Design of the planer of Team *AnnieWAY*'s autonomous vehicle used in the *DARPA Urban Challenge* 2007

Tobias Gindele, Daniel Jagszent, Benjamin Pitzer and Rüdiger Dillmann

***Abstract*— This paper reports on the behaviour decision and execution unit of *AnnieWAY*, an autonomous vehicle that is capable of driving through urban scenarios and that has successfully entered the finals of the *DARPA Urban Challenge* 2007 competition. Starting with a short description of the car and its major hardware components, we outline the underlying software structure and focus on the design of the behavior decision module. The selection of maneuvers necessary to accomplish the mission is conducted online via a concurrent hierarchical state machine that specifically ascertains behavior in accordance with California traffic rules. The states and transitions used to model a adequate behaviour are described. We conclude with a report of the results achieved during the competition.**

I. INTRODUCTION

If you take a look at the tasks of a human driver, you will notice how complex they are. Starting with the capability to seamlessly perceive the vehicle environment, the driver must be able to analyze the situation, assess developments and to choose and execute an appropriate behavior in a controlled way. To automate the process of driving you have to find and realize solutions for all these problems. For the sake of vehicular comfort, efficiency, and safety, research groups all over the world have worked on building autonomous technical systems that resemble such capability (cf. e.g. [1], [2], [3], [4], [5]).

The DARPA Urban Challenge 2007 has been a competition introduced to expedite mainly US research on autonomous vehicles. Its finals took place on Nov. 3rd, 2007 in Victorville, CA, USA. As in its predecessors, the Grand Challenges 2004 and 2005, the vehicles had to conduct missions fully autonomously and unmanned without any intervention of or interaction with the teams. In contrast to the earlier competitions, the Urban Challenge required operation in 'urban' traffic, i.e. in the presence of other vehicles operated either autonomously themselves or by the organizer. The major challenge imposed was collision-free driving in traffic in compliance with traffic rules (e.g. right of way at intersections) while completing the given missions that included overtaking maneuvers, U-turns, parking, and merging into regular flow of traffic. Finally, recovery strategies had to be demonstrated in deadlock situations or in traffic congestion that cannot solely be handled with traffic rules.

T. Gindele, D. Jagszent and R. Dillmann are with Insitute for Computer Science and Engineering, University of Karlsruhe, D-76128 Karlsruhe, Germany.
{gindele|jagszent|dillmann}@ira.uka.de,
benjamin.pitzer@us.bosch.com

II. SYSTEM OVERVIEW

The basis of the AnnieWAY automobile is a 2006 VW Passat Variant B4 (figure 1). The Passat has been selected for its ability to be easily updated for drive-by-wire use by the manufacturer.

Fig. 1. *AnnieWAY*'s autonomous vehicle

AnnieWAY relies on an off-the-shelf AMD dual-core Opteron multiprocessor PC main computer whose computing power is comparable to a small cluster, yet offers low latencies and high bandwidth for interprocess communication. All sensors connect directly to the main computer which offers enough processing capacity to run almost all software components. The main computer is augmented by a dSpace AutoBox that operates as electronic control unit (ECU) for low-level control algorithms. It directly drives the vehicles actuators. Both computer systems communicate over a 1 Gbit/s Ethernet network. The drive by wire system as well as the car odometry are interfaced via the Controller Area Network (CAN) bus. The DGPS/INS system allows for precise localization and connects to the main computer and to the low-level ECU (AutoBox). The chosen hardware architecture is supported by a real-time-capable software architecture. The main sensor is a Velodyne HDL-64E rotating laser scanner that produces highly accurate 3D scans of the surounding environment with 10 frames per second.

A. Software architecture

The sofware architecture consists mainly of four modules. Like shown in figure 2 the first one is the perception module. It analyses all the sensor data and classifies all seen objects and maps the environment. This data is send to the planner where it is integrated in a global high-level scene description. Based on this description the planer analyses the current situation and choses an appropriate behavior. This behavior

Preprint submitted to 2008 IEEE Intelligent Vehicles Symposium.
Received January 14, 2008.

is then executed and generates a trajetory which is passed to the next module, the low-level-collision-avoidance. Because the trajectory is generated on abstract information it has to be checked for drivability by taking into account the overall environment. If it is probable that the car will hit an obstacle than the collision avoidance plans an alternative trajectory. At the last stage the control module drives the car according to the trajectory.

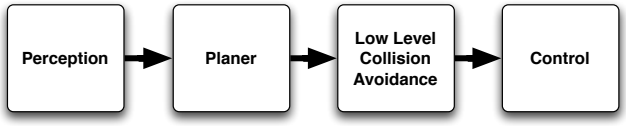

Fig. 2. main modules and information flow

III. REPRESENTATION OF THE ENVIRONMENT

The autonomous vehicle uses an internal world model of its environment to plan its actions. This model is permanently updated and extended as soon as new information becomes available. It consists mainly of a virtual map which contains information about the whole drivable road network. In addition it holds all the knowledge about dynamic and static objects in a close range like other traffic participants and road blockades.

The map is represented as a directed graph, whose edges describe lane segments. Figure 3 shows a comparison between a real road network and the internal representation. The nodes represent points in the real world connected over global coordinates. The nodes form by definition the start and end points of segments and lie in the center of the lanes. Edges feature additional attributes defining e.g. the length, width or type of the lane edge boundaries. Nodes could also have additional attributes to display a stop-line or checkpoint.

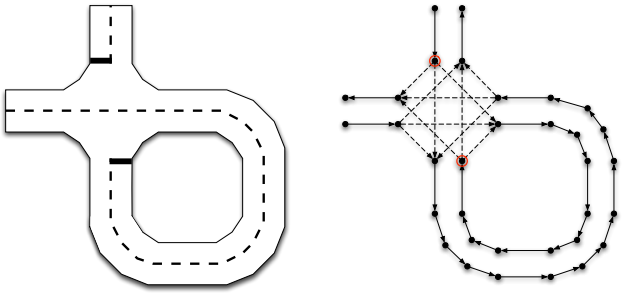

Fig. 3. real road network and internal representation

Edges could have additional annotations to mark special kinds of lane-segments. These are:

- Virtual
- Zone
- LaneChange
- KTurn
- Blocked
- PriorityLane

Virtual lanes designate lanes in intersection areas for which no edge boundaries exist. The *Zone* attribute marks abstract connections in unstructured areas. The *KTurn* attribute marks areas where a k-turn is possible and analogous *LaneChange* where lane changes are possible.

Edges have links to adjacent lanes marking their type of relationship at the same time. Thereby it is easy to look up if a lane has left or right neighboring lanes, oncoming lanes or crossing lanes and which these are. This allows comfortable traversing of the graph at the scene analysis.

The missions that the vehicle has to accomplish consists of a list of checkpoints that has to be passed in the given order. The routes between the checkpoints are not specified hence the mission planning component searches the optimal route in the road network with an A* algorithm [6]. The optimum criterion is defined as shortest travel time. Edges are thereby weighted by their expected travel time. The result is stated as sequence of edges that are annotated with high level maneuvers, like a k-turn or a left turn, etc..

IV. DESIGN OF THE STATE MACHINE

The basic idea of a hierarchical state machine is to design and group the states in the way that each substate is a specialization of its parent state so that only corresponding differences have to be modeled similar to inheritance principle of the object oriented programming paradigm. Thereby the functional redundancy of the states is reduced as well as the amount of transitions needed hence it is easier to create and extend complex state machines.

In the state machine of *AnnieWAY* with exception of some special states, every state represents a behavior. A decision that had an leading impact on the design was to operate the state machine with a fixed cycle time. This makes it easy to make propositions about the runtime and to guarantee real time execution. Because state machines operate event based we created a special event called *EvProcess* that is created with a fixed frequency. Each state implements a reaction on this event. In this routine all relevant aspects of the current scene concerning the actual behavior and the possible transitions are analyzed and a transition to a new state is executed if required.

The state machine was implemented with the C++ programming language and the boost state chart library [7].

A. Overview of the state machine

The state machine is based on three basic modes of operation. These are *Pause*, *Active* and *Error*. The vehicle starts in state *Pause* and goes in state *Active* by sending a start signal. In *Pause* the breaks are activated whereas in *Active* the car acts free to accomplish its mission. It is possible to interrupt the actions of the vehicle at any time by sending a pause signal thereby setting it in state *Pause*. By doing this the current state of *Active* and all its substates are saved and the vehicle instantly starts to break til it stands. If

Preprint submitted to 2008 IEEE Intelligent Vehicles Symposium.
Received January 14, 2008.

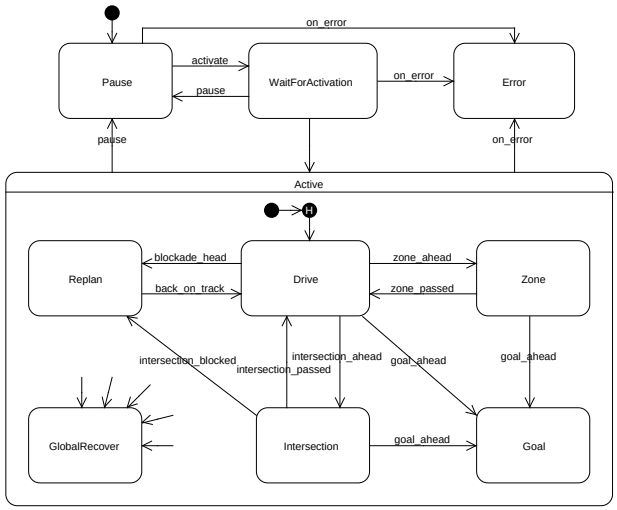

Fig. 4. UML Diagram of the main states

the car gets reactivated later the saved state is restored and the mission directly resumed. If the car gets moved in *Pause* which is detected by the localization module the saved state is discarded and the state machine reinitialized. If an error occurs at any time that can not be handled by the current state the state *Error* gets activated which stops the car in a controlled way and prints out the error message.

The automat is designed in a way that divides movement in three different types which are handled differently. The normal driving on streets is handled by state *Drive*. In intersection areas *Intersection* is active and for all areas with unstructured environments the state *Zone* is used. This state allows the car to navigate in areas beside the road network where no lanes are marked e.g. in parking lots. As soon as the autonomous vehicle reaches its goal it remains in state *Goal*.

Sometimes the car gets in situations where no progress is made, that means that no improvement is measurable by the progress criterion defined in the current state for a longer time. This can happen for two reasons. First the abilities of the state machine are to limited to handle the current situation or second the other cars are not acting in a correct way for example if they simply do not drive. To even accomplish the mission in such cases the state *GlobalRecover* is used but only if all other more specific fallback recover mechanisms have failed. In the global recover mode the module for free navigation is used to plan a path from the current place to the next checkpoint. Thereby no traffic rules and only the most outer lane boundaries are considered. Dynamic obstacles are evaded and static obstacles are circumnavigated. As soon as the checkpoint is reached the state machine changes back to *Drive*. The global recovery state is a dangerous behavior and not useable in its current form for every day road traffic but made sense in the context of the urban challenge because it was more important not to get stuck than breaking a traffic rule.

B. Drive state

The state *Drive* (see Fig. 5) represents driving on a lane. It includes all relevant maneuvers. The start state *DriveStart* analyzes the current situation an branches to the corresponding sub-state. *DriveOnLane* is active during normal driving on a road. *LaneChange* is active on any kind of lane change to ensure a safe merging with moving traffic. *DriveKTurn* coordinates a K-turn. *DriveStop* handles the special case of a stop-line without an intersection. *DriveRecover* tries to bring *AnnieWAY* into a better position if it got stuck.

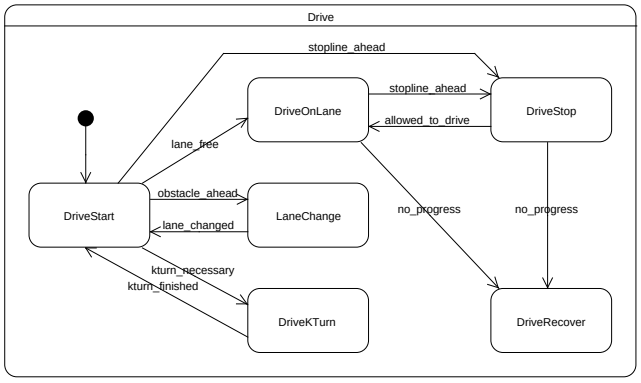

Fig. 5. UML diagram of state *Drive*

1) DriveOnLane state: This state is used for regular driving on roads with no special characteristics. The generated path of the state *DriveOnLane* only depends on the current lane and possible vehicles in front. Thereby the maximal allowed speed and the speed of the vehicle ahead is taken into account.

2) LaneChange state: The state *LaneChange* is responsible for all kinds of lane changes. Overhauling is thereby regarded as a combination of two lane changes. Overall three different types are handled, lane change to the left neighboring lane, to the right and overhauling on the oncoming lane. Figure 6 displays the state diagram with the different states.

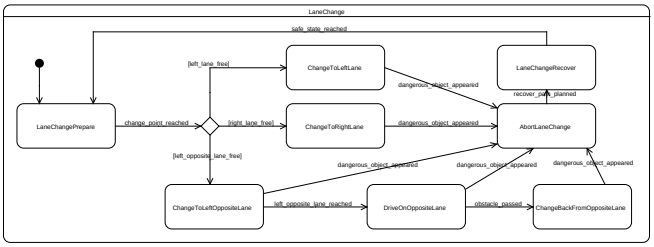

Fig. 6. UML diagram of state *LaneChange*

A lane change can have different causes:

- The mission planning dictates a lane change. This is often the case before intersections when the lanes get separated in straight and turning lanes.

- The current lane is blocked by a obstacle e.g. by a standing vehicle or a road blockage.
- The vehicle ahead drives significantly slower than the own desired velocity.

When a lane change is initiated the state *LaneChangePrepare* is activated. This state is responsible for planning the details of the lane change and to start the execution at the right time. If the direction of the lane change is not dictated by the mission it is analyzed which lanes are applicable. Therefor all adjacent lanes are examined and checked if they are adjacent long enough to finish the change process. This is most relevant for an overhaul operation on the oncoming lane. Additionally it is asserted that the corresponding lane does not cross an intersection during the change and is not blocked. If a left neighboring lane fulfills all these conditions it is preferred over a right one. As a last option a left oncoming lane for an overhaul operation is considered. If no lane is applicable for a lane change and it is mandatory the state *Replan* is activated to find an alternative route otherwise *Drive* is reactivated.

If the lane for the change is chosen the point where the car starts the lane change called change point is calculated based on the distance to the obstacle ahead. Starting at this point a path is planned in shape of a ramp to other lane and the end point is calculated. For the overhaul operation the corresponding points for the change back are estimated analogous. Using this points the area of lane change is defined and estimations of the movement of all relevant vehicles are calculated to test if they conflict with the own car during the change process. Because the autonomous vehicle is still in the approaching phase it stops at the change point if one of the test fails. If the reason for the lane change disappears in this phase, e.g. that the blocking vehicle ahead starts to drive again, the lane change is aborted and vehicle gets back to *Drive*. As soon as the car reaches the change point the state machine does a transition in the corresponding state *ChangeToLeftLane*, *ChangeToRightLane* or *ChangeToLeftOppositeLane*.

As soon as the new lane is reached normal driving in state *Drive* is continued after an adjustment of the mission graph. In case of an overhaul operation on the oncoming lane a maneuver of driving on the opposite line in state *DriveOnOppositeLane* and lane change back on the old lane with *ChangeBackFromOppositeLane* is executed before.

In all phases of a lane change the situation of the autonomous car and its evolvement is assessed for potential conflicts. If an acute danger emerges, e.g. a car appears out of the sight shadow, the state *LaneChangeAbort* is activated which responsible for planning a save abortion of the lane change. Subsequently the state *LaneChangeRecover* executes the the abortion and navigates to the planned retreat point. If for example the autonomous vehicle is starting to change to the opposite lane and suddenly sees an oncoming car it stop the operation, changes back to the old lane and retries to change as soon as the other lane is free again.

3) K-Turn state: U-turns or K-turns were necessary during the *Urban Challenge* in following situations:

- Mission planning schedules a K-turn because it is the best (and possibly only) means to reach a checkpoint. E.g. planning out of a stub road.
- A K-turn needs to be made spontaneous because *AnnieWAY* detected a road blockade.

Only the first situation is handled with the state *DriveKTurn*. Dynamically necessary K-turns are considered a special case of the state *Replan*/*GetBackOnTrack*.

The path planning for K-Turns is done with the path planning component of the zone planner. With its help *AnnieWAY* can handle non-parallel oncoming lanes and obstacles between the lanes. The path planner gets parameterized to only use its circle/tangent/circle heuristic and a B-formed configuration space. With this parameter the zone planner generates a path for a three point turn on the oncoming lane.

For a description of the zone planer and its used algorithms and heuristics see [8].

In the state *DriveKTurn* *AnnieWAY* is stopped, the path start and end points are calculated, the zone planner is parameterized and executed.

4) DriveStop state: Normally stop points belong to an intersection and stopping at those points is handled in the *Intersection* state. But the data structures allow a stop point to be placed anywhere in the road network. The *DriveStop* state handles those stop points by stopping at them and unconditionally go on afterwards.

5) DriveRecover state: There is a tranistion from every sub-state of *Drive* to *DriveRecover* that is executed if *AnnieWAY* does not make progress for a certain amount of time depending on the actual sub-state. E.g. *DriveOnLane*'s timeout is small because a clear passage is assumed when the state machine is in this state. *DriveRecover* uses the zone path planning component to get *AnnieWAY* a certain distance further in the mission.

C. Replan state

It can easily happen that assumptions that are stated during the mission planning process turn out to be wrong in reality. In such cases the autonomous vehicle must be able to replan so that current circumstances can be considered. This is the task of the state *Replan*. The corresponding UML chart is shown in figure 7.

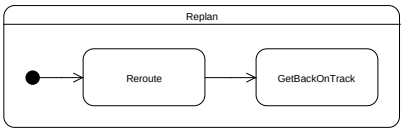

Fig. 7. UML diagram of state *Replan*

If a road blockage is recognized the corresponding lane edge is marked as blocked and therefore can not be traversed during the rerouting process.

The state *Reroute* searches for a new route to absolve the mission depending on the current position and orientation of

the own car. To find the best edge in the graph to continue the mission the eges are iterated according to their distance and direction. The first edge in this order from which an path exists to the next checkpoint in the mission and that is not blocked is used as the beginning of the new route. From there the paths between the checkpoints are planned with the A* Algorithm as described earlier. After rerouting the vehicle is navigated to the beginning of the new mission with the state *GetBackOnTrack*.

It is checked in every state if the car is off the track measered by its distance to the planned route. This situation can emerge if the vehicle missed a lane change or after rerouting. In this case the *GetBackOnTrack* aproximates the best reentry point on the next lane in the mission. The vehicle uses free navigation to reach this point and continues from there in the normal driving mode.

D. Intersection state

In proximity of any kind of intersection the state machine is put into the state *Intersection* (Fig. 8).

IntersectionApproach is the start state for *Intersection*. It remains active until *AnnieWAY* is close to the intersection unless another vehicle is detected in front of *AnnieWAY*. Then *IntersectionQueue* is activated and the state machine remains in it as long as either the vehicle passes the intersection, is not detected anymore, or *AnnieWAY* has reached the intersection.

If *AnnieWAY* is in state *IntersectionApproach* and near the intersection, the state machine branches on the basis of the type of the intersection and the current traffic situation:

(a) *IntersectionStop* if *AnnieWAY* is on a yield road,
(b) *IntersectionPrioDriveInside* if it crosses the intersection on the priority road and no other vehicle on a priority road has precedence over *AnnieWAY*.
(c) *IntersectionPrioStop* if *AnnieWAY* is on the priority road but needs to wait for other traffic. (E.g. left turn crossing the priority road with oncoming traffic)

In case (a) *AnnieWAY* stops at the stop-line and the state is switched to *IntersectionWait*. In this state all vehicles standing at other stop-lines are registered. According to the 4-way-stop rule this vehicles have precedence over *AnnieWAY*. If these vehicles passed the intersection and the moving traffic check (MTC, [9]) concludes that *AnnieWAY* can safely merge into the moving traffic the state is changed to *IntersectionDriveInside* and the intersection is passed.

In case (b) the intersection is passed without further state changes - except if *AnnieWAY* is near the intersection entry, a new vehicle gets detected on the oncoming priority road, and the MTC identifies a possible collision. In this case a state transition to *IntersectionPrioStop* is executed and the situation is handled like in case (c).

In case (c) *IntersectionPrioStop* brings *AnnieWAY* to a full stop at the line of sight. Afterwards state *IntersectionPrioWait* is activated. The state machine stays in this state as long as there are vehicles with right of way and the MTC identifies unsafe situations for pulling in. After waiting the active state is changed to *IntersectionPrioDriveInside* and the situation is handled like in case (b).

IntersectionRecover can be reached from every intersection state. It gets activated when *AnnieWAY* did not made progress for a certain amount of time. This state assesses the current situation and tries to maneuver *AnnieWAY* to a better position. If this recovery is unsuccessful the current lane segment or intersection is annotated as blocked and a transition to state *Replan* is issued.

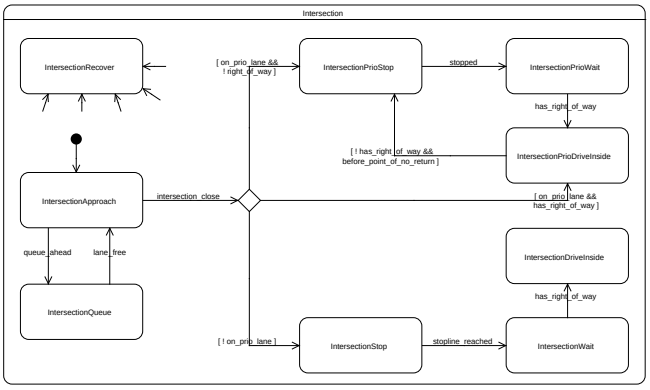

Fig. 8. UML diagram of state *Intersection*

E. Zone state

The *Zone* state handles situation when *AnnieWAY* has to cope with unstructured environment - so called zones. Parking lots or obstacle fields are modeled as zones. Because there are no lanes in zones which the path finding component could leverage there is a special path finding component responsible in zones. This component searches a traversable path in the configuration space with the help of an A* algorithm. It is also responsible for constantly checking the traversability of this path and making adaptions. Used heuristics, algorithms for the target domain are described in [8].

The state machine only controls the path finding component and monitors its progress. It can therefore be modeled on a very high level. Fig. 9 depicts the UML state chart of the *Zone* state.

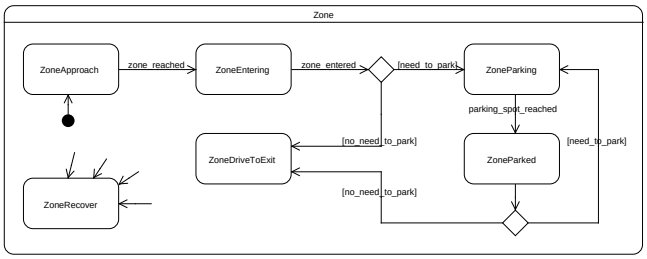

Fig. 9. UML diagram of state *Zone*

When a zone is next in *AnnieWAY*'s mission the state *Zone* gets activated. *ZoneApproach* is its inital state. Here the special path planning component gets initialized while *AnnieWAY* is slowed down so that it enteres the zone at a reasonable slow speed. If the zone is reached, *ZoneEntering* is activated. This state stops *AnnieWAY* and gives the special path finding component the coordinates and direction of the next checkpoint in the mission. This can be a parking spot - where the next state is *ZoneParking*, or the exit of the zone - where the next state is *ZoneDriveToExit*. These two states monitor the progress of the special path planning component and initiate a state transit if the checkpoint is reached. If the state machine was in *ZoneDriveToExit*, it exits the *Zone* state and returns to *Drive*. In the other case the state tranistion goes to *ZoneParked*, where *AnnieWAY* parks for 5 seconds. After parking the state transition are the same as in state *ZoneEntering*.

During driving in a zone the normal safety and recovery mechanisms are not active. The special path planning component takes care of safety and recovery itself. Despite this, the state machine monitors the progress and can skip a checkpoint if the special path planning component fails to recover and gets stuck. Therefore *ZoneRecover* is activated.

V. RESULTS

One of the primary requirements to reach the finals of *DARPA Urban Challenge* was to show that the car drives safely. This means not only to follow the traffic rules but also to drive predictable and appropriate in every situation. For this reason only 11 of originally 89 Teams entered the final which one of them was team *AnnieWAY*.

In the semi-final which only 36 teams reached these qualities were tested. There, Team *AnnieWAY* was able to accomplish the safe conduction of a variety of maneuvers with the presented approach including

- regular driving on lanes
- turning at intersections with oncoming traffic
- lane changing maneuvers
- vehicle following and passing
- following order of precedence at 4-way stops
- merging into moving traffic

AnnieWAY entered the final drove collision-free for several minutes up to a point where it stopped due to a software exception in one of the modules.

Figure V depicts three examples of the vehicles actual course taken from a log-file and superimposed on an aerial image. The rightmost figure shows the stopping position in the final. The competition was won by CMU, followed by the teams of Stanford and Virginia Tech.

VI. CONCLUSIONS AND FUTURE WORKS

We presented the design of the planner component of the autonomous vehicle of Team *AnnieWAY*. The process of behavior decision is solved with a hierarchical finite state machine that enables the vehicle to behave properly in various situations. With this architecture *AnnieWAY* was able to reach the finals of the *DARPA Urban Challenge* 2007.

The main goal for the future is to improve the current state machine design to enable the handling of more complex situations and to possibly see at which point it gets untractable. It is also planned to investigate different approaches for behavior decision and compare them to the current approach.

VII. ACKNOWLEDGMENTS

The authors gratefully acknowledge the contribution of the German collaborative research center "SFB/TR 28 – Cognitive Cars" granted by Deutsche Forschungsgemeinschaft.

REFERENCES

[1] C. Thorpe *et al.*, *Vision and Navigation: The Carnegie Mellon Navlab*. Annual Reviews, 1990.

[2] H. Nagel, W. Enkelmann, and G. Struck, "FhG-Co-Driver: From Map-Guided Automatic Driving by Machine Vision to a Cooperative Driver Support," *Mathematical and Computer Modelling*, vol. 22, no. 4, pp. 185–212, 1995.

[3] U. Franke, D. Gavrila, A. Gern, S. Görzig, R. Janssen, F. Paetzold, and C. Wöhler, "From Door to Door: Principles and Applications of Computer Vision for Driver Assistant Systems," *Intelligent Vehicle Technologies*, pp. 131–188, 2001.

[4] E. Dickmanns, R. Behringer, D. Dickmanns, T. Hildebrandt, M. Maurer, F. Thomanek, and J. Schiehlen, "The seeing passenger car'VaMoRs-P'," *Intelligent Vehicles' 94 Symposium, Proceedings of the*, pp. 68–73, 1994.

[5] M. Bertozzi, A. Broggi, and A. Fascioli, "Vision-based intelligent vehicles: State of the art and perspectives," *Robotics and Autonomous Systems*, vol. 32, no. 1, pp. 1–16, 2000.

[6] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *Systems Science and Cybernetics, IEEE Transactions on*, vol. 4, no. 2, pp. 100–107, July 1968.

[7] "The boost statechart library." [Online]. Available: http://www.boost.org/libs/statechart/doc/index.html

[8] J. Ziegler, M. Werling, , and J. Schröder, "Navigating car-like robots in unstructured environments using an obstacle sensitive cost function," 2008, to be accepted for IV08.

[9] M. Werling, T. Gindele, D. Jagszent, and L. Gröll, "A robust algorithm for handling moving traffic in urban scenarios," 2008, to be accepted for IV08.

Preprint submitted to 2008 IEEE Intelligent Vehicles Symposium.
Received January 14, 2008.