Hybrid-State System Modelling for Control, Estimation and Prediction in Vehicular Autonomy

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Arda Kurt, B.S., M.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2012

Dissertation Committee:

Ümit Özgüner, Advisor
Ashok Krishnamurthy
Keith A. Redmill
Abstract

This thesis studies the Hybrid-State System models and their properties for different pieces of the urban autonomy problem. For autonomous vehicles that operate in real-life, mixed-mode traffic, a number of parallels between the human-driven system and the autonomous counterpart can be identified and captured in the hybrid-state system setting. For the control subproblem of the urban autonomy, this thesis proposes a system architecture, related design approaches for autonomous mobile systems and guidelines for self-sufficient operation. Development of a tiered layout for a hybrid-state control in a series of stages as well as the integration of such a controller in the overall autonomy structure are proposed and demonstrated as part of multiple examples, including The Ohio State University participation in Defense Advanced Research Projects Agency Urban Challenge 2007. The hierarchical layout and the iterative design methodology enable design flexibility through compartmentalization of the overall system and helps prepare for various contingencies, as illustrated on specific development cycles. The sensing and perception part of the autonomy implementation relies on a probabilistic hybrid-state system modelling method that is developed for driver-behavior analysis and prediction. The model fits into and captures the central modules of the existing Human Driver Model. The stochastic models, based on the observable actions of the driver/vehicle interaction, are useful in representing the behavior of human-driven vehicles in certain urban decision-making
scenarios. The Driver Intention Estimator presented utilizes the developed stochastic models to detect and predict high-level, abstract decisions of observed drivers through traffic scenarios and it can be expanded to form scenario safety estimation tools as demonstrated. As for the analysis of the developed estimators and as a useful tool for hybrid-state systems in general, this study develops an encoding scheme for discrete-state systems as part of a hybrid-state hierarchy. The codes are command-based, in the sense that the interactions of the discrete states with the continuous states are exploited to attach significance to what each discrete state does to the continuous subsystem. The resultant codeset is independent of how the discrete-state transitions are designed and conventional binary tools such as truth tables and K-maps are easily applicable in the binary representation of the codes. Code-based representation of every possible combination of commands/behaviors governed by the discrete subsystem is useful in a number of design scenarios, an example of which is the generation of a consistent norm for discrete states. Such a norm is demonstrated to be useful in hybrid-state estimation.
To Fügen and Olcayto Kurt, without whom I would not be here. And to Burçak, for being my strength and my joy through all this.
Acknowledgments

I thank all the members of the Control and Intelligent Transportation Research and Cyber-Physical Systems Groups, past and present; especially Dr. Keith Redmill and Mr. Scott Biddlestone for their insight, help and understanding over the years; and Dr. Ashok Krishnamurthy and Mr. Vijay Gadepally for the discussions we had.

The studies represented in this dissertation were partially supported by the National Science Foundation under the Cyber-Physical Systems (CPS) Program (ECCS-0931669), and partially by Honda R&D of Americas, Inc.
Vita

April 29, 1980 .......................... Born - Izmir, Turkey

2003 ........................................ B.S. Electrical Engineering,
Bilkent University

2005 ........................................ M.S. Electrical Engineering,
Bilkent University

2005-present ............................. Graduate Research Associate,
The Ohio State University.

Publications

Research Publications


Fields of Study

Major Field: Electrical and Computer Engineering
# Table of Contents

<table>
<thead>
<tr>
<th>Abstract</th>
<th>ii</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dedication</td>
<td>iv</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>v</td>
</tr>
<tr>
<td>Vita</td>
<td>vi</td>
</tr>
<tr>
<td>List of Tables</td>
<td>x</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xii</td>
</tr>
<tr>
<td>1. Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2. Hybrid-State Systems for Autonomous Mobile-Agent Control</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Hybrid-State System Definition</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Autonomous Vehicle Architecture</td>
<td>9</td>
</tr>
<tr>
<td>2.2.1 Modules and Connections</td>
<td>10</td>
</tr>
<tr>
<td>2.2.2 Hybrid-State System Layout of Controllers</td>
<td>11</td>
</tr>
<tr>
<td>2.3 Finite-State Machine Development for Hybrid-State System Controllers</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Scenarios, Meta-states and Completeness</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Capability Grafting</td>
<td>20</td>
</tr>
<tr>
<td>2.3.3 Equivalence Classes of Discrete States</td>
<td>23</td>
</tr>
<tr>
<td>2.3.4 Properties of Graft Types</td>
<td>27</td>
</tr>
<tr>
<td>2.4 Autonomous Mobile Agent Examples</td>
<td>30</td>
</tr>
<tr>
<td>2.4.1 OSU-ACT and the Meta-State Machine</td>
<td>30</td>
</tr>
<tr>
<td>2.4.2 Development Example for the Grafting Process</td>
<td>33</td>
</tr>
<tr>
<td>2.4.3 MAGIC 2010 Capabilities</td>
<td>41</td>
</tr>
</tbody>
</table>
3. Probabilistic Modeling of Mobile Agents for Estimation

3.1 Autonomy in Mixed-Mode Traffic

3.1.1 Human Driver Model

3.1.2 Hybrid-State System Model for Driver/Vehicle Interaction

3.1.3 Decision-Making

3.1.4 State Selection

3.2 Hybrid-State Estimation and Detection

3.2.1 Estimator Architecture

3.2.2 Discrete-State Detection

3.3 Stochasticity of Control Decisions

3.3.1 State-Transition Probabilities

3.3.2 Decision Probabilities as Functions of Risk

3.3.3 Probability Assignment and Updates

3.4 Example Sets of Driving Decisions in Urban Traffic

4. Discrete-State Encoding in Hierarchical Hybrid-State Systems

4.1 Finite-State Machine Encoding and Norms over Hybrid-State Domains

4.1.1 Hybrid-State Systems

4.2 Command-Based Codes

4.2.1 Definitions

4.2.2 Encoding Scheme

4.3 State Encoding Examples

4.3.1 Hybrid-State Controller Development and Representation in Codes

4.3.2 Performance Metric for Hybrid-State System Estimator Using Code-Based Norm

5. Contributions and Future Work

Bibliography
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Equivalence classes for discrete states in Figure 2.16.</td>
<td>35</td>
</tr>
<tr>
<td>2.2 Equivalence classes for discrete states in Figure 2.18.</td>
<td>37</td>
</tr>
<tr>
<td>2.3 Equivalence classes for discrete states in Figure 2.21.</td>
<td>39</td>
</tr>
<tr>
<td>2.4 Equivalence classes for discrete states in Figure 2.24.</td>
<td>41</td>
</tr>
<tr>
<td>3.1 State transition conditions for the lane-change FSM model.</td>
<td>69</td>
</tr>
<tr>
<td>3.2 Dynamic probability update conditions and corresponding change directions for the lane-change FSM model.</td>
<td>72</td>
</tr>
<tr>
<td>3.3 Initial probability assignments with end-to-end probabilities equalized.</td>
<td>74</td>
</tr>
<tr>
<td>3.4 Initial probability assignments with statistical analysis. Differences are marked bold.</td>
<td>75</td>
</tr>
<tr>
<td>3.5 Combination of intentions for a vehicle pair, based on predicted intentions (Stop, Go through, Left turn, Right turn) and approach direction. Unsafe combinations are marked with U, while safe ones are marked S.</td>
<td>84</td>
</tr>
<tr>
<td>4.1 State codes for ACC example</td>
<td>104</td>
</tr>
<tr>
<td>4.2 Discrete-to-continuous interface.</td>
<td>104</td>
</tr>
<tr>
<td>4.3 ACC example state-transition table with command-based coding of discrete states.</td>
<td>105</td>
</tr>
</tbody>
</table>
4.4 Command blocks and possible behaviors for each controllable continuous state in the FSM-based driver model. ............................... 108

4.5 Code assignments for each state in the HSS driver/vehicle estimator FSM. ................................................................. 109
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Hybrid-State System representation.</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Meta states, substates, low-level controller and the mobile robot.</td>
<td>11</td>
</tr>
<tr>
<td>2.3</td>
<td>Hybrid-State System with system modules.</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>Low-level controller connection in the continuous-state feedback loop.</td>
<td>13</td>
</tr>
<tr>
<td>2.5</td>
<td>Meta states and internal sub-state machines.</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>Simple state machine with three states.</td>
<td>20</td>
</tr>
<tr>
<td>2.7</td>
<td>New state machine after STATE D is grafted as an <em>expansion</em>.</td>
<td>21</td>
</tr>
<tr>
<td>2.8</td>
<td>Alternate state machine after STATE D is grafted as an <em>insertion</em>.</td>
<td>22</td>
</tr>
<tr>
<td>2.9</td>
<td>Equivalence classes of discrete states according to the equivalence relation $\sim_{in}$. $S_1 \sim_{in} S_5$ and $S_3 \sim_{in} S_4$.</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>Equivalence classes of discrete states according to the equivalence relation $\sim_{out}$. $S_1 \sim_{out} S_2$ and $S_3 \sim_{out} S_4$.</td>
<td>25</td>
</tr>
<tr>
<td>2.11</td>
<td>Equivalence classes of discrete states according to the equivalence relations $\sim_{in}$ and $\sim_{out}$ (a) before any graft operations, (b) after an insertion graft and (c) after an expansion graft.</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>OSU-ACT at DARPA Urban Challenge 2007.</td>
<td>31</td>
</tr>
<tr>
<td>2.13</td>
<td>OSU-ACT Meta-state connections.</td>
<td>32</td>
</tr>
</tbody>
</table>
2.14 The indoor testbed at The Ohio State University Control and Intelligent Transportation Research Laboratory.

2.15 First Stage. No obstacles, single lane, waypoint following.

2.16 Road meta state at Stage I. Event E1 stands for reaching the end of the road.

2.17 Second Stage. Lane can be blocked.

2.18 Road meta state at Stage II. Event E2 is detecting the current lane being blocked.

2.19 “Expansion graft” of F2 onto F1 in block diagram.

2.20 Third Stage. Lane can be blocked, there is a second lane on the left with the opposite direction. Corresponding FSM is in Fig. 2.21.

2.21 Road meta state at Stage III. Event E3 is having the passing lane occupied.

2.22 The new block, F3 is “expanded” over the previous stage, connected to both F1 and F2.

2.23 Fourth stage. Lane can be blocked, second lane on the left is in unknown direction.

2.24 Road meta state at Stage IV. Event E4 is having a solid lane divider, which means one needs to stop before passing.

2.25 F4 is “inserted” between existing blocks. F1-F2 direct connection is severed. Also an alternate state trajectory is formed, making the graft a “hybrid.”

2.26 (a) “Peri” UGV, built for MAGIC 2010; (b) Pioneer 3-AT mobile robots at OSU CITR; (c) Robots during a mission at CITR testbed.

2.27 (a) “Static OOI” meta state for disruptor robots before the graft; (b) “Static OOI” meta state after the line-of-sight capabilities were grafted.
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Agents of differing capabilities, communicating at different levels, all connected to a single situational awareness module.</td>
</tr>
<tr>
<td>3.2</td>
<td>General architecture of Human Driver Model.</td>
</tr>
<tr>
<td>3.3</td>
<td>Simplified FSM model and transition probabilities for driver decisions, where an approaching driver has two possible choices.</td>
</tr>
<tr>
<td>3.4</td>
<td>Module connections for driver-state estimation and prediction.</td>
</tr>
<tr>
<td>3.5</td>
<td>Trellis representation for the simplified, four-state FSM model.</td>
</tr>
<tr>
<td>3.6</td>
<td>Seven-state FSM model for driver decisions and end-to-end paths for state transitions.</td>
</tr>
<tr>
<td>3.7</td>
<td>Two-lane road with the tracked (red) approaching a leader-follower pair (green) on the adjacent lane.</td>
</tr>
<tr>
<td>3.8</td>
<td>FSM representation of the merge behavior of the approaching vehicle.</td>
</tr>
<tr>
<td>3.9</td>
<td>Trellis representation for lane-change/gap-acceptance driver model.</td>
</tr>
<tr>
<td>3.10</td>
<td>Target (tracked) vehicle in blue, approaching an intersection with four possible choices, stop, go through, turn left and turn right. The host (observing) vehicle is marked in red.</td>
</tr>
<tr>
<td>3.11</td>
<td>The Finite State Machine model for driver behavior at intersection approach.</td>
</tr>
<tr>
<td>3.12</td>
<td>Continuous states (distance to the intersection, velocity), detected discrete state and predicted final state for a vehicle approaching an intersection and making a left turn.</td>
</tr>
<tr>
<td>3.13</td>
<td>Continuous states (distance to the intersection, velocity), detected discrete state and predicted final state for a vehicle approaching an intersection and going through straight.</td>
</tr>
<tr>
<td>3.14</td>
<td>Overall system and connections of modules for the unsafe condition estimator.</td>
</tr>
<tr>
<td>3.15</td>
<td>Map database viewer for detailed data.</td>
</tr>
</tbody>
</table>
3.16 Intersection detail from the map database. ........................................ 81

3.17 Scenario Safety Estimator flow-chart for two-vehicle intersection approach. .................................................................................. 82

3.18 Example case for parallel approach, tracked vehicle is going straight through the intersection at 45mph, host vehicle is turning left. ....... 85

3.19 Example case for parallel approach, tracked vehicle is slowing down from 45mph and stopping, host vehicle is turning left. ............... 86

3.20 One vehicle (target) is going at 45mph, straight through the intersection, the other (host) is turning left. ................................. 87

3.21 Software for intention and unsafe condition estimators. FSM display of the GUI is shown. .............................................................. 89

4.1 Two dimensional discrete-state space spanned by two controllable substates of $2^m$ and $2^{n-m}$ commands. ................................. 97

4.2 Discrete states for simple ACC example. ........................................... 102

4.3 Input bits for simple ACC example. ................................................. 103

4.4 FSM model for driver/vehicle estimation for intersection approach. ........ 107
Chapter 1: Introduction

Autonomous and semi-autonomous vehicles, or cars that partially or fully drive themselves, are leaving the domain of science fiction and becoming reality as we observe. Partial autonomy solutions such as lane assistance and automatic lane keeping[32], advanced cruise control[43] and automatic parallel parking[46] are already available as optional accessories for commercially available automobiles. The study of high-level vehicle behavior as more and more vehicles are partially, and eventually fully computer-controlled is an important part of Intelligent Transportation Systems (ITS) research.

In this dissertation, the main focus is a particular type of system that is encountered throughout ITS research. Hybrid-State Systems (HSS) capture the interaction between a higher-level mode changer or decision maker and the continuous-state plant controlled in a compartmentalized way. The use of HSS in autonomous control and estimation for traffic scenarios form the two main chapters of this work. The third chapter develops a mapping/encoding method for analysis of HSS for both control and estimation purposes and is a supplement to the first two chapters.

In Chapter 2, a hierarchical control structure for autonomous mobile systems is proposed, a methodical approach to capability additions or “grafts” to such systems
is analyzed and the place of such a controller in the broader system architecture is discussed.

The proposed architecture and design guidelines are generic to the extent that they are applicable to a range of cyber-physical systems, but they will be demonstrated on specific implementations including tactical Unmanned Ground Vehicle control in a cooperative setting and vehicular autonomy, all of which were developed in The Control and Intelligent Transportations Research Lab of The Ohio State University. For specific low-level details of implementation on certain platforms, one can refer to studies with that level of focus. For instance, in [42], Moore et al. investigate the use of various software modules provided by National Institute of Standards and Technology Real-Time Control System (RCS) library for intelligent control and provide design examples, including an automated highway system.

The main focus of Chapter 2 will be on the design and development approaches for the Discrete Event System (DES) portion of the HSS layout. This study can be traced back to [15], where the authors compared different representations of the same event-driven dynamical system, including Markov chains, Petri nets, automata and finite-state machines. In [48], the control-theoretic concepts such as controllability and observability were applied to DES and [45] presented the notions of stability and stabilization through dynamic feedback. Earlier HSS studies on both the formulation [20, 21], where the DES concepts mentioned above were expanded to the hybrid domain of HSS and application [47], where simple examples were used to demonstrate the flexibility of the formulation structure, can be referred for further details. Earlier hierarchical design guidelines on vehicular autonomy [44], in which the authors explore the supervisory control in HSS structures for automated highway systems,
also formed a basis for this study. The main contribution of this particular study is providing step-by-step development guidelines towards a hierarchy of situations over a hierarchy of controllers for autonomous vehicles.

In Section 2.2, the general architecture of the modules responsible for the control branch of the autonomy, their connections and tasks associated with each module are discussed. The reasoning and analogies behind the hierarchical structure can also be found in this section.

Meta states, selection criteria, development approaches for FSM design and related definitions and analysis are demonstrated in Section 2.3. The “capability grafting” concept is introduced and a number of rules and important distinctions are established.

Section 2.4 gives an in-depth description and analysis of the high-level decision-making process at a behavioral action-reaction level and the associated mechanisms and structures, studied on various examples ranging from Ohio State University - Autonomous City Transport (OSU-ACT) implementation to mobile-robot control.

As mentioned above, mixed-mode traffic, where autonomous, semi-autonomous and human-driven vehicles interact, is an increasingly important arena, in which a sizable portion of the Intelligent Transportation Systems applications need to operate as more driver-assistance and semi-autonomous systems are deployed in existing infrastructure and driver demography. The human drivers and their behavior need to be detected, interpreted and even predicted by the machine intelligence in order to make sense of the multi-agent, large-scale, cyber-physical environment.
Human-driver modeling and driver-vehicle interaction studies focused on a number of different approaches. In [30], the authors compare the psychological risk-vs-reward processes for driver acceptance in Advanced Cruise Control (ACC) for varying individual characteristics and perceptions. In [13] model-tracking and parameter-identification were studied that aim at describing one particular decision in the single-lane car-following problem in detail. And decision-tree methods for specific maneuvers such as gap acceptance were investigated in [3], where the specifics of the lane-change gap-acceptance decision were broken into three steps, consisting of consideration of lane change, selection of the lane and search for a suitable gap. The particulars of the cited gap-acceptance study helped with the development of the third chapter, in which we take a stochastic approach to a limited set of driver decisions.

Liu and Özgüner presented a modular Human Driver Model (HDM) in [38], which is further described in Chapter 3. This study itself was based on the COSMOS DRIVE (Cognitive Simulation Model of the Driver) model by the French Institute for Transportation Research [35], where the behavior modules and perception modules of the cognitive process representation were separated, and the continuing cognitive studies at PATH (Partners for Advanced Transportation TecHnology) [53, 19], focusing on building human-driver models to compare with automated control. Later studies of the same model focused on actual vehicles as test-beds to capture the specific cyber-physical interactions for data-acquisition and parameter identification [2] to help with active-safety-system development.

In Chapter 3, the parallels between the HDM and the HSS model are investigated and the analysis of driver-behavior leads to the development of decision or intention estimators usable in mixed-mode traffic scenario evaluation and situational awareness.
The observable outcomes of the driver decisions, when modeled as a Hybrid-State System (HSS), are used to drive a number of estimators and predictors to capture the intention of the observed driver.

The stochasticity of certain portions of the Human Drive Model developed in Chapter 3 lead into probabilistic analysis of prediction of behavior that are particularly useful in mixed-traffic autonomy and driver-assistance systems. The ability to recognize, avoid or mitigate the threat of a traffic accident by modeling what is probable can lead to lives saved as a significant portion (21%) of fatal intersection crashes in the United States were due to not realizing other drivers running stop signs, or failing to yield to crossing main road traffic (an additional 23%), according to a National Highway Traffic Safety Administration (NHTSA) study[14].

Driver assistance and semi-autonomy applications can help in limited-visibility, impaired-judgment or inattentive driver scenarios by augmenting the information available to the driver. It is important to filter, process and summarize the available sensory data presented to the driver in order to avoid information overload, so the machine needs to be able to make sense of the scenario and deduce the safety level in the near future to help prevent/reduce collisions. Two specific examples considered in the final section of Chapter 3 (Section 3.4) are lane changes under limited visibility and intersections where the decisions of other vehicles are significant.

The Human Driver Model, the stochasticity of the decision-making process in the model, the estimator architecture to capture and predict behavior and implementation examples on the overall system are presented in Sections 3.1, 3.2 and 3.4.

The reachability, stability or formal verification analysis of the Hybrid-State Systems are not straightforward processes as their continuous-state counterparts, due to
the hybrid domain of the states and different mechanisms governing state transitions in each domain. In [10], the authors proved that the stability and controllability problems for even simple classes of nonlinear systems in general were either undecidable or computationally intractable. Algorithmic approaches [4] to define the hybrid automata in a certain language and to explore the reachability problem gained traction for formal verification of hybrid systems, yet most of the developed software tools are limited to certain restrictions of the overall hybrid automata definition such as piecewise linearity.

One important tool that is useful for this type of analysis is the definition of a norm between arbitrary discrete states; since a norm, or a distance measure is crucial in definitions of attractivity, invariant subspaces and stability. So far, DSS or HSS norm definitions [20] in the literature have been based on the graph topology of the finite-state/sequential automata that governs the DSS portion of the HSS. A state-encoding method that is independent of the interconnection of the discrete states is developed in this study for the specific purpose of defining a new norm over the discrete states.

An encoding scheme, in its simplest definition is a one-to-one mapping between the finite states of an FSM and a finite set of integers, usually represented in binary. Armstrong in [5], compared a certain subset of these assignments, or mappings called “pr mappings” in terms of efficiency. The method proposed in Chapter 4 starts with numerical encoding of the discrete states based on the CSS interactions and does not use the state-machine transitions. The generated codeset is demonstrated to be useful to form a norm over the discrete-states.
Code-assignment schemes for the internal states of finite-state automata can be traced back to early studies [28], which focused on comparing different methods of code assignment for the purposes of optimality in logic implementation. Different code assignments like Gray Code[27] or Johnson encoding[33], where the codes for two successive states differ only in one bit, try to minimize the number of bits changing during state transitions, while performance-based coding schemes like One-Hot coding aim to maximize the speed of the implementation on FPGAs by assigning as many bits to the code as there are discrete states. While these studies dealt mainly with FSMs in isolation, the hybrid-state hierarchy and the DSS-CSS interaction is the focus of this study.

In Sections 4.1 and 4.2, the authors will describe norms over hybrid-state system hierarchy used in this study and the developed code-assignment methodology based on the commands in the overall hierarchy and interconnections. The methods will be demonstrated in Section 4.3 on two specific examples based on autonomous vehicles and driver/vehicle modelling and estimation.

Finally, Chapter 5 lists the contributions of this dissertation and discusses the potential future extensions on the work presented.
Chapter 2: Hybrid-State Systems for Autonomous Mobile-Agent Control

2.1 Hybrid-State System Definition

A Hybrid-State System (HSS), as used in this study, consists of the combination of a continuous-state system (CSS) and a discrete-state system (DSS), interacting through a number of interfaces and signals, a simple representation of which can be seen in Figure 2.1. Most generally, the discrete-state system is driven by events and operate on abstract, discrete modes; while the continuous-state system operates on continuous signals, states and trajectories. Most mode-switching controllers of continuous plants have a combined state of the discrete mode and the continuous plant state, forming a hybrid state for the overall system. In more specific examples, a human-driven automobile, with the position and velocity of the vehicle in continuous domains and the discrete decisions of the driver, is the fundamental hybrid-state system that most vehicular autonomy attempts are based on.

In the systems that are encountered in mobile and vehicular autonomy, the interaction between the two systems is hierarchical, leading to the discrete-state system manipulating certain aspects of the continuous-state system.
2.2 Autonomous Vehicle Architecture

In observing human directed motion of mobile, physical systems (cars on the road, lawn-movers, remote-controlled robots, etc.) it is clear that a number of functions handled by the driver are hierarchically at a higher level than others. Conscious-level decisions such as which road to take, what speed to maintain or which way to dodge an obstacle are inherently at a higher level in the hierarchy of the overall thought and action process. This division of behavior is observed to be common in a wide array of cyber-physical systems where a physical system interacts with the abstract behavior of a controller, estimator or a human being.

On the lower portions of the hierarchy, semi or subconscious decision or actions such as steering corrections to keep the vehicle in a lane or maintaining the velocity have a more transparent and “learned, but not always thought-about consciously” quality.
This apparent hierarchy lends itself to a tiered architecture in mobile robot controller design. The proposed hierarchical organization was investigated in [1], in which the authors utilized the physical structure of the system, rather than its functionality. The modular architecture was implemented for a number of preceding autonomous vehicles and robots, including automated highway system applications [49], the off-road autonomous navigation competitions DARPA Grand Challenge 2004 [17] and DARPA Grand Challenge 2005 [18, 50], as well as hybrid estimator applications [37] for driver-assistance systems to avoid fatal intersection crashes, all of which following this design philosophy.

Similar to the human driver analogy, another human-operated system, a naval vessel, represents a higher degree of correspondence to the designed modules. In a boat or a ship of large enough scale, command and control functions are distributed among a number of individuals. Sensing and sensory processing for mobile robots, which normally make up a comparable portion of the autonomy architecture as described on the example of DARPA Grand Challenge 2005 vehicle in [50], is beyond the scope of this study, as this chapter focuses on the control side of such modular systems.

2.2.1 Modules and Connections

The control branch of the autonomy can be seen in Fig. 2.2. The tiers from top to bottom are the High-Level Controller, the Low-Level Controller and the Plant. The high-level control (HLC) module is responsible for the top-level decisions such as stopping for an obstacle, initiating a lane change, selecting a speed and a lane for the vehicle or dynamic replanning in case of a road block when the controller is designed
for vehicles; selection of obstacle-free navigation paths and search-patterns or general change of modes in the case of mobile robots.

The lower portion of the control branch, the low-level control (LLC) module, is in turn responsible for maintaining the mobile-agent (plant) speed and course according to the higher-level commands, much like the function of the helmsman of the naval vessel.

These modules will be investigated further in the upcoming subsections.

2.2.2 Hybrid-State System Layout of Controllers

The control branch of the hierarchical layout has two distinct levels, high level and low level control modules, as described earlier.

**High-Level Controller (HLC):** The higher-level decisions are highly situational and be classified into several cases according to the robot and environment conditions. This is the main reason for designing a Finite State Machine (FSM) for the high-level
controller (HLC). HLC is responsible for *the situations and events that are discrete in nature*. One example would be the detection of objects and related decisions.

**Low-Level Controller (LLC):** The low-level controller (LLC), on the other hand, is a more conventional, continuous controller that interacts with the mobile agent, be it a robot or a drive-by-wire vehicle as it maintains the desired course and the speed. LLC is responsible for *the regulation/control of the states that are continuous in nature*. Examples include position, orientation and velocity of the mobile agent.

This connection between a discrete-state system on top and a continuous-state feedback loop at the bottom, as seen in Fig. 2.3, in two separate layers is the classic description of a hybrid-state system (HSS), where the system has portions with discrete states while the remaining modules have continuous states.

![Figure 2.3: Hybrid-State System with system modules.](image)
The continuous state system (CSS) is not just a controller, but the connection of the low-level control module and the robot/vehicle model, which can be simplified into the Dubins’ Car [23], given in equation (2.1), for the purposes of this study.

\begin{align*}
\dot{x}_v &= \epsilon \cos \theta_v \\
\dot{y}_v &= \epsilon \sin \theta_v \\
\dot{\theta}_v &= \eta
\end{align*}

(2.1)

The feedback connection as seen in Fig. 2.4, is from the vehicle states for position and orientation \(x, y\) and \(\theta\), fed back to the low-level controller, which in turn generates the yaw rate and velocity control inputs \(\eta\) and \(\epsilon\) respectively. The low-level controller also has the external input \(u\), which accounts for the connection of the continuous-state system to the world, just like the discrete-level system has the same connection through input \(U\). The desired path, which the robot is planned to follow, comes from the higher level, and is generated in the discrete-to-continuous interface, \(\Psi\).

On the mission level, the application-specific planner generates a path between two points in the mobility domain as a sequence of waypoints to be traversed. These waypoints can be generally too sparsely distributed to be used directly in the low level. As the high-level controller updates robot progression on a specific mission, it maintains the nearest waypoint ahead as the goal waypoint. This waypoint and a number
of previous ones with the vehicle position is then used to generate an interpolated continuous path. This process takes place in the discrete-to-continuous interface, $\Psi$, and uses the interpolation scheme of choice such as splines, potential fields or elastic bands. In open environments, where the mobility domain does not naturally lead to a graph representation as a road network does, other trajectory optimization techniques can be employed, considering the vehicle dynamics and other constraints. For example, the authors of [39] utilized receding horizon (RH) optimization over a spatial, state-dependent cost-to-go function, while [56] employed nonlinear programming for optimizing the unmanned ground vehicle (UGV) trajectory in a model-predictive framework.

The overall connection of the HSS can be detailed more formally in equation (2.2), and signal and system labels can be found in Fig. 2.3.

\[
X(k+1) = F(X(k), U(k), s(k)),
\]
\[
y(k+1) = f(y(k), x(k), u(k), S(k)),
\]
\[
x(k+1) = v(x(k), y(k)),
\]
\[
s(k+1) = \Phi(x(k+1)),
\]
\[
S(k+1) = \Psi(X(k+1)).
\]

In the set of system equations, $F$ is the state-transition function of the high-level controller in DSS, $f$ is the low-level system equation of the continuous controller, $v$ is the state-transition function of the continuous plant, $S$ and $s$ are the interface signals generated for discrete-to-continuous and continuous-to-discrete connections, and $\Psi$ and $\Phi$ are the actual interfaces for these connections.
$X \in \Sigma = [X_1, X_2, ..., X_N]$ is the discrete state of the single-state FSM in DSS, with system $F$ providing the state $X$ as output to the interface. $x = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ is the real-valued, n-dimensional vector of states for the continuous-state plant, while $y \in \mathbb{R}^m$ is the real-valued, m-dimensional vector of states and also the output of the continuous-state controller. The states of all three dynamical systems, $F$, $f$ and $v$ are also the outputs of each system.

$U$ and $u$ are inputs from external world to discrete and continuous level controllers respectively. $U(k) = [U_1, U_2, ..., U_K]^T$ and $s(k) = [s_1, s_2, ..., s_L]^T$ are $K$ and $L$-dimensional vectors of binary indicator variables that carry state-transition events to the finite-state machine $F$. $u(k) \in \mathbb{R}^l$ and $S(k) \in \mathbb{R}^j$ are real-valued, $l$ and $j$-dimensional vectors of inputs to the continuous controller $f$, from the outside world and the discrete state controller respectively.

The system as a whole can be denoted with the combination of the various state transition functions $A(F, f, \Psi, \Phi, v)$.

For FSM mechanics, the discrete-state-transition function $F$ of equation (2.2) can be further formulated as a linear function of current state and external and lower-level inputs.

Given $X \in \Sigma = [X_1, X_2, X_3, ..., X_m]$, and $X(k) = X_i = [0, 0, 0, ..., 0, 1, 0, ..., 0]^T$ as a column vector of binary indicator variables, where $i^{th}$ row is 1 when the system is in $i^{th}$ state, one can use the time-varying linear state transition function,

$$X(k+1) = F(X(k), U(k), s(k)),$$

$$= E(k)X(k), \quad (2.3)$$
\[ E(k) = \begin{bmatrix}
  e_{11} & e_{12} & \cdots & e_{1m} \\
  \vdots & \vdots & \ddots & \vdots \\
  e_{m1} & e_{m2} & \cdots & e_{mm}
\end{bmatrix} \]

and \( e_{ij} \) is the binary state transition event from state \( j \) to state \( i \) at time \( k \). The external and lower-level inputs \( U \) and \( s \) affect the function by changing the events. This state-transition matrix of events will further be exploited in the formulated development cycle of the next section.

### 2.3 Finite-State Machine Development for Hybrid-State System Controllers

The autonomous controller layout outlined in the previous section needs to be developed under a number of considerations. The finite-state nature of the discrete controller may result in incorrect behavior for the entire system if an unexpected situation, for which there is no contingency designed, is encountered. For this reason, a sense of complete coverage of acceptable scenarios as well as means to develop structured decision-making processes for each covered scenario are of utmost importance.

In the following subsections, one further tier of hierarchy is introduced to help compartmentalize the design process and an iterative method of building the state machines is described and analysed.

#### 2.3.1 Scenarios, Meta-states and Completeness

In the tiered controller structure described in the previous section and shown in Fig. 2.2, the High-Level Controller internally utilizes another layer of hierarchy. This hierarchical FSM consists of a meta-state machine, seen in Fig. 2.5.

A finite state machine is inherently case based, flexible enough to scale and easy to visualize. To further exploit this case-by-case nature of the regular state machines,
Figure 2.5: Meta states and internal sub-state machines.
the even-higher-level “meta-state machine” was developed, in which the meta-states or “states made of states” correspond to general scenarios and each one contains a fully functional state-machine for that specific scenario.

A meta state, by definition deals with a particular class of scenarios that can be grouped, or a particular task that the mobile agent is supposed to handle. Each meta state has its internal state machine (substates) to deal with that class of scenarios.

In a mission-based application, the meta-state selection can be coupled with mission definitions. Each specific type of mission can be handled by a meta state, with an internal sub-state machine to cover that particular class of scenarios. One example would be a search and rescue application, where “exploration” or “search” actions are different enough from the decisions and actions related to “rescue” mode of the mobile agent. So a two-meta-state FSM, one for Search and one for Rescue can be developed to sufficiently compartmentalize the development procedure.

In case the mobile agent is to be designed to deal with a complex environment, the meta states can each be designed to handle one particular portion of the overall operation. For a robot to operate in mixed settings such as Urban, Suburban and Rural areas, with different rules for each setting; meta states can cover differing conditions and rules efficiently.

To summarize, meta-state selection can be based on:

- **Missions/Tasks:** When one task is inherently different from another due to differing checks and decisions, meta states help compartmentalize the design. For instance, Exploration tasks and Rescue tasks can be grouped within their own meta states.
• **Rules/Settings:** Handling the same input may require different actions depending on the rules associated with a particular environment. Meta states help differentiate the reactions based on the rules. Dodging an obstacle on a road, in the middle of an intersection or in a parking zone require different manoeuvres.

Meta-state selection based on the above criteria should also consider the following pointers in order to achieve a degree of completeness, so that the affect of unexpected inputs/scenarios can be minimized:

• Consider every *general situation* that the mobile agent is required to encounter. Do not treat every possible scenario at this uppermost level, but each *class of scenarios*.

• Try to group the rules governing autonomous operation into *sets of consistent behavior*. An autonomous vehicle obeys same or very similar rules on a one-lane road and a two-lane road, while the rules change drastically in a parking zone.

• List the *quantifiable measures of success* in the given autonomy application. Try to answer the following questions: Does the robot have a list of tasks that are expected to be handled autonomously? How can these tasks be divided into classes? Are “general exploration” and “search for specific item” close enough to be in the same meta state, or should they be separated?

Once these pointers are considered, the detailed design of each meta state, the selection of substates and events, can lead to further differentiation of general scenarios and corresponding meta states. If one meta state is found/tested to be insufficient to handle the entire class of scenarios it was assigned to, new ones can be generated
to match the evolving requirements. The meta-state selection and sub-state design follow an iteratively coupled process.

### 2.3.2 Capability Grafting

Throughout the development cycle of the high-level controller described above, a multi-stage approach called “Capability Grafting” was utilized, first proposed in [36].

The process starts with a simple and minimally complete set of capabilities, and exploits the finite state machine characteristics to *graft* new abilities onto the existing FSM structure. This way, a design and implementation timeline is easier to establish as testing simpler scenarios and addition of more complicated capabilities later can be done progressively.

Consider the simple three-state FSM in Fig. 2.6. Assume that the state transitions and possible state trajectories are limited, but sufficient for the tasks at hand. Given the event to switch from *State B* to *State C*, the tasks carried out in *State C* are satisfactory.

![Simple state machine with three states.](image)
Now assume either the requirements/tasks of the system change, or the understanding of the problem evolve so that carrying out the tasks in State C are not always the appropriate response to an event in State B.

![State machine diagram](image)

Figure 2.7: New state machine after STATE D is grafted as an expansion.

Fig. 2.7 shows the addition, or “graft” of the new state, D, to work before C under certain conditions. This addition did not in any way change or destroy the existing possible state trajectories. State B can still transition directly to State C, but now there is an alternate path through State D. This type of graft is defined as an Expansion Graft.

It is important to note that the parallel nature of the expansion graft only preserves the existing state transition paths if the new transition does not eliminate alternative paths due to the transition rules. If the state transition into the newly-grafted state is conditioned on rules that are always true, the grafted path destroys existing, alternative paths in practice and it is not an expansion graft in effect. These extreme cases of grafts need to be checked, as the rules governing the transitions into the new states clearly affect the behavior of existing states under certain conditions.
On the other hand, if there is a need to take additional steps in between existing substates, the new state, D, may need to be inserted between States A and B. This type of graft, seen in Fig. 2.8, in contrast to the expansion, modifies the existing state trajectories, as State A can no longer transition to State B in one step. This type is defined as an **Insertion Graft**.

The difference between Insertion and Expansion grafts are in essence similar to the difference between a series and parallel connection in circuit elements. As it is the case in the circuit counterpart, newly-grafted states can also be connected in parallel to some states while they are inline with the others. These grafts show a mixture of both insertion and expansion characteristics and are called **Hybrid Grafts**.

The overall procedure will further be demonstrated in the next sections in a few stages that go from simple to more complex in the development cycle of an hypothetical high-level controller for an autonomous vehicle on a simplified world.
2.3.3 Equivalence Classes of Discrete States

In this part, a number of equivalence relations on the discrete states of the HSS and the corresponding classifications will be defined to help formally define the above-presented graft types.

Starting with the adjacency matrix $A$ that describes which state pairs are connected in the FSM discrete-state subsystem of HSS $A$, where

$$
A = \begin{bmatrix}
    a_{11} & a_{12} & \cdots & a_{1m} \\
    \vdots & \ddots & \ddots & \vdots \\
    \vdots & \ddots & a_{ij} & \vdots \\
    a_{m1} & a_{m2} & \cdots & a_{mm}
\end{bmatrix},
$$

(2.4)

$$
a_{ij} = \begin{cases}
1 & \text{if } p_{ij} = p(X(k + 1) = X_j|X(k) = X_i) > 0 \\
0 & \text{otherwise}
\end{cases},
$$

Let $c_{in}(X)$ and $c_{out}(X)$ be two functions from the set of discrete states $\Sigma$ to the set of nonnegative integers $\mathbb{Z}^*$, representing the number of incoming and outgoing state transitions for each discrete state, respectively.

$$
c_{in}(X_i) = \sum_{j \in [1,m]} a_{ji} \quad \text{(2.5)}
$$

$$
c_{out}(X_i) = \sum_{j \in [1,m]} a_{ij}.
$$

Since each outgoing state transition from one state is an incoming state transition to either the same state or another, the sums of these two function over all states should be equal.

$$
\sum_{i \in [1,m]} c_{in}(X_i) = \sum_{i \in [1,m]} c_{out}(X_i). \quad \text{(2.6)}
$$
Using these two functions, two equivalence relations, $\sim_{in}$ and $\sim_{out}$ can be defined:

\[
X_i \sim_{in} X_j \text{ if } c_{in}(X_i) = c_{in}(X_j)
\]

\[
X_i \sim_{out} X_j \text{ if } c_{out}(X_i) = c_{out}(X_j).
\]

The corresponding equivalence classes of discrete states are demonstrated in Figures 2.9 and 2.10 on a simple FSM example.

![ FSM Diagram ]

Figure 2.9: Equivalence classes of discrete states according to the equivalence relation $\sim_{in}$. $S_1 \sim_{in} S_5$ and $S_3 \sim_{in} S_4$.

The graft operations defined in the previous subsection can now be further formalized using the equivalence relations and classes defined above. Given the original HSS $A(F, f, \Psi, \Phi, v)$, let $\tilde{A}_{ins}(\tilde{F}, \tilde{f}, \tilde{\Psi}, \tilde{\Phi}, \tilde{v})$ be a new HSS after a single state is added to the existing finite-state machine via an insertion graft, where $\tilde{X}_{ins} \in \tilde{\Sigma} = [\tilde{X}_1, \tilde{X}_2, \tilde{X}_3, \ldots, \tilde{X}_{m+1}]$. 

24
Figure 2.10: Equivalence classes of discrete states according to the equivalence relation $\sim_{out}$. $S_1 \sim_{out} S_2$ and $S_3 \sim_{out} S_4$.

Since the insertion graft is defined above as a new state added between previously-connected states,

$$\forall i \in [1, m], X_i \sim_{in} \tilde{X}_i, \text{ as } c_{in}(X_i) = c_{in}(\tilde{X}_i),$$

$$\forall i \in [1, m], X_i \sim_{out} \tilde{X}_i, \text{ as } c_{out}(X_i) = c_{out}(\tilde{X}_i).$$

In contrast, let $\tilde{A}_{exp}(\tilde{F}, \tilde{f}, \tilde{\Psi}, \tilde{\Phi}, \tilde{v})$ be a new HSS after a single state is added to the existing finite-state machine via an expansion graft, where $\tilde{X}_{exp} \in \tilde{\Sigma} = [\tilde{X}_1, \tilde{X}_2, \tilde{X}_3, ..., \tilde{X}_{m+1}]$.

For this HSS, the expansion graft creates alternate state trajectories, according to the definition in the previous subsection, resulting in one existing state with a new outgoing transition and one existing state with a new incoming transition.
\[ \exists i \in [1, m], \text{s.t. } X_i \not\sim_{in} \bar{X}_i, \text{ as } c_{in}(X_i) + 1 = c_{in}(\bar{X}_i), \quad (2.9) \]

\[ \exists i \in [1, m], \text{s.t. } X_i \not\sim_{out} \bar{X}_i, \text{ as } c_{out}(X_i) + 1 = c_{out}(\bar{X}_i) \]

Figure 2.11: Equivalence classes of discrete states according to the equivalence relations \( \sim_{in} \) and \( \sim_{out} \) (a) before any graft operations, (b) after an insertion graft and (c) after an expansion graft.
The described effects of insertion and expansion grafts on the equivalence classes of states based on the number of incoming and outgoing state transitions are demonstrated in Figure 2.11, where the number of incoming/outgoing transitions are displayed on each state before and after the graft.

### 2.3.4 Properties of Graft Types

The major difference between an expansion and an insertion graft is the preservation of existing state trajectories. The following definitions, formulated in [20], are important in the analysis of this difference.

**Definition 1** The set \([X^b_{x^b}] \subseteq \begin{bmatrix} X \\ R^n \end{bmatrix}\) is reachable from the set \([X^a_{x^a}] \subseteq \begin{bmatrix} X \\ R^n \end{bmatrix}\) if \(\forall X(0) \in X^a, x(0) \in x^a, \exists \Gamma(k), k = 0, 1, ..., K; \text{ such that } X(K) \in X^b, x(k) \in x^b\).

**Definition 2** The set \([X^a_{x^a}] \subseteq \begin{bmatrix} X \\ R^n \end{bmatrix}\) is 1-step returnable if \(\forall X(0) \in X^a, x(0) \in x^a; \exists \Gamma(0) \text{ such that } X(1) \in X^a \text{ and } x(1) \in x^a\).

**Definition 3** The set \([X^a_{x^a}] \subseteq \begin{bmatrix} X \\ R^n \end{bmatrix}\) is stabilizable if it is reachable from \([X^0_{x^0}]\) and 1-step returnable. The sets \(X^0\) and \(x^0\) consist of possible initial conditions.

As an expansion works in parallel to the earlier system, any fault in the new states themselves or trigger generation related to the newer events results in a functional, albeit handicapped controller.

On the other hand, an insertion replaces the connection between previously attached states, so any failure in the new addition leaves a portion of the system dysfunctional.

This distinction can be clearly observed in Fig. 2.7 and Fig. 2.8. With the failure of the expansion graft State D in Fig. 2.7, existing states in B and C maintain
their reachability from any state in A and one-step returnability. However, when
D malfunctions in Fig. 2.8, D and C are not reachable from any state in A any
more. Hence the stabilizability of states after an insertion graft is dependent on the
functionality of the inserted states.

**Theorem 1** The insertion graft of the set of states \([X^b_{x^b}] \subseteq [X_{R^m}]\), into the existing, one-step returnable set of states \([X^a_{x^a}] \subseteq [X_{R^m}]\), breaks the one-step returnability of the existing set \(a\), while an expansion graft onto the same set preserves the one-step returnability of the existing set of states.

**Corollary 1** The insertion graft of the set of states \([X^b_{x^b}] \subseteq [X_{R^m}]\), into the existing, stabilizable set of states \([X^a_{x^a}] \subseteq [X_{R^m}]\), breaks the stabilizability of the existing set \(a\), while an expansion graft onto the same set preserves the stabilizability of the existing set of states.

Note that this analysis does not necessarily lead to one type of graft being superior. It will not always be possible to graft new capabilities in parallel to the existing ones, and even when it is possible, it might be costly to have a larger number of new states compared to an insertion. As usual, there is a trade-off between robustness and cost.

Further differentiation of the two different graft types is possible through investigation of the state-transition matrix \(E\), defined in the previous section.

Given the original, pre-graft system \(A\), with discrete-state transition function \(F\) and \(m\) states,

\[
F(X(k), \Gamma(k), s(k)) = E(k)X(k),
\]
\[ E(k) = \begin{bmatrix} \epsilon_{11} & \epsilon_{12} & \cdots & \epsilon_{1m} \\ \vdots & \ddots & \ddots & \vdots \\ \epsilon_{m1} & \epsilon_{m2} & \cdots & \epsilon_{mm} \end{bmatrix} = E_{11}, \]

and the post-graft system, \( \tilde{A} \), with the transition function \( \tilde{F} \),

\[ \tilde{F}(\tilde{X}(k), \Gamma(k), s(k)) = \tilde{E}(k)\tilde{X}(k), \]

\[ \tilde{E}(k) = \begin{bmatrix} \tilde{\epsilon}_{11} & \tilde{\epsilon}_{12} & \cdots & \tilde{\epsilon}_{1m+1} \\ \vdots & \ddots & \ddots & \vdots \\ \tilde{\epsilon}_{m1} & \tilde{\epsilon}_{m2} & \cdots & \tilde{\epsilon}_{mm} \\ \tilde{\epsilon}_{m+11} & \tilde{\epsilon}_{m+12} & \cdots & \tilde{\epsilon}_{m+1m+1} \end{bmatrix} = \tilde{E}_{11}, \]

assuming that only one new state was grafted and the new number of states is \( m+1 \).

Since any graft of larger number of states is simply a series of a single-state grafts, this assumption does not cause a loss of generality.

As described in Subsection 2.3.2, an expansion graft does not break any connections between existing states.

**Theorem 2** Post-graft state transition matrix, \( \tilde{E} \), includes the pre-graft matrix, \( E \), if the capability graft is of the type “expansion”.

\[ \tilde{E}(k) = \tilde{E}_{11} = \begin{bmatrix} E_{11} & E_{12} \\ E_{21} & E_{22} \end{bmatrix}, \tilde{\epsilon}_{ij} = \epsilon_{ij} \forall i, j \in [1, m], \quad (2.10) \]

where \( E_{11} \) is the state transition matrix of the earlier, pre-graft system, while the other sub-matrices \( E_{ij} \) describing the parallel connection of the new state to the existing ones.

*This inclusion, however, is not valid for an insertion graft, since existing state transitions and paths, the information contained in \( E_{11} \), are not preserved when the new state is connected in series.*

The preservation of state-transition matrices, or connectivity of the graph in general, is especially useful when dealing with a large number of states and the computational
cost of the tools used for reachability and in extension formal verification [29] is of concern. On the other hand, as mentioned in the previous subsection, not all additions to the world model can be solved via expansions, and insertion grafts have their uses, even if they restructure the state-transition matrix completely.

2.4 Autonomous Mobile Agent Examples

The Control and Intelligent Transportation Research (CITR) group at the Ohio State University has developed a number of autonomous platforms over the years, most of which participated in international autonomous vehicle and mobile robot competitions.

The methods described in the previous sections were utilized in the most recent examples in this series of mobile autonomy implementations. In this section, a fully-autonomous vehicle operating in urban scenarios, indoor mobile robots of the CITR testbed and outdoor mobile robots for tactical exploration and neutralization scenarios are used to illustrate the above mentioned ideas.

2.4.1 OSU-ACT and the Meta-State Machine

Ohio State University Autonomous City Transport (OSU-ACT) was developed to participate at DARPA Urban Challenge 2007\footnote{The Urban Challenge is the most recent in a series of autonomous-vehicle competitions, initiated by the Defense Advanced Research Projects Agency. The main difference between the Urban Challenge and the preceding autonomous challenges is that the scope of autonomous capabilities has shifted from highway or off-road environments to an urban setting.}. As mentioned before, the highly situation-dependent nature of urban driving led to the FSM-based design approach for the high-level decision-making process of OSU-ACT.
OSU-ACT represents the latest iteration in a series of autonomous vehicles designed at The Ohio State University, both for earlier DARPA challenges and other events. A certain amount of legacy from autonomous highway system solutions such as the Demo ‘97 vehicles [49], to more recent implementations for Grand Challenges 2004 [17] and 2005 [18, 50], can be traced down to the design philosophies associated with ACT, seen in Fig. 2.12.

The urban autonomy application of the Urban Challenge 2007 provided a number of distinct situations with accompanying sets of rules. The rules that a vehicle is required to follow on a roadway are different from the ones for an intersection, where precedence and stopping locations are significant. Similarly, a parking zone with no marked roads provides enough of a separate set of behaviors to mark it as a different meta state.

The interconnection of the meta states for the specific example of Urban Challenge HLC can be seen in Fig. 2.13.
As described in Section 2.3.1, the classification of scenarios into specific rule sets is an important consideration for meta-state selection. OSU-ACT meta states followed this rule for the design of the top-level of the control hierarchy. In the final implementation, the following meta states were selected to cover the urban autonomy of OSU-ACT:

- **Start**: Initialization routines and preliminary mission parsing and planning.

- **Finish**: End-of-mission declaration and setting the vehicle to be safe for human interaction.

- **Road**: Roadway navigation, avoidance of partial blocks, lane changes and car following.

- **Intersection**: Obeying the precedence order in stop-sign controlled intersection, merging into moving traffic and navigating partially blocked intersections.
• **U-Turn:** Specific 5-point u-turn maneuver for fully blocked roadway and dynamic replanning for the remainder of the mission.

• **Zone:** Navigation in obstacle zones without roads, parking and deparking manoeuvres.

The “Road” meta state in the meta-state interconnection above, was developed very similarly to the development example of Section 2.4.2 and other aspects of the OSU-ACT has been used as examples through the earlier sections.

### 2.4.2 Development Example for the Grafting Process

The indoor testbed at OSU Control and Intelligent Transportation Research Laboratory, dubbed *SimVille*, seen in Figure 2.14, has been used to study the feasibility of many urban autonomy implementations on a smaller, easily-controllable scale over the years [9]. Details on the actual architecture of the testbed, the mobile agents and sensors can be found in [8] and [54].

Autonomy of the robots in the mock-urban setting used in SimVille were initially achieved through a simple high-level controller for a vehicle on a simplified road. Further capability grafts to enable operation under V2I traffic-lights and convoying behavior were also included in later revisions.

The simplest scenario one can think of on a road is cruising on an empty, single-lane road, following waypoints, as illustrated in Fig. 2.15.

This world model does not require more than the simplest state machine, and one suitable solution can be seen in Fig. 2.16. Using the equivalence relations defined in the previous section, the equivalence classes for each discrete state can be tabulated in Table 2.1.
Figure 2.14: The indoor testbed at The Ohio State University Control and Intelligent Transportation Research Laboratory.

Figure 2.15: First Stage. No obstacles, single lane, waypoint following.

Under the assumptions that the continuous states of interest are the longitudinal speed of the vehicle and the lateral position \( \mathbf{x} = [\dot{x}_{\text{lon}}, x_{\text{lat}}]^T \), and the continuous controller maintains a certain velocity under waypoint following conditions, the set \([\mathbf{x}_a, \mathbf{X}_a]^c\), where the continuous states are within closed velocity and lateral position...
Figure 2.16: *Road* meta state at Stage I. Event E1 stands for reaching the end of the road.

<table>
<thead>
<tr>
<th>State #</th>
<th>$c_{inc}(X_i)$</th>
<th>$c_{out}(X_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.1: Equivalence classes for discrete states in Figure 2.16.

brackets and the discrete state is either S1 or S2, is both reachable and one-step returnable, therefore stabilizable according to the definitions of the previous section.

One can complicate this scenario by including an obstacle on the single lane, as in Fig. 2.17. This means adding external sensors on the sensing branch of the overall architecture, and detection of an obstacle triggering further action in the controller.

Figure 2.17: Second Stage. Lane can be blocked.
In Fig. 2.18, a new capability is grafted onto the existing FSM. The new states, S4 and S5 open up alternate state trajectories. Instead of doing the S1→S2→S3 trajectory, it is now possible to visit the new states and follow S1→S2→S4→S5→S2→S3 route when an obstacle is detected. So the existing trajectory is preserved while a new one is added.

Figure 2.18: *Road* meta state at Stage II. Event E2 is detecting the current lane being blocked.

Since the new trajectory works in parallel to the existing ones, the new block representing the additional states, F2 in Fig. 2.19, are connected to the earlier system in parallel. The graft is an “expansion” as it does not directly modify the prior system.

Table 2.2 lists the same equivalence relations after the graft operation. The new graft being an “expansion,” the numbers of both the incoming and outgoing transitions increased by one for state S2, as described in the previous section.

The reachability, one-step returnability, and stabilizability of the set \([x_u, X_u]^t\), as described above, were preserved under the expansion graft, as expected according to the theorems of the previous section.
Table 2.2: Equivalence classes for discrete states in Figure 2.18.

<table>
<thead>
<tr>
<th>State #</th>
<th>$c_{inc}(X_i)$</th>
<th>$c_{out}(X_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.19: “Expansion graft” of F2 onto F1 in block diagram.

Further capabilities are required when the world model is expanded to include a second lane, strictly in the opposite direction, as in Fig. 2.20. Now the autonomous vehicle needs to have the lane-change capability.

As with the previous stage, one needs to stop before the blocking vehicle before doing anything else. After the stop, however, there are two possibilities: pass the obstacle on the left, or wait. Fig. 2.21 depicts this new possibility with the addition of the new pass state, $S1 \rightarrow S2 \rightarrow S4 \rightarrow S5 \rightarrow S6 \rightarrow S2 \rightarrow S3$ being the new trajectory. The newly-expanded block is also shown in Fig. 2.22.
Figure 2.20: Third Stage. Lane can be blocked, there is a second lane on the left with the opposite direction. Corresponding FSM is in Fig. 2.21.

Figure 2.21: *Road* meta state at Stage III. Event E3 is having the passing lane occupied.

A second expansion graft increased the number of incoming and outgoing state transition, as seen in Table 2.3.

In the last stage of development in this example, the direction of the second lane is left unknown, to make the world model more generic. This new world model (Fig. 2.23) is applicable to a larger number of situations, without actually changing the structure, but by relaxing the assumptions.
Table 2.3: Equivalence classes for discrete states in Figure 2.21.

<table>
<thead>
<tr>
<th>State #</th>
<th>$c_{inc}(X_i)$</th>
<th>$c_{out}(X_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>S6</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2.22: The new block, F3 is “expanded” over the previous stage, connected to both F1 and F2.

Figure 2.23: Fourth stage. Lane can be blocked, second lane on the left is in unknown direction.
Figure 2.24: *Road* meta state at Stage IV. Event E4 is having a solid lane divider, which means one needs to stop before passing.

The required capability is grafted as S7, seen in Fig. 2.24. This new graft is different in terms of its connection to the existing states. As S7 is located on an existing state trajectory, between S2 and S4, it was called an “insertion graft” in the previous section. It modifies an existing trajectory so that a direct transition between previously connected states is no longer possible. However, S7 also provides and alternate path from S2 to S6, so this operation has the characteristics of both types, and is a “hybrid graft.”

The new graft being a hybrid, the insertion part of the operation does not change the numbers of incoming and outgoing transitions of states S2 and S4, while the new alternate path between S2 and S6 increases the number of incoming transitions to S6. The changes can be seen in Table 2.4.

Fig. 2.25 further demonstrates the significance of an “hybrid graft” as blocks F1 and F2 are no longer connected directly, as the state or states within F4 reside in between them due to the “insertion” portion of the hybrid.
Table 2.4: Equivalence classes for discrete states in Figure 2.24.

<table>
<thead>
<tr>
<th>State #</th>
<th>$c_{inc}(X_i)$</th>
<th>$c_{out}(X_i)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>S2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>S3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>S4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>S5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>S6</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>S7</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Figure 2.25: F4 is "inserted" between existing blocks. F1-F2 direct connection is severed. Also an alternate state trajectory is formed, making the graft a “hybrid.”

### 2.4.3 MAGIC 2010 Capabilities

OSU CITR has recently developed the higher-level software modules for the mobile robots to participate in MAGIC 2010\(^2\), as part of the finalist Team Cappadocia.

The higher-level software modules developed at OSU for the “Peri” mobile robots, seen in Fig. 2.26(a), include the top level Dynamic Mission Planner, on-board and central Sensor Fusion and the robot High-Level Control.

\(^2\)The Multi-Autonomous Ground-robotic International Challenge (MAGIC) 2010 is a research competition organized by U.S. and Australian Departments of Defense. The challenge requires competitors to demonstrate multi-vehicle autonomous robotic teams that can execute an intelligence, surveillance and reconnaissance mission in a dynamic urban environment.
The decision-making modules of the robots were designed using a series of capability grafts, and the overall system was tested on the indoor testbed (Fig. 2.26(c)) with the robots seen in Fig. 2.26(b).

Figure 2.26: (a) “Peri” UGV, built for MAGIC 2010; (b) Pioneer 3-AT mobile robots at OSU CITR; (c) Robots during a mission at CITR testbed.
Two snapshots from the growth cycle of the discrete-controller FSM can be seen in Fig. 2.27.

A subset of the autonomous robots, named “disruptors” are responsible for neutralization of static objects-of-interest during MAGIC 2010. These robots and the corresponding FSM, as seen in Fig. 2.27(a), were initially designed to go to the static object-of-interest (sOOI), wait for neutralization confirmation from the central command station and commence neutralization.

After this initial design, it was deemed necessary to deal with objects that are at the right neutralization distance, but are also partially or fully occluded. This was a particular problem when sOOI were right around a corner, when the shortest path leading to the sOOI does not have clear line of sight when the correct range was achieved.

The line-of-sight-seeking capability was grafted onto the existing FSM, as seen in Figure 2.27(b), following the proposed design methodology. The new state, “find LOS to SOOI” is an expansion graft that generates an alternate action after reaching the correct range, but before asking for confirmation.
Figure 2.27: (a) “Static OOI” meta state for disruptor robots before the graft; (b) “Static OOI” meta state after the line-of-sight capabilities were grafted.
Chapter 3: Probabilistic Modeling of Mobile Agents for Estimation

3.1 Autonomy in Mixed-Mode Traffic

As discussed in the Introduction, operation of autonomous or semi-autonomous vehicles in mixed traffic, where fully manual vehicles are used with vehicles that have different degrees of autonomy, relies not only on the ability to control the behavior of one vehicle, but also on the perception of every other vehicle that are interacted with.

Since the first fully autonomous vehicle will need to build situational awareness in an environment with mostly human-driven vehicles and a few semi-autonomous ones, it is important to talk about the framework where each of these different information sources can be combined into a single model.

Figure 3.1 shows different modes and levels of information that can be obtained from vehicles (agents) of differing capabilities. For the purposes of this study, the “estimator” modules are used to make sense of the information obtained and have a meaning attached to the raw data, unless directly-applicable information is available from an intelligent source such as another autonomous vehicle. The estimators can be continuous filters or probabilistic models as discussed in the following sections.
A most basic agent, such as an old car or a bicycle, needs to be sensed first before the estimators, so the active and passive sensing capabilities remain a requirement even when high-level communications between some vehicles are available. A vehicle with basic communication capabilities can send a number of continuous measurements such as the location and speed, so that layers of estimators can be utilized as described on the following pages. Some vehicles can have more sophisticated equipment that generate their own behavior estimates and also broadcasts the estimated future paths and decisions, so the situational awareness can be linked directly to these vehicles. And lastly, autonomous vehicles can broadcast their decisions, goals and a future path, which will have a higher certainty that the human-driven vehicle, as the decision or intent is not an estimate any longer.

Figure 3.1: Agents of differing capabilities, communicating at different levels, all connected to a single situational awareness module.

In order to develop the estimators for the various levels of autonomy capabilities involved, this chapter starts with a human-driver model, investigates the existing
hybrid-state system model to capture various aspects of the human/vehicle interaction and builds behavior estimators and predictors on this model using the stochasticity of observations. The overall process is demonstrated on a number of common scenarios in traffic.

### 3.1.1 Human Driver Model

Liu and Özgüner developed the Human Driver Model presented below, based on a compartmentalized architecture capturing the sensory and decision-making modules representing a driver, interacting with the vehicle and the environment.

![General architecture of Human Driver Model](image)

Figure 3.2: General architecture of Human Driver Model.

HDM modules, interconnections of which are seen in Figure 3.2, can be briefly described as follows:
Environment: This module represents everything that the driver and vehicle interact. It includes both the external world, such as other vehicles, pedestrians, road and obstacles; and the sensory apparatus that interacts with the direct perception of the driver such as driver assistance systems and electromechanical sensors of the vehicle.

Perception: The perception module models the biological sensory capabilities of the human driver such as the range and accuracy of vision and hearing, which interfaces the environment with the decision making process.

Task Planning: The task planning block is responsible for the overall goal of the human driver, such as going from point A to point B, or finding and parking at a parking spot.

Decision Making: The central decision-making module to the overall HDM architecture is the high-level controller for the entire system. It is responsible for the decisions that are based on discrete events captured from the environment through the perception module. Performing a lane-change maneuver due to a blocked lane or utilizing the turn-signal on intersection approach are examples of decisions that this module is tasked with.

Driver Characteristics: This module alters the way decisions made in the decision-making module, based on the preferences, reaction time and habits of the individual driver. Various threshold values and delays in the decision making module are essentially functions of these driver characteristics.

Implementation: Implementation module stands for the low-level control of the vehicle according to the decisions made in the decision-making module. The
continuous-state control of the vehicle such as maintaining a certain speed and following a lane are performed in this block.

**Emergency Management:** This module is responsible for the reflexive behavior that affects the low-level control. The interaction between the emergency management and implementation blocks account for sudden reactions to the environment that do not necessarily go through the conscious decision-making process.

Since this study is mainly focusing on the decision-making process and the stochasticity of said process, the corresponding central module and the interaction with the implementation block will be further investigated in the following subsections.

### 3.1.2 Hybrid-State System Model for Driver/Vehicle Interaction

The interaction of a discrete-event system and a continuous-state plant in a hierarchic setting has been modeled as a Hybrid-State System (HSS) for a wide variety of applications including autonomous vehicles and initial HSS estimation studies [37] over the years, some of which already mentioned in the previous chapter.

This coupling of the discrete-state system (DSS) and continuous-state system (CSS), as seen in Figure 2.1 reflects the connection between the combined Task Planning / Driver Characteristics / Decision Making assembly to the Implementation block of Figure 3.2. Simply put, the blocks in the dotted lines in Figure 3.2, correspond to the HSS hierarchy in Figure 2.1.

The Decision-Making block correspond to the high-level controller module in an HSS, described in the previous chapter. The actual implementation of HLC may include nuances that account for the Driver Characteristics block of the HDM, and
depending on the mission-planning capabilities, the Task Management block can also be incorporated in the DSS part of the HSS.

The low-level control module of the vehicle-control architecture described in the previous chapter correspond to the Implementation and Emergency Management blocks of the HDM, as both vehicle control based on higher-level decisions and reflexive behavior for direct external inputs are handled in the CSS portion of the HSS.

Since this study focuses on the decision-making and driver-intention portions of the HDM, the following sections will investigate how the FSM-based DSS in a HSS can be used to capture these higher and abstract processes.

### 3.1.3 Decision-Making

The discrete-state-transition function \( F \) of Equation 2.2, in case of a finite-state machine (FSM) model, can be represented as a linear function of current state and external and lower-level inputs.

At any discrete-time instance, the discrete state \( X \in \Sigma = [X_1, X_2, X_3, ..., X_N] \), and \( X(k) = X_i = [0, 0, 0, ..., 0, 1, 0, ..., 0]^T \) is a column vector of binary indicator variables, where \( i^{th} \) row is 1 when the system is in \( i^{th} \) state. The time-varying linear state transition function was written in Section 2.2.2 as,

\[
X(k+1) = F(X(k), U(k), s(k)),
\]

\[
= E(k)X(k),
\]

\[
E(k) = \begin{bmatrix}
e_{11} & e_{12} & \cdots & e_{1N} \\
\vdots & \vdots & \ddots & \vdots \\
e_{N1} & e_{N2} & \cdots & e_{NN}
\end{bmatrix}. \quad (3.1)
\]
and $e_{ij}$ is the binary state transition event from state $j$ to state $i$ at time $k$. These descriptions and equations are repeated in this section for they will be referred to throughout this chapter.

The state transition topology of the FSM can be intuitively represented as a graph. A generic and simple driver model FSM is given in Figure 3.3. A driver approaches a point on the road where a certain decision needs to be made. Having two choices, the driver either takes action $a$ or action $b$ and exits the area of interest. The entirety of the discussion in this chapter boils down to ways to detect the decision between actions $a$ and $b$, so that the other parties can react accordingly.

Figure 3.3: Simplified FSM model and transition probabilities for driver decisions, where an approaching driver has two possible choices.
3.1.4 State Selection

A given scenario for a single driver or a group of drivers can be captured in a number of different finite-state automata structures, which is one of the reasons for the existence of a wide variety of driver and decision models. Therefore, it is imperative to select the states and build the transition models according to the needs of the particular applications.

Since this study focuses on the stochasticity of decisions and resultant states, one needs to consider the observability and verification of the resultant probabilistic model while building the FSM. The following requirements are essential for FSM models that aim at probabilistic representations:

**Observability:** In order to detect the state transitions when they happen, so that statistical methods can be employed to collect probability distributions, the state changes of the FSM need to be immediately observable.

As stated above, the actual decisions of the driver may not always be immediately apparent through measurements on the continuous state of the vehicle. Therefore, the states of the FSM should reflect the actual responses of the HSS that can be observed. The decisions should be captured in the state transitions, while observable actions and distinct states should be represented in the FSM states. In Figure 3.3, the decision between *Action A* and *Action B* may be made long before the actual state transitions, but it is important to model the observable changes in states and leave the decision to the connections between states.

**Resolution:** Coupled with the observability requirement, the ability to differentiate a wider number of scenarios and decisions is based on the resolution of the
selected FSM states. If an existing state represents the vehicle behavior under a number of different decisions/intentions for the driver; and if it is possible to divide that state into multiple, observable states that help identify the different decisions, then the resolution of the FSM needs to be increased by adding the necessary states.

**Coverage:** There is a trade-off between the increased resolution and graph simplicity, so the selection of decisions and scenarios to capture needs to be weighted against the implementation and calculation complexity.

Every single decision that a driver may ever make, under any situation, cannot be captured in an FSM without the curse of dimensionality making the overall structure unfeasibly large. So every driver model is a simplification of actuality. The selection of which decisions to include and where to increase the resolution without causing “state bloat” is an important aspect of the probabilistic FSM design.

### 3.2 Hybrid-State Estimation and Detection

Earlier studies on hybrid-state system estimation generally involved either a top-down approach, which can be summarized as “estimate the discrete state, use the corresponding continuous model to estimate the continuous state” [6] or a holistic methodology that defines the system in its entirety and does not utilize the discrete-continuous separation [24]. For our applications, which involves vehicles in urban traffic, direct output on the driver state is scarcely available. So our method starts from the easier-to-observe states of the lower level, such as vehicle velocity, position and orientation, and builds the estimate of the higher-level state using the estimate of the continuous state. The system connections of said method can be seen in Figure 3.4.
3.2.1 Estimator Architecture

The information flow in the system can be summarized as follows: Sensing or vehicle-to-vehicle (v2v) communication equipment on the observing vehicle, designated “the host” generates measurements, $z$, on the continuous state of the observed vehicle, “the target”. These measurements are filtered through continuous filtering, which may incorporate particle filters or a variety of Kalman filters to generate an estimate of the continuous states $\hat{x}$, formulated as a MIMO estimator block in Equation 3.2, which is in turn fed into the driver state detector described in the next subsection. For our implementations, the continuous-filtering block was named the Vehicle Tracking System and abbreviated VTS in Equation 3.2.
The measurement signal $z$ carries all the information that the outside observer can access. The information in $z$ includes both the continuous-system-state measurements, as mentioned above, and also optional data on various vehicle systems such as turn-signal status, acceleration-pedal level and steering-wheel position if the vehicle is equipped to relay these over v2v communication.

\[ z(k) = v(x(k)), \]
\[ \hat{x}(k) = VTS(z(k)) \]

The stochasticity of the decision-making process is utilized in the final module, in which the detected FSM state is used to predict the future states of the driver behavior.

### 3.2.2 Discrete-State Detection

The tracking and estimation of the instantaneous driver state is accomplished through an internal model of the discrete-state system, which is run by the estimated continuous parameters. A generic and simple driver model FSM developed for demonstration purposes was given in Figure 3.3.

The continuous-state estimates $\hat{x}$ are used to drive a number of events, either directly in the FSM, or through secondary calculations on continuous parameters such as distance between vehicles and binary parameters embedded directly into the measurements, such as v2v messages (turn signals or “brake applied” flags), if available. The events are set on continuous-parameter estimates.
So the estimated continuous state, $\hat{x}$, drives the signal generator $\Phi$, which in turn generates the estimate of the signal $s$ so that we can generate an instantaneous estimate of the discrete state, as given in Equation 3.3.

$$\hat{s}(k) = \Phi(\hat{x}(k)),$$

$$\hat{X}(k+1) = F(\hat{X}, U(k), \hat{s}(k))$$

The output of the instantaneous driver state estimator, $\hat{X}$ is one of the inputs of the predictor, which will be detailed in the following section.

3.3 Stochasticity of Control Decisions

In order to capture the stochasticity of the FSM decision-making process either on the uncertainty of a single driver or the distribution of a number of drivers, assignment and update of probabilities for state-transitions are required. The following subsections define the necessary steps for defining stochasticity of the FSM.

3.3.1 State-Transition Probabilities

This subsection makes use of a representation structure called a trellis, which found extensive usage through the Viterbi Algorithm in communication applications.

The Viterbi Algorithm [25], in simplest terms, is used on the observation sequence of a Hidden Markov Model (HMM) to find the most likely sequence of hidden states. The dynamical-programming algorithm was extended by Bouloutas et al.[11] to account for relaxing the alignment requirement of the original, where no missed or extra observations were allowed. For both the original algorithm and the extensions, the visual representation of the trellis is a useful tool to help define the necessary concepts.
The trellis is used to represent a finite state machine, similar to the state diagram, and it visually helps probabilistic calculations and predictions. The corresponding trellis for a given state diagram is presented in Figures 3.3 and 3.5.

The trellis, in simplest terms, is a directed graph or a tree, which starts from known initial state(s), and ends at the designated final state(s). In our example FSM, the initial state is $S0$ and the only possible final state is $S3$. Each column of nodes in the trellis represents a new discrete-time instance after transitions from previous column of nodes. So at the second column, there are two possible nodes for $S1$ and $S2$, the possible states that the initial state can transition into; and the third column contains the possible states at the next transition.

![Trellis representation for the simplified, four-state FSM model.](image)

In order to have any stochastic application with a trellis, one needs to assign probabilities (or inversely costs) to the edges of the trellis, which correspond to state transitions of the FSM. Once the edge probabilities are assigned, it is possible to run a graph search from any state to possible final states or the states that are $n$-transition away, to determine the maximum-likelihood estimate of the defined outcome.
So if \( X \in \Sigma = [X_1, X_2, X_3, \ldots, X_N] \) is the set of discrete states and the state-to-state transition probabilities \( p_{ij} = p(X(k+1) = X_j | X(k) = X_i) \), are known, as in Figure 3.3, they can be directly applied to the trellis, as seen in Figure 3.5. These state-transition probabilities are further tied into the binary event variables defined in Equation 3.1, simply as \( p_{ij} = p(e_{ji} = 1) \).

We can further define the probability of the \( n \)-step aggregate trajectory connecting state \( i \) to state \( j \) as \( p_{ijn} = p(X(k+n) = X_j | X(k) = X_i) = \sum (p_{io} * p_{ol} * \cdots * p_{qm} * p_{mj}) \), where the sum is among all individual state trajectories that connect \( X_i \) to \( X_j \) in \( n \) steps. More formally, \( p_{ijn} \) can be defined as an element of the \( n \)-step probability matrix \( P^n \), as given in Equation 3.4.

\[
P = \begin{bmatrix}
p_{11} & p_{12} & \cdots & p_{1N} \\
\vdots & \ddots & \ddots & \vdots \\
p_{N1} & p_{N2} & \cdots & p_{NN}
\end{bmatrix},
\]

\[
P^n = \overset{n\text{-times}}{P * P * \cdots * P},
\]

(3.4)

\[
P^n = \begin{bmatrix}
p_{11n} & p_{12n} & \cdots & p_{1Nn} \\
\vdots & \ddots & \ddots & \vdots \\
p_{N1n} & p_{N2n} & \cdots & p_{NNn}
\end{bmatrix}.
\]

The \( P \) matrix can also be interpreted as the term-by-term probability matrix of the transpose of \( E \) matrix of the function \( F \) from Equation 3.1 being equal to the \( N \)-by-\( N \) matrix of 1s.

**Theorem 3** Given the estimate of the current state of the state machine \( F \), \( \hat{X}(k) = X_i \), the maximum-likelihood estimate of the state in \( n \) state transitions is \( \hat{X}_n(k) = X_{\kappa} \), where \( \kappa = \arg\max_j (p_{ijn}) \).
The $i^{th}$ row of the n-step probability matrix $P^n$ consists of $[p_{i1n}, p_{i2n}, \ldots, p_{iNn}]$, which are the probabilities of being in states 1 through $N$ in $n$ state transitions. Picking the maximum value in this row of probabilities correspond to the maximum-likelihood estimate of the discrete state $n$-transitions in the future.

### 3.3.2 Decision Probabilities as Functions of Risk

An expanded interpretation of the state transition probabilities is linked to the Driver Characteristics module of the Human Driver Model given in Subsection 3.1.1. Given a specific scenario and a corresponding decision, with all the physical parameters such as vehicle state and environment status being constant, the actual decisions made by two individual drivers can be different. The reason behind this variation is the difference between individual driver characteristics, and the definition of risk as a factor in the decision-making process can help model this difference.

Let $Y = [X, x, U, u]^T$ be the state of everything that the driver is aware of, including the discrete and continuous states $X$ and $x$ and inputs from the environment into the discrete and continuous subsystems $U$ and $u$. The state transition probability from state $i$ to state $j$ at time $k$ was already defined as $p_{ij} = p(X(k+1) = X_j | X(k) = X_i)$.

Using this all-encompassing state, we can define the risk of making the decision of changing the discrete state from $i$ to $j$, for driver $d$ as,

$$r_{ij}^d = E\{c^d(X(k+1) = X_j | Y(k) = Y_i)\}, \quad (3.5)$$

where $Y_i = [X_i, x_i, U_i, u_i]$ is the state of the world as driver $d$ sees it, and $c^d(X(k+1) = X_j)$ is the cost of being at discrete state $X_j$ at time $k+1$. 

59
Since the driver does not know the future state \( Y(k+1) \) entirely at time \( k \), which includes \( U(k+1) \) and \( u(k+1) \) that come from the outside world, the risk is an expected value of the cost over all possible substates of \( Y \),

\[
\begin{align*}
    r^{d}_{ij} &= E\{c^{d}(X(k+1) = X_j)|Y(k) = Y_i\} \\
    &= \int \int \int c^{d}([X_j, x, U, u]^T|Y(k) = Y_i) \\
    &\quad \pi_x \pi_u dxdUdu \\
\end{align*}
\]

where \( \pi_x, \pi_U, \pi_u \) are the p.d.f. of the continuous state, discrete-level input and the continuous-state input.

The driver characteristics mentioned above are the psychological factors shaping the function \( c^{d} \) uniquely for each driver, since the same world state \( Y_j \) may have different risk levels for different drivers.

It is also possible to define the expected reward of a decision, following the same mechanism as the cost definition, but it is safe to assume an individual driver makes decisions that minimizes his/her interpretation of the expected risk, without loss of generality. Therefore, the probability of a particular state transition \( p_{ij} \) is inversely proportional to the risk of the corresponding decision.

\[
\begin{align*}
    p_{ij} &\propto \frac{1}{r^{d}_{ij}} \\
    p_{ij} &= \frac{1}{N-1} \sum_{l \neq j} r^{d}_{il} \\
    &= \frac{1}{N-1} \sum_{l \in [1,N]} r^{d}_{il}. \quad (3.7)
\end{align*}
\]

And we can define the risk-based state-transition probability \( p_{ij} \) via the sum of the risks of all the other state transitions from state \( i \), as seen in Equation 3.7.
3.3.3 Probability Assignment and Updates

The probabilities $p_{ij} = p(e_{ji} = 1)$, as discussed in the previous subsections, need to correspond to actual decision and transition probabilities if the built model is to be used for estimation, prediction or any general study of the human-driver behavior.

In most cases, direct observation of a group of drivers can lead to collection of length-$o$ state sequences $X(k, k + o)$, since the observability of state transitions is a requirement as defined in Subsection 3.1.4, while the underlying decision-making process of risk minimization defined in Subsection 3.3.2 is inherently unobservable.

Once the observed discrete-state sequence is obtained, statistical methods such as Baum-Welch Algorithm [55], which itself computes the probabilities via maximum-likelihood estimation through expectation maximization, can be used to calculate the approximate transition probabilities, as this task is essentially a parameter-estimation problem. Hidden Markov Model (HMM) studies [7] can be of further help since the probabilistic decision making process studied as a piece of the driver model in this study has close ties with the HMM. For instance, in [41], Mitrovic et al. use collected acceleration and velocity data as observation sequences of HMMs to recognize driving events.

Alternatively, approximating the situation parameters such as gap lengths and speeds into Gaussian distributions through unscented transforms, Bayesian formulations of decision probabilities can be structured as presented in [52]. As this leads to a very scenario-specific estimator for a single decision, the FSM models presented in this study are proposed as a slightly bigger picture of the situation.
For a limited set of scenarios, where the number of state transitions is small, the state-transition probabilities can also be assigned intuitively as an initial model and can later be modified according to collected data.

For the FSM model with a higher number of states than the model in Figure 3.3, as shown in Figure 3.6, the probability assignment for prediction can be done empirically through equalizing end-to-end probabilities of possible scenarios. In other words, each state trajectory that starts at an initial state and ends at a final state is given equal weight, and these trajectories vote on the transitions they occupy. From our seven-state FSM, the trajectories from \textbf{S0} to \textbf{S3} and \textbf{S0} to \textbf{S6} vote or weigh on the transitions they pass through and the number of end-to-end trajectories passing through a particular transition determines the likelihood of that edge on
the trellis. This method approximates a uniform probability distribution over the
driver intentions. Each end-to-end scenario correspond to one particular path that
the driver can decide to follow, so assigning equal weights to each end-to-end path
and approximating individual state-transition probabilities through voting results in
a set of probabilities where each driver intention is equally likely.

Starting with the adjacency matrix $A$, first defined in Equation 2.4, for the FSM
discrete-state subsystem of HSS $S$, where

$$A = \begin{bmatrix}
  a_{11} & a_{12} & \cdots & a_{1N} \\
  \vdots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & a_{ij} \\
  a_{N1} & a_{N2} & \cdots & a_{NN}
\end{bmatrix}, \quad (3.8)$$

$$a_{ij} = \begin{cases}
  1 & \text{if } p_{ij} > 0 \\
  0 & \text{otherwise}
\end{cases}$$

And similar to the powers of the probability matrix $P$, as described in Equa-
tion 3.4, $n$-th power of the adjacency matrix $A$ gives the information on the connect-
tivity in $n$ transitions:

$$A^n = \underbrace{A \ast A \ast \ldots \ast A}_{n\text{-times}},$$

$$= \begin{bmatrix}
  a_{11n} & a_{12n} & \cdots & a_{1Nn} \\
  \vdots & \vdots & \ddots & \vdots \\
  \vdots & \vdots & \ddots & a_{ijn} \\
  a_{N1n} & a_{N2n} & \cdots & a_{NNn}
\end{bmatrix}, \quad (3.9)$$

$$a_{ijn} = \begin{cases}
  1 & \text{if } p_{ijn} > 0 \\
  0 & \text{otherwise}
\end{cases}$$

If $\Sigma_b \subset \Sigma$ is the set of all discrete initial discrete states and $\Sigma_e \subset \Sigma$ is the set of
all possible discrete end states, for each $a_{ijn} = 1$ with $X_i \in \Sigma_b$, $X_j \in \Sigma_e$ and $n \in \mathbb{Z}^+$
there exists a state-transition path of length $n$ that connects an initial state to an end state.

The probability assignment method that equalizes the end-to-end path probabilities can then be formulated as follows:

1. Initialize an $N \times N$ counter matrix $C$ with $c_{kl} = 0 \forall k, l \in [1, \ldots, N]$,
2. For all $i$, where $X_i \in \Sigma_b$,
3. For all $j$, where $X_j \in \Sigma_e$,
4. For all $n \in \mathbb{Z}^+, N \leq \text{steplimit}$, where $\text{steplimit}$ is a design parameter to constrain the length of paths to be considered,
5. If $a_{ijn} = 1$, find the length-$n$ state-transition path from $i$ to $j$ and increment $c_{kl}$ by one for every state transition the path.
6. Once every end-to-end path of length less than $\text{steplimit}$ are found and corresponding $c_{kl}$ are incremented, $p_{ij} = \frac{c_{ij}}{\sum_{o=1}^{N} c_{io}}$.

Since step 5 of the algorithm consists of finding the actual path of length $n$ from state $i$ to state $j$, which for nontrivial cases involves a graph search; this methodology is more suitable for simpler graph topologies, for which the required graph searches are manageable. Even with complicated graph structures, the complexity of the above algorithm does not affect the actual estimation/prediction implementation, as the initial probability assignment is done offline and only once per FSM structure.

When the underlying risk-vs-reward notations (or decision costs) are not deducible, and when the observations on a specific discrete controller to build a statistical model cannot be readily obtained, the end-to-end equalized probabilities are
a good initial approximation for the unknown decision maker, since the method assumes that every scenario (every end-to-end state path) is equally likely.

As it is the case in the original Viterbi Algorithm, the initial cost assignment, independent of the method used, is not always sufficient for an healthy representation of decision probabilities as the scenario evolves. For a particular vehicle-based application, the state transition probabilities can also be dynamically updated depending on the measurements on the tracked vehicle.

After a set of initial probabilities are calculated, certain events can be tied into modified probabilities. Detecting a certain behavior pattern can force certain probabilities higher and others lower to shift the prediction into one direction or the other. Gradually increasing and decreasing the probabilities according to detected behavior patterns or external flags lead to a more accurate instantaneous representation of the state-transition stochasticity.

The dynamic probability updates can be assigned to binary flags (brake applied, turn signal on, etc.) in the measurement data, for instance as described to be part of the v2v safety message for a DSRC driver-assistance system [37]. Or it can be conditioned to change according to certain thresholds on the observed continuous states (deceleration harder than given value).

After the initial assignment of the probabilities and as the dynamic updates are taking place, a series of graph searches can be conducted from the observed current state to each of the possible significant outcomes. The case with the highest path probability or the maximum likelihood outcome is selected as the predicted state, $\hat{X}_{\text{final}}$ output, as formulated in Theorem 3.
As the maximum-likelihood estimate of the discrete state in \( n \) steps is one of the many possible outcomes, it is still possible for one of the lesser-likelihood states to be the actual discrete state after \( n \) transitions. On the other hand, if the predicted state is the only possible state in \( n \) transitions, the prediction is guaranteed to come true, as described in Theorem 4.

**Theorem 4** Given the estimated current discrete state being \( \hat{X}(k) = X_i \), the \( n \)-step predicted discrete state is guaranteed to be the same as the estimated discrete state \( n \) time steps in the future, \( \hat{X}_n(k) = \hat{X}(k + n) = X_j \), iff \( a_{ijn} = 1 \) and \( \sum_{l=1}^{N} a_{iln} = 1 \).

The proof for the above theorem follows that given a current discrete state, \( a_{ijn} = 1 \) and \( \sum_{l=1}^{N} a_{iln} = 1 \) implies the existence of a single \( n \)-step path starting from \( X_i \) and the end of that \( n \)-step path is at \( X_j \). Therefore, the maximum-likelihood estimate defined in Theorem 3 will find the only non-zero probability \( (p_{ijn}) \) for the maximum-probability \( n \)-step predicted state.

**Corollary:** The maximum horizon of correct prediction can be defined as \( n_{\text{max}} = \max_{n \in [1, \infty]} n \) s.t. \( \exists j : a_{ijn} = 1 \) and \( \sum_{l=1}^{N} a_{iln} = 1 \).

### 3.4 Example Sets of Driving Decisions in Urban Traffic

This section illustrates the modeling and estimation of stochastic FSM in driver and vehicle hybrid-state systems with two specific examples, one focusing on a highway convoying/merging decision and the second one capturing a number of potential scenarios where a vehicle is approaching an urban intersection. The second model is also expanded to build a scenario-safety estimator for driver assistance.
3.4.1 Convoying and Gap Acceptance

Gap-acceptance decision for lane-change behavior was widely studied in transportation applications, mainly due to the fact that the major parameter in effect is the relation of the merging vehicle speed to the size of the gap to be merged in. Both deterministic studies focusing on flow-charts of lane-change/gap-acceptance behavior such as [26] and more probabilistic research focusing on the distribution of gap-acceptance behavior [40] heavily utilized the “critical gap” that defines the minimum gap-length that a driver is willing to take at a given speed. These studies were later expanded to estimate the parameters of the critical-gap distribution functions to better predict highway capacities and delays for on-ramp mergers [16]. For autonomous vehicles, the machine decision to perform a lane change has also been studied and parallels with the human decision are utilized in [51].

The designed FSM captures a specific lane-change scenario, in which one vehicle has the possibility to merge into a convoy of two other vehicles on the adjacent lane, on a two-lane road with both lanes in the same direction. The vehicle positions and the overall scenario is illustrated in Figure 3.7.

Since the gap-acceptance and speed-matching decisions of a merger are unique to this type of lane-change, the generalization of the FSM model to generic lane-change behaviors requires either significant expansion of the states or oversimplification. For these reasons, the current model is designed to be limited in scope, but demonstrative in nature, in accordance with the trade-off described in Subsection 3.1.4.

On Figure 3.7, $x_{\text{car}}$, $x_{\text{leader}}$ and $x_{\text{follower}}$ are the longitudinal positions of the approaching, tracked vehicle; leading vehicle on the adjacent lane and the following
vehicle on the adjacent lane respectively. \( x_{\text{gap}} = \frac{x_{\text{leader}} + x_{\text{follower}}}{2} \) is the longitudinal position of the gap between the convoying vehicles. \( y_{\text{car}} \) is the lateral position of the tracked vehicle. The continuous states will be abbreviated as \( x_c, x_l, x_f, x_g \) and \( y_c \) for the rest of this example.

Figure 3.8 shows the FSM model developed to estimate and predict the lane-change decision of the approaching, potentially-merging vehicle. The main focus is on the speed difference between the gap in the convoy and tracked vehicle, as well as the longitudinal position difference between the gap and the merger.

This particular treatment is based on the similarities between taking the gap between two vehicles, and taking a turn at an intersection, which is part of the more detailed example that follows. Both scenarios require matching the required speed and initiating the necessary lane-change or turn maneuver at a specific position. Therefore, a merge-into-convoy scenario can be treated broadly similar to an intersection scenario.

The length of the gap also factors into the prediction of the decision by modifying the state transition probabilities, as discussed further below.
Figure 3.8: FSM representation of the merge behavior of the approaching vehicle.

The transitions between the states on Figure 3.8 are tabulated and the transition conditions are described on Table 3.1.

Table 3.1: State transition conditions for the lane-change FSM model.

<table>
<thead>
<tr>
<th>State Transition</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0 → S1</td>
<td>$x_{g} - x_{c} \leq d_{range}$</td>
</tr>
<tr>
<td>S1 → S2</td>
<td>$\dot{x}<em>{c} - \dot{x}</em>{g} &lt; v_{tot}$</td>
</tr>
<tr>
<td>S1 → S6</td>
<td>$x_{c} - x_{g} &gt; d_{tot}$</td>
</tr>
<tr>
<td>S2 → S1</td>
<td>$\dot{x}<em>{c} - \dot{x}</em>{g} \geq v_{tot}$</td>
</tr>
<tr>
<td>S2 → S3</td>
<td>$\dot{x}<em>{g} - \dot{x}</em>{c} \geq v_{tot}$</td>
</tr>
<tr>
<td>S2 → S4</td>
<td>$\dot{y}<em>{c} \geq v</em>{lc}$</td>
</tr>
<tr>
<td>S2 → S6</td>
<td>$x_{c} - x_{g} &gt; d_{tot}$</td>
</tr>
<tr>
<td>S3 → S2</td>
<td>$\dot{x}<em>{g} - \dot{x}</em>{c} &lt; v_{tot}$</td>
</tr>
<tr>
<td>S3 → S0</td>
<td>$y_{g} - x_{c} &gt; d_{range}$</td>
</tr>
<tr>
<td>S4 → S5</td>
<td>$y_{c} \geq d_{lc}$</td>
</tr>
</tbody>
</table>

In Table 3.1, $v_{tot}$ is a tolerance value that is the half width of the window around the speed of the gap, $\dot{x}_{g}$, in which the approaching vehicle can merge into the gap. When $|\dot{x}_{g} - \dot{x}_{c}| \leq v_{tot}$, the speeds are roughly matched and merging is possible. Similarly,
$d_{tol}$ is the tolerance value that defines the window on the longitudinal position of the approaching vehicle that helps us use $|x_g - x_c| \leq d_{tol}$, within which merger is possible.

$d_{range}$ is the tracking range, within which the approaching car is marked as relevant and the lane-change FSM starts to track the behavior. $v_{lc}$ is the threshold on the lateral speed of the merging vehicle, by which the lane-change maneuver is detected. 

$d_{lc}$ is the lane-width, which helps detect the completion of the lane-change maneuver.

The tolerance and threshold values listed above are both driver and scenario specific, as different speeds require different gaps and different drivers accept different gaps. For implementation of the described model, statistically collected averages of these parameters can be used. Alternatively, developed estimators can have training periods, in which they learn the driver-specific parameters through observations.

As already mentioned in the previous section, the trellis structure tends to get complicated as the number of states increases, and for this case with seven discrete states, the trellis, seen in Figure 3.9 is visually dense. Starting from the single possible initial state $S_0$, each column representing possible states at one time step, the trellis ends at the possible final states $S_5$ and $S_6$.

For the prediction of the final outcome for merger scenarios, once the state-to-state transition probabilities for the FSM in Figure 3.8 are set via HMM-based methods or empirically, the following conditions can be used to update the probabilities dynamically to generate more accurate prediction results. $\uparrow$ denotes an increase in the probability of the given state transition, while $\downarrow$ represents the opposite.

Table 3.2 defines probability updates to make a lane change into a gap more likely when the gap and merging vehicle are aligned and the turn signal of the merging car is on. Similarly, when the gap length is less than a certain threshold, the probability
that the new vehicle will stay in its original lane is increased. Also, behavior patterns with trying to catch up to a gap of sufficient length and trying to slow down to match a gap of sufficient length affect the probabilities to favour a merge.

Newly defined parameters in Table 3.2 include $a_{tol}$, which is a threshold to check for deceleration or acceleration to match the speed of the gap and $d_{veh}$, which is the vehicle length to check for enough room in the gap to merge. The static gap-length check is a simplification of the more complicated gap-acceptance decision-making process.

### 3.4.2 Intersection Approach

The probabilistic decision making model can also be shown on an ”intersection approach/access” example, where a simple generic intersection and possible decisions of an approaching vehicle are analyzed, estimated and predicted as part of an earlier study [37]. Possible driver actions are given in Figure 3.10.
Table 3.2: Dynamic probability update conditions and corresponding change directions for the lane-change FSM model.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Updated Probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>turn signal + (</td>
<td>x_g - x_c</td>
</tr>
<tr>
<td></td>
<td>↑ S3 → S2</td>
</tr>
<tr>
<td></td>
<td>↑ S2 → S4</td>
</tr>
<tr>
<td>(x_l - x_f &lt; 3d_{veh})</td>
<td>↓ S2 → S4</td>
</tr>
<tr>
<td></td>
<td>↑ S2 → S6</td>
</tr>
<tr>
<td>(</td>
<td>x_g - x_c</td>
</tr>
<tr>
<td>+ (\ddot{x}<em>c &lt; -a</em>{tol})</td>
<td>↑ S2 → S4</td>
</tr>
<tr>
<td>(</td>
<td>x_g - x_c</td>
</tr>
<tr>
<td>+ (\ddot{x}<em>c &gt; a</em>{tol})</td>
<td>↑ S2 → S4</td>
</tr>
</tbody>
</table>

The FSM driver model, given in Figure 3.11, is developed to have as much state resolution as possible, while the significant final states are kept separated through a number of discrete speed and acceleration brackets. The model is built to fit and differentiate a list of important intersection approach scenarios. The events that trigger the discrete-state transitions, defined as functions of the estimated continuous state, are also selected to have various thresholds that can be fitted to a wide range of instances. One example is that state S1 (Go Through) transitions to the state S3 (Slow 1), when the target vehicle speed is less than 20 mph.

Using both the statistical analysis and end-to-end equalization methods described in Subsection 3.3.3, Tables 3.3 and 3.4 were populated with state-transition probabilities used as initial values in final-state prediction.

The data set used for statistical analysis consists of real-life data recorded on an experimental vehicle, driving in actual traffic in Southfield, MI. The vehicle was driven by a single driver, approaching four separate intersections, a number of times for each, and performing various maneuvers that cover the scenarios of interest.
Figure 3.10: Target (tracked) vehicle in blue, approaching an intersection with four possible choices, stop, go through, turn left and turn right. The host (observing) vehicle is marked in red.

Since the statistical analysis is based on the collected decisions of a single driver, the resultant transition probabilities can be expected to be tuned to this particular individual and his/her habits. However, it is noteworthy that the end-to-end equalized probabilities of Table 3.3 are a close approximation to the probabilities obtained statistically.

In this particular application, there are a number of binary flags in the v2v safety message, which was the main measurement used in the model. These binary flags are used to change the probabilities of relevant transitions accordingly, as mentioned in the probability update suggestion of the previous section. For instance, if “brake
Figure 3.11: The Finite State Machine model for driver behavior at intersection approach.

Table 3.3: Initial probability assignments with end-to-end probabilities equalized.

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.50</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.50</td>
<td>0.00</td>
</tr>
<tr>
<td>S2</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.285</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S5</td>
<td>0.00</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.428</td>
<td>0.142</td>
</tr>
<tr>
<td>S8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.20</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S9</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

applied” flag is set in the v2v message, the transitions toward slowing and stopping are incrementally set to have higher probabilities, while maintaining the sum of the
probabilities of all transitions from each state as one. Further details on state transition conditions, the particular trellis structure used and the dynamic probability update rules can be found in [37].

Using the FSM model, initial probability assignments and dynamic probability updates as described above, the data collected from DSRC radios on actual vehicles in traffic was processed to generate successful prediction of the driver decision/intention on intersection approach for a number of important scenarios. Two examples, where drivers approaching an intersection make different decisions, are presented in Figures 3.12 and 3.13.

In Figure 3.12, the distance of the tracked vehicle can be seen to decrease as it gets closer to the intersection, while the speed stays above 10 meters per second, but decreases as the vehicle gets closer to the intersection. The instantaneous discrete state is tracked to be going straight, slowing down and turning before exiting. The

<table>
<thead>
<tr>
<th></th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
<th>S4</th>
<th>S5</th>
<th>S6</th>
<th>S7</th>
<th>S8</th>
<th>S9</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S1</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.62</td>
<td>0.38</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S2</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S3</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S5</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.33</td>
<td>0.00</td>
<td>0.00</td>
<td>0.06</td>
<td>0.61</td>
</tr>
<tr>
<td>S7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.66</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S8</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.34</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>S9</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Figure 3.12: Continuous states (distance to the intersection, velocity), detected discrete state and predicted final state for a vehicle approaching an intersection and making a left turn.

turn maneuver is predicted on the last plot, close to the 14-second mark, 10 seconds before the actual turn.

In Figure 3.12, the speed of the observed vehicle stays above 20 meters per second through the intersection. The instantaneous discrete state of going through straight and exiting are tracked on the third plot, while the prediction stays constant at “going through straight.”
Figure 3.13: Continuous states (distance to the intersection, velocity), detected discrete state and predicted final state for a vehicle approaching an intersection and going through straight.

For these intention estimation results, the probabilities generated by the end-to-end equalization method were used, yet the initial probability assignment methodology plays a smaller role than the dynamic probability updates, as both probability sets (from Tables 3.3 and 3.4) generated almost identical results.

As seen in Figures 3.12 and 3.13, the developed estimator successfully predicted the driver decisions several seconds in advance.
After further expanding the single-vehicle decision-making model into a two-vehicle scenario safety example in the following subsection, these results will be used for safety estimation.

3.4.3 Combination of Predictions into Scenario Safety

The intersection-approach model of the previous subsection was initially developed to predict unsafe intersection approaches between two human-driven vehicles. In this subsection, the possible uses of the developed probabilistic decision making model are demonstrated through one particular driver-assistance system.

Using the probabilistic model of Subsection 3.4.2, it is possible to combine the predicted behaviors of multiple vehicles into the estimate of the overall safety of a given scenario. The developed software suite runs on DSRC radios installed on experimental vehicles and receives the information about the other vehicles over vehicle-to-vehicle communication safety messages.

Given two vehicles that are expected to communicate via V2V, the connections of the modules that form the unsafe condition estimate are given in Figure 3.14. Both the host vehicle, where the observer/user is and the observed/target vehicle generate measurements. These measurements are fed into the estimator through the internal communication bus of the vehicle on the host side, and transmitted in V2V safety messages on the target side.

Two intention estimators, one for each vehicle, run in parallel to generate the predicted intentions of the host and the target. These estimates, combined with data from the map of the environment, are interpreted by the actual safety check module.
and an estimate on the condition safety is generated, as well as warning messages and directions in case of an unsafe condition.

Since a vehicle pair within communication distance can in fact be approaching completely unrelated intersections with no interaction, the safety estimator requires knowledge of the environment. For this task, a map database processor was developed. The parser/processor can read both navigation-centric Navteq database extracts and additional data collected for ITS purposes. For the intersections of interest in the area that the tests were conducted, precise-GPS data was collected to give a better definition of the individual lanes of the roads and intersection shapes.
Once detailed map data is parsed, a graph structure is generated in the map database processor, which can be used to search for the relevant intersection for each vehicle. A separate GUI, as seen in Figure 3.15, was developed by Mr. John Martin of CITR Lab at OSU to test and verify the data returned by the database processor.

Figure 3.15: Map database viewer for detailed data.

Given the location and orientation of a vehicle, a graph search over the map representation returns the details of the intersection that the vehicle is approaching. This detail set includes the center-lines of inbound and outbound lanes, intersection center coordinates, stop-line locations as well as specific rules for the lanes such as “right-turn only”. A snapshot of the intersection details displayed on the GUI can be seen in Figure 3.16.
The observed continuous states are translated into real-time estimates of the discrete states, which in turn are used to predict the final discrete state as described in Subsection 3.3.1. Once the predicted final discrete states, or the predicted decisions of the drivers for both vehicles are at hand, the flow chart given in Figure 3.17 is utilized to estimate the overall safety of the two-vehicle scenario.

The intersections are uniquely identified via node ID numbers. These unique ID numbers are used in the safety check module to compare the relevancy of vehicles in range.

The overall operation of the safety-check mechanism is as follows:
1. The safety estimator starts running on the host vehicle and gets the unique ID number of the intersection that the host is approaching using the internal map database.
2. If there are no targets, the condition is safe. If there is a target, the ID of the intersection that the target is approaching is fetched from the same database.

3. If the intersections of the host and target do not match, the condition is safe. If the match, the velocities and distances to the intersection are used to calculate the time to intersection for each.

4. If the arrival times are further apart than a given threshold (5 seconds in this specific implementation), the condition is safe. Otherwise, intention estimates of the host and target and approach direction of the target are fetched.

5. If the intention estimates do not intersect, according to a decision lookup table, the condition is safe. If they intersect, declare unsafe condition and issue warning with target approach direction. The lookup table uses the geometric approach direction of the observed vehicle, as seen in Figure 3.10 and the combination of the predicted decisions.

6. Keep track of vehicles exiting their current intersections using the current state of the vehicle FSM and fetch new intersection as needed.

For the intention safety check, the intersection is divided into four quadrants, as seen in Figure 3.10, and the approaching target vehicle is classified into one of four cardinal directions (from left, from right, from ahead (head-on) and same as host), from the perspective of the host vehicle. This classification is also listed on Table 3.5, and the combination of target direction, target intention and host intention result in either “Safe” or “Unsafe” condition estimates. For instance, an example target vehicle approaching from right and turning left would result in a crash for a host...
vehicle going straight through. The same target would be safe if the host vehicle
is turning right. This table representation of combinations of intentions was first
discussed by Y. Mochizuki of Honda R&D in a simpler form and it was expanded to
account for approach directions.

Table 3.5: Combination of intentions for a vehicle pair, based on predicted inten-
tions (Stop, Go through, Left turn, Right turn) and approach direction. Unsafe
combinations are marked with U, while safe ones are marked S.

<table>
<thead>
<tr>
<th>Approach</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>G</th>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>Same</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Ahead</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>S</td>
<td>S</td>
<td>U</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>S</td>
<td>U</td>
<td>S</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>S</td>
<td>S</td>
<td>U</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Left</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>S</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>S</td>
<td>U</td>
<td>U</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Right</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>S</td>
<td>U</td>
<td>U</td>
<td>U</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>L</td>
<td>S</td>
<td>U</td>
<td>U</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td></td>
<td>R</td>
<td>S</td>
<td>U</td>
<td>S</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>

Further details of implementation as well as expansion into a full driver-assistance
and safety-assessment system can be found in an earlier study [37]. Three particular
scenarios, with recorded data from actual vehicles in traffic, is given in Figures 3.18,
3.19 and 3.20.

Example 1
Tracked or “target” vehicle goes straight through the intersection, without slowing down significantly. The host vehicle is slowing down and turning left. The continuous states of both vehicles, as generated by the vehicle tracking system, are plotted versus time, as well as the predicted vehicle states and safety estimates.

The upper left plot in Figure 3.18 shows the estimated distance to the intersection over time for both the host (thin blue line) and the target (thick black line) vehicles. The upper right figure is the vehicle speed over time, in meters per second, for both vehicles and the lower two plots show the predicted state (driver intention) for both vehicles as well as the estimated condition safety due to the combination of intentions.

Figure 3.18: Example case for parallel approach, tracked vehicle is going straight through the intersection at 45mph, host vehicle is turning left.
The discontinuity in the distance plot is due to the fact that the vehicles exit their current intersection (at the point that the distance is zero) and the next intersection is fetched for each of them. For the target, the next intersection is about 400m away, while host has 650m to its next intersection.

From the intentions plot, host is predicted to turn left and the target is predicted to exit (go through), so the condition estimate is unsafe as soon as the target is started to be tracked. Once the host exits the intersection, the intersections of host and the target do not match and the condition is safe again.

Example 2

![Graphs showing distance, velocity, and other data over time for two vehicles approaching an intersection.]

Figure 3.19: Example case for parallel approach, tracked vehicle is slowing down from 45mph and stopping, host vehicle is turning left.
In the second example, seen in Figure 3.19, target vehicle slows down and stops at the stop line. The host vehicle is turning left. The continuous states of both vehicles, as generated by the vehicle tracking system, are plotted versus time, as well as the predicted vehicle states and safety estimates.

The host is approaching, slowing down and turning left; while the target is slowing down to a stop. As soon as the target is predicted to be stopping, the condition is estimated to be safe. Before that point, the target was predicted to be going through, so the initial condition estimate is unsafe.

Example 3

Figure 3.20: One vehicle (target) is going at 45mph, straight through the intersection, the other (host) is turning left.
For the last example, both vehicles slow down and turn left. The continuous states of both vehicles, as generated by the vehicle tracking system, are plotted in Figure 3.20 versus time, as well as the predicted vehicle states and the safety estimate.

Since the target approaches head-on and both vehicles turn left, the condition estimate is safe. Prediction of left turns can be observed on the state plot.

In addition to a console-only version of the estimator software, which can be ran with recorded data, the complete estimator software is also developed to have a GUI version at OSU-CITR, which runs persistently and can be hooked to real-time data or alternatively can be fed recorded data. The GUI developed by Mr. Scott Biddlestone displays the intention prediction for the host and the target as well as continuous estimates on the host and the target such as distance, speed and time to intersection.

The central portion of the GUI can be set to display either the map of the intersection with vehicle positions or it can display the driver intention model FSM, with current end predicted states for both the host and the target, as seen in Figure 3.21.

In Figure 3.21, the target vehicle is currently in “go through” state with a “exit” prediction, as displayed in the upper halves of the state bubbles; while the host is currently in “slow 1” and is predicted to “turn left”, as displayed in the lower halves of the state bubbles.
Figure 3.21: Software for intention and unsafe condition estimators. FSM display of the GUI is shown.
Chapter 4: Discrete-State Encoding in Hierarchical Hybrid-State Systems

4.1 Finite-State Machine Encoding and Norms over Hybrid-State Domains

As discussed through this entire study, the hybrid-state systems combine a plant with continuous-states and a discrete-event system with abstract states. The hybrid domain of the combined state makes it more difficult to define relations between or operations on two different states, as the discrete states do not correspond to real-valued scalars or vectors without mappings or transformations. Conventional continuous-state systems, on the other hand, can use all the existing algebraic operations and definitions to improve the analytical theory on CSS behavior.

One basic operation that would be useful in the hybrid domain is a simple norm, yet it is not as straightforward as the continuous counterpart. The definition of a norm in the continuous case enables the distance definition, which in turn leads to invariance, domains of attractivity and stability definitions for control theory. Without the distance between two points in the state space, the time-evolution of a system cannot be analysed for convergence or divergence.
Similar norm developments for discrete-event and hybrid-state systems would let us analytically define inter-state distance, metrics on error and performance and would eventually lead to better mathematical constraints of hybrid-state system behavior.

Traditionally, the need for a norm over discrete-event systems and automata was filled via graph-based approaches. Since the discrete states of a finite-state machine do not have inherent relations such as one being greater than other, the constructed relations between them, namely the state transitions, were used as means to define distances. The number of transitions or “hops” between two discrete states is one simple distance measure, yet as the graph structures of FSMs get more complex, and multiple paths between two discrete states become commonplace, more complex graph searches are required, not to mention the need to refresh the entire set of distances when a state transition is added or removed.

In this chapter, the foundations for a norm definition over the discrete states of a hybrid-state system is developed and demonstrated. The method relies on the interaction between the discrete and continuous states and it is independent of the discrete-state graph topology.

4.1.1 Hybrid-State Systems

As it was given in Section 2.2.2, $X \in \Sigma = [X_1, X_2, ..., X_N]$ is the discrete state of the single-state FSM in the DSS part of the HSS, with system $F$ providing the state $X$ as output to the interface, and $x = [x_1, x_2, ..., x_n]^T \in \mathbb{R}^n$ is the real-valued, n-dimensional vector of states for the continuous-state plant.

The combined state of the HSS, $[X, x]^T \in \Sigma \times \mathbb{R}^n$ in the hybrid-state domain does not inherently have an intuitive norm, as discussed above. The continuous
states can utilize Euclidean or $L^2$ norm, just as well as $L_{\infty}$ norm or any other norm conventionally defined over real-valued vectors.

The discrete states, on the other hand, do not have the necessary operations over their domains. The addition or subtraction of two discrete states in $\Sigma$ are not normally defined, unless the application calls for a very specific definition that only works for the particular example. On top of these limitations, any concept of “distance” between two discrete states is confined to the graph topology of the finite-state machine.

Under the assumption of static graph topologies, where state transitions are set/designed.MODELED once and never change, using the number of transitions between two discrete states as the distance between these two states leads to a usable metric of distance. However, the state transitions in a FSM does not need to be bidirectional, which may lead to non-symmetric distances between two states. In addition, complex FSM structures require graph searches to find the minimal number of transitions between distant states.

Due to the complications mentioned above, this chapter attempts to define a mapping from the discrete states of an HSS to a known, conventional vector field in order to utilize conventional norms. The developed method, utilizing the interaction between the discrete and continuous states, exploits the fact that the discrete-state systems that are of interest to this study do not exist in isolation, but are hierarchically coupled with a continuous-state system.

4.2 Command-Based Codes

The discrete-state portion of the HSS is not inherently different from any other discrete-state system (DSS). The existing set of tools for coding and analysis of DSS
still apply, yet the interaction of the DSS with the continuous states can be exploited for further insight. Command-based codes for the discrete states are developed with this idea as the basis, as described in the following sections.

4.2.1 Definitions

Definition 4 A command to the continuous-state system, as part of a hybrid-state system, is part of the discrete-to-continuous signal $S$, issued to provide the reference input to the continuous controller for a one-dimensional continuous substate, $x_i, i \in [1,n]$ of the continuous-state plant, according to the requirements of the discrete state of the hierarchically higher level.

Each substate of the continuous plant state $x \in \mathbb{R}^n$ can potentially receive a command, but not all substates are controllable by the continuous controller.

A command can be sent to one substate of the continuous plant, for which the discrete-state controller can set the reference value; or it can be sent to one substate that the discrete-state controller can select the direction of change (set the sign of the first derivative), therefore controlling through the continuous controller if that substate will increase, stay constant or decrease.

Theorem 5 A command can be sent to substate $x_i \in \mathbb{R}$ of the continuous-state plant in an HSS controller if $x_i$ is a continuous state of the system $v(x,y)$ that is controllable by the input $y$; and $y$ is the state and output of the system $f(y,x,u,S)$, that is controllable by the input $S$ from the discrete-state level.

Proof: The controllability requirement of continuous substate $x_i$ by input $y$ in system $v(x,y)$ directly follows from the command definition in Definition 4. If the
continuous output and state \( y \) of system \( f(y, x, u, S) \) is controllable by the output 
\( S \) of the interface \( \Psi \), the second part of the command definition, which requires the 
control of \( x_i \), by \( y \), according to \( S \), is satisfied.

By design, possible commands that the discrete states can send via \( S \) for the 
same continuous substate are mutually exclusive. Increasing a continuous state and 
decreasing it cannot occur at the same time. Each discrete state that controls a par-
ticular continuous substate picks one command from the list of all possible commands. 
The command-based encoding scheme utilizes the combinations of these commands, 
issued by each discrete state.

**Definition 5** The code or the encoding scheme is a **one-to-one mapping** (an in-
jection) from the finite discrete states of the DSS to a finite set of nonnegative-integer-
valued vectors of length \( M \), \( M \) being the number of controllable substates among the 
continuous plant states.

Each discrete state is assigned a single vector of nonnegative integers to identify 
that state, and no vector in the coding scheme is used more than once. 
A random assignment of integers (or vectors of integers of arbitrary length) to 
discrete states would also form an encoding scheme by the definition above, if the 
number-state correspondence is one-to-one. However, a numbering/encoding scheme 
that purposefully captures certain properties of the states or the overall system is 
inherently more useful and it is the main focus of this study.

The earlier methods developed to compare nuances of code assignments such as 
[5], in which the author compares the “pr mappings” as mentioned in Chapter 1, 
and [28], where efficiency comparisons are based on physical implementation of the
sequential machine are applicable to compare the merits of resultant codes for hybrid-state systems. However, the initial selection of the code represents the overall system in finer detail when both the discrete states and their interactions with the continuous states are considered, and this level of detail is achieved by going above the absolute minimum number of digits/bits in the code space and losing the strict implementation efficiency of the studies mentioned above.

4.2.2 Encoding Scheme

The assignment of Command-Based Codes, in the sense of the word “code” that was described above, starts by identifying the set of states of the continuous subsystem that are dependent on the discrete subsystem as defined in Definition 4. This dependence, as described in Subsection 4.1.1, is through the discrete-to-continuous interface $\Psi$ and the continuous controller $f$.

The overall methodology of identifying and assigning command-based codes is given in the following steps:

1. Identify the continuous states that the DSS can control through the interface $\Psi$ and the continuous controller $f$.

2. List the number of possible commands/selections that are available to the DSS for each controllable continuous state.

3. For each controllable continuous substate, number the possible commands in order as nonnegative integers. These are the possible codes that each discrete state chooses from.
4. For each discrete state, combine the codes of the commands that the particular state issues to the continuous subsystem as a vector.

The codes generated by this method are based on how the discrete states individually interact with the continuous states. Hence, the codes do not depend on the state transition function $F(X, U, s)$. As listed on the above summary, the actual conditions for state transitions are not used. The same set of discrete states can be connected in any graph topology, and the same codes will apply as long as there are no changes to what each state does.

If the continuous state is $x(k) = [x_1(k), x_2(k), ..., x_n(k)]^T$, a number of these $n$ continuous substates are independent of the discrete state $X(k)$. The others, through their controllability by input $y$, $y$ being a function of $S$ and $S$ being a function of $X$, are controlled in accordance with the discrete state $X$. Each of these controllable continuous substates is a command recipient.

Once the continuous states that can receive commands are identified, the possible commands that the discrete-state system can send to these continuous-state system are listed, individually for each state. Each command in a given list of commands for a particular state gets a nonnegative integer that uniquely identifies the command within the set of commands for that continuous state.

After all the states that can receive commands are identified and all possible commands are listed and numbered, each discrete-state in the DSS can be encoded with the vector of individual command codes in a selected order, based on which of the listed commands that the particular discrete state issues.
Command-based encoding scheme $C$ is a mapping from the discrete-state space of the HSS to the set of nonnegative-integer-valued vectors,

$$X_i \xrightarrow{C} [c_1, c_2, \ldots, c_M]^T,$$

$$c_j \in \mathbb{Z}^*, \forall X_i \in \Sigma, i \in [1, N], j \in [1, M],$$

where $M$ is the number of controllable continuous substates of the HSS among $x \in \mathbb{R}^n$, with $M \leq n$ and $c_j$ is the code piece corresponding to the command that state $X_i$ issues for the $j^{th}$ controllable substate, for $j \in [1, M]$.

Figure 4.1: Two dimensional discrete-state space spanned by two controllable substates of $2^m$ and $2^{n-m}$ commands.

**Theorem 6** The discrete-state space is mapped into an $L$-dimensional vector space, where $L$ is the number of command blocks, through state encoding based on the commands/behaviors that can be generated by the discrete-state level.

**Proof:** A vector space $V$, by definition [12], over a field $G$, together with the sum and multiplication binary operators, holds the following properties:
• Associativity of addition: \( \mathbf{v}_1 + (\mathbf{v}_2 + \mathbf{v}_3) = (\mathbf{v}_1 + \mathbf{v}_2) + \mathbf{v}_3 \),

• Commutativity of addition: \( \mathbf{v}_1 + \mathbf{v}_2 = \mathbf{v}_2 + \mathbf{v}_1 \),

• Identity element of addition: \( \mathbf{0} \in V \),

• Inverse element of addition: \( \forall \mathbf{v} \in V, -\mathbf{v} \in V : \mathbf{v} + (-\mathbf{v}) = \mathbf{0} \),

• Distributivity of scalar multiplication over vector addition: \( g(\mathbf{v}_1 + \mathbf{v}_2) = g\mathbf{v}_1 + g\mathbf{v}_2, g \in G \),

• Distributivity of scalar multiplication over field addition: \( (g_1 + g_2)\mathbf{v} = g_1\mathbf{v} + g_2\mathbf{v}, g_1, g_2 \in G \),

• Respect of scalar multiplication over field multiplication: \( (g_1g_2)\mathbf{v} = (g_1g_2)\mathbf{v} \),

• Identity element of scalar multiplication: \( 1\mathbf{v} = \mathbf{v} \).

The field of integers, \( \mathbb{Z} \), with integer addition and multiplication, forms the integer-valued vector field, with the zero vector being the additive identity element and integer 1 being the multiplicative identity element. Therefore, by going from the discrete-state space onto the vectors of nonnegative integers \( (\mathbb{Z}^+) \), which is a subset of integers, encoding scheme introduced is a mapping into a vector field.

An \( n \)-bit code with two command sets can represent all \( 2^n \) combinations of the two commands, as shown in Figure 4.1. Even if a particular combination of commands/behavior are never used in a given FSM controller, the corresponding discrete states already have codes in the given scheme, independent of the FSM graph connectivity and topology.

With the four-step method defined above, all the possible combinations of commands that can be issued by the discrete level are captured in the code. Each discrete
state in the DSS gets a vector of nonnegative integers based on which command combination that particular state uses. Some combinations of commands, therefore some codes, will not be used in a given HSS implementation. These unused codes may correspond to not useful or unsafe combinations of commands (infeasible states), or they can correspond to possible/safe, yet not needed combinations of commands (feasible states that were not used).

Since the code space captures all possible states under the DSS/CSS structure, the discrete-to-continuous interface, $\Psi$, becomes a simple decoder of discrete state codes into commands.

Both inputs ($U$ and $s$) can be defined as combinations of binary events as the inputs into the discrete tier are discrete and abstract. State-transition events can be defined as binary variables over external/internal continuous variables and thresholds or they can be inherently binary by definition.

These binary events, fed as inputs to the discrete-event system do not need to be grouped, as they are naturally binary and independent, so “external event 0” can be captured in the least significant bit of $U(k)$ and “internal event 0” can be captured in the least significant bit of $s(k)$, while the higher order bits carry the remaining events. The command-based codes for discrete states can also be represented in binary as they are vectors of nonnegative integers.

As all the possible states and inputs of the discrete-event system can be represented in binary, a truth table is sufficient to represent the entire finite state machine (FSM). Further binary logic tools such as K-maps[34] can also be utilized for embedded controller design for these systems, but the application of such tools is a straightforward extension, and thus beyond the scope of this study.
Truth tables can be visually less intuitive than graph-based FSM representations. The use of command-based codes, however, does not necessarily eliminate graph-based design methods from the overall system development. One can work on the initial design of the FSM via a state transition graph, and calculate the codes for each state if the need for the codes arises. One implementation can be seen in Section 4.3, along with further demonstrative examples. The binary representation is not a requirement of the actual encoding scheme, but the nonnegative nature of the codes combined with the binary nature of most external detection events lead to the tools of binary logic being applicable after encoding.

As mentioned at the beginning of this chapter, the command-based codes can be used to define a norm over the discrete-state space, which in turn can be used in HSS analysis as discussed in [22]. Unless the discrete states are naturally defined on vector spaces, such as each state being a partition of Cartesian plane, the mapping onto command-based codes, therefore onto M-dimensional nonnegative-integer-valued vectors enables the definition of distance relations between them.

A norm, by definition given in [31], holds the following properties:

\textbf{Definition 6} Given a vector space \( V \) over a subfield \( G \) of the complex numbers, a norm on \( V \) is a function \( p : V \to \mathbb{R} \) with: \( \forall a \in G \) and \( u, v \in V \),

1. Positive homogeneity: \( p(au) = |a|p(u) \),

2. Triangle inequality: \( p(u + v) \leq p(u) + p(v) \),

3. If \( p(v) = 0 \) \( \Rightarrow v \) is the zero vector.
Given a Hybrid-State System with $M$ controllable continuous substates ($M \leq n$, $n$ being the number of dimensions of the continuous state $x$), and the command-based code of a discrete state as an $M$-dimensional non-negative integer vector, $c = [c_1, c_2, ..., c_M]^T \in \mathbb{Z}^*$, where $c_i$ is the code piece for the command that the particular state issues for controllable continuous substate $i$,

**Theorem 7** $L^2$ distance over vector field $(\mathbb{Z}^*)^M$ defines a norm over the discrete states of the HSS.

**Proof:** Since $c$ is a vector of nonnegative integers, by Definition 5, Euclidean, or $L^2$ norm satisfies the positive homogeneity, triangle inequality and zero-vector requirements, as the same norm satisfies the same requirements on the set of integers ($\mathbb{Z}$); and nonnegative integers $\mathbb{Z}^*$ as a subfield ($G$) is closed under multiplication and addition that the requirements call for.

Given two codes, $c$ and $d$, the Euclidean distance is used as follows:

$$
||c|| = \sqrt{c_1^2 + c_2^2 + ... + c_M^2},
$$

$$
||c - d|| = [(c_1 - d_1)^2 + (c_2 - d_2)^2 + ... + (c_M - d_M)^2]^{1/2}.
$$

The mapping of states onto $m$-dimensional integer vectors lets us use the well-defined Euclidean norm in a straightforward manner.

### 4.3 State Encoding Examples

In this section, two simple examples on control and estimation of hybrid-state systems will be used to demonstrate the command-based coding scheme described in the previous section.
4.3.1 Hybrid-State Controller Development and Representation in Codes

A simple advanced cruise control (ACC) example for an autonomous vehicle controller can be constructed as follows.

The vehicle model is a one-dimensional point mass model with first order friction:

\[ m \ddot{x} = -a \dot{x} + f_{in}, \]  

(4.2)

where \( m \) is the mass of the vehicle, \( a \) is the friction coefficient, \( f_{in} \) is the input force and \( x \) is the position of the vehicle.

The discrete-to-continuous interface, \( \Psi \), generates the input \( f_{in} \) to the continuous system. The discrete-state system is an FSM with three possible states, “accelerate”, “decelerate” and “maintain speed”, as shown in Figure 4.2. The behavior of the ACC can be summarized as:

- If there are no vehicles ahead, maintain the speed input by the user,
- If there is a slower vehicle ahead, slow down to match speed.

Figure 4.2: Discrete states for simple ACC example.
The sensor subsystem of the ACC generates the following inputs to be used in the discrete controller, assuming the sensing range of the vehicle being $D$ meters: “there is another vehicle in front of us, within $D$ meters”, “the vehicle in front is slower than this vehicle”, “the vehicle in front is faster than this vehicle”, “this vehicle is below the set speed from user input”. These inputs can be considered as a mixture of external input $U$ shown in Figure 2.3 and the internal signal $s$, as they combine external sensors and the state of the vehicle.

In order to minimize oscillations and fast switching, the comparisons listed above are assumed to have built-in hysteresis, such that “the vehicle in front is faster than this vehicle” event is not triggered unless the speed of the lead vehicle is $l$ mph higher than the self vehicle.

The inputs defined above can be coded in four bits in a straightforward manner, as seen in Figure 4.3. The number of bits can be reduced, but the event-to-bit correspondence is generally more intuitive for debugging purposes.

\[
[U(k), s(k)] = 0 \ 0 \ 0 \ 0
\]

Figure 4.3: Input bits for simple ACC example.
Since there are three possible commands within the same continuous substate (the acceleration of the vehicle), two bits and a single command block are needed, seen in Table 4.1.

<table>
<thead>
<tr>
<th>Code</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Maintain speed (cruise)</td>
</tr>
<tr>
<td>01</td>
<td>Accelerate</td>
</tr>
<tr>
<td>10</td>
<td>Decelerate</td>
</tr>
</tbody>
</table>

The discrete-to-continuous interface, as a simple state decoder, is given in Table 4.2.

In order to keep the design simple and illustrative, the detection range \( D \) is assumed to be large enough to react to slower/stopped vehicles with the above listed input forces. In real applications, these input forces need to be functions of the lead vehicle speed and the instantaneous headway in order to guarantee safe operation.

<table>
<thead>
<tr>
<th>State</th>
<th>( S(k) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>( f_{in} = f_{friction} = a\dot{x} )</td>
</tr>
<tr>
<td>01</td>
<td>( f_{in} = 2f_{friction} = 2a\dot{x} )</td>
</tr>
<tr>
<td>10</td>
<td>( f_{in} = -f_{friction} = -a\dot{x} )</td>
</tr>
</tbody>
</table>

Under all the assumptions and descriptions listed above, the state-transition table given in Table 4.3 was generated to capture the desired FSM behavior with command-based coding of the states.
Since the states and input events are represented over a total of six bits, two for the state and four for the input, there are $2^6 = 64$ possible combinations for these bits. Therefore, the complete truth table includes 64 rows. Table 4.3, on the other hand, only lists the eleven significant rows, in the sense that these are the only combinations of states and input events that result in state transitions. The omitted lines would have the same state on both $X(k)$ and $X(k + 1)$ columns, as those particular state-input combinations do not change the state of the DSS.

Table 4.3: ACC example state-transition table with command-based coding of discrete states.

<table>
<thead>
<tr>
<th>X(k)</th>
<th>[U(k),s(k)]</th>
<th>X(k+1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0011</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>1000</td>
<td>01</td>
</tr>
<tr>
<td>00</td>
<td>1011</td>
<td>10</td>
</tr>
<tr>
<td>00</td>
<td>1101</td>
<td>01</td>
</tr>
<tr>
<td>01</td>
<td>0000</td>
<td>00</td>
</tr>
<tr>
<td>01</td>
<td>0011</td>
<td>10</td>
</tr>
<tr>
<td>01</td>
<td>1011</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>0001</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>1001</td>
<td>00</td>
</tr>
<tr>
<td>10</td>
<td>1101</td>
<td>01</td>
</tr>
</tbody>
</table>

As discussed in Subsection 4.2.2, the codes assigned to inputs and states up to this point are independent of how the FSM transitions worked. The state codes were based on the continuous substate(s) they control, and the transition conditions defined in the truth table can be changed without reassigning codes to each discrete state.
4.3.2 Performance Metric for Hybrid-State System Estimator Using Code-Based Norm

An HSS estimator for driver/vehicle modelling and prediction was developed and demonstrated in the previous chapter. As in most estimation applications, a metric to measure the success of the estimator is useful. As discussed above, the non-negative integer vectors of the command-based codes can be represented in any numeral system, and this particular example will demonstrate the decimal representation.

A simple approach that was earlier utilized is to check if the estimated discrete state and the actual discrete state are the same. When $X(k) = \hat{X}(k)$ holds, $X(k)$ being the actual state of the DSS in an HSS, and $\hat{X}(k)$ being the estimate, it is obvious that the estimation is successful.

However, the amount of error is not easy to measure when the estimation and actuality differ. Defining $\|X(k) - \hat{X}(k)\|$, when $X(k) \neq \hat{X}(k)$ is non-trivial.

Usual methods utilize the graph connectivity of the discrete states, yet the single-directional state transitions do not comply with the symmetry requirements of a formal norm between discrete states. If $X_1$ transitions into $X_2$, but if the transition in the opposite direction is not possible, claiming $\|X_1 - X_2\| = \|X_2 - X_1\|$ is not very intuitive.

Command-based codes provide a method to assign values to each discrete state independent of such connectivity issues, as discussed in Subsection 4.2.2, which leads to a measure of distance as defined in the same subsection, between any given state pair that is useful in estimation performance metrics.
The FSM-based driver model in Figure 4.4 was developed to capture a number of intersection approach scenarios that resulted in fatal traffic accidents. As such, the model does not cover every possible scenario.

The main focus is on speed profiles of an observed vehicle, since the speed is both easily observable via sensors or vehicle-to-vehicle communication, and indicative of the general behavior of the driver.

![FSM model for driver/vehicle estimation for intersection approach.](image)

**Figure 4.4:** FSM model for driver/vehicle estimation for intersection approach.

Investigating the possible behaviors that the model captures, the overall span of the discrete states can be represented via four separate controllable continuous states: “Longitudinal Location”, “Speed”, “Acceleration” and “Steering.”

As defined in Subsection 4.2.2, the continuous states that the DSS can represent are:

\[ x(k) = [x_l, \dot{x}_l, \ddot{x}_l, \dot{\theta}]^T, \]
Table 4.4: Command blocks and possible behaviors for each controllable continuous state in the FSM-based driver model.

<table>
<thead>
<tr>
<th>Location</th>
<th>Speed</th>
<th>Acceleration</th>
<th>Steering</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out of range</td>
<td>0</td>
<td>Decel.</td>
<td>Turn Left</td>
</tr>
<tr>
<td>In range</td>
<td>1</td>
<td>Cruise</td>
<td>Straight</td>
</tr>
<tr>
<td>Past intersection</td>
<td>2</td>
<td>Accel.</td>
<td>Turn Right</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slow III</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stopped</td>
<td></td>
</tr>
</tbody>
</table>

where \( x_l \) is the longitudinal position and \( \theta \) is the yaw of the observed vehicle. The remaining continuous states of the vehicle are not significant enough to be represented in the discrete-state abstraction model, therefore they do not have command blocks by above definition.

The command blocks and the commands that the DSS can send for each controllable continuous state are listed in Table 4.4. Each command/abstraction over a controllable state is represented as decimal nonnegative integers, as described earlier.

Using these command blocks in the order given, each state in Figure 4.4 can be assigned a unique command-based code vector, as seen in Table 4.5.

These codes can directly be used to measure the distance between the actual state and the estimate calculated by the HSS estimator. Using the norm defined in Subsection 4.2.2, the following example distances between states are calculated,

\[
\|S_7 - S_8\| = \sqrt{((0 - 0)^2 + (2 - 3)^2)} + (0 - 0)^2 + (1 - 1)^2}^{1/2}
\]

\[
= 1,
\]

\[
\|S_7 - S_9\| = \sqrt{((0 - 0)^2 + (2 - 3)^2)} + (0 - 1)^2 + (1 - 2)^2}^{1/2}
\]

108
Table 4.5: Code assignments for each state in the HSS driver/vehicle estimator FSM.

<table>
<thead>
<tr>
<th>State #</th>
<th>Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>Out of range</td>
<td>[0,0,1,1]</td>
</tr>
<tr>
<td>S1</td>
<td>Go through</td>
<td>[1,0,1,1]</td>
</tr>
<tr>
<td>S2</td>
<td>Accelerate</td>
<td>[1,0,2,1]</td>
</tr>
<tr>
<td>S3</td>
<td>Slow I</td>
<td>[1,1,0,1]</td>
</tr>
<tr>
<td>S4</td>
<td>Exit</td>
<td>[2,0,2,1]</td>
</tr>
<tr>
<td>S5</td>
<td>Stop</td>
<td>[1,4,1,1]</td>
</tr>
<tr>
<td>S6</td>
<td>Turn Left</td>
<td>[1,2,1,0]</td>
</tr>
<tr>
<td>S7</td>
<td>Slow II</td>
<td>[1,2,0,1]</td>
</tr>
<tr>
<td>S8</td>
<td>Slow III</td>
<td>[1,3,0,1]</td>
</tr>
<tr>
<td>S9</td>
<td>Turn Right</td>
<td>[1,3,1,2]</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\|S7 - S6\| &= [(0 - 0)^2 + (2 - 2)^2]^{1/2} \\
&= \sqrt{3}, \\
&= \sqrt{2}.
\end{align*}
\]

Interpreting the above distance measures, if the estimate is \( \hat{X}(k) = S8 \) when the actual state is \( X(k) = S7 \), the estimation error is 1 unit, while estimating \( \hat{X}(k) = S6 \) gives an error of \( \sqrt{2} \) units.

Comparing these results to the simple number-of-state-transitions distance between states by using Figure 4.4, both \( S6 \) and \( S8 \) are one transition away from \( S7 \), yet the command-based codes captured the difference between “wrong estimation of speed” and “wrong estimation of speed and yaw rate”.

109
Chapter 5: Contributions and Future Work

This dissertation presents the discussion on hierarchical controllers and controller design, driver intention estimators and scenario interpreters, and analysis tools for autonomy in mixed-mode traffic, utilizing the hybrid-state system models.

The system architecture and design approaches were analysed for hierarchical mobile-robot control and illustrated on the latest iterations in a series of autonomous ground vehicles and robots developed in The Ohio State University. The expansion of the inherent hierarchy of an HSS to a higher level of meta-states or situations, which allows a compartmentalized and step-by-step development strategy led to the capability grafting methods.

Both the modular layout of the general system and the hybrid-state nature of the controller branch are structured to be highly analogous to a number of non-autonomous systems. The hierarchy derived from and expanded on these analogies are twofold, as it represents itself both in the modular connections and tasks, and in the highest level control for the more abstract decision-making process.

A controlled development procedure for designing certain aspects of such a hierarchy was also presented with examples. The performance of a real-life system, OSU-ACT, designed under listed guidelines was tested in DARPA Urban Challenge 2007, where it was one among the safest and most successful semifinalists. Similarly
developed controllers for mobile ground robots were used for Team Cappadocia in MAGIC 2010, which led to the world finals in Australia. A point of major importance when designing a capability-based FSM is preparing for most possible situations and scenarios one expects to face, and a stage-by-stage development scheme enables the addition of such contingencies.

The performance of proposed controller architecture, implemented for and tested in a number of applications, was satisfactory within the realm of capabilities as demonstrated through examples.

For the situational awareness of the autonomous vehicles, a probabilistic model for the decision-making module in the human driver model was developed and used in a hybrid-state estimation architecture to capture driver behavior and predict intention.

The parallels between the hybrid-state system hierarchy (HSS) and the interaction between the decision-making and implementation modules of the HDM were utilized to define observable discrete-states through observability of the continuous states. Finite-state machine design guideline for the proposed modelling methodology were discussed and example models were given to demonstrate the ideas.

In the final chapter, an encoding scheme that is uniquely useful in Hybrid-State Machines was developed. The hierarchy of the discrete states over continuous subsystems was exploited to define discretely-controllable continuous states of the overall system, where each controllable continuous state provides a dimension of separability in the discrete-state space.

In an HSS setting, there is more information to be used than the isolated DSS, namely the interaction of the DSS portion with the CSS. Using this information to identify and encode the discrete states independent of the state transitions, it was
possible to capture the interactive roles of each discrete state, which in turn was useful in defining a norm over them.

The resultant methodology and the command-based codes were demonstrated on two examples, including a simple autonomous vehicle controller and an HSS estimator. The codes were shown to be independent of the FSM state transitions and connectivity, which is interpreted as an advantage over graph-topology-based methods in the sense that the codes are less likely to change under minimal design adjustments to the FSM.

Possible future extensions of this study include investigating the actual interactions between mobile agents of varying levels of autonomy and how these interactions affect scenario estimation; expanding the capability grafting methods into a full set of fundamental FSM operations including removal of states; and linking the design and analysis methods developed in this work to verification and synthesis tools for hybrid-state systems.
Bibliography


116


