Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Hybrid State System Development for Autonomous Vehicle Control in Urban Scenarios

Arda Kurt* and Ümit Özgüner**

**Electrical and Computer Engineering Department, The Ohio State University, Columbus, OH 43210, USA (Tel: 614-940-9670; e-mail: kurt.12@ osu.edu).*
***Electrical and Computer Engineering Department, The Ohio State University, (e-mail: umit@ece.osu.edu)*

Abstract: This paper analyzes a hybrid-state-system-based controller for an autonomous vehicle in urban traffic and provides development procedures for hybrid-state systems for automatic control. The Ohio-State University Autonomous City Transport utilizes a discrete-state system, based on a finite state machine for high-level decision making and a continuous-state controller for low-level lateral and longitudinal control. The design procedure for the overall hybrid controller involves a series of capability grafts, each improving the ability of the vehicle to handle diverse situations. The design methodology, as demonstrated in a number of development steps, and architecture are capable of handling various urban scenarios, as demonstrated in a June 2007 site visit by Darpa officials.

1. INTRODUCTION

This paper analyzes the control scheme and the design methods that were used to develop Autonomous City Transport (ACT) of The Ohio State University, built to participate in 2007 Darpa Urban Challenge (DUC). The setting of the competition consists of several urban scenarios, for which a Hybrid State System (HSS) was selected to be the suitable control scheme.

The 2007 Darpa Urban Challenge is a natural extension of 2004 and 2005 Darpa Grand Challenge competitions. While the Grand Challenges were based on off-road ground vehicle automation, the Urban Challenge tackles the problems associated with city scenarios for an autonomous vehicle, where the roads, intersections and parking lots are among the possible environments, each with its distinct rules and regulations.

A Hybrid States System consists of two distinct parts, the Discrete State System (DSS), in which the state assumes only a finite and discrete set of values, and the Continuous State System (CSS), where the system state varies continuously.

This type of interaction between systems of various state domains is common in *process control, flexible manufacturing systems* (FMS), etc. as the continuous operation of the plant generates a number of discrete events, which need to be handled by separate procedures in the controller.

The DSS definition as part of the hybrid state system can be considered a subset of Discrete Event Systems (DES), which were studied extensively in (Özveren, 1989), (Ramadge and Wonham, 1989), (Cao and Ho, 1990). The main advantage of restricting the discrete part of the system definition to DSS is that the automaton nature of the discrete system leads to a state-equation representation developed in (Doğruel and Özgüner, 1997a). This representation of the DSS in state equations is helpful both in terms of familiarity and ease of use when investigating the connections of such systems.

As an alternative to the HSS approach, it is also possible to model the systems of various state domains with a general and unifying representation. Both the discrete and continuous systems can be covered in such a singular formulation.

The reason for selecting the HSS approach for the design of OSU-ACT was the natural affinity of the system towards a situation-dependent solution for different scenarios and an underlying continuous model for the vehicle. Modelling examples and analysis tools for the general HSS architecture can be found in (Doğruel and Özgüner, 1997b) and (Passino and Özgüner, 1991).

The main contribution of this paper is in developing a structured design methodology for HSSs and demonstrating the power and flexibility of the HSS approach for automation in complex and largely uncontrolled environments. The system architecture for the OSU-ACT, subsystem definitions, the design procedure and comments on the performance can be found in the following sections.

2. SYSTEM ARCHITECTURE

2.1 The HSS layout

The OSU-ACT was developed for the Darpa Urban Challenge, of which the setting, rules and regulations are highly situational.

The autonomous vehicles are expected to obey the general rules of traffic for roads of varying curvature, lane numbers and directions; intersections with or without stop signs and parking zones without structured lanes of traffic. Each of these situations demand detailed regulations, some of which are stricter than the day-to-day traffic rules a human driver is expected to obey.

In order to be able to capture the situation-dependent nature of the challenge, the controller for OSU-ACT is layered into a Low-level Controller (LC), and a High-level Controller (HC). The "human driver" analogy is helpful in visualizing the distinct layers. The HC is responsible for the conscious-level decisions such as lane changes, obeying the speed limits and handling intersections, while the LC handles the subconscious control of steering to stay in the lane and throttle/brake control to maintain the speed.

This layered structure is comparatively easy to model in a HSS representation. In Fig. 1, the discrete-state nature of the HC is connected to the continuous-state system, which includes the LC, through an interface, $\boldsymbol{\Psi}$ and the events generated in LC is signalled back to HC through the second interface, $\boldsymbol{\Phi}$.

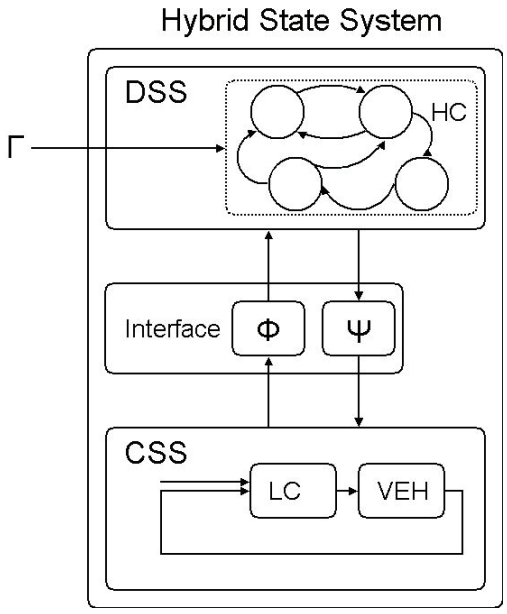

Fig. 1, HSS layout. The HC is in the discrete-state system and the LC is in the continuous-state system.

The external input to the DSS, $\boldsymbol{\Gamma}$, consists of discrete events generated by the sensing and situation analysis system of OSU-ACT. The details of each control subsystem from lowest to highest layer can be found in the following subsections.

2.2 The vehicle model

OSU-ACT is based on a commercially available, hybrid Sport Utility Vehicle (SUV), the image of which with a number of sensors mounted can be seen in Fig. 2. For modelling this front-wheel steering vehicle, widely utilized Dubins' car (Dubins, 1957) is used. This model fits our implementation as the rate of change in yaw ($\boldsymbol{\eta}$) and speed ($\boldsymbol{\varepsilon}$) inputs defined in the model (1) are either readily available, or easy to access in a hybrid vehicle.

Fig. 2, OSU-ACT with front and rear looking LIDARs and GPS antenna visible.

$$\begin{aligned} \dot{x} &= \varepsilon \cos(\theta) \\ \dot{y} &= \varepsilon \sin(\theta) \\ \dot{\theta} &= \eta \end{aligned} \tag{1}$$

In this 2D model, $\boldsymbol{x}$ and $\boldsymbol{y}$ stand for corresponding coordinates of the vehicle, $\boldsymbol{\theta}$ is the yaw and , $\boldsymbol{\eta}$ and $\boldsymbol{\varepsilon}$ are the steering (as the steering wheel position is proportional to the rate of yaw change) and speed inputs as mentioned earlier.

2.2 The low-level controller (LC)

The low-level controller is responsible for steering and speed control of the vehicle. As illustrated in Fig. 3, LC has desired path and velocity as inputs *p* and *v* from higher level, through interface $\boldsymbol{\Psi}$, feedback input from the vehicle in the form of position and orientation $\boldsymbol{x}$,$\boldsymbol{y}$ and $\boldsymbol{\theta}$, and generates control inputs for the vehicle, $\boldsymbol{\eta}$ for steering and $\boldsymbol{\varepsilon}$ for velocity.

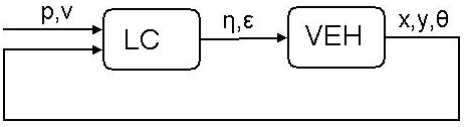

Fig. 3, Input/output connections for the low-level controller.

2.3 The interfaces

There are two subsystems in the interface layer of the HSS. $\boldsymbol{\Phi}$ passes the event signals from continuous-state system to

discrete-state system, and $\boldsymbol{\Psi}$ works in the opposite direction, generating the continuous signals to be used in the CSS.

$\boldsymbol{\Phi}$ is the simpler of two interface subsystems. The discrete events required by the DSS (and the HC within) are either threshold checks such as "*distance to intersection is less than 30 meters*" or command completions generated by LC such as "*the vehicle stopped*". These signals are used in the higher level and the details can be examined in the HC subsection.

The interface from DSS to CSS, $\boldsymbol{\Psi}$, is slightly more complex, as the low-level controller required continuous signals representing velocity and desired path commands.

On the global scale, the route of the vehicle is defined by a sequence of discrete waypoints. On the higher-end, discrete system, the destination at a given time instance is as simple as "***waypoint n***", ***n*** being the index of the current waypoint target in the overall waypoint sequence. The actual continuous path for the LC to follow is generated in $\boldsymbol{\Psi}$, by fitting a spline to a number of these waypoints.

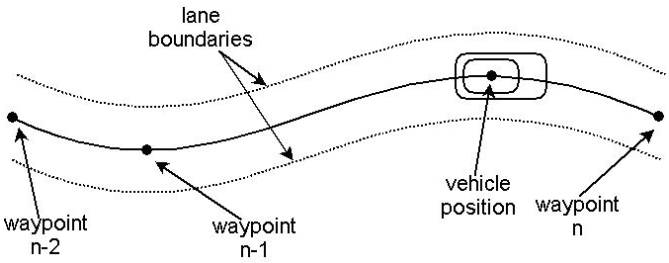

Fig. 5, Spline-based continuous trajectory, generated by interface Ψ.

As seen in Fig. 5, three coordinate pairs from waypoints and the position of the vehicle are used to generate a continuous trajectory for the LC to follow. A Catmull-Rom spline is used to interpolate these points. The Catmull-Rom spline (Catmull and Rom, 1974) was selected among a number of interpolation options, mainly because of the forced nature of the spline that makes the trajectory pass through all the points. This was important in our application, since the missions in DUC are defined as a sequence of special, "must-go-through" waypoints that are called "checkpoints".

2.4 The high-level controller (HC)

As mentioned above, the high-level controller is responsible for emulating the higher-level, conscious decision-making process of a human driver. A number of these decisions are initiating lane changes and passing manoeuvres, speed selection, car following, u-turns, handling the order precedence in an intersection, merging into or crossing moving traffic.

One can observe that this brief list of required capabilities is highly situation dependent. A general control scheme of "*If event **A** happens, check for **C** and **D**; and decide to take action **X** or **Y** accordingly*" is repeatedly employed in the decision-making process of a human driver.

The Finite State Machine (FSM) was the selected method of control that mimics this manner of decision-making. FSMs are scalable and easy to trace by nature and the flexibility of this method has so far proven useful in a number of previous autonomous vehicle applications.

Since the most basic classification of the high-level controller capabilities is dependent on the structure of the environment, a number of *meta-states* are defined to cover these basic situations. Depending on the position of the vehicle and the mission, the high-level controller is in one of these meta-states, the list of which includes the following:

- Mission Start
- Mission End
- Road
- Intersection
- U-Turn
- Parking Zone

This overall list of meta-states are connected in a FSM, as seen in Fig. 6, while each meta-state is a FSM in itself.

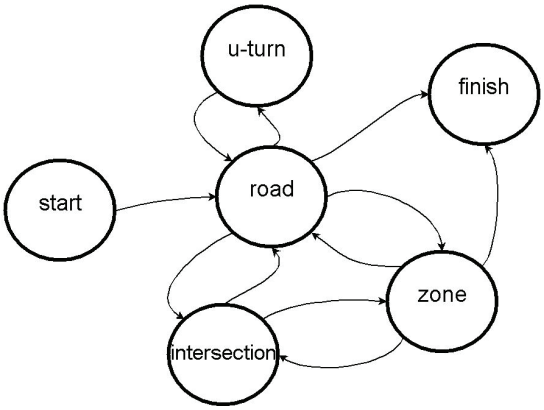

Fig. 6, Meta-state connections.

3. DESIGN PROCEDURE

3.1 Formulation

The DSS part of the controller, based on a FSM as described above, was designed in various stages that correspond to an increasing set of capabilities.

This design approach, which we call "*capability grafting",* relies on the scalable nature of the FSMs, and the final controller is reached by means of a sequence of expansions on top of a core set of capabilities. The overall methodology and terminology is demonstrated in this section, by using the initial stages of development for the "road" meta-state.

The HSS architecture described in 2.1 can be detailed more formally in order to obtain the system equations in (2). The detailed block diagram is in Fig. 7.

$$
\begin{aligned}
X(k+1) &= F(X(k), \Gamma(k), s(k)), \\
x(k+1) &= f(x(k), y(k), S(k)), \\
y(k+1) &= v(x(k+1)), \\
s(k+1) &= \Phi(y(k+1)), \\
S(k+1) &= \Psi(X(k+1)).
\end{aligned} \tag{2}
$$

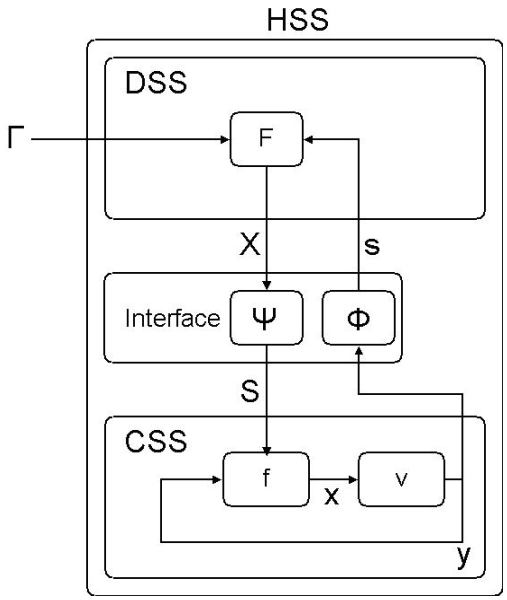

Fig. 7, HSS blocks used in system formulation.

In this formal block diagram, ***F*** is the high-level controller in DSS, ***f*** is the low-level controller part of the CSS, ***v*** is the vehicle model, ***S*** and ***s*** are the control signals generated for discrete-to-continuous and continuous-to-discrete connections, ***Ψ*** and ***Φ*** are the interfaces for these connections as described above, ***X*** and ***x*** are the states of the high and low level controllers and ***y*** is the vehicle state that is fed back to the low-level control and up to the DSS through appropriate interface.

The development stages of block ***F*** and the related design decisions will be the focus of this section, as the design methodologies for the standalone continuous controller have been widely studied.

3.2 Development Stages

A series of snapshots from the development of an example controller for the "road" meta-state, which is an FSM in itself, will be illustrated in this subsection.

The design direction is from a core set of capabilities to a flexible and more capable architecture. The first stage is the most basic capability associated with the road situation, the general waypoint following that can be seen in Fig. 8.

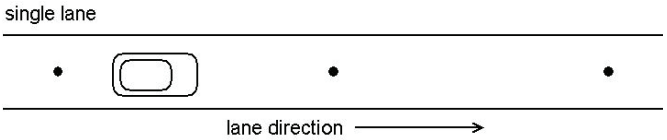

Fig. 8, Stage I: simple waypoint following.

At this stage, a single waypoint following state, with connections into and out of this meta-state is sufficient, as can be seen in Fig. 9. The state-transition triggers, or "*events*", an example of which is ***E1*** in Fig. 9, can either be generated within the HSS and fed into ***F*** in signal ***s***, or come from the sensing and situation analysis systems of the autonomous vehicle as represented in signal ***Γ***.

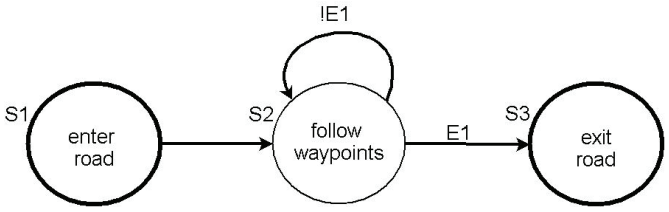

Fig. 9, FSM for stage I. ***E1***: *End of the road is reached*.

The second stage involves adding the capabilities related to external stimuli. The case example here is a blocking obstacle on the followed path. The world model still consists of a single lane, so a generic stop-and-wait capability is required.

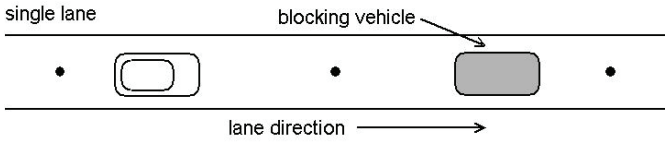

Fig. 10, Stage II: single lane with blocking vehicle.

In Fig. 11, this capability is "*grafted*" into the existing state machine by means of adding two new states and a new event. This type of graft forms an alternate state trajectory. The existing state trajectory of ***S1*** → ***S2*** → ***S3*** is still available in the absence of an obstacle and the new trajectory ***S1*** → ***S2*** → ***S4*** → ***S5*** → ***S3*** works in parallel to the existing system.

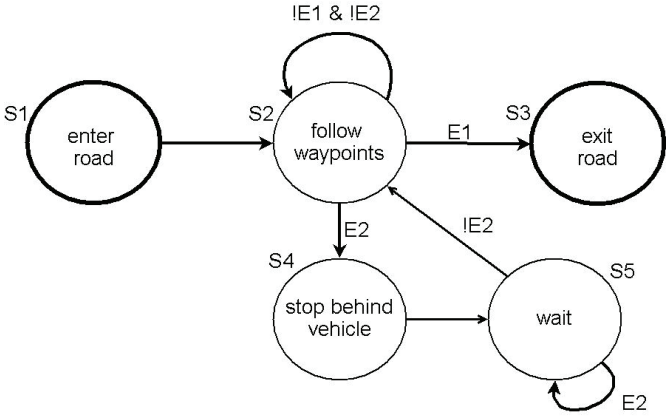

Fig. 11, FSM for stage II. ***E2***: *Current lane is blocked*.

A graft of this type, where the new state trajectory is parallel to the existing ones, is an "*extension*", and can be visualized

as a parallel connection in the block diagram, as seen in Fig. 12. The grafted system, ***F2*** has connections to the existing system and utilizes the same interface to the CSS part of the HSS.

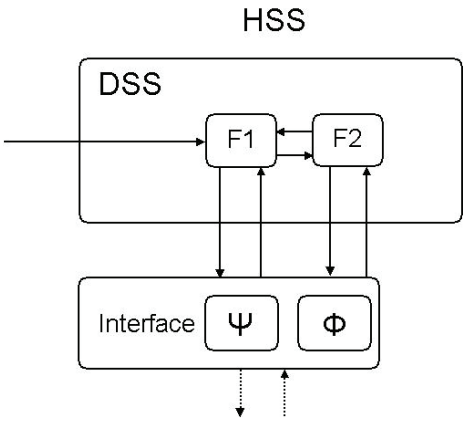

Fig. 12, Parallel connection of an "*extension graft*", ***F2***, to the existing system, ***F1*** at the end of Stage II.

For the next stage, the world model is expanded to include a second lane as seen in Fig. 13, under the strict assumption of an opposing direction of traffic and a solid lane boundary. Under the rule set of DUC, this situation requires a complete stop, followed a lane-change pass around obstacles, if the passing lane is free of obstacles.

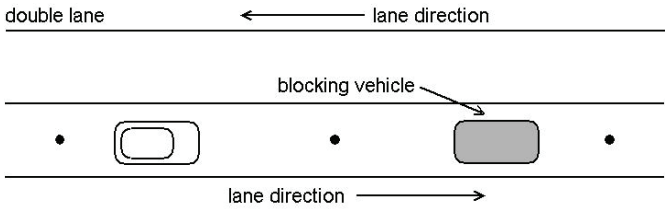

Fig. 13, Stage III: double lane, opposite direction, blocking vehicle.

This new capability is added into the system by a second "*extension graft*" that can be seen in Fig. 14. The new state, ***S6*** forms an alternate trajectory, ***S1→S2→S4→S5→S6→S3***, and connects to the exiting system in parallel as demonstrated in Fig. 15.

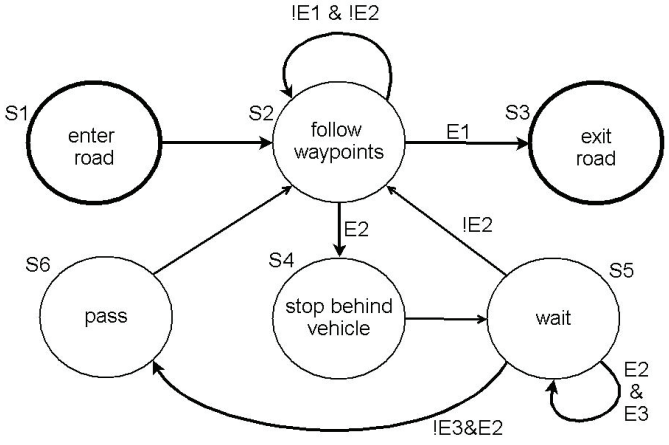

Fig. 14, FSM for stage III. ***E3***: *Passing lane is occupied*.

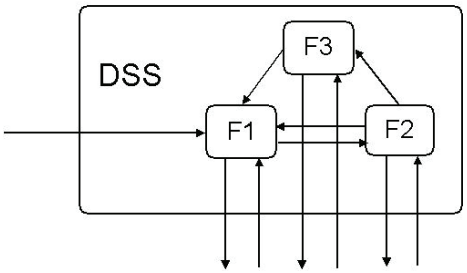

Fig. 15, The extension graft of ***F3*** for Stage III, utilizing same interface to the continuous level.

For the final stage of this example sequence, the assumption on travel direction of the next lane is relaxed, as in Fig. 16. Depending on the direction of travel on the passing lane, or equivalently the lane divider being broken or solid, the autonomous vehicle needs to perform either a complete stop and pass, or a non-stop lane change.

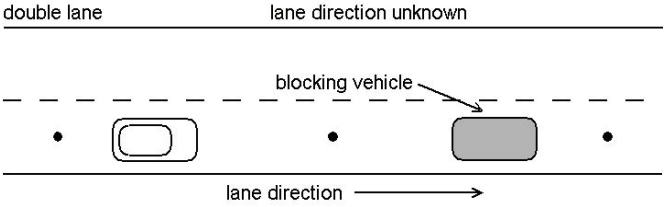

Fig. 16, Stage IV: double lane, next-lane direction unknown, blocking vehicle.

The added capability is handled by the new graft, ***S7***, as displayed in Fig. 17. The connection of this state demonstrates the second type of graft, which we name an "*insertion*". ***S7*** is inserted into an existing state trajectory passing from ***S2*** to ***S4*** and this manner of graft is more akin to a serial connection as it does not preserve all existing trajectories.

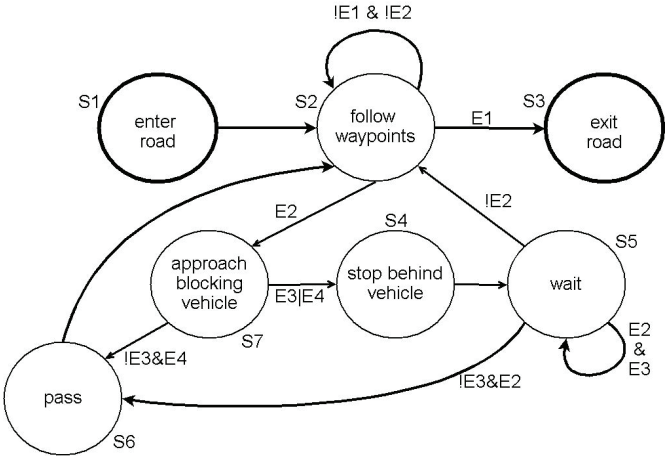

Fig. 17, FSM for stage IV. ***E4***: *Lane divider is solid*.

The "*insertion graft*" can also be seen in Fig. 18, where the new system ***F4***, severed the direct connection of ***F1*** and ***F2***.

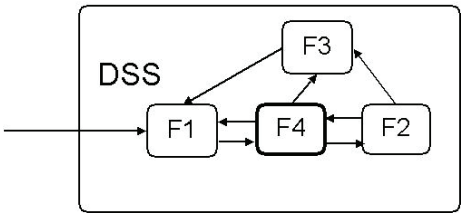

Fig. 18, "Insertion graft" of ***F4***. The direct route from ***F1*** to ***F2*** is destroyed.

The major difference between the two types of grafts is that the *extension graft* preserves existing state trajectories, so any failure or removal of the grafted system results in a working system. On the other hand, *insertion grafts* are placed in serial connection, severing existing state trajectories. Hence, the malfunction or removal of an insertion graft creates unreachable subsystems, which in turn impacts the reachability and stability of the overall system, as described in (Doğruel and Özgüner, 1997a).

However, it might be more problematic to implement new capabilities solely with extensions in order to avoid aforementioned insertion risks, since a parallel-connected system to handle the same capability generally contains a higher number of states and events. So, both types of grafts are needed in this design methodology and it is important to note the particular risks of each.

4. SYSTEM PERFORMANCE

As of September 2007, OSU-ACT, with the above-described, HSS-based controller, is selected to be among 36 semi-finalists for the 2007 Darpa Urban Challenge.

During the elimination stages for the initial group of 89, the most recent tests included a site visit by the Darpa officials. During this site visit, OSU-ACT was asked to perform a series of tasks, which required a subset of the final set of capabilities to be functional. The performance of the vehicle was found to be satisfactory by Darpa reviewers, which led to the OSU team getting into the next stage of the DUC. Following is a sample list of capabilities, implemented in a series of insertion and extension capability grafts as described above and successfully demonstrated on the site visit:

- Route planning for a given mission,
- Waypoint following without crossing lane boundaries,
- Performing u-turn manoeuvres within specified boundaries,
- Performing passing manoeuvres by changing lanes around stopped vehicles,
- Handling a queue formations at an intersection,
- Establishing an order of precedence at an intersection and obeying this order,

5. CONCLUSIONS

The hybrid-state system implementation for controlling an autonomous vehicle in Darpa Urban Challenge was analyzed and a flexible design methodology was formulated in this paper.

The controller is divided into a continuous-state system that handles the low-level control, and a discrete-state system that uses a FSM, segmented into meta-states, for high-level control. These layers of the HSS are connected via continuous-to-discrete and discrete-to-continuous interfaces.

The design stages involve a sequence of expansions called "*capability grafts*", the two types of which were named "*extension grafts*" and "*insertion grafts*".

This implementation was tested and found to be successful during a number of events, latest of which was run by Darpa officials. The autonomous vehicle is currently under further development for the national qualification event and the final race of the Darpa Urban Challenge.

6. ACKNOWLEDGEMENTS

The authors of this paper are grateful for discussions and help of the OSU-ACT Team, especially Dr. Keith Redmill.

REFERENCES

Catmull, E. E. and Rom, R.J. (1974) A class of local interpolating splines. In: *Computer Aided Geometric Design*, (Barnhill, R.E. and Riesenfeld, R.F (Ed)) Academic Press, Orlando, 317—326.

Cao X.R. and Ho Y.C. (1990). Models of Discrete Event Dynamic Systems. *IEEE Control Systems Magazine*, June.

Doğruel, M. and Özgüner, Ü. (1997a). Discrete and Hybrid State System Modeling and Analysis. *Turkish Journal of Electrical Engineering and Computer.* **5**, no. 2, 263—286.

Doğruel, M. and Özgüner, Ü. (1997b). Stability of a Set of Matrices with Applications to Automatic Control. *Turkish Journal of Electrical Engineering and Computer.* **5/2**, 247—262

Dubins, L.E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics.* **79/3**, 497—516.

Özveren, C.M. (1989). Analysis and Control of Discrete Event Dynamic Systems: A State Space Approach, *Ph.D. Thesis*, Massachusetts Institute of Technology.

Passino, K.M. and Özgüner, Ü. (1991). Modeling and Analysis of Hybrid Systems: Examples. *Proceedings of the IEEE International Symposium on Intelligent Control.* 251—256.

Ramadge, P.J.G. and Wonham, W.M. (1989). The Control of Discrete Event Systems. *Proceedings of the IEEE*, **77**, 81—98.