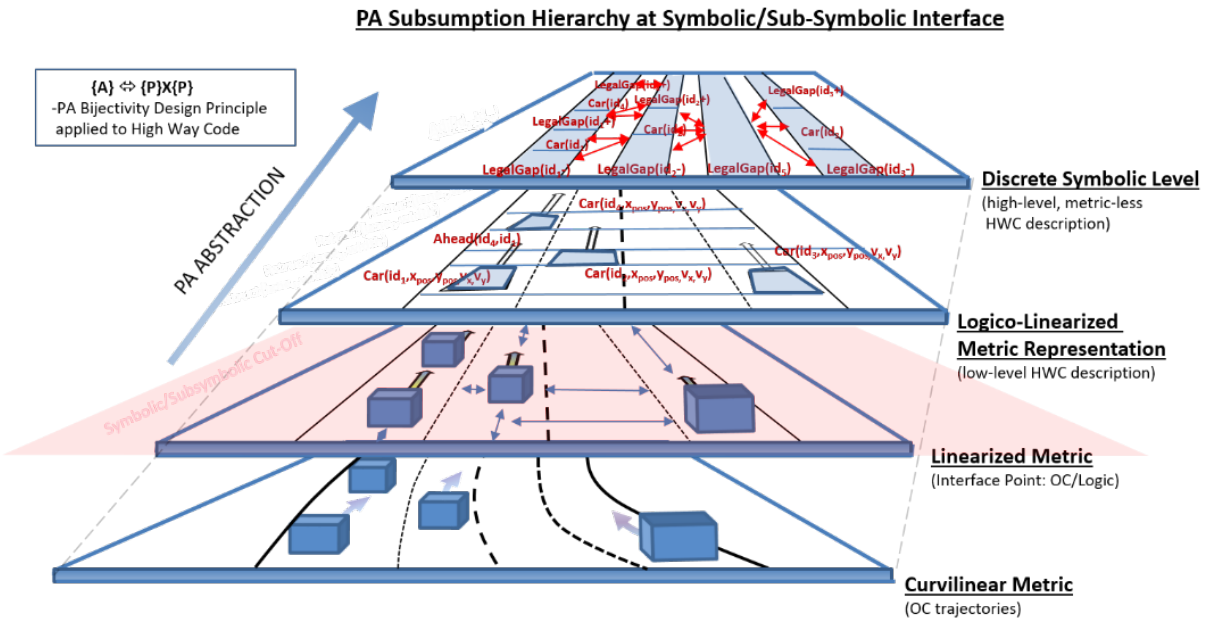

Figure 7: Run-time system PA hierarchy (OC refers to the optimal control trajectories existing at the physical layer).

Depending on which side of the perception-action divide is being favoured, there are in effect *two distinct forms of subsumption* implicit in the highway code in relation to intentions. The first is classical Brookesian task subsumption, within which fulfilment of a high-level intention (e.g. 'overtaking') may depend contextually on the fulfilment of a lower-level goal (e.g. 'pulling out into overtaking lane'). The second form of subsumption is the aforementioned *perceptual* subsumption, in which different perception-action goals are apparent at different subsumptive levels of representation. For clarity and consistency, it is the latter form which dictates the design of LRM level and level-interfacing architecture (in a pure PA hierarchy, i.e. one that is grown bottom-up by motor babbling [13], [14], the two distinct forms of subsumption are exactly equivalent).

However, both aspects of subsumption are implicit in the majority of driving intentions. A task fulfilment criterion (i.e. perceptual target) may thus be defined upon any perceptual level. The notion of subsumption thus necessarily implies that a perceptual transition at one layer is upwardly transmitted (so that e.g. traversing a lane-boundary at the metric level necessarily implies a transition of discrete lane markers on the logico-symbolic level). Conversely, the LRM embodies a 'principle of lower-level veto', such that information concerning a failure of attainment of a perceptual goal on a lower level is upwardly transmitted, and the higher-level goal imperative overruled by the lower level imperative failure (however, the higher-level intention itself remains active).

2.2.2.2 Interlayer Interface Structure of the LRM

The HWC refers to both discrete symbolic entities (cars, lanes, signs, gaps, etc.), as well as linearized-metric entities —i.e. metric entities expressed in terms of distance-to/time-to and distance-from/time-from other entities described in relation to the Ego Car.

At the high-level node (Figure 7), lane-wise road configurations are characterised in the LRM via a logical-list format: ordered in-lane lists of cars and gaps, with (the equivalent of) predicatized assertions as to which cars/gaps are legally adjacent to which others.

At the immediately lower level of the LRM, the (symbolic/sub-symbolic) node is characterised by annotated metric bounding boxes relating to legal transitions produced by a two-stage process, corresponding to the two stages of subsumption at the apex of the PA hierarchy listed above (the annotation aspect of the metric bounding boxes thus correlates to their high-level representation, illustrating the progressively grounded nature of symbols generated in a PA hierarchy).

Contextual metric information (distances to and velocities of other cars), received from the agent are hence converted into a non-metrical list of cars and gaps by means of linear extrapolation according the HWC protocols (i.e. assuming constant speeds and legally-specified reaction times). This list is passed as the second level in-put to the LRM as a declaratively-enacted script format, from which a set of high-level legal intentions with uniform priors are generated (these are uniform since road protocols do not distinguish between legal intentional possibilities *a priori*).

2.2.3 Bottom-up Communication from the Pre-LRM Layer During Runtime (Semantic Annotation & Logico-Legal Scene Representation)

The bottom-up semantic annotation function of the runtime system thus involves communication from the Codriver through the various levels of the LRM in the form of abstractions of the perceptual data consistent with the outlined notion of perceptual subsumption:

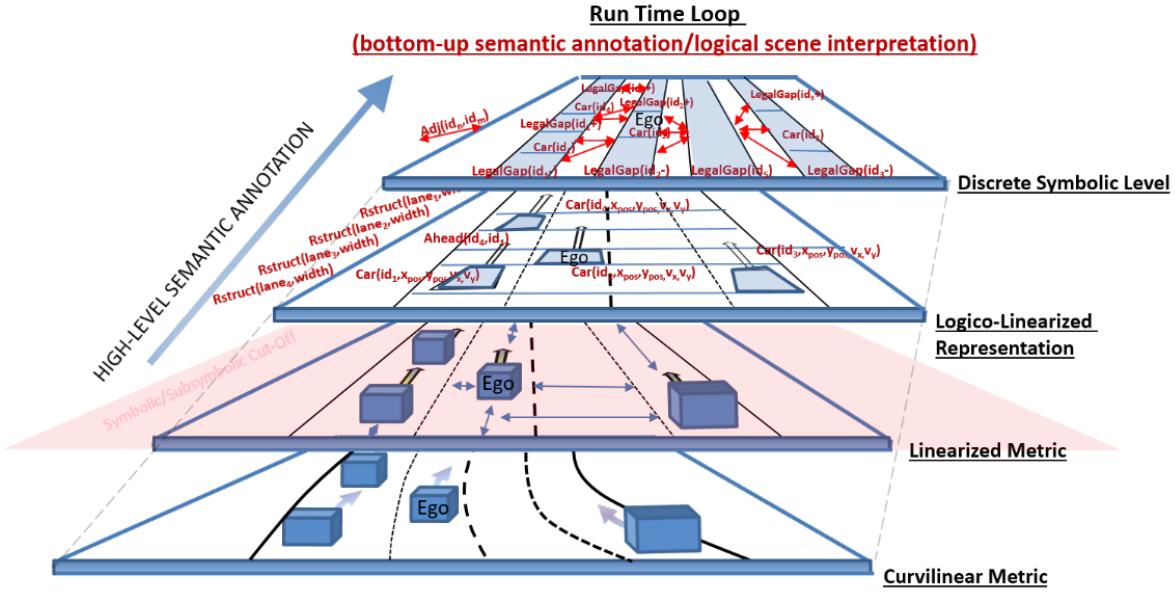

Figure 8: Run-time loop; bottom-up semantic annotation

At the symbolic/sub-symbolic interface layer (Linearized Metric Layer in Figure 8), geometric details such as the exact shape of the lanes are hence discarded, while the topology and linear distance (constituting a higher-level legal-symbolic parametrisation) is retained. The speed and distance of individual objects in relation to the Ego car, and road configuration information in the form of lane numbering, width, lane marker types (e.g. whether lane change is allowed) etc. are passed to the LRM.

The net result of the bottom-up communication of road configuration, after processing by the logical-reasoning system, is thus a high-level symbolic representation of both the legal status of, and the legal possibilities with respect to, the current road configuration. This hence constitutes a semantic annotation of the road situation described with respect to a (legal) intentional frame, or equivalently the high-level scene interpretation.

2.2.4 Top-down communication from the LRM (Legal intention Grounding)

The logic-symbolic reasoning process, as well as providing the high-level interpretation of the road circumstances indicated above, also serves to provide a full set of Herbrand (i.e. logically-self consistent) interpretations of the future legal action possibilities (for example, whether it is legal to change lane in the current context).

These Herbrand sets are then grounded —i.e. propagated downwards (as instantiated hierarchical variables in the run-time mode)— through the perception-action subsumption hierarchy so that, at the point of interface with the Codriver, they manifest as a set of binary saliency indicators attached to legally-designated areas in the linearized metric space (Figure 9).

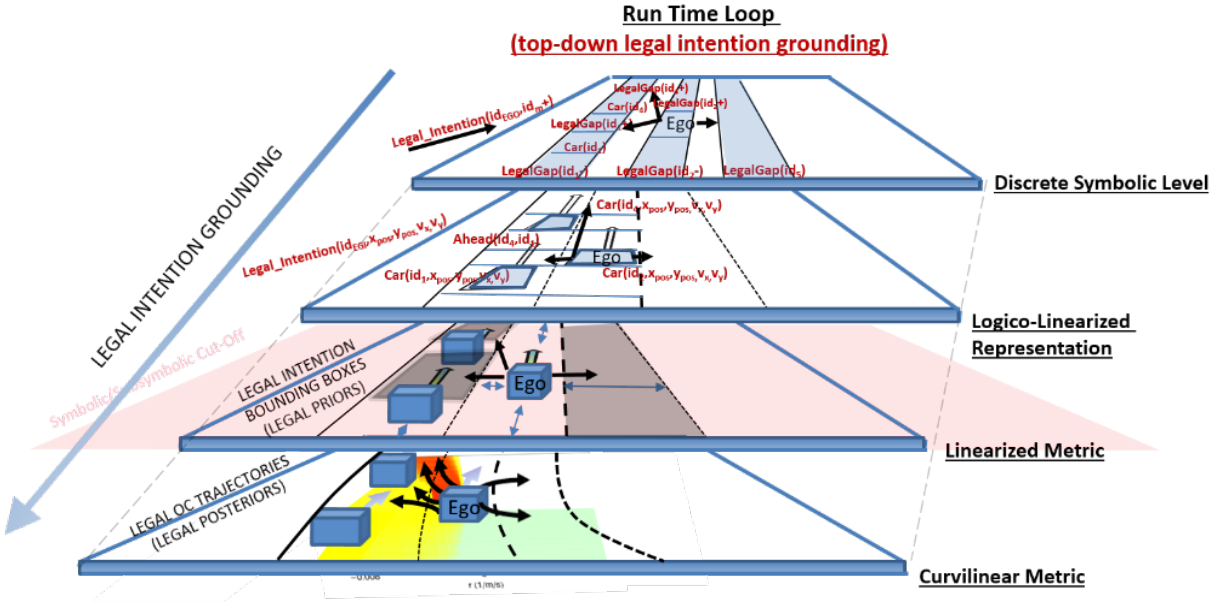

Figure 9: Legal-Perceptual Intentional Grounding from the LRM

The communication from the LRM thus take the form of bounding boxes augmented by their discrete legal saliency indicators. These bounding boxes are then used to compute the biasing matrix, described in section 2.2.1.

The bounding boxes, together with their hierarchical intentional annotations, legal saliencies and adjacency relation thus constitute the high-level semantic annotation; i.e. the annotation represents the high-level interpretation of the scene as represented by the end locations of legally-definable actions. The scene annotation thus simultaneously satisfies the requirements of perception-action bijectivity and legal self-consistency. It is this bijectivity that allows the bounding box annotation to be directly interpretable at the motor cortex, such that it can be utilized directly in action selection.

Note that the top-down LRM logical annotation process is exhaustive, such that a complete Herbrand-interpretation of the scene is generated as the annotation output (this is a natural consequence of the logic program being applied recursively until an inferential fixed point is arrived at).

This means that, in the event of incomplete input data, the system generates a full range of self-consistent 'completion' sets, which are effectively the equivalent of equally-weighted 'possible worlds' (in the modal logic sense) consistent with the input, composed of alternative groundings of predicate variables with the available constants.

2.2.5 Dreaming Initiation via Top-down Communication of Legal-Perceptual Priors (LRM Percept-Motor Babbling).

Where no input is given to the LRM, there are in effect no grounded logical configuration variables asserted at the symbolic/sub-symbolic interface. In principle, this allows the LRM to automatically bootstrap the dreaming process (i.e. initiate high-level percept-action babbling) without any modification of the system's top-down logical structure.

That is, *exactly the same mechanism for legal biasing* using the above protocols can be utilised for dreaming, since the Herbrand fixed-points in the absence of any assertion as to road configuration (i.e. no assertions relating to either road topology or to vehicle traffic using that topology) are simply a uniform set of possible worlds consistent with the legal constraints on the road configurations in general (being, in their essence, simply a set of clauses applied to asserted configuration predicates, the LRM's logical axioms necessarily have only a nominal distinction between *intentional* rules and *environmental-consistency* rules).

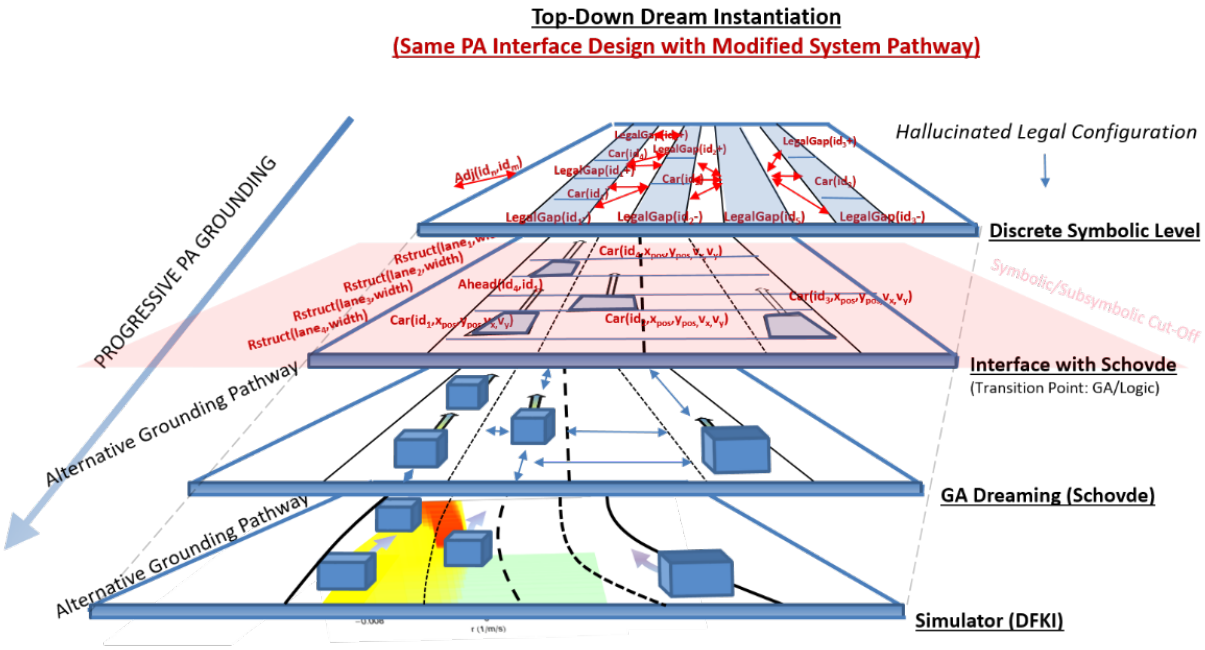

Figure 10: Dream instantiation via the top-down functionality of the LRM

2.3 Action selection loops

Action selection in the Codriver system has been organised in a hierarchical fashion (Figure 11). There are two distinct action selection modules, acting at the symbolic (the LRM) and sub-symbolic (physical) levels of description. The levels are differentiated firstly by their differing inputs, and secondly by the differing timescales over which their decisions are made. Each action selection loop is comprised of a multi-hypothesis sequential probability ratio test algorithm (MSPRT, see section 2.3.1).

The higher-level action selection loop takes the outputs of the logical reasoning module (LRM) as its inputs. The LRM, in particular, outputs the parameters of "bounding boxes" which detail the locations of legal gaps and obstacles in a road-centred coordinate system. The high-level action selection module first assigns, at each time step, scalar weights representing the "desirability" of each of the LRM's bounding boxes. These weights are learned, this way enabling the agent to learn long-term strategies (see also D1.3 section 3.2.3).

Once the high-level action-selection loop has concluded its decision-making process, the conclusion can be passed down to the lower level (Figure 11). The high-level algorithm will have identified one bounding box to be the preferred target. It may have identified others to be legal but less desirable, and the remaining bounding boxes as being dismissed as possible targets (as explained in Figure 6).

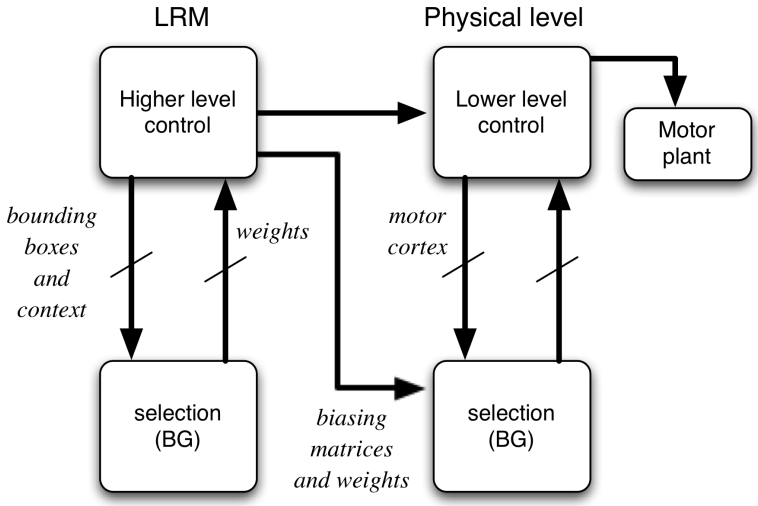

Figure 11: Action selection architecture. Left: Logical Reasoning Module identifies bounding boxes around legal and illegal locations. High-level Selection loop chooses which of the legal bounding boxes are best and assigns priority weights. The output of the High-level selection loop biases low level control via biasing matrices and weights. The low-level selection loop combines weights with the motor cortex as in section 2.2.1 and completes the final selection.

2.3.1 The MSPRT Algorithm

Neurally inspired action selection in the agent takes the form of a computational model of the basal ganglia. We have built on our previous work [15] demonstrating that the basal ganglia could be performing a form of action selection known as multi-hypothesis sequential probability ratio test (MSPRT). The algorithm sums evidence for each action over time, and finds the log likelihood that each channel is drawn from a distribution with a higher mean than the other channels. Once the log likelihood crosses a threshold, the action becomes selected. The threshold has to be tuned such that some predetermined error rate is permitted. Subject to a few assumptions, the algorithm can be shown to be optimal in decision time, given a particular error rate.

MSPRT has many advantages, with the main benefit being robustness of the decision-making to noisy input data. Other advantages have been previously outlined in deliverable D1.2 and will therefore not be restated here.

The standard MSPRT algorithm has been extended to function well in nonstationary environments, i.e. environments in which the statistical parameters of the input data are not constant [16]. The non-stationary MSPRT deals with the changing input statistics by forgetting inputs that occurred some time ago. The evidence for a particular decision is therefore accrued only over *a time window of finite size*. The size of this time window is variable, depending on which level of the subsumption architecture the MSPRT is working: higher level decisions such as which manoeuvre to perform next will need to accumulate evidence over ~1-10 s time scale, whereas fine motor control and path planning decisions will need to accumulate evidence over much shorter ~0.1-1 s time scales.

The MSPRT implementation is described in Table 1. The fundamental step is 4: the computation of the log-likelihood as follows:

$$L = -g \cdot mc + Log(Total(Exp(g \cdot mc))) \quad (4)$$

where mc is the accrued motor cortex (step 3), g is a gain tensor (that is used to model biasing) and $g \cdot mc$ is the elementwise multiplication of the two.

The term $Log(Total(Exp(g \cdot mc)))$ models the competition between channels (Exp is computed element wise).

If the channel with the minimum L (the log likelihood) is below the specified threshold, the channel location in the motor cortex is selected and returned. Else, if the threshold is not reached within the deadline, the best candidate is returned.

The implementation of the algorithm is quite simple. It is however worth noting that it can be parallelized. Given tensors $g \cdot mc$, L is actually the logarithm of the SoftMax function of $g \cdot mc$.

Table 1: The MSPRT algorithm.

MSPRT Algorithm, Requires: $gain, threshold, current\_motocortex, deadline, forgettime$
1: Start with empty $accumulated\_motorcortex$
2: $accumulated\_motorcortex \leftarrow$ **Append**$(current\_motorcortex)$
3: $acc\_mc \leftarrow$ Total$(accumulated_motorcortex)$
4: $log\_likelihood \leftarrow -gain\ acc\_mc +$ **Log**(**Total**(**Exp**$(gain\ acc\_mc)$))
5: $m \leftarrow$ **Min**$(log\_likelihood)$
6: $i, j \leftarrow argmin(log\_likelihood)$ // position of the minimum
7: **If** $m < threshold$ **Then**
forget frames before $forgettime$ in the $accumulated\_motorcortex$
Return i, j *and* m // selection before deadline
8: **If** $deadline$ is elapsed **Then**
reset the $accumulated\_motorcortex$
Return i, j and m // selection after deadline

2.4 Cerebellar stream

The cerebellar loop is used both inline (“wake” state) and offline (“dream state”). The inline use is for state estimation/control and failure detection (see section 3 of this document). The offline use is for embodied simulations (e.g., D1.3 section 6).

In both cases, everything relies on the learning of forward/inverse models, which has been presented in D3.1, where section 6.1 deals with the learning of forward models with neural networks and section 6.2 with state space and analytical-parametric models. A comparison between these approaches is going to be submitted as a journal publication [17].

The software implementation for inline use simply requires loading and *running* the neural networks models (this deliverable section 3), which in turn uses a deep learning foundation (in our case MXNet).

The software implementation for the offline use (as it was for the dorsal stream) requires, in addition, the ability to *train* neural networks, which is achieved by means of a number of command scripts (either Python/Keras or Wolfram Mathematica as front ends and MXNet as backend).

3 Integration of the agent in real and virtual vehicles

3.1 Vehicle control paradigm

The control scheme is important for concretely benefitting of the Codriver abilities. In particular, it is the key for interoperability (the Codriver being easily adapted to different vehicles) and for adaptation to changes in vehicle dynamics (the Codriver adapting to either failures or environmental changes).

3.1.1 Traditional approaches

Traditional vehicle control is usually based on two possible approaches: 1) trajectories are planned in the cartesian space and then tracked, e.g., using Model Predictive Control; 2) trajectories are planned directly in the actuators space.

The former approach, separating trajectory planning from trajectory tracking leaves to the trajectory tracking level the arbitrary decision of how aggressively corrections of deviations have to be. In this scheme, if a deviation from the planned trajectory occasionally occurs, the tracker will try (more or less aggressively) to return to the original trajectory, even if the deviation is irrelevant for the goal at hand. The “minimum intervention principle” [18] states that only deviations that affects the final goal should be corrected. This is clearly possible only if the trajectory planning and the trajectory execution levels are not disjoint. With planning and tracking as separate modules, aggressive corrections that are in facts unnecessary, may arise and may have the drawback of using resources, in particular tire friction, that are unnecessarily close to the manoeuvrability limits (as an example, imagine a human driver that realizes being slight off-centre in one lane; on a slippery road, an aggressive correction to return in the centre might cause slippage; a human instead would evaluate the minimum amount of correction to manoeuvre for the next curves).

The latter approach, conversely, aims to provide the exact command to the actuators (steering wheel angle, engine and brake actuation) without stepping in the intermediate level of a cartesian trajectory (i.e., trajectories are planned straight in the actuator space, see one example in our previous work [19]). The main drawback is that an accurate model of the vehicle is necessary. This model is often a parametric analytical model (one example of optimal feedback control based on learned dynamics is given in [20]). The model may be inaccurate either because the parameters are not perfectly known and/or vary in operation and because the model includes many simplifications, the effect of which may be not perfectly known in all the situations. Furthermore, even if a detailed (necessarily complex) model were available, planning optimal control with such nonlinear complex models may be very demanding from the computational point of view, and may require iterative algorithms (and the latter are disliked by the automotive industry unless a formal proof of convergence in the available times is given, which is difficult to provide).

3.1.2 Inverse model approach

The control approach used in Dreams4Cars is based on two levels (execution of kinematic trajectories) but, unlike 1), the two levels are tightly coupled. With reference to Figure 2, motor control corresponds to the blue arrow labelled “motor output”, where movement is initially conceived at some level of abstraction (i.e., abstractions of longitudinal and lateral control in the motor cortex) and then converted into individual muscle commands in the cerebellum/brainstem and spinal cord.

Such control approach has been anticipated and tested in D3.1 (D3.1 section 6.4.1 and D3.1 Figure 35) and D4.1, section 3. Figure 12 shows the control scheme (at a level more general than the cited deliverables). The motor plans instantiated in the dorsal stream are encoded in the abstract longitudinal/lateral control space of the motor cortex (longitudinal jerk and curvature rate). They are converted into the command of the actuators of a specific vehicle by means of inverse models of the vehicle dynamics, which are learned with methods mentioned in section 2.4. Updates to motor plans are continually produced by the Codriver (hence there are no two nested trajectory and tracking loops). In case of failure of the Codriver, the last trajectory may be used for a while, hence driving the host vehicle along the latest plan as explained in D4.1, section 3. However, in normal operation the Codriver updates motor plans at a rate that is similar to the inverse model rate (in our implementation the Codriver rate is 50 ms; the inverse model up-samples the Codriver output to 10 ms for the generation of the commands sent to the actuators).

With this scheme the minimum intervention principle is satisfied, because deviations are corrected by the Co-driver itself, leading to the generation of an optimal new trajectory for the current Codriver goal.

The learning of inverse models via neural networks may be, with care, quite accurate and even more accurate than traditional parametric analytical models [17].

Any inaccuracy of the inverse model (together with environmental disturbances) may in turn cause deviations. The stability of the control loop of Figure 12 has thus been studied in [21], concluding that there are very large margins of stability even if the inverse model is not adapted.

Finally, the actual stochastic vehicle forward dynamics may be learned in parallel with the inverse model (D3.1 section 6.1) and used to train the dorsal stream loops (D2.1 sections 3.3 and 3.4 and D3.1 section 6.4). This way the salience instantiated in the motor cortex will take into consideration the forward dynamics of the vehicle/context. For example, with reference to Figure 3, left, in slippery conditions the relevant forward model would produce a much weaker peak for the option of turning right (*b*). Hence, a much stronger bias (i.e., willingness to turn) is required if the road is slippery. This means the agent would normally give up turning in close curves if conditions are slippery. On snow the peak would probably disappear leading the agent not to consider at all the possibility of sharp curves.

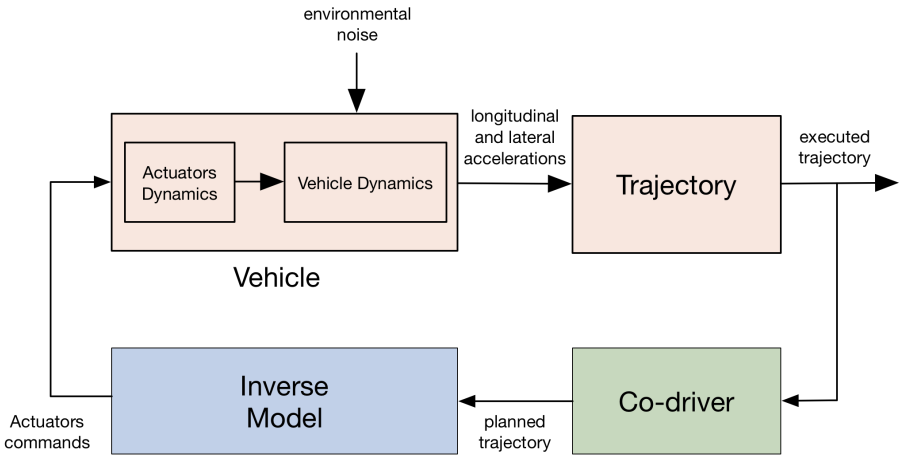

Figure 12: Inverse model approach for vehicle control.

3.1.3 Software (functional) implementation

The software implementation is schematically shown in Figure 13. It was significantly revised from the original implementation in previous AdaptIVe project.

In Dreams4Cars, the Codriver can be adapted/customized to different environments by using a configuration file (CONF.JSON in Figure 13). Also, sharing the Codriver among partners was eased by the creation of a library (CODRIVER LIBRARY) which wraps the Codriver software modules acting as a unique interface, in a way that any modification of the software is transparent to the users.

The CODRIVER LIBRARY allows the user to run the Codriver either locally (in the same CPU) or remotely (in a different dedicated CPU) via UDP communications, depending on how the software is initialised.

When selecting the remote functioning, the red blocks of Figure 13 are enabled and the UDP communication is established between the library (which acts as a client in remote mode) and the server. This operating mode is especially suitable for Software in the Loop, Hardware in the Loop prototyping and for the CRF Vehicle (see D5.1). On top of the server architecture there is the SERVER MANAGER which takes care of instantiating the communication parameters as specified in the configuration file such as IP address and port for the application.

The local operating mode (light blue boxes), conversely, is used for simulation with Model in the Loop (CarMaker MIL and OpenDS) and for the DFKI Vehicle. Because of the lack of UDP and hence lack of occasional packed losses, this operation modality is also perfectly reproducible.

In both operating modes the CODRIVER AGENT/MANAGER are the same. The MANAGER is a wrapper, which provides the initialisation of the agent parameters as specified in the user-customisable configuration file.

In addition, the MANAGER loads the appropriate forward/inverse models that are situation-specific. This way switching between models is possible. Interoperability and forward/inverse model adaptation are thus achieved by selecting the suitable parameters (including the indication of the forward/inverse models to use) in the configuration file CONF.JSON.

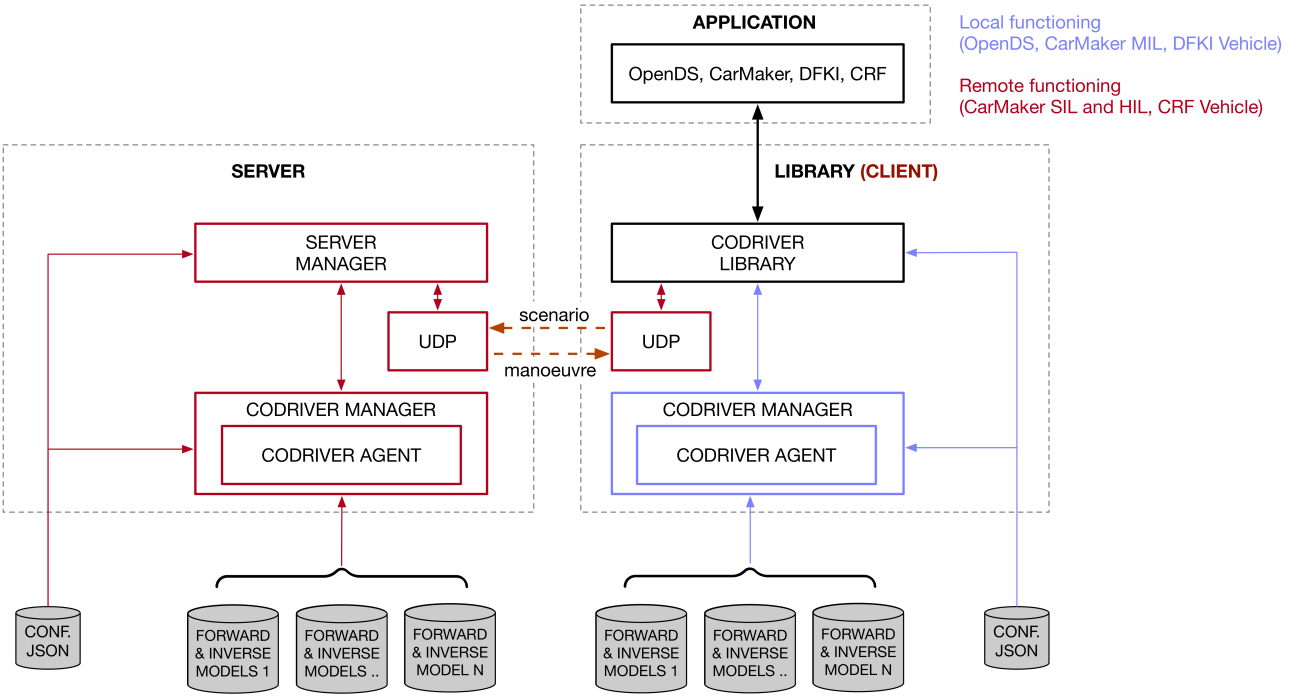

Figure 13: Functional diagram of software implementation.

3.2 Interoperability of high-level behaviours in vehicles of different types

As was described, the transportability of the Codriver in different testing environments (MIL, SIL, HIL, Prototype Vehicle), and on different vehicle models (Jeep Renegade and MIA) is given by the fact that the abstraction level used by the Codriver (coordinates, speed, etc.) is independent on the vehicle and sensor set. Also, common interfaces with the rest of the vehicle (scenario and manoeuvre messages) allow easy transportation among vehicles and simulation environments. The part of the system that depends on the specific vehicle, that is the preparation of sensor data, and chassis-level vehicle control, is executed by other modules outside of the Codriver, that interface the Codriver on the specific vehicle.

In particular, interoperability between vehicles of different types is obtained with the use of the appropriate inverse/forward models learned from the vehicle recorded behaviours. These models may also be changed (if necessary) and finely tuned when environmental conditions vary.

Two relevant aspects are involved here:

1) The *learned forward model* is used when training the dorsal stream computation of the salience. This way the same trajectory looks more or less affordable depending on the driven vehicle. There hence is a first form of adaptation to the vehicle.
2) The *learned inverse model* is used to map the Codriver planned control from the abstract generic cartesian space into the actual inputs for the actuators. There hence is a second form of adaptation to the vehicle.

The actual implementation of the control scheme in Figure 12 may vary, depending on which processing unit is used for running the inverse model.

3.2.1 CRF vehicle

In the CRF vehicle there are two processing units (D4.1), which are shown in Figure 14. The Codriver unit, which may either be the NVIDIA DRIVE PX2 or the VBOX PC (see D4.1), runs at 50 ms cycle time. The central unit, which is the dSPACE MicroAutoBox (with real-time OS), runs at 10 ms and, among the others, drives the actuators.

In the CRF vehicle the inverse model is implemented in the Codriver unit, and hence runs at 50 ms. Installing the inverse model in the MicroAutoBox would be very problematic, because of the lack of reliable deep learning frameworks that are necessary to run the inverse model neural network.

At every cycle (every 50 ms) the output of the Codriver Agent is given by motor control primitives $j(t-t_0)$ and $r(t-t_0)$ as functions of the elapsed time $t-t_0$. These represent the longitudinal and lateral control at the abstraction level of the dorsal stream (longitudinal jerk j and steering rate r). The future curvature $\kappa(t-t_0)$ and acceleration $a(t-t_0)$ are then also obtained. This may be carried out either by matching the actual initial curvature and acceleration of the vehicle, or, by means of an integrator block that may correct drifts in the executed manoeuvre.

The curvature and acceleration functions are then sampled at future times $k\,\Delta T - t_0$ and used, with the current state $x(t_0)$ to compute the steering angle and engine/brake commands. These are passed to the MicroAutoBox unit, where they are up-sampled, and used to feed the actuators taking the sample that corresponds to the time elapsed from t_0 and the current cycle. This way, delays in the Codriver unit loop are compensated and, in case the Codriver unit fails to update the motor plans, the last valid plan is used, which allows some resilience against failures of the Codriver unit.

Albeit not mentioned here, the OpenDS and CarMaker simulation environments follow a very similar arrangement.

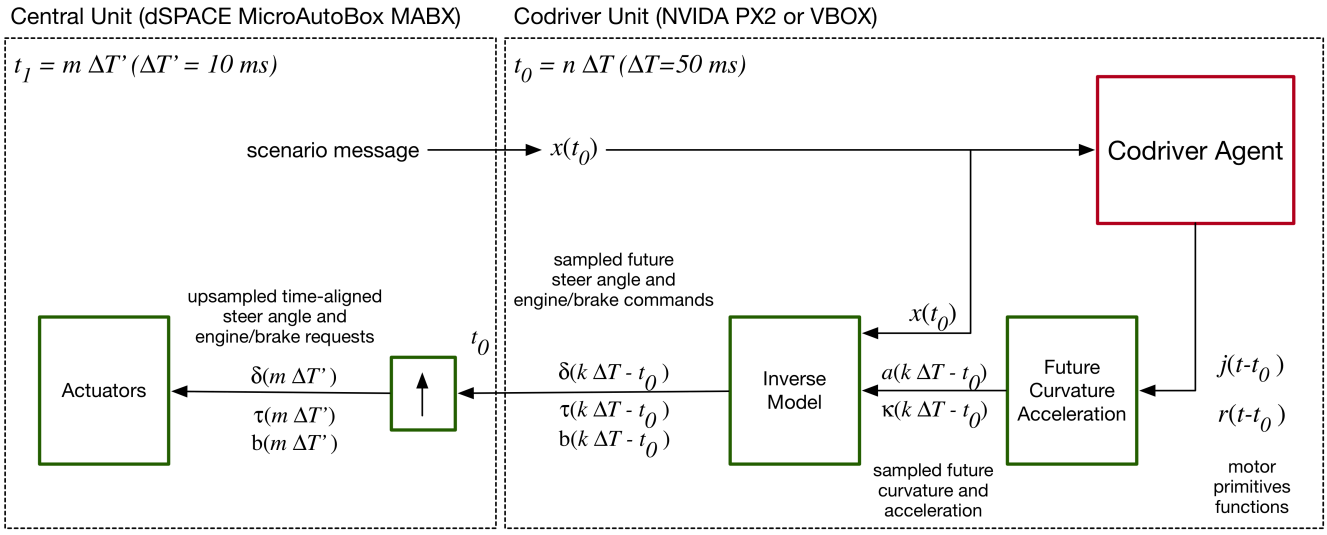

Figure 14: Inverse model implementation in the CRF vehicle.

3.2.2 DFKI vehicle

In the DFKI vehicle the Codriver unit is the NVIDIA DRIVE PX2 (see D4.1), which also runs at 50 ms cycle time. The rest of the vehicle is managed in a ROS environment which has several asynchronous processes.

In this case the inverse model is implemented in the ROS environment. Hence the Codriver unit sends the curvature $\kappa(t - t_0)$ and acceleration $a(t - t_0)$ functions to ROS. They are sampled and used with the current state $x(t)$ to compute the steering angle and engine/brake commands. This process may be carried out whenever necessary and may use a fresh $x(t)$ every time.

With the following arrangement, it is easy to switch among different inverse models, as well as implementing failure recovery strategies.

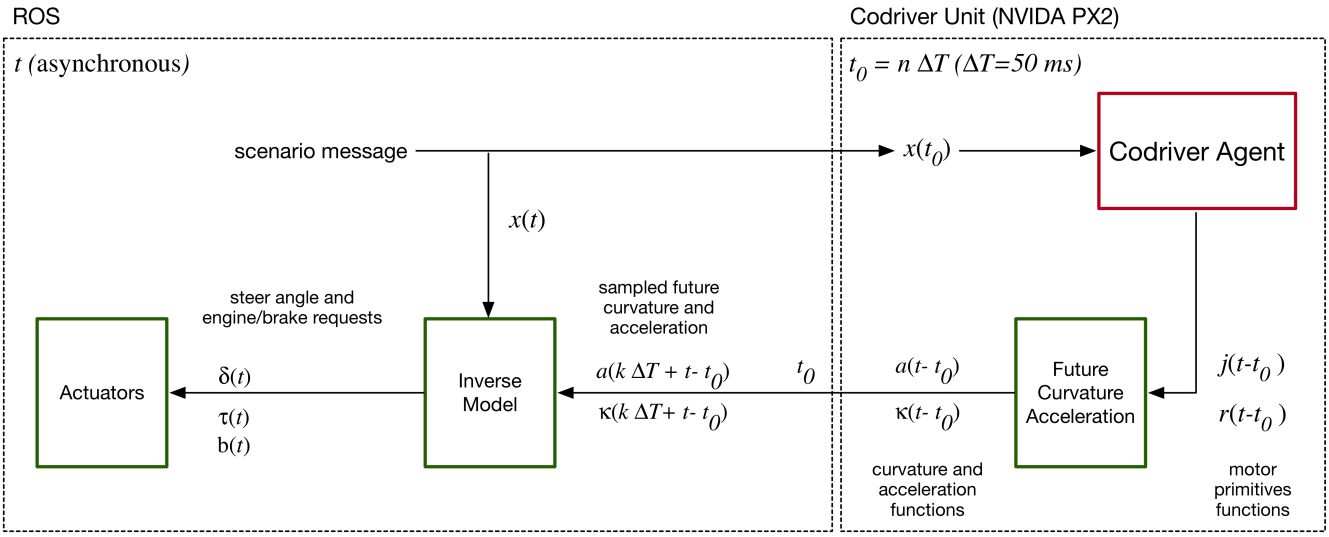

Figure 15: Inverse model implementation in the DFKI vehicle.

4 Bibliographical References

[1] P. Cisek, «Cortical mechanisms of action selection: the affordance competition hypothesis», *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 362, n. 1485, pagg. 1585–1599, set. 2007.

[2] G. Pezzulo e P. Cisek, «Navigating the Affordance Landscape: Feedback Control as a Process Model of Behavior and Cognition», *Trends Cogn. Sci.*, vol. 20, n. 6, pagg. 414–424, giu. 2016.

[3] K. Meyer e A. Damasio, «Convergence and divergence in a neural architecture for recognition and memory», *Trends Neurosci.*, vol. 32, n. 7, pagg. 376–382, lug. 2009.

[4] M. Da Lio, G. P. Rosati Papini, e R. Donà, «Hierarchical Robust Path Control for Automat-ed Driving in Roads Structured with Lanes», vol. Manuscript in Preparation, 2018.

[5] P. Viviani e T. Flash, «Minimum-jerk, two-thirds power law, and isochrony: converging approaches to movement planning.», *J. Exp. Psychol. Hum. Percept. Perform.*, vol. 21, n. 1, pagg. 32–53, mar. 1995.

[6] P. Bosetti, M. Da Lio, e A. Saroldi, «On the Human Control of Vehicles : an Experimental Study of Acceleration», *Eur. Transp. Res. Rev.*, 2013.

[7] P. Bosetti, M. Da Lio, e A. Saroldi, «On Curve Negotiation: From Driver Support to Automation», *IEEE Trans. Intell. Transp. Syst.*, vol. in press, 2015.

[8] R. S. Sutton, A. G. Barto, e R. J. Williams, «Reinforcement learning is direct adaptive optimal control», *IEEE Control Syst. Mag.*, vol. 12, n. 2, pagg. 19–22, 1992.

[9] M. Da Lio, A. Plebe, e D. Bortoluzzi, «On Reliable Neural Network Sensorimotor Control in Autonomous Vehicles», *IEEE Trans. Intell. Transp. Syst.*, pag. submitted.

[10] M. Da Lio *et al.*, «Artificial Co-Drivers as a Universal Enabling Technology for Future Intelligent Vehicles and Transportation Systems», *IEEE Trans. Intell. Transp. Syst.*, vol. 16, n. 1, pagg. 244–263, 2015.

[11] M. Da Lio, A. Mazzalai, e M. Darin, «Cooperative Intersection Support System Based on Mirroring Mechanisms Enacted by Bio-Inspired Layered Control Architecture», *IEEE Trans. Intell. Transp. Syst.*, vol. submitted.

[12] M. Da Lio, A. Mazzalai, K. Gurney, e A. Saroldi, «Biologically guided driver modeling: The stop behavior of human car drivers», *IEEE Trans. Intell. Transp. Syst.*, vol. 19, n. 8, pagg. 2454–2469, 2018.

[13] D. Windridge, «Emergent Intentionality in Perception-Action Subsumption Hierarchies», *Front. Robot. AI*, vol. 4, ago. 2017.

[14] D. Windridge e S. Thill, «Representational fluidity in embodied (artificial) cognition», *Biosystems*, vol. 172, pagg. 9–17, ott. 2018.

[15] R. Bogacz e K. Gurney, «The basal ganglia and cortex implement optimal decision making between alternative actions», *Neural Comput.*, vol. 19, n. 2, pagg. 442–477, 2007.

[16] L. F. Nunes e K. Gurney, «Multi-alternative decision-making with non-stationary inputs», *Open Sci.*, vol. 3, n. 8, pag. 160376, ago. 2016.

[17] S. James, A. Sean, e D. L. Mauro, «Longitudinal Vehicle Dynamics: A Comparison of Linear State-space, Nonlinear Physical and Neural Network Models», *Be Submitt.*

[18] E. Todorov e M. I. Jordan, «A Minimal Intervention Principle for Coordinated Movement», *Adv. Neural Inf. Process. Syst.*, vol. 15, pagg. 27–34, 2003.

[19] E. Bertolazzi, F. Biral, M. Da Lio, A. Saroldi, e F. Tango, «Supporting drivers in keeping safe speed and

safe distance: The SASPENCE subproject within the European framework programme 6 integrating project PReVENT», *IEEE Trans. Intell. Transp. Syst.*, vol. 11, n. 3, pagg. 525–538, 2010.

[20] D. Mitrovic, S. Klanke, e S. Vijayakumar, «Adaptive Optimal Feedback Control with Learned Internal Dynamics Models», *Robotics*, vol. 264, pagg. 65–84, 2010.

[21] R. Donà, G. P. Rosati Papini, M. Da Lio, e L. Zaccarian, «On the Robustness and Stability of Hierarchical Vehicle Lateral Control with Inverse/Forward Dynamics Quasi-Cancellation», *IEEE Trans. Intell. Transp. Syst.*