UP-Drive

Automated Urban Parking and Driving

Research and Innovation Action - Horizon 2020
H2020-ICT-24-2015: Robotics
Grant Agreement Number 688652

Project start date: January 1, 2016, Duration: 48 months

Deliverable D7.2

First development and integration cycle of decision making and navigation

Coordinating partner: Volkswagen A.G., Group Research
Coordinating person: Wojciech Derendarz

Lead contractor for this deliverable: Volkswagen A.G.
Deliverable editor: René Waldmann

Due date of deliverable: July 31, 2017
Revision: 1.0

Dissemination level: Public

Executive Summary

Deliverable 7.2 (D7.2) provides insights of the current software architecture representing the decision-making and navigation framework assigned to WP7 of the *UP-Drive* project which is eager to create a car capable of self-driving in urban traffic scenarios up to 30 km/h. The presented tool chain has been evolved during other EU funded projects like *V-Charge*, *InteractIVe* or *AdaptIVe* [13, 12, 11]. D7.2 describes the first development and integration cycle. Moreover, reflecting the experience gained in recent projects, significant architecture changes compared to Deliverable (D7.1) will be motivated and explained.

An introduction is given in section 1. The task formulation is shortly being recapped. Then some shortcomings of the architecture designed in D7.1 are highlighted motivating the algorithmic and system changes presented in the following sections.

Section 2 presents an overview of existing architectures proposed in the last decade to provide sophisticated decision-making behavior combined with advanced navigation and trajectory planning capabilities.

Then our proposed trajectory planner is explained in Section 3. Section 4 elucidates the resulting architecture changes.

Finally, Section 5 concludes the deliverable.

Contributing Authors/Partners

Partner	Author	Contribution
Volkswagen	René Waldmann	document editor
Volkswagen	Simon Großjohann	contributor

Contents

1 Introduction

The Deliverable 7.1 contributes to the *UP-Drive* (Automated Urban Parking and Driving) project which aims to create a car capable of self-driving in unconstrained urban environments with 30 km/h speed limits. *UP-Drive* goals are specified in the Description of Work [3] and the Deliverable 1.1 [2].

The *UP-Drive* consortium intends to build up a demonstrator platform consisting of a sensor-rich electric VW Golf sharing data with a cloud environment to demonstrate automated transportation in urban environments. *UP-Drive* exploits know-how and the technology of the previous *V-Charge* [13] project, in which partners VW and ETH Zurich participated.

Deliverable 7.2 will provide insights on the first integration cycle of the decision making and navigation stack. However, the main focus of D7.2. resides on the trajectory planner and how strategic decisions are going to be embedded into the framework.

In order to understand why the system introduced in D7.1 is suboptimal, the task formulation of WP7 is shortly recapped.

1.1 Task formulation

WP7 aims to provide all required functionalities for routing, decision-making, and trajectory planning. To this end, mandatory vehicle capabilities resulting from this WP definition have been derived (see [2], section 4.6):

- Smooth driving within own lane while keeping the track error very low
- Keeping safety distance to leading vehicle
- Constantly adapt the lateral distance to static objects within the own lane
- Decide whether to wait behind or to pass a vehicle standing in the own lane
- Compute collision free trajectories to handle pedestrians crossing the street
- Stop at zebra crossings to let pedestrians pass by
- Plan collision free trajectories to overtake pedestrians walking along the lane
- Compute a safe strategy to overtake and pass obstacles in presence of oncoming traffic
- Search for free parking spots and park in and out
- Do lane changes when necessary
- Follow traffic rules while passing intersections

Note that this is only a very rough summary of the system requirements. A detailed overview is available in Deliverable 1.1. However, the list implies a substantial complexity that comes with inner-city traffic.

In such scenarios, it is insufficient to generate trajectories while not taking into account either some high level strategy to reach the goal or the intention of other traffic participants. Furthermore, non-holonomic mechanics of the ego vehicle further constrain the planning process. In particular, this is relevant in more or less low-speed scenarios like urban traffic

where lateral and longitudinal motion directions cannot be handled separately anymore using simple polynomial trajectory representations [20, 19].

For instance parking out and smoothly squeezing into the flowing traffic requires good understanding about what is going on around the vehicle as well as precise motion planning in terms of curvature constraints, steering wheel limits and target poses. Please refer to Section 3.1 for detailed examples.

2 Related Work

In the past, many complex and impressive autonomous vehicle systems have been demonstrated which internally hosted navigation modules controlled by behavior layers. The first prominent example is *Stanley*, the vehicle that won the DARPA Grand Challenge in 2005 [17]. Although, only one single route guided the competitors through the Mohave dessert, *Stanley* already had a state management system with a global driving mode maintained in a finite state machine. However, it remains unclear how this driving mode exactly worked. The navigation component consisted of a global path planner smoothing the road network data and a local path planner which was capable of avoiding obstacles and guiding the vehicle back to the reference path.

The vehicle *Junior* [8] which competed in the DARPA Urban Challenge (2007), featured a hierarchical state machine with 13 states to handle urban traffic scenarios. Typical driving behavior modes were *drive forward*, *stop at intersections*, or *wait at stop signs*. On the navigation side, a global path planner has been applied to compute an optimal route from every location in the map to the next check point using dynamic programming.

The winning vehicle *Boss* [18] was able to perform three different driving categories, namely *lane driving*, *passing intersections*, and *parking* (which was called *achieving a zone pose* by the authors). Furthermore, a mission planner connected the road network data to a graph structure where each node contains an optimal path to the goal. This information has constantly been updated. Then, a motion planner computes a set of feasible trajectories along the reference path to reach the goals provided by the mission planner.

The vehicle *Odin* which achieved the 3^{rd} place, had a driving behavior layer which internally managed three different driver types, i.e., *target point driver*, *speed driver*, and *lane driver* [15]. The selection of the preferred driver type has been conducted by using a *winner-takes-it-all* strategy. A motion planner is considered as decision-making layer between the driving behavior module and the vehicle interface converting the incoming target points, lane and speed goals into low level vehicle commands using a trajectory planner. However, before computing the trajectory, a navigation strategy has been generated by roughly subdividing the scenarios into lane navigation and zone navigation (where parking takes place).

Other well known urban challenge vehicle examples are *Caroline* [14], *AnnieWAY* [6], *Skynet* [7], or *Knight Rider* [9] which all featured comparable driving behavior, decision-making, and navigation stacks.

Another real-time decision-making and navigation approach has been presented by Furda and Vlacic[4]. They propose a decision layer consisting of two states. The first stage generates a set of feasible driving maneuvers, i.e., maneuvers which are safe and follow the traffic rules. To this end, an event based system has been set up forwarding the incoming events to a Petri Net model. Furthermore, route planning information are fed to the Petri Net. The latter is important as overtaking another vehicle right before a planned turn might not be feasible. The second stage takes the set of all feasible driving maneuvers as input and selects one maneuver according to objectives as efficiency and comfort.

However, *V-Charge* [5, 13] as predecessor project of *UP-Drive* used a system where at a first stage a global, drivable path has been computed once, guiding the vehicle over the parking lot. Then a mission has been planned generating a set of tasks necessary to reach the parking spot. Typical tasks were *switch on engine* and *follow lane*. So-called task processors for each task have been defined. For example, the trajectory planner was capable to follow the lane or to pass intersections. At the beginning of a mission, every task processor registered his capabilities at a central coordinating instance, which was called *mission executive*. This module continuously processed the task list and activated the responsible task processors. Decision-making took place in situations were the pre-planned task list did not hold, e.g., when the parking spot was occupied. Then the system had to decide which strategy to follow next to reach another parking spot.

All systems presented so far have in common that strategic layers and trajectory planning modules are strictly separated. In contrast, this deliverable describes an approach were tactical layers and path planning layers are highly integrated which diminishes the problem of computing consistent target pose sets using traditional rule based methods.

Dolgov *et al.* [1] introduced the well-known hybrid-A* path planner which resembles our trajectory planner most. The authors compute a set of discrete motion primitives evolving in space which is discretized into continuous state cells. If more than one state falls into the same cell, the best solution is being kept. Similar methods use state-lattice approaches where motion primitives are used as well to crawl through the search space [10],[16]. Please note that there are tons of trajectory planning proposals in the literature where most methods are out of the scope of this deliverable. However, our method is capable of handling parking, urban traffic, and highway scenarios at the same time. To the best of our knowledge the algorithm described here outperforms today's state-of-the-art approaches in terms of speed, flexibility, and quality of the solutions.

3 Trajectory Planning with Dynamic Programming

3.1 Motivation

At the beginning of the last decade, driver assistance systems became more and more popular. To this end, simple state machines were used to feed controller commands to the vehicle. However, as illustrated in Fig. 1, the higher the degree of automation and complexity, a substantial increase of the development time can be observed.

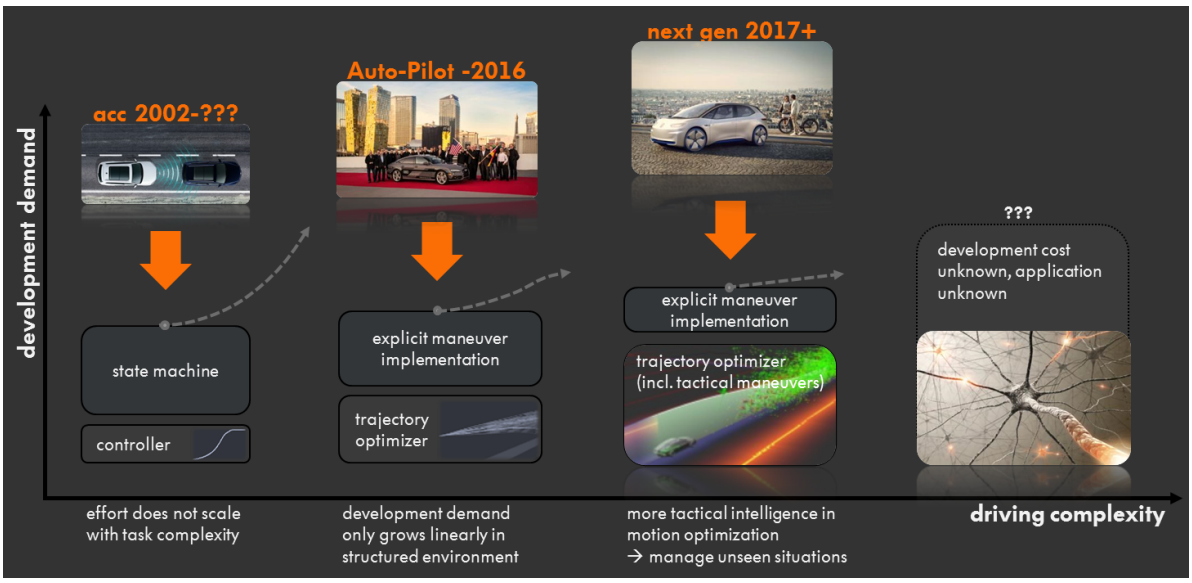


Figure 1: Development demands depending on self-driving system complexity influence the design of the architecture. However, future characteristics are still unclear.

Researchers of the automotive industry constantly adapted the system architecture the more experience was gained with level 3 self-driving functions. Here, tactical layers generated a set of vehicle maneuvers which were transformed to suitable trajectories by advanced optimization routines (cf. Fig. 1). Typically, such systems have been designed for highway scenarios were the behavior of other traffic participants is more or less easy to predict compared to urban scenarios. Thus, state machines sufficed to handle the traffic complexity. In the course of the investigation of inner-city scenarios, we learned that a system with more tactical capabilities directly embedded in the motion optimization routine decreases the development demands significantly when conquering urban traffic with a push towards level 5 automation. Moreover, with increasing situational complexity, many descisions require extensive knowledge about your cars and possibly environmental constraints, which descision making layers typically do not have to sufficient extent.

An example can be seen in Fig. 2. The red car constitutes the ego vehicle. Its goal is marked by the red cross. Therefore, the vehicle has to find a way over the intersection. All other cars intend to go straight as implied by the red arrows and the pedestrians are going to cross the road. The complexity of this scene is manifold. First of all, the ego vehicle has to detect the pedestrians as well as the yield traffic sign. Consequently, it has to give way to all vehicle of the scene. The overall situation is very complex in terms of maneuver generation given a suitable state machine which needs to compute the right target states at every point of time including a correct priority order (cf. [19], pp 16). This example shows that pure state machine approaches on the tactical layer need to count (nearly) all possible situation which renders the tactical module cumbersome to implement and to maintain. Therefore, a new trajectory planning method is being described in this deliverable which absorbs many tasks of the tactical layer by capturing the overall scene in the motion optimization routine.

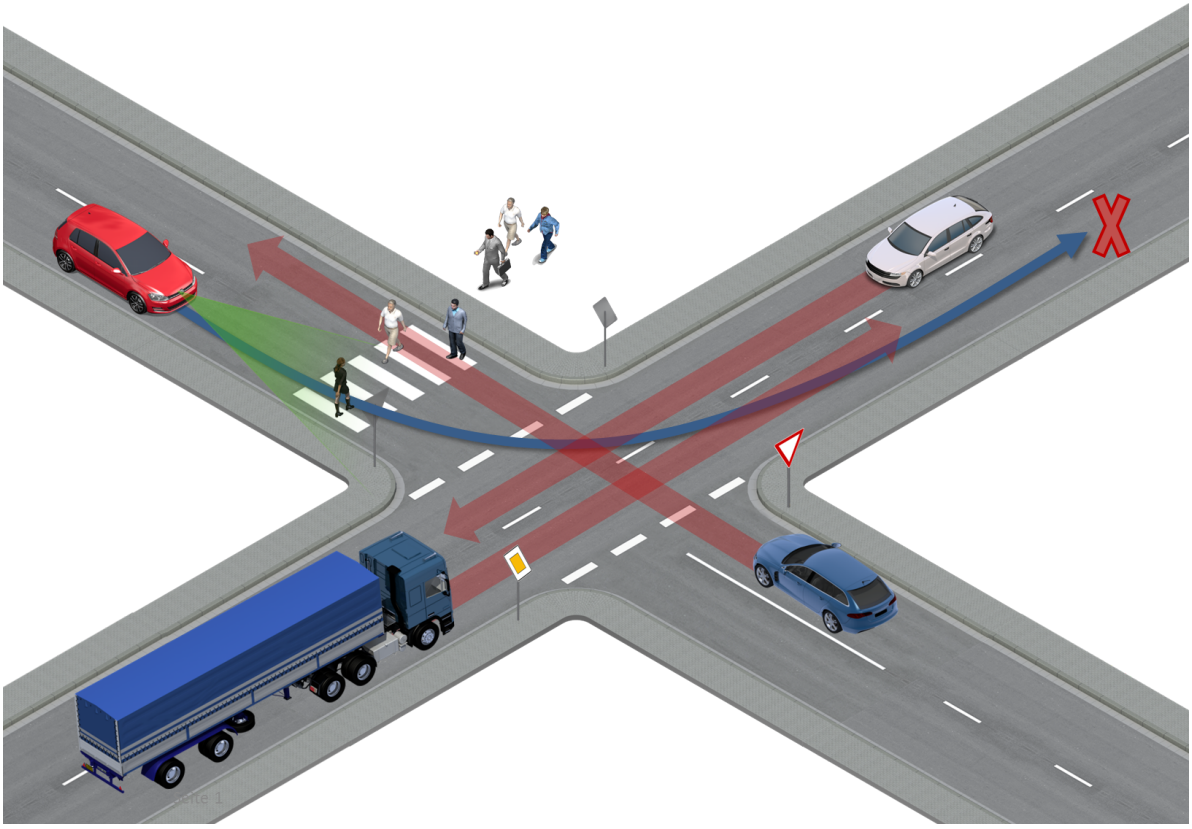


Figure 2: Traffic scene elucidating the difficulty of state machines on the decision layer. The ego vehicle is represented by the red car. Its target is marked by the red cross. This example shows that it is hard to account for all peculiarities when computing motion target using pure state machine approaches.

3.2 Principles of the planner

3.2.1 Dynamic Programming

The functionality of the presented trajectory planner is based on Bellman's principle of optimality which refers to the field of dynamic programming. Dynamic programming can be applied to problems which can be further divided into homogeneous sub problems, i.e., the problem can be broken down into recursion. The proposed method constitutes a holistic approach tackling the trajectory planning problem by finding a global optimum of the cars 2D optimization problem by exploiting massively parallelized dynamic programming technique. Dynamic programming is referred to as DP in the remainder of this deliverable.

Many DP algorithms start by taking the target state into account and move backwards until the initial state has been reached. However, we initially use the current vehicle state and explore the world in time with a flexible combination of actions for steering wheel angle and gas/brake pedal, which are then integrated with a suited vehicle model.

Currently, the vehicle dynamics are defined by a car-like model which is captured by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} cos\,(\theta) \cdot v \\ sin\,(\theta) \cdot v \\ \frac{v}{L} tan\,(\phi) \\ a \end{bmatrix} \tag{1}$$

Here, the vehicle state is defined by $(x, y, \theta, v, a, \phi)$ which refers to its position, orientation,

velocity, acceleration, wheel angle ϕ and a constitute control inputs to the planner to generate motion primitives for short periods of time using polynomials with different configurations. The overall DP principle is being illustrated in Fig.3. At time $t_0 = 0$ the planner starts to work given the current vehicle configuration. Then, the aforementioned variations of the control inputs are applied to move forward for a certain time step Δt (e.g., 5 seconds). The integration of the vehicle dynamics result in a set of vehicle states at time t_1, i.e., 5 in this example.

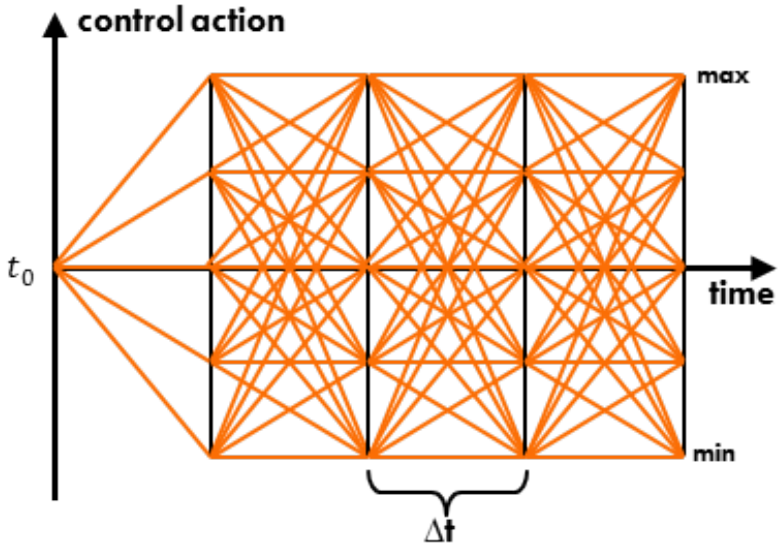


Figure 3: Control action over time

The computation of motion primitives is triggered again for each of the 5 new states. Technically, this approach yields 25 states. Consequently, the growth of states has exponential characteristics and thus going deeper in time would be intractable. Therefore, the state space is being discretized yielding a high dimensional array. If several states fall into the same motion state cell, only the best one is being chosen for further exploration while the rest is discarded. Therefore, the growth is close to linear now as illustrated in Fig.3. A more realistic example can be seen in Fig.4. Given the current vehicle state, 128 successors are computed by applying different lateral and longitudinal control shapes called actions. The states fall into a 11×11 spacial grid (in fact this is a simplification to the actually used motion state cells, which are of much higher dimensions such as space, velocity, acceleration, wheel angle, yaw angle, etc., and should consequently be called hyper cubes). From each motion state cell which hosts a vehicle state, 128 new successors are being computed entering a 11×11 grid again. Only the strongest successors of a motion state cell survive while the rest is discarded. In this way, the complexity is reduced from $11 \times 11 \times 128$ new states to 11×11 yielding a linear growth.

This approach is very similar to the one presented by Dolgov *et al.*[1]. The main difference here is that the motion primitives are generated by integrating a sufficiently realistic vehicle model and therefore the resulting trajectory is driveable without any further optimization. Moreover, the method of Dolgov *et al.* yields paths and no trajectories. Here, exploration is performed for the 2D problem in an extremely flexible hyperspace with arbitrary dimension compared to three dimensions of [1]. In order to keep the computational complexity tractable, massive parallelization takes place by shifting the motion generation to a graphical processing unit (GPU) using the Nvidia CUDA framework. Typical notebook

GPUs like Nvidia Quadro M200M (1,300 GFlops, 640 Cuda cores) suffice to manage the computational burden. However, the more computational power is available (in terms of GFlops and CUDA cores), the more solutions can be checked substantially increasing the quality of the resulting trajectory.

For the sake of clarity, the following item list summarizes the planner scheme:

1. Upload valid initial vehicle state from CPU to GPU
2. Generate actions from available source states (GPU)
3. Integrate actions with vehicle dynamics up to target time (GPU)
4. Compute motion costs during integration (GPU)
5. Determine new motion state space discretization (GPU)
6. MIN-selection: pick the best candidate per motion state space cell (GPU)
7. Go back to 2
8. Shift current solution from GPU to CPU parallel to 7

Here, target time refers to the maximum time horizon used for state generation, e.g., 30 seconds. Note, that the planner never stops working since the algorithm jumps to action generation after the best solution has been picked up. Choosing the right target state is a very critical step and will be elucidated in section 3.2.2.

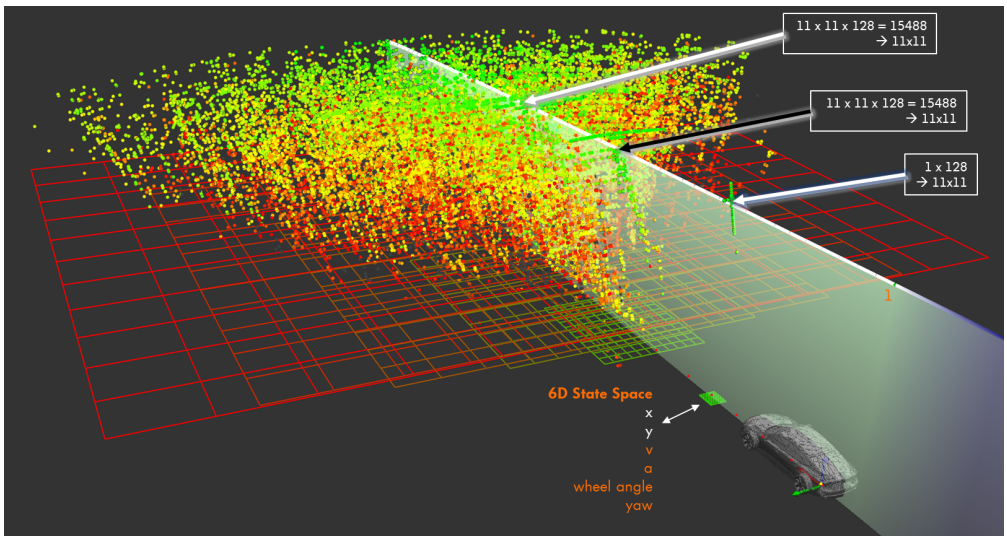


Figure 4: The planner here explores a $6d$ state space. In this example, the motion state space is visibly discretized into an 11×11 spacial grid with an ever decreasing resolution the more exploration along the time axis has been performed. If several successor states fall into the same state space cell, only the best successor is chosen while the rest is discarded.

3.2.2 Solution Assessment

Steps 6 and 8 of the described algorithm rise the problem of how to assess the state candidates. In fact, state assessment can be modeled as a multidimensional decision problem

as illustrated in Fig. 5. On the one hand there are motion costs of all candidates, leading to a pre-selection of favourable motion candidates. On the other hand, there are objectives, such as reaching a target pose or following navigation and making quick progress. These are some of the most obvious factors that influence the decision making process:

Acceleration High acceleration and deceleration should be avoided even if target proximity needs to be sacrificed.

Jerk Should be close to zero since jerk feels really bad for the driver.

Wheel angle change Penalizing strong wheel angle changes helps to avoid swerving.

Strategic cost The driving maneuvers should be comprehensible. This implies that the unnecessary lane changes have to be penalized even when it helps to reach the target *make as much longitudinal progress as possible.*

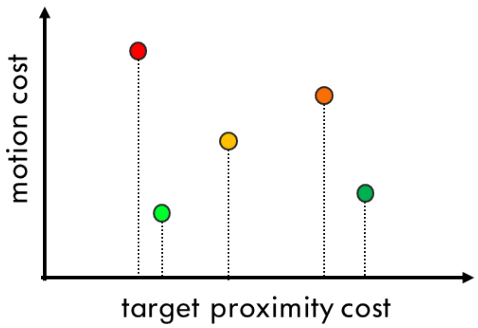
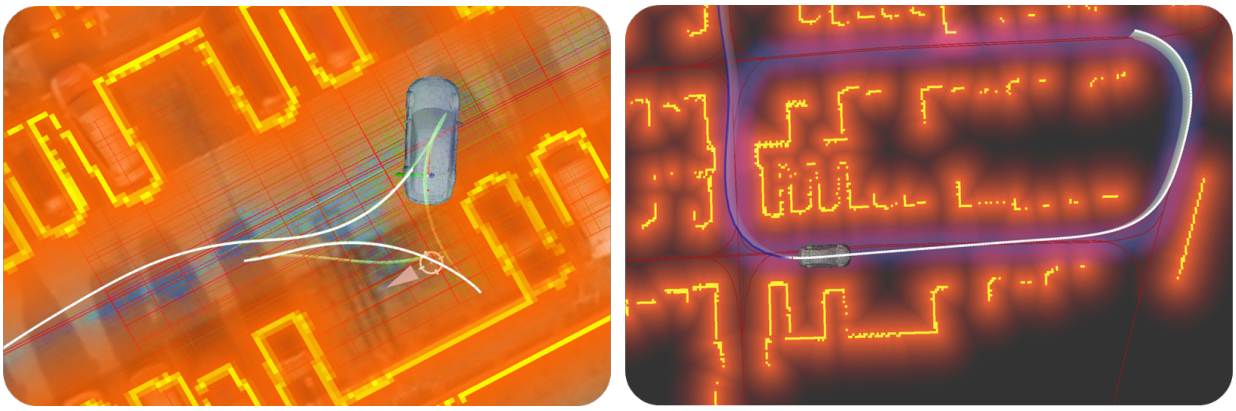


Figure 5: Assessment of the state candidates requires to consider motion costs as well as target proximity cost. The vehicle should be as close to the target as possible but not at any price.

In contrast, the proximity to the target is being evaluated in step 8. Currently, two possible targets can be selected. The first one refers to parking, i.e., the vehicle should be as close to a predefined parking pose (x_p, y_p, θ_p) as possible. The second target is defined as to make as much progress along the road as possible given a dedicated time budget. Of course, this target must be constrained by traffic rules (see strategic cost), e.g., passing an intersection even if the traffic light is red is not a good idea. Note, that Fig. 5 suggests that both cost types are evaluated at the same time. However, this is not the case since the algorithm linearizes the problem by just taking the motion costs first and then creating a valid target state if the target has been reached with a sufficient precision.

3.2.3 State Actions

Lateral state actions point from the current state's wheel angle ϕ_t to new target wheel angles $\phi_{i,t+1}$ with $i \in$ lateral action count, which are bound by maximum and minimum wheel angles $\phi_{max/min}$. An action from time t to $t+1$ is therefore sampled in $\phi_{i,t+1} \in [\phi_{min}, \phi_{max}]$ and connects to ϕ_t by a multitude of suitable action functions, such as constant wheel angle rates, polynomials or interval functions. The same method is being applied for longitudinal states, where the current state's acceleration a_t is connected to a plethora of $a_{j,t+1} \in [a_{min}, a_{max}]$ and $j \in$ longitudinal action count with individual sets of suited action functions, such as constant jerk, acceleration profiles and again functions defined on intervals.



(a) Parking scenario (b) Driving on parking lot

Figure 6: Parking and driving scenarios. Static obstacles are represented by a so-called scene mask (see section 3.2.4) resembling a potential field approach.

3.2.4 Strategic Decisions

Strategic navigation decisions are implicitly computed by using the concept of the so-called *scene mask*. Technically, the scene mask resembles a grid data structure providing costs for visiting the cells. It aggregates heterogeneous information about many aspects of traffic scenes, e.g.,

- Static obstacles
- Dynamic obstacles
- Lane centers
- Route data
- Road boundaries

Thus, the scene mask is made of several stacked layers where each layer influences the strategic motion costs. It is computed w.r.t. world coordinates and each vehicle state generated during planning is being projected onto each scene mask layer accumulating the cost.

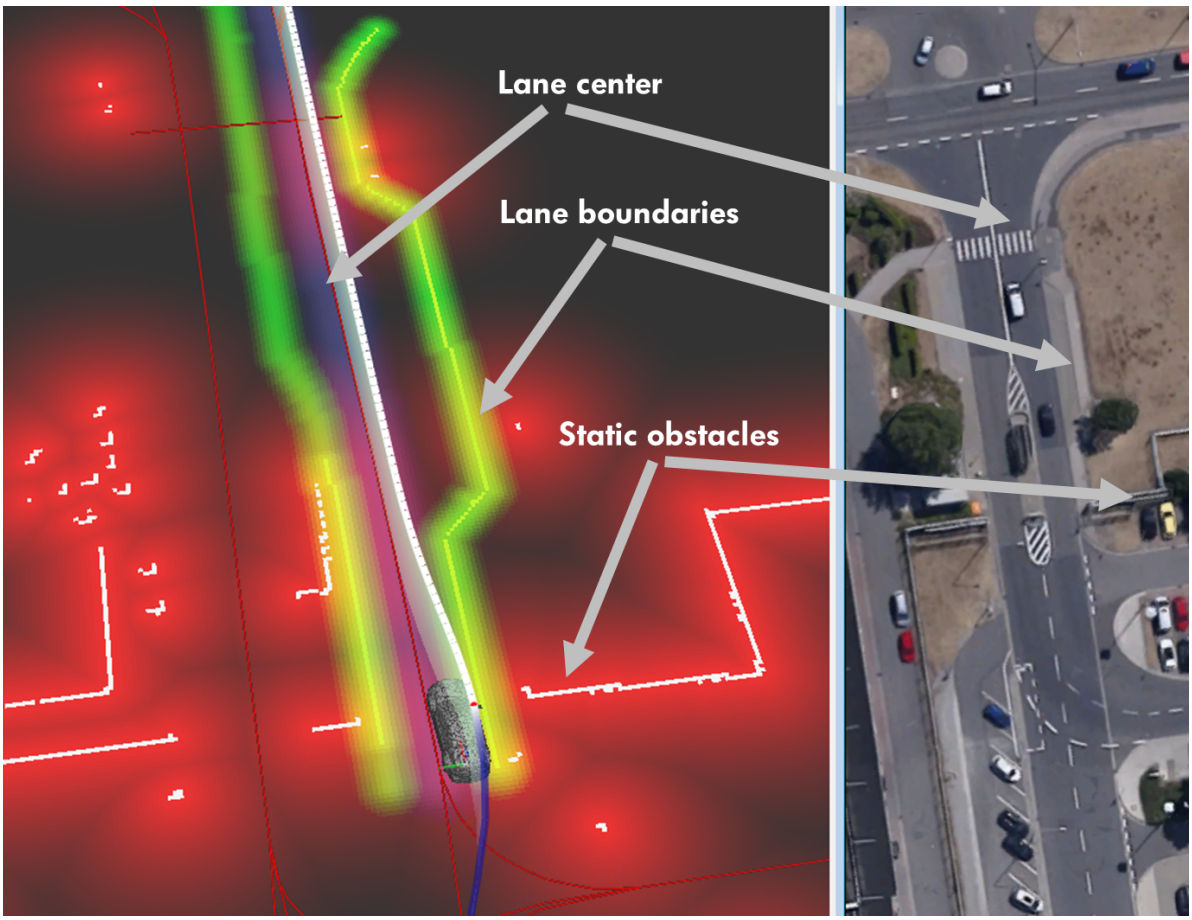
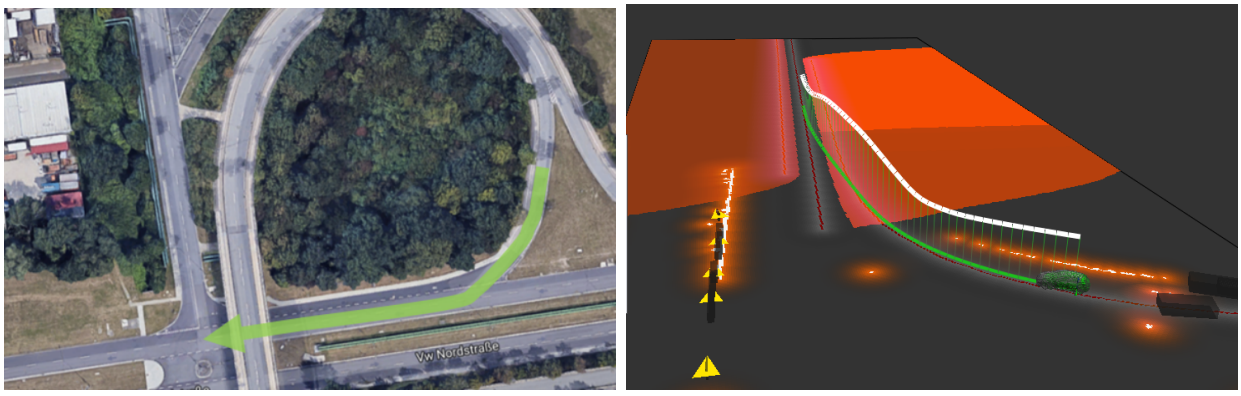


Figure 7: Aggregated scene mask displaying lane center, lane boundaries and static obstacles.

Fig.7 shows an example of several layers representing static infrastructure. Static obstacles, lane center and lane boundary data are being stacked in order to motivate the vehicle to follow the lane. Each layer is a potential field pushing the vehicle states away from areas which shall not be passed. For example, the closer the vehicle gets to the wall on the right hand side, the higher the motion costs. This example shows that the planner has basically no restrictions. The human driving behavior as well as strategic decisions result from the optimization routine considering the scene mask.



(a) (b)

Figure 8: The lane change is forced by the scene mask which produces high strategic motion costs when not following the lane advice.

Another examples demonstrating the usage of route data is shown in Fig.8. Here, the red areas reflect the influence of the so-called lane advice which is simply a list of consecutive edges of the road graph (cf. [19]) guiding the vehicle to its goal. Every area which is not part of the lane advice induce high travel costs pushing the optimization routine towards the right lane center.

Any right of way rules are managed by WP 6 *Scene Understanding*. Other vehicles are predicted accordingly, e.g., on intersections where the ego vehicle needs to give way. I.e., other vehicles are predicted over the intersection without any stop motion. This is a simple method to make the ego vehicle stop and drive correctly.

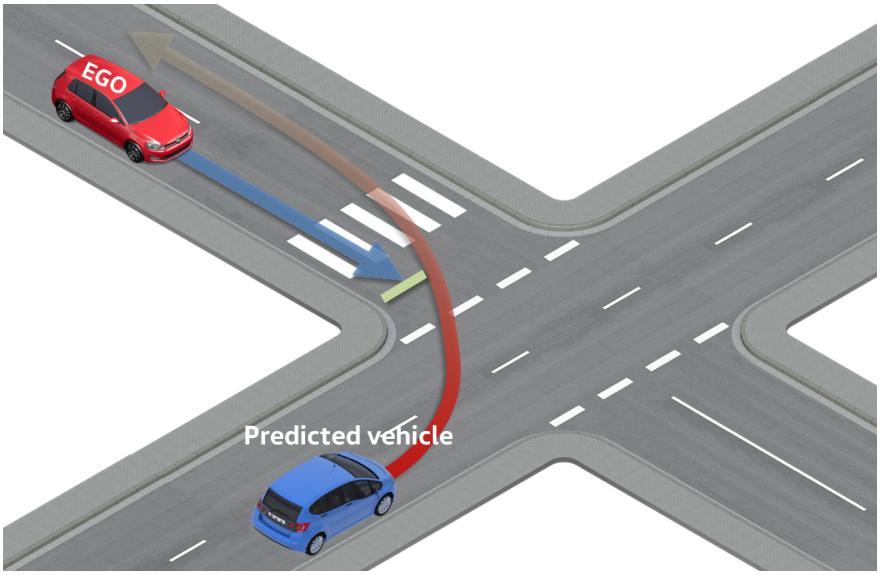


Figure 9: The blue vehicle is predicted over the intersection since it has right of way. The red (ego) vehicle is being forced to stop as the trajectory planner detects a collision otherwise.

To this end, dynamic obstacles are being handled by a distinct layer which currently under investigation and will be explained in the next deliverable.

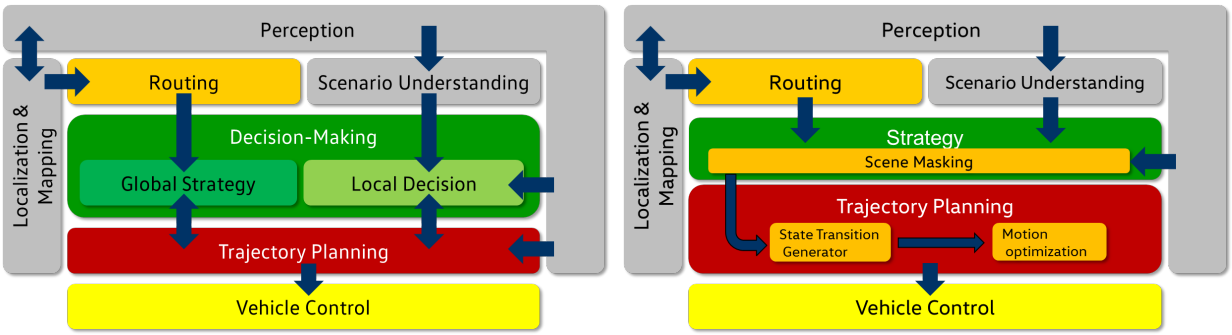
4 Change of Architecture

Deliverable 7.1 has explained all modules in detail which influence the decision making and navigation stack (cf. [19]). For the sake of clarity, the layers described in D7.1 are roughly recapped:

Routing provides functionalities to compute a global reference path to the user-defined target. Here, a global path is represented by a set of consecutive edges guiding the vehicle to its goal.

Decision-Making consists of a global strategy module and a local decision one. The global strategy orchestrates all different modules required for a (safe) navigation. E.g., this includes trajectory planners for both standard traffic and parking scenarios. In contrast, local decision frequently updates the routing information and places target poses for the trajectory planner.

Trajectory Planning hosts all path and trajectory planning modules. Two planners used to be active in this layer, i.e., a trajectory planner for road following and a parking planner.



(a) (b)

Figure 10: (a) Old architecture with an explicit decision-making layer (b) Our recent architecture feeds routing and scenario understanding output directly to the trajectory planner

Vehicle Control takes the trajectories as input and computes the according control commands,e.g., acceleration and steering wheel velocity.

The data flow is already indicated in Fig. 10(a). However, our current architecture has evolved much in recent months as shown in Fig. 10(b). The main difference compared to the architecture presented in D 7.1 is that we do not have a dedicated decision-making layer anymore but a slim strategy layer responsible for computing the right scene masks as explained in section 3.2.4. Then, the right motion behavior directly results from the 2D optimization algorithm. Global strategy has been erased as we do not have several planners anymore which need to be orchestrated but only one solution.

5 Conclusion

This deliverable elucidates our latest trajectory planner concept including significant changes of the navigation and decision architecture. D7.1[19] described a system which used two completely separated trajectory planners for highway and parking scenarios, respectively. Additionally, a complex strategy layer computed the right target points. Thus, the system constituted a solution *counting* all possible scenarios. However, section 3.1 motivates why such a system does not meet our requirements for complex inner-city traffic. Therefore, a novel trajectory planner based on the well-known dynamic programming paradigm is introduced in section 3.2. In summary, the planner computes a flexible set of motion primitives given an initial vehicle configuration which yields a set of successor states. Each successor applies individual motion primitives again and so on. The motion state space is subdivided into a high dimensional array. Whenever several states fall into the same motion state space cell, only the best local solution is being kept and the rest is discarded which keeps the computational complexity tractable. The planner associates motion and target proximity costs with every state in order to assess every possible solution. This concept allows for arbitrary vehicle motion patterns and hence the novel planning approach facilitates both parking and highway driving. Furthermore, no target point generator is required anymore because the planner gets the global target (i.e., make as much progress along the route as possible or park in) and the correct driving behavior results from the optimization routine, introducing a new level of robustness and descision making competence. To this end, the concept of the *scene mask* has been introduced in section 3.2.4. It constitutes several stacked grid layers where each cell carries information about the costs to visit the cell. For example, the layer representing static obstacles provides a traditional potential field to push the vehicle away from walls, curb stones, etc.. Eventually, the scene mask concept replaces the decision making layer introduced in D7.1. In summary, the following changes of the architecture have been presented and described:

- The architecture hosts only one trajectory planner.
- The trajectory planning paradigm has been completely revised.
- The trajectory planner basically has no restrictions. The correct behavior is the result from the optimizer.
- The decision-making layer from D7.1 has been replaced by a slim strategy component generating the scene mask
- The scene mask carries information where to drive and where not

Future work will focus on dynamic obstacles as well as traffic rule handling. In particular, the latter rises important questions how to generically encode traffic rules, e.g., traffic lights.

References

[1] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, Chicago, USA, June 2008. AAAI.

[2] Sergiu Nedevschi et al. Initial version of requirements definition, system architecture and component specification. Deliverable 1.1, Project UP-Drive consortium, May 2016.

[3] W. Derendarz et al. Project UP-Drive, Description of Work, Technical Annex, December 2015.

[4] Andrei Furda and Ljubo B. Vlacic. Enabling safe autonomous driving in real-world city traffic using multiple criteria decision making. *IEEE Intell. Transport. Syst. Mag.*, 3(1):4–17, 2011.

[5] Paul Furgale, Ulrich Schwesinger, Martin Rufli, Wojciech Derendarz, Hugo Grimmett, Peter Mühlfellner, Stefan Wonneberger, Julian Timpner Stephan Rottmann, Bo Li, Bastian Schmidt, Thien Nghia Nguyen, Elena Cardarelli, Stefano Cattani, Stefan Brüning, Sven Horstmann, Martin Stellmacher, Holger Mielenz, Kevin Köser, Markus Beermann, Christian Häne, Lionel Heng, Gim Hee Lee, Friedrich Fraundorfer, René Iser, Rudolph Triebel, Ingmar Posner, Paul Newman, Lars Wolf, Marc Pollefeys, Stefan Brosig, Jan Effertz, Cédric Pradalier, and Roland Siegwart. Toward Automated Driving in Cities using Close-to-Market Sensors, an Overview of the V-Charge Project. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 809–816, Gold Coast, Australia, 23–26 June 2013.

[6] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagszent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, Oliver Pink, Christian Frese, and Christoph Stiller. Team annieway's autonomous system for the DARPA urban challenge 2007. In *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA*, pages 359–391, 2009.

[7] Isaac Miller, Mark Campbell, Dan Huttenlocher, Aaron Nathan, Frank-Robert Kline, Pete Moran, Noah Zych, Brian Schimpf, Sergei Lupashin, Ephrahim Garcia, Jason Catlin, Mike Kurdziel, and Hikaru Fujishima. *Team Cornell's Skynet: Robust Perception and Planning in an Urban Environment*, pages 257–304. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[8] M. Montemerlo, J. Becker, S. Bhat, H. Dahlkamp, D. Dolgov, S. Ettinger, D. Haehnel, T. Hilden, G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, and S. Thrun. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 2008.

[9] Benjamin J. Patz, Yiannis Papelis, Remo Pillat, Gary Stein, and Don Harper. *A Practical Approach to Robotic Design for the DARPA Urban Challenge*, pages 305–358. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[10] Mihail Pivtoraiko and Alonzo Kelly. Generating state lattice motion primitives for differentially constrained motion planning. 2012.

[11] Adaptive - automated driving. `https://www.adaptive-ip.eu/`, 2014-2016. European Commission funded project, FP7. Accessed: 2016-06-17.

[12] Interactive - accident avoidance by active intervention for intelligent vehicles. `http://www.interactive-ip.eu/`, 2009-2013. European Commission funded project, FP7. Accessed: 2016-07-07.

[13] Automated valet parking and charging for e-mobility (V-Charge). `http://www.v-charge.eu/`, 2011-2015. European Commission funded project, FP7. Accessed: 2016-03-18.

[14] Fred W. Rauskolb, Kai Berger, Christian Lipski, Marcus A. Magnor, Karsten Cornelsen, Jan Effertz, Thomas Form, Fabian Graefe, Sebastian Ohl, Walter Schumacher, Jörn-Marten Wille, Peter Hecker, Tobias Nothdurft, Michael Doering, Kai Homeier, Johannes Morgenroth, Lars C. Wolf, Christian Basarke, Christian Berger, Tim Gülke, Felix Klose, and Bernhard Rumpe. Caroline: An autonomously driving vehicle for urban environments. In *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic, George Air Force Base, Victorville, California, USA*, pages 441–508, 2009.

[15] Charles Reinholtz, Dennis Hong, Al Wicks, Andrew Bacha, Cheryl Bauman, Ruel Faruque, Michael Fleming, Chris Terwelp, Thomas Alberi, David Anderson, Stephen Cacciola, Patrick Currier, Aaron Dalton, Jesse Farmer, Jesse Hurdus, Shawn Kimmel, Peter King, Andrew Taylor, David Van Covern, and Mike Webster. *Odin: Team VictorTango's Entry in the DARPA Urban Challenge*, pages 125–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[16] Alexandru Rusu, Sabine Moreno, Yoko Watanabe, Mathieu Rognant, and Michel Devy. State lattice generation and nonholonomic path planning for a planetary exploration rover. In *65th International Astronautical Congress 2014 (IAC 2014)*, volume 2 of *65th International Astronautical Congress 2014 (IAC 2014) Our World Needs Space, ISBN: 978-1-63439-986-9*, page 953, Toronto, Canada, September 2014.

[17] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, September 2006.

[18] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, M. N. Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, Michele Gittleman, Sam Harbaugh, Martial Hebert, Thomas M. Howard, Sascha Kolski, Alonzo Kelly, Maxim Likhachev, Matt McNaughton, Nick Miller, Kevin Peterson, Brian Pilnick, Raj Rajkumar, Paul Rybski, Bryan Salesky, Young-Woo Seo, Sanjiv Singh, Jarrod Snider, Anthony Stentz, William "Red" Whittaker, Ziv Wolkowicki, Jason Ziglar, Hong Bae, Thomas Brown, Daniel Demitrish, Bakhtiar Litkouhi, Jim Nickolaou, Varsha Sadekar, Wende Zhang, Joshua Struble, Michael Taylor, Michael Darms, and Dave Ferguson. Autonomous driving in urban environments: Boss and the urban challenge. *J. Field Robot.*, 25(8):425–466, August 2008.

[19] Rene Waldmann. Software specification and architecture for decision making and navigation. Deliverable 7.1, Project UP-Drive consortium, September 2016.

[20] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenét frame. In *ICRA*, pages 987–993. IEEE, 2010.