QUANTITATIVE MEASURE OF REGULAR LANGUAGES FOR
SUPERVISORY CONTROL OF ENGINEERING APPLICATIONS

A Thesis in
Mechanical Engineering
by
Xi Wang

© 2003 Xi Wang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2003
The thesis of Xi Wang was reviewed and approved* by the following:

Dr. Asok Ray  
Distinguished Professor of Mechanical Engineering  
Thesis Advisor, Co-Chair of Committee

Dr. Shashi Phoha  
Professor of Electrical Engineering  
Co-Chair of Committee

Alok Sinha  
Professor of Mechanical Engineering

Marc Carpino  
Professor of Mechanical Engineering

John Metzner  
Professor of Computer Science and Engineering

Richard C. Benson  
Professor of Mechanical Engineering  
Head of the Department of Mechanical Engineering

*Signatures are on file in the Graduate School.
Abstract

This dissertation formulates a signed real measure of sublanguages of regular languages based on the principle of automata theory and real analysis. The measure allows total ordering of any set of partially ordered sublanguages of a regular language for quantitative evaluation of the controlled behavior of deterministic finite state automata (DFSA) under different supervisors. The computational complexity of the language measure algorithm is polynomial in the number of DFSA states. An online parameter identification procedure is presented for computation of the language measure parameters.

A discrete event behavior-based multi-robot system has been designed and constructed to validate the language measure theory and its applications to supervisory control in the discrete-event setting. Each robot is equipped with multiple sensors and multiple actuators. The interactions between the robot(s) and the (possibly) dynamically changing environment are characterized by discrete-event and continuous models, and the design and analysis of the robotic system are presented in both continuous-time and discrete event domains. The robustness and reliability of the controlled behavior is guaranteed in the continuous-time domain. For example, visual servoing is applied to robot navigation during ‘approaching target’ and the vector field historgam (VFH) method is used for robust ‘obstacle avoidance’. The discrete event interactions between behaviors are formulated as a supervisory control theory problem, where multiple supervisors are synthesized and implemented online for robot control under different specifications. The efficacy of the language parameter identification procedure is demonstrated in real-time supervisory control through experiments on the mobile robotic system as well as on a high-fidelity robot simulator. A quantitative performance measure has been used to evaluate various discrete event supervisory (DES) controllers and is validated through experiments. The performance measure is consistent with other heuristic measures in the evaluation of the robotic system performance.

An optimal DES controller synthesis procedure based on the language measure theory is presented and applied to the automated design of DES control polices.
for different experimental scenarios of the mobile networked robotic system. The simulation results validate the optimal synthesis procedure. For a hierarchically structured cooperative multi-robot system, it turns out that the language measure theory is also applicable for design and performance analysis of the high level coordination.
# Table of Contents

List of Figures viii

List of Tables x

List of Symbols xi

Acknowledgments xii

1 Introduction 1

1.1 Models of computation ........................................ 1
  1.1.1 Notation ............................................... 1
  1.1.2 Continuous time systems ................................. 2
  1.1.3 Discrete event systems ................................. 4
  1.1.4 Hybrid systems ......................................... 5
  1.1.5 Limitations of continuous models ....................... 8

1.2 Fundamental problems in systems under different model paradigms 9

1.3 Supervisory control of DESs .............................. 12

1.4 Proposed solution ........................................ 14

1.5 Implementation issues in modeling of DESs ............... 15

1.6 Thesis outline and contribution .......................... 17

2 Performance Measure of Discrete Event Systems 20

2.1 Introduction ............................................... 20

2.2 Qualitative Measure of Discrete Event Supervisors ....... 21
  2.2.1 Algebraic Preliminaries .............................. 21
  2.2.2 Qualitative Design of Discrete Event Supervisors .... 23

2.3 Quantitative Measure of Discrete Event Supervisors ....... 26
  2.3.1 Problem Formulation .................................. 26
  2.3.2 Basics of Measure Theory ............................. 27
  2.3.3 Quantitative Measure of Regular Languages .......... 28
## List of Figures

1.1 A piecewise-continuous behavior $\beta$ and a piecewise-constant behavior $\beta_1$
with $\beta_u = q_1, q_2, q_3, q_4, q_5$ ........................................... 2
1.2 Continuous time systems ...................................................... 3
1.3 Discrete event systems ......................................................... 5
1.4 Hybrid systems ................................................................. 7
1.5 Standard DES modeling and implementation ............................... 17
1.6 DES modeling with the separation of $\Sigma_c$ and $\Sigma_u$ .................. 18
1.7 Thesis Outline ................................................................. 19

2.1 Example 1 ........................................................................ 41
2.2 Finite State Automaton Model of the Plant .................................. 48
2.3 Controller 1 for Specification #1 ............................................ 48
2.4 Controller 2 for Specification #2 ............................................ 49
2.5 Controller 3 for Specification #3 ............................................ 49

4.1 DES behavior based robotic system architecture .......................... 65
4.2 Robotic system lab and Pioneer 2 AT specifications ......................... 67
4.3 System identification of the Pioneer 2 AT robot ............................ 69
4.4 Robot configuration in experiment ............................................. 70
4.5 Sensor readings in experiment ................................................ 70
4.6 Robots configuration in simulator ............................................ 70
4.7 Sensor readings in simulator .................................................. 70
4.8 (a) Model of Pioneer 2 AT robot; (b) The side-view of P2AT with
a camera facing downward at a tilt angle $\phi > 0$ ............................. 73
4.9 The closed-loop vision-guided navigation system for the Pioneer 2
AT robot ........................................................................ 74
4.10 Experimental validation of robot visual servoing ............................ 77
4.11 Object detection. (a) ACTS color blob detection (overlay mode)
(b) ACTS color blob detection (background subtraction) (c) Color
blobs interpreted by Player ................................................... 78
4.12 DES supervisory control of Pioneer 2 AT (behaviors: \textit{approach, reach, grab}) ........................................ 78
4.13 Some samples of SICK LMS200 laser range finder ................. 82
4.14 Samples in Figure 4.13 after filtering .................................. 83
4.15 One measurement ...................................................... 83
4.16 Simple histogram ..................................................... 83
4.17 VFH real experiment .................................................. 84
4.18 VFH simulation .......................................................... 84
4.19 Trajectory in experiment ............................................... 84
4.20 Trajectory in simulation ................................................ 84
4.21 A trajectory of the real robot in experiment ......................... 85
4.22 DES plant model $G$ for the experiment scenario .................... 85
4.23 DES supervisor $S$ for the experiment scenario ...................... 86
4.24 Convergence of some non-zero $\tilde{\Pi}$ elements in experiment ........ 86
4.25 Convergence of some non-zero $\tilde{\Pi}$ elements in simulation .... 88
4.26 Cumulative performance comparison in simulation .................. 89

5.1 Robot simulation environment .......................................... 99
5.2 Robot behavioral model .................................................. 100
5.3 Battery failure model and gripper failure model ...................... 100
5.4 Some non-zero elements of $\tilde{\Pi}$-matrix ............................. 101
5.5 Cumulative performance comparison of all controllers ............... 103

6.1 DES hierarchical control structure .................................... 110
6.2 Hierarchical structure of multi robot system ......................... 113
6.3 The Mealy machine of DES plant model $G$ .......................... 113
6.4 The high level abstraction of DES plant model $G$ .................... 114
6.5 Two simulated robots without coordination .......................... 115
6.6 Real robot behaviors in coordination .................................. 117
6.7 Simulated robot behaviors in coordination ............................ 118
6.8 Some non-zero elements in $\tilde{\Pi}$-matrix of $G_{hi}$ ................... 119
6.9 Cumulative performance comparison of high level open and closed loop systems ......................................................... 120
6.10 Cumulative performance comparison of individual robot with and without coordination ........................................... 121
List of Tables

2.1 Plant Automaton States ........................................ 47
2.2 Plant Event Alphabet ........................................... 47
2.3 State Transition and Event Cost Matrix ...................... 50

4.1 CPU usage of various applications ............................ 71
4.2 The discrete event set $\Sigma$ for Pioneer 2 AT robot .......... 72
4.3 The state set $Q$ of the plant model $G$ and its $X$-vector ........ 87
4.4 $\Pi$-matrix ($17 \times 17$) for the discrete-event model $G$ ......... 87

5.1 List of discrete events ........................................... 101
5.2 Iteration of $\mu$ synthesis .................................... 101
5.3 Simulation statistics of 400 missions ........................ 102
5.4 Iteration of $\mu$ synthesis with $(\Pi, X)$ obtained in simulation .......... 104
5.5 Iteration of $\mu$ synthesis with $(\Pi, X)$ obtained in experiment .... 104
5.6 Comparison of $Q$-learning and $\mu$-selection .................... 106

6.1 The discrete event set $\Sigma$ for Pioneer 2 AT robot ............ 116
List of Symbols

\( \mathbb{N} \) natural numbers, p. 1
\( \mathbb{R} \) real numbers, p. 1
\( \emptyset \) empty set, p. 11
\( G \) a deterministic finite state automaton (DFSA), p. 5
\( Q \) discrete state space, p. 5
\( \Sigma \) the set of discrete events, p. 5
\( \delta \) (possibly partial) transition function, p. 5
\( q_0 \) initial state, p. 5
\( Q_m \) the set of marked states, p. 5
\( L(G) \) the generated language of DFSA \( G \), p. 13
\( L_m(G) \) the marked language of DFSA \( G \), p. 28
\( \Pi \) state transition cost matrix, p. 17
\( \tilde{\Pi} \) event cost matrix, p. 17
\( \mathbf{X} \) state characteristic vector, p. 29
\( \mu \) language measure vector, p. 2.3.15
Acknowledgments

The work described in this thesis could not have been accomplished without the help and support of others. First and foremost, I would like to thank my thesis advisor Professor Asok Ray for his generous and constant support, for many stimulating discussions, for being an unexhaustable source of energy and new ideas, and for the patience throughout my graduate study. His vision and ideas maintained my interest and spurred my creativity.

I thank Professor Shashi Phoha for her guidance, encouragement, and many helpful discussions during various stages of my thesis research and the research projects that I have participated and assisted.

I would like to thank Professor Alok Sinha, Professor Mark Carpino, and Professor John Metzner for their valuable suggestions to my proposal, reading this dissertation and for being on my committee.

I give my special appreciation to Professor W. M. Wonham for many discussions through emails and for giving me research insight and vision in the area of discrete event supervisory control during my visit to his group at University of Toronto, Canada.

The research work reported in this dissertation has been supported in part by Army Research Office under Grant No. DAAD19-01-1-0646, NASA Glenn Research Center under Grant No. NAG3-2448 and NAG3-2940, and National Science Foundation under Grant No. CMS-9819074.

I would like to offer my gratitude to all the fellow graduate students and research professionals whom I have worked with, in particular, members of the Networked Robotic System Laboratory: Peter Lee, Jinbo Fu, Eric Keller, Amol Khatkhate, Eric Grele, Jason Douglas, Steve McGuire, and Brian Hirth who all contributed in helping me finally reach my goal. Peter deserves special recognition for his assistance on the robotic experiment, the greatest time we had in the lab, and his great friendship. Eric Grele and Jason Douglas have also helped me for improving the knowledge of software engineering and programming skills. I thank all lab members for offering their support to my leadership of the MURI CSF research project.
as well as the development of the coordinated multi-robot system laboratory. I am also grateful to Amit Surana, Jinbo Fu, Ishanu Chattopadhyay and Venkatesh Rajagopalan for their collaboration on research, technical discussions, and many other joint efforts.

I would like to thank my wife, Junning Liu, for her love, patience, and encouragement. Without her support, the completion of this work would have been nearly impossible. The last thanks goes to my parents for their quiet support, encouragement, and believing in me even though far away.
Dedication

to my wife and my parents
Chapter 1

Introduction

“The purpose of control is to alter the dynamical behavior of a physical system in accordance with man’s wishes”. Since control theory attempts to find systematic ways to influence the system behavior through observing the state of the system and then computing appropriate control actions, the ability to exercise control depends significantly on the mathematical modeling framework for dynamic systems. Below three major mathematical modeling formalisms are presented.

1.1 Models of computation

1.1.1 Notation

Mathematical modeling formalisms described in this thesis can be viewed as dynamical systems whose states evolve in time either explicitly or implicitly. For continuously varying dynamical systems, time ranges over the set of real numbers. For discrete event systems, the time set is a discrete set that is order isomorphic to \((\mathbb{N}, \leq)\). This does not necessarily correspond to a fixed-interval discretization. The time domain is denoted as \(T = \mathbb{R}^+\). A time sequence is a monotonically increasing set of time points \(\{t_k \in T \mid k \in \mathbb{N}\}\).

Definition 1.1.1. (Temporal Behavior) [21] A temporal behavior over a topological space \(S\) is a partial function \(\beta : T \rightarrow S\) whose domain is an interval \([0, \kappa)\) for some \(\kappa \in T \cup \{\infty\}\). The metric length of \(\beta\) is denoted by \(|\beta|\) and \(\beta\) is infinite if \(\kappa = \infty\).
Definition 1.1.2. (Piecewise-continuous Behavior) [21] A behavior $\beta$ is piecewise-continuous if it admits a time sequence $\Gamma(\beta) = 0, t_1, t_2, \ldots$ such that for every $k \in \mathbb{N}$, $\beta$ is continuous on the interval $I_k = [t_k, t_{k+1})$.

Note that there may be many realizations of time sequences $I_0, I_1, \ldots$ on which a behavior $\beta$ is piecewise-continuous. A piecewise-constant behavior is a special case of piecewise-continuous behaviors where $\beta$ is constant on every interval $I_k$. The untimed abstraction of a piecewise-constant behavior $\beta$ is the sequence $\beta_u = q_0, q_1, \ldots$ such that for every $k$, $q_k = \beta(t_k)$. Figure 1.1 shows an example of piecewise-continuous and piecewise-constant behaviors.

![Figure 1.1](image)

**Figure 1.1.** A piecewise-continuous behavior $\beta$ and a piecewise-constant behavior $\beta_1$ with $\beta_u = q_1, q_2, q_3, q_4, q_5$

### 1.1.2 Continuous time systems

Much of modern system and control theory has been based on a modeling framework known as continuous time system (CTS), as defined follows.

**Definition 1.1.3.** A continuous dynamical system is $\mathcal{C} = (\mathcal{X}, U, f)$ where

1. $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space, where $n \in \mathbb{N}$
2. \( U \subseteq \mathbb{R}^m \) is the system input, where \( m \in \mathbb{N} \)

3. \( f : \mathcal{X} \times U \to \mathbb{R}^n \) is a continuous vector field

The behavior of the system is governed by the differential equations:

\[
\begin{align*}
\dot{x} &= f(x, u, t), \quad x(t_0) = x_0 \\
y &= g(x, u, t)
\end{align*}
\] (1.1) (1.2)

where \( t \in \mathbb{R}, t \geq t_0 \) is the time index. At any time instant \( t \), \( x \in \mathcal{X} \) is the \( n \)-dimensional state of the system, \( u \in U \) is the \( m \)-dimensional input of the system, \( x_0 \) is the initial condition of the state vector \( x \) at \( t_0 \), \( y \in \mathbb{R}^l \) is the \( l \)-dimensional output or measurement of the system, and \( \dot{x} \) is the derivative of state vector \( x \) with respect to time \( t \).

As time evolves, the dynamic behavior \( \psi : \mathcal{T} \to \mathcal{X} \) of the system \( \mathcal{C} \) starting from \( x_0 \in \mathcal{X} \) is captured by the state vector \( x(t) \), which is the solutions to the initial value problem of the differential equation, and governed by the control input \( u(t) \), as shown in Figure 1.2. This class of systems is referred as time-driven system. It is well-known [32] that there exists a unique solution to Equation 1.1 for every initial condition in \( \mathcal{X} \) if \( f \) is globally Lipschitz in \( x \). The continuous system \( \mathcal{C} \) is deterministic in the sense that it generates a unique trajectory given \( x_0 \) and \( u \).

Due to significant advances in computer and digital technology in the past few decades, the use of (embedded) digital controllers has successfully improved the automation level of physical plants. A classical example is the engine control
unit (ECU) in a car, which interacts with the physical processes in the internal combustion engine as well as with the events, e.g., acceleration and deceleration, generated by the driver. Another example is control of a robotic system whose behavior (for example, unexpected detection of obstacles) cannot be modeled in terms of differential equations. Such abrupt changes are best described as discrete events and are an integral part of the robot interactions with its environment. Such systems involve both continuous and discrete event dynamics, resulting in the modeling framework of discrete event systems and hybrid systems.

1.1.3 Discrete event systems

Discrete event systems are characterized by event-driven dynamics where state variables are usually discrete and change only as a result of the occurrence of events that are located discretely on the time axis, instead of continuously-changing over time, as depicted in Figure 1.3. Examples of discrete event systems include database operations, flexible manufacturing systems, communication protocol design, traffic systems, among others.

There have been several frameworks for the modeling of discrete event systems, including Petri nets, finite state automata, and process algebras. There is a trade-off between descriptive power and complexity for the choice of a modeling framework. It has been shown that the class of languages generated by Petri nets strictly contains the class of languages generated by finite state automata and thus Petri nets are more powerful than finite state automata in terms of modeling of discrete event systems. However, for the same reason, control problems in Petri nets and process algebras are more undecidable than those in finite state automata. These modeling frameworks can also be classified into untimed and timed models.

In an untimed model, only the logical behavior of the discrete event system is of interest, i.e., whether or not the system will enter a particular state. It is not important to know when the system enters that state or how long the system remains there. In a timed model, both logical behavior and timing information are important and characterized.

Below the discrete event system modeling framework in finite state automata is presented. In this modeling paradigm, the dynamic behavior of the systems
is represented by event sequences that occur possibly asynchronously (i.e., at irregular intervals). Therefore, the system can be modeled by an automaton and characterized by its generating language. Formally,

**Definition 1.1.4.** A discrete event system is modeled as a deterministic finite state automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$, where $Q$ is the discrete state space; $\Sigma$ is the set of events; $\delta : Q \times \Sigma \rightarrow Q$ is the (possibly partial) transition function; $q_0 \in Q$ is the initial state; and $Q_m \subseteq Q$ is the set of marked (as known as accepted) states.

Given an initial state $q_0 \in Q$, the behavior of the deterministic finite state automaton (DFSA) $G$ is a sequence $\phi : \mathbb{N} \rightarrow Q$ such that $\phi(0) = q_0$. For every $k \in \mathbb{N} \cup \{0\}$, $\phi(k + 1) \in \delta(\phi(k), \sigma)$, for some $\sigma \in \Sigma$. The generating language of $G$ is

$$L(G) = \{ s \in \Sigma^* \mid \delta(q_0, s) \in Q \} \quad (1.3)$$

It is important to note that in studying the behavior of an *untimed* discrete event system, the *ordering* information between transitions, not mapping of event occurrence to real metric time, is important.

### 1.1.4 Hybrid systems

*Hybrid systems* are dynamical systems with interacting continuous-time dynamics and discrete-event dynamics. Applications of hybrid systems can be found in...
many areas, such as automated highway systems, air traffic management systems [74], automatic control systems (as in automotive engines [6], industrial plants), chemical processes [41], real-time circuits, and robotics [2]. The mathematical model for hybrid systems is called hybrid automaton, a DFSA model augmented with a finite number of real-valued variables that change continuously according to the governing differential equations or differential inequalities [1].

**Definition 1.1.5.** A hybrid system, denote by \( H \), is modeled as a hybrid automaton

\[
H = \left( Q, X, \Sigma, \delta, X_0, I, f \right),
\]

where \( Q \) is the discrete state space representing a set of operation modes of the system; \( X \subseteq \mathbb{R}^n \) is a bounded continuous state space. The state space of \( H \) is thus \( \mathcal{X} = (Q \times X) \) and states of \( H \) are of the form \( (q, x) \in \mathcal{X} \) with \( q \in Q \) and \( x \in \mathbb{R}^n \); \( \Sigma \) is the set of events; \( \delta : (Q \times X) \times \Sigma \times (Q \times X) \to 2^X \) is the transition function capturing mode switching by transition guards \( g(\sigma) \), where \( \sigma \in \Sigma \), and reset mapping \( r(\sigma, x) : X_0 \subseteq \mathcal{X} \) is the set of initial states; \( I : Q \to 2^X \) is the staying/invariant conditions \( I(q) \) that constrain the value of the continuous variables \( x \) for being at a discrete state \( q \in Q \); \( f : Q \to (X \times \mathbb{R}^m \to \mathbb{R}^n) \) assigns a continuous vector field \( f_q(x, u) \) on the continuous state \( x \in X \), given an input \( u \in \mathbb{R}^m \), to every discrete state \( q \in Q \).

The transition guard, denoted \( g(\sigma) \), is defined to characterize the behavior of a hybrid system

\[
g(\sigma) = \{ x \in I(q) \mid \delta((q, x), \sigma) = (q', x'), \text{for some } x' \in I(q') \}
\]

and the set-valued reset map \( r(\sigma, x) : \)

\[
r(\sigma, x) = \{ x' \in I(q') \mid \delta((q, x), \sigma) = (q', x') \}
\]

Therefore, when the automaton is in a discrete state \( q \) and \( x \in g(\sigma) \), it makes a transition from \( q \) to \( q' \), generates the discrete event \( \sigma \), and resets the continuous variables to \( x' \in r(\sigma, x) \). The conditions of a discrete transition in hybrid systems can be given as follows.

**Definition 1.1.6.** A discrete transition occurs from one discrete state \( q \in Q \) to another discrete state \( q' \in Q \), i.e., \( \delta((q(\tau), x(\tau)), \sigma) = (q'(\tau), x'(\tau)) \), in a hybrid system \( H \) at time \( \tau \) if one of the following two conditions is true:
Figure 1.4. Hybrid systems

1. \( \exists \epsilon > 0, \) such that \( \forall 0 < \gamma \leq \epsilon, \)

\[
(x(\tau - \gamma), u(\tau - \gamma)) \in I(q(\tau - \gamma)), \text{ but } (x(\tau), u(\tau)) \notin I(q(\tau));
\]

2. \( \exists \sigma \in \Sigma, \) such that \( (x(\tau), u(\tau)) \in g(\sigma) \) and \( x'(\tau) \in r(\sigma, x) \)

Condition 1 states that a discrete transition is \textit{forced} to occur if the invariant \( I(q) \) at state \( q \) is violated for the first time at time \( \tau \). Condition 2 states that a discrete transition is \textit{enabled} if one of the guards from state \( q \) is triggered. Condition 2 reflects the nondeterministic nature of hybrid systems. When a discrete event \( \sigma \) takes place, the discrete state of the system changes from \( q \) to \( q' \), and the continuous variable \( x \) is reset to a new value \( r(\sigma, x) \). While both conditions above are monitored in continuous time, it is up to the high level automaton to decide whether condition 2 is to be enabled and thus take the discrete transition.

\textbf{Definition 1.1.7.} A trajectory of a hybrid system \( H \) starting from any initial state \( (q_0, x_0) \in X_0 \) is a pair of behaviors \( \gamma = (\psi, \phi) \) of the same metric length where \( \psi : T \rightarrow X \) is piecewise-continuous and \( \phi : T \rightarrow Q \) is piecewise-constant satisfying the following conditions.

1. \textit{Initialization:} \( \psi(0) = x_0 \) and \( \phi(0) = q_0. \)
2. Continuous evolution: for every interval $I_k = [t_k, t_{k+1}) \in \Gamma(\phi)$ such that $\phi_k = q$, $x$ evolves according to $f_q(x(t), u(t))$ for every $t \in [t_k, t_{k+1})$.

3. Discrete transition: for every $t_k \in \Gamma(\phi)$, such that $\phi_{k-1} = q$, $\phi_k = q'$, and $\delta(q, \sigma) = q'$, $\psi(t_k^-) \in g(\sigma)$ and $\psi(t_k) = r(\sigma, \psi(t_k^-))$ where $\psi(t_k^-)$ denotes the left limit of $\psi$ at $t_k$.

Therefore the behavior of a hybrid system $\mathcal{H}$ consists of concatenations of continuous evolutions and discrete transitions, as shown in Figure 1.4. Continuous evolutions keep the discrete part of the state $q$ constant, and the continuous part $x$ evolves over time according to the differential inclusions $\dot{x} \in f_q(x, u)$ provided that $x$ remains inside the invariant set $I(q)$. The continuous evolution can be denoted by the relation $\xrightarrow{c}$. At some point of the continuous evolution, if $x \in g(\sigma)$ for some $\sigma \in \Sigma$, then a discrete state transition occurs. The state of the hybrid system then changes from $(q, x)$ to a state $(q', x')$ that satisfies $x' \in r(\sigma, x)$ by the discrete event $\sigma$. The process then repeats, and the continuous evolution of the system will obey the differential inclusions $\dot{x} \in f_{q'}(x, u)$. This type of transitions due to discrete jumps is denoted by the relation $\xrightarrow{d}$. It should be noted that the hybrid state transition occurs instantaneously in both discrete and continuous spaces.

It is clear that a hybrid system can be reduced to a continuous time system captured by a set of ordinary differential equations (ODEs) if there is no discrete state transition as in [20]. Similarly, a hybrid system can be reduced to a discrete event system captured by a finite state automaton if its continuous dynamics is not considered.

1.1.5 Limitations of continuous models

The traditional formalism for mathematical modeling of dynamic systems has exhibited great success. There is a variety of approaches for control synthesis of continuous dynamic systems. Some of them are widely used in practice, e.g., PID control, and some are of a purely theoretic interest. In a rich mathematical domain such as the Euclidean space or more generally a differentiable manifold over the real number field, the dynamics of the system under consideration can be described using functions constructed from arithmetic, algebraic, and trigonometric relationships without logical elements. Together with the physical laws such
as those of mechanics, electricity or thermodynamics, system and control theory played a dominant role in those systems that can be specified by differential or difference equations. However, there are several reasons that the continuous time modeling formalism is insufficient for describing real systems.

1. The dynamics of many physical components of plants and controllers can not be modeled purely by continuous models. For instance, the saturation phenomena of sensors and actuators. The behaviors of valves and switches are best modeled as discrete transition.

2. High level intelligence might require sequential behaviors that can not be described in continuous trajectories. For instance, a moving robot is designed to search for objects of interest while avoiding obstacles.

3. The linear approximation of highly non-linear system is valid only in some region of the state space. When the system leaves this region, a different linear model should be used.

4. Many control systems are part of larger systems and interact with entities such as human operators or computers. The controller thus may receive discrete commands that will activate or suspend its execution or force it to switch to another mode of operation.

Both the discrete event system and hybrid system modeling formalisms are motivated by the fact that it is often natural to hierarchically decompose complex dynamic systems into a lower-level component representing the time-driven continuous behavior and a higher-level component representing the instantaneous event-driven abstraction of the physical systems. It is important to note that the event-driven dynamics is decoupled from the time-driven dynamics in discrete event systems and therefore can be considered as a special case of hybrid systems.

1.2 Fundamental problems in systems under different model paradigms

Control theoretic tools enable the design of continuous controllers in a single model of operation. Although nonlinear switching control laws can be applied for systems
with several modes of operations, the techniques are generally only applicable for simple systems with relatively few modes. In addition, the existence of such nonlinear switching control laws is case dependent, and the properties such as stability, convergence, and reachability only apply within certain bound of the operating points.

Systematic methods are necessary for design and analysis of hybrid and discrete event systems since systems modeled in these two frameworks are often complex and safety critical. A theory must address the fundamental problems in hybrid and discrete event systems, and computer tools play an important role during the automated design process. Therefore, below several fundamental problems are addressed in the design of these systems.

1. **Verification.** The goal of formal verification is to prove that the system performs as expected. As today’s automated systems increase complex in scale and complexity, the chance of having subtle errors is much greater. Therefore it is critical to ensure the designed system is correct (no error). The simplest verification problem is reachability analysis problem, for example, are “bad” states reachable from the initial state of the system?

2. **Synthesis.** The goal of synthesis is to design controllers/supervisors such that the controlled system meets a set of given specifications. Examples of synthesis problem are safety (i.e., prevent the controlled system from reaching states at which a disaster may occur) and liveness (i.e., guarantee that the controlled system can reach the goal state(s) infinitely often)

Let $P_c(\mathcal{X}(q))$ be the set of continuous successors of a set of states $\mathcal{X}(q) = (q, X(q)) \subseteq \mathcal{X}$ where $q \in Q$ and $X(q) \subseteq X$, then

$$P_c(\mathcal{X}(q)) = \{(q, x') \in \mathcal{X} | \exists (q, x) \in \mathcal{X}(q), (q, x) \xrightarrow{c} (q, x')\}$$

Similarly, the set of discrete successors of $\mathcal{X}(q)$, denoted by $P_d(\mathcal{X}(q))$ is defined as

$$P_d(\mathcal{X}(q)) = \{(q', x') \in \mathcal{X} | \exists (q, x) \in \mathcal{X}(q), (q, x) \xrightarrow{d} (q', x')\}$$

**Definition 1.2.1.** Given a hybrid system $H = (Q, X, \Sigma, \delta, X_0, I, f)$, the set of reachable states $\mathcal{R} \subseteq \mathcal{X}$ is defined as
1. \( \mathcal{R}^{(0)} = \mathcal{X}_0 \cap \{(q, x) \in \mathcal{X} | x \in I(q)\} \)

2. \( \mathcal{R}^{(i+1)} = \mathcal{P}_c(\mathcal{R}^{(i)}) \cup \mathcal{P}_d(\mathcal{R}^{(i)}), \forall i \geq 0 \)

3. \( \mathcal{R} = \bigcup_{i=0}^{\infty} \mathcal{R}^{(i)} \)

Let \( Q_b \subseteq Q \) be a set of bad discrete states and \( X_b \subseteq X \) be a set of bad continuous regions, Let \( \mathcal{B}_X = \{(q, x) \in \mathcal{X} | q \in Q_b, x \in X_b\} \). The hybrid system verification problem is to check if

\[
\mathcal{R} \cap \mathcal{B}_X = \emptyset \tag{1.4}
\]

It has been shown in [1] [3] that the verification problem for general hybrid systems is **undecidable** because of the uncountability of the state space. This far, the verification problem is decidable only for several special cases of hybrid systems, such as initialized rectangular automata [31] [3] and hybrid systems with linear differential equations [46]. Another drawback is due to its computational complexity as well as the lack of modularity/compositionality in the case of continuous dynamic systems models. These negative results force us to consider hybrid systems with either simpler discrete dynamics or simpler continuous dynamics [21] [57].

On the other hand, reachability in discrete event system is relatively straightforward since only the discrete transition structure is under consideration. Given a discrete event system \( G = (Q, \Sigma, \delta, q_0, Q_m) \), let \( p \rightarrow q \) represent a state \( p \in Q \) is a predecessor of a state \( q \in Q \), equivalently, \( q \) is a successor of \( p \), if there exist an event \( \sigma \in \Sigma \) such that \( \delta(p, \sigma) = q \). The sets of predecessor and successor states of \( p \in Q \) are

\[
\begin{align*}
P_b(p) &= \{q \in Q \mid q \rightarrow p\} \quad \tag{1.5} \\
P_f(p) &= \{q \in Q \mid p \rightarrow q\} \quad \tag{1.6}
\end{align*}
\]

The set of states that are accessible from \( p \) in exact two transitions is \( P_f(P_f(p)) \), and is denoted \( P^2_f(p) \). In general, \( P^i_f(p) \) denotes the set of states that are accessible from \( p \) after exact \( i \) transitions. \( P^i_b(p) \) can be defined similarly. Then

\[
\begin{align*}
P^*_b(p) &= \bigcup_{i \in \mathbb{N}} P^i_b(p) \quad \tag{1.7} \\
P^*_f(p) &= \bigcup_{i \in \mathbb{N}} P^i_f(p) \quad \tag{1.8}
\end{align*}
\]
are the set of states that are backward and forward reachable from \( p \), i.e., all possibly accessible states in \( Q \) from \( p \). In particular,

\[
\mathcal{R}(q_0) \equiv P_f^*(q_0) = \{ q \in Q \mid \exists s \in \Sigma^*, \delta(q_0, s) = q \}
\]

(1.9)
is the set of reachable states of the DFSA \( G \). It can be easily shown [42] that the reachability of a DFSA \( G \) can be determined in polynomial time \( O(n) \), where \( n \) is the number of states in \( G \). Let \( Q_b \subseteq Q \) be a set of bad states, therefore the verification problem in DFSA, i.e., if \( \mathcal{R}(q_0) \cap Q_b = \emptyset \), is decidable in \( O(n) \) time.

The basic synthesis problem in hybrid systems is to partition the state space into many regions with overlaps. Within each region, a possibly different continuous control law with predictable performance can be designed as usual such that stable equilibrium points lie in the overlap with other regions. In this way, it is possible to control the transition from mode to mode with predictable performance. It is difficult in all but the simplest topological spaces to have stable equilibria in the given regions. Therefore, partitioning and determining the switching surfaces so that all trajectories generated by the system satisfy some performance criteria is challenging in hybrid system design. For instance, at present, only hybrid systems with linear continuous dynamics and uncertain, bounded input is solvable [3].

### 1.3 Supervisory control of DESs

There is a large body of literature in the modeling, analysis and control of discrete event systems. Several fundamental problems in the design of discrete event systems have been addressed. The verification problem, which addresses the correctness of the controller design, has been first resolved by Ramadge and Wonham [66]. The notion of controllability was introduced and shown to be a necessary and sufficient condition for the existence of a supervisor that achieves a desired controlled behavior for a given discrete event system under complete observation of events. Lin and Wonham [48] introduced the observability condition for the existence of supervisor under partial observation. In the more general case of decentralized control, the condition of co-observability is introduced in place of observability by Cieslak et al [19] and later further generalized by Rudie and Wonham [68].
and Jiang and Kumar [36]. The synthesis problem, which addresses the design of a maximally permissive supervisor so that the closed loop system can engage in some maximal behavior and still maintain the prescribed behavioral constraint, has been studied in [66] [43] [45] [11] [70]. According to [66] and [91], the basic supervisory control of discrete-event system can be stated as follows.

A supervisory control for a discrete-event system $G$ is any feedback map $V : L(G) \to \Gamma$, where the control pattern $\Gamma = \{ \gamma \in 2^\Sigma \mid \gamma \supseteq \Sigma_u \}$ and $\Sigma = \Sigma_c \cup \Sigma_u$ is partitioned into two mutually exclusive sets, the controllable event set $\Sigma_c$ and uncontrollable event set $\Sigma_u$. The pair $(G, V)$ is written $V/G$ to suggest ‘$G$ under the supervision of $V$’. The generated behavior of $V/G$ is defined to be the language $L(V/G) \subseteq L(G)$ described as follows.

1. $\epsilon \in L(V/G)$
2. If $s \in L(V/G), \sigma \in V(s)$, and $s\sigma \in L(G)$, then $s\sigma \in L(V/G)$.

The marked behavior of $V/G$ is

$$L_m(V/G) = L(V/G) \cap L_m(G)$$

A supervisor $V$ is nonblocking if $pr(V/G) = L(V/G)$, where the prefix closure of a language $K$, denoted $pr(K) \subseteq \Sigma^*$ is the language

$$pr(K) = \{ s \in \Sigma^* \mid \exists u \in K, t \in \Sigma^*, s.t., u = st \}$$

A language $K \subseteq \Sigma^*$ is controllable with respect to $G$ and $\Sigma_u$ if

$$\forall s \in pr(K), \forall \sigma \in \Sigma_u, \text{such that } s\sigma \in L(G) \Rightarrow s\sigma \in pr(K)$$

equivalently

$$pr(K) \Sigma \cap L(G) \subseteq pr(K)$$

**Theorem 1.3.1.** (Theorem 3.1 in [42]) Let $K \subseteq L(G)$ be nonempty and prefix closed. There exists a nonblocking supervisory control $V$ for $G$ such that $L(V/G) = K$ if and only if $K$ is controllable.
Theorem 1.3.2. (Theorem 3.2 in [42], Theorem 3.4.2 in [91]) Let $K \subseteq L_m(G)$ be nonempty. There exists a nonblocking supervisory control $V$ for $G$ such that $L_m(V/G) = K$ if and only if $K$ is controllable.

As shown above, the synthesis of supervisors can be realized by checking if the conditions are hold. In many application domains, the purpose of a discrete event supervisor is to make decisions. One and only one controllable event is expected to be issued by a supervisor whenever it needs to make decisions. On the other hand, the conventional supervisor synthesis procedure allows more than two controllable events enabled at a state of a supervisor provided that this is allowed in the specification language, there is no preference by the supervisor on which controllable event will give better performance. Second, for two controllable supervisors such that one is not subset of the other, based on the qualitative measure of maximal permissiveness which will be defined in Chapter 1, one can not tell by looking at their generating languages or transition structures which controller has better performance than the other. Finally, it does not necessarily suggest that the more permissive a controller is, the better performance it will be.

1.4 Proposed solution

Complementary to the qualitative measure, a language measure for quantitative performance evaluation of different discrete event supervisors with respect to the same plant model is proposed in this dissertation. There are two sets of parameters in the language measure. Elements in one set can be obtained from real experiments and hence are intrinsic parameters of the given plant model of a physical process while elements in the other set are chosen by the designer based on the perception of their role in the system performance. It can be proven that there exists an optimal supervisor in terms of this measure. It is shown that such an optimal supervisor will enable one and only one controllable event whenever it is required to make a decision. A recursive synthesis procedure for the optimal discrete-event supervisor is then given.

A mobile robotic system is chosen to validate the language measure theory. The design of a mobile robotic system has been the subject of interest for many researches over the past few decades in a variety of topics, ranging from autonomous
control of single robot to coordination of a team of robots. Some of the approaches were justified only in simulation while others were validated in real world environment. In this dissertation, two major issues in the design of a complex dynamic system, a Pioneer 2 AT robot, are addressed.

1. **Design.** Robust design of the robot to interact with environment in a predictable way. The design should be flexible (to be able to coordinate with another robot if required) and yet purposeful (have an objective by itself).

2. **Evaluation.** Quantitative performance evaluation of the design of a single robotic system. Many high level strategic controllers can be designed given the robot’s capability. Their performance can further be compared.

The design of the robotic system is addressed by careful modeling of control and sensing strategies in both continuous and discrete domain. Both real robotic experiments in real world environment and simulation are carried out to demonstrate the design of the mobile robotic system. The following experiments are conducted to verify the consistency of the simulator and the real world robotic system.

1. obstacle avoidance which enables the robot to freely explore unknown space without collision in the presence of obstacles.

2. real-time visual-servoing to guide the robot to a target (object or destination) after it is detected.

3. coordinating between two robots in high level control strategies.

The effectiveness of the proposed performance measure is validated through extensive simulation study. Preliminary simulation results are provided using the same performance measure to evaluate coordination of multiple robot operations under hierarchical discrete event supervisory control.

### 1.5 Implementation issues in modeling of DESs

In this section, two important issues of successful implementation of a DES under a nonblocking supervisor are discussed. First, event generation. Second, separation of controllable and uncontrollable events. In virtually all existing literature
on supervisory control of DES, no particular attention has been paid to the subtle difference between controllable and uncontrollable events due to the assumption of instantaneous occurrence of discrete events. Figure 1.5 and Figure 1.6 present two different ways of modeling DESs. In the standard DES modeling approach Figure 1.5, both controllable and uncontrollable events are allowed to occur at the same state. When a supervisor makes a decision, i.e., enabling or disabling a controllable event, it will wait and expect this decision to be implemented by the plant. Since there exists a possibility that an uncontrollable event may occur before the occurrence of a controllable event enabled by the supervisor. Therefore it is possible to observe several uncontrollable events occurred consecutively in the plant, even though the supervisor enables at least one controllable event each time right after observing such an uncontrollable event. In implementation of this modeling framework, the synchronization of the plant and supervisor is assumed, the supervisor does not take the transition right after it enables a controllable event, instead, it waits for what actually happens in the plant next. It could be the enabled controllable event. It could also be another uncontrollable event. In Figure 1.6, a different way of modeling and implementation of a DES under supervisory control is proposed. It requires a procedure called \textit{separation of controllable and uncontrollable events} to reconstruct the original DES model such that there is no controllable and uncontrollable events both defined at any state of the resulting DES model. In this way, the occurrence of an uncontrollable event is always followed by the occurrence of a controllable event decided by the supervisor. Whenever the supervisor enables a controllable event, it immediately takes that transition, proceeds to next state, and waits for next uncontrollable event to occur. The occurrence of that controllable event in the plant will also be instantaneous. This new DES modeling approach fits well with the instantaneous event occurrence assumption. It is also consistent with classical feedback control theory. In addition, it will be explained in later chapters that this new approach facilitates the implementation of a DES optimal control strategy, an application of the proposed quantitative language measure for DESs.
1.6 Thesis outline and contribution

Chapter 2 of the thesis first reviews the general lattice theory on which the qualitative measure of maximal permissiveness of discrete event supervisory control is based. The quantitative measure theory of regular languages is then presented.

Chapter 3 introduces an identification procedure for language measure parameters and provide its convergence analysis.

Chapter 4 describes the design and development of the robotic system in both continuous-time and discrete-event domains. This includes the low level continuous-time control law based on the models of individual subsystems, data acquisition and real-time signal processing strategies used in the development of individual behaviors (e.g., visual servoing, obstacle avoidance), and the construction of a behavior-based discrete-event system model for the operation of the robot. The validation of a high fidelity simulator with respect to the real robot is also presented in this chapter. The performance evaluation of both real experiments and simulations in language measure is studied and comparison with results of other heuristic measure commonly-used in robotics research community are given.

Chapter 5 presents the existence of an optimal supervisor in terms of the language measure and a recursive synthesis procedure. Such an optimal supervisor enables one and only one controllable event at a given state where there are possibly many controllable events defined in the plant. The monotonicity of the optimal supervisor synthesis procedure guarantees its convergence in finite number of iteration. Given $(\bar{\Pi}, X)$, the optimal supervisor can be effectively constructed by this procedure. The effectiveness of the optimal supervisor is then validated by simulation of a robotic scenario.

Chapter 6 provides the hierarchical design of a multi-robot system in a discrete

\[ \Sigma_c \]

\[ \Sigma = \Sigma_u \cup \Sigma_c \]

**Figure 1.5.** Standard DES modeling and implementation
event setting. A quantitative performance evaluation in terms of language measure is proposed for this hierarchical control structure and further validated through simulation.

Chapter 7 summarizes the contributions of this dissertation and discusses some open problems and possible future extensions and directions of the presented work. Figure 1.7 shows the connections among chapter 3, 4, 5, and 6. The contributions of this dissertation are as follows.

1. Development of a quantitative language measure theory for the performance evaluation of DES controllers and optimal/robust control of discrete event systems.

2. A parameter identification procedure for the computation of language measure.

3. Design and construction of a behavior-based distributed multi-robot system and its simulator, including its individual behavior design (e.g., visual servoing, obstacle avoidance), event generation and detection through continuous-time control, signal processing, and communication, and supervisory control in the discrete event setting.

4. Validation of the proposed quantitative language measure through real robot experiments and simulations.

5. Development of an optimal DES controller synthesis algorithm using the language measure. Its computational complexity is polynomial in number of states in the discrete event plant model. Therefore, it is simpler than the popular Q-learning approach widely used in robotics.

![Figure 1.6. DES modeling with the separation of $\Sigma_c$ and $\Sigma_{\mu}$](image-url)
Figure 1.7. Thesis Outline
Chapter 2

Performance Measure of Discrete Event Systems

2.1 Introduction

For analysis and supervisory control of a discrete event system which is based on the theories of formal languages and automata, the legal behavior of a physical plant, denoted by $G$, is often represented by a deterministic finite-state automaton (DFSA) that is equivalent to a regular language [33], [53], [25]. The closed-loop behavior of the plant $G$ under a discrete event supervisor, denoted by $S$, is a sublanguage of the uncontrolled plant language [66], [42]. Mathematically it can be represented by synchronous composition [42] or parallel composition [17] of the plant automaton and the supervisor automaton. It is assumed that the event alphabets of $G$ and $S$ are the same, i.e., $\Sigma_S = \Sigma_G = \Sigma$. The class of prefix-closed controllable sublanguages of a given specification language $E$, denoted $C(E)$, is closed under arbitrary unions and intersections, thus forms a complete lattice. However, the qualitative measure [66] of supervisory control theory of discrete event systems, which gives the supremal and infimal elements of $C(E)$, does not provide a total order of this set and hence can not quantitatively evaluate the performance of the controlled plant under different supervisors. To address this issue, Wang and Ray [81] formulated a signed measure of regular languages followed by Ray and Phoha [67] who constructed a vector space of formal languages and defined a metric based on the total variation measure of the language. This chapter
presents the theory of language measure for discrete event systems. Section 1 first provides some algebraic preliminaries. Some results on qualitative measure of discrete event supervisor is then presented in Section 2. Section 3 formulates the problem of quantitative measure of discrete event systems and presents the theory of language measure. Two different approaches to compute the quantitative language measure are given in section 4. An engineering example on a gas turbine engine is presented in section 5. Some related work and how the theory presented here differs from others are presented in section 6. Section 7 summarizes the work and briefly discuss the extensions of quantitative measure theory.

2.2 Qualitative Measure of Discrete Event Supervisors

2.2.1 Algebraic Preliminaries

Some necessary concepts from lattice theory are given below, following more or less closely [91] and [42].

**Definition 2.2.1.** Let $X$ be a set, a binary relation on $X$ is a subset of $X \times X$, the cartesian product of $X$ with itself.

**Definition 2.2.2.** Let $\leq$ be a binary relation on set $X$, the relation $\leq$ is called a partial order (p. o.) on set $X$ if the following holds.

- reflexive:  
  $(\forall x \in X) x \leq x$

- transitive:  
  $(\forall x, y, z \in X)(x \leq y) \& (y \leq z) \Rightarrow x \leq z$

- antisymmetric:  
  $(\forall x, y \in X)(x \leq y) \& (y \leq x) \Rightarrow x = y$

The pair $(X, \leq)$ is a partially ordered set, or simply poset.

Elements $x, y \in X$ are comparable if either $x \leq y$ or $y \leq x$. A partial order is a total ordering if every two elements of $X$ are comparable. It is clear that the pair $(\mathbb{R}, \leq)$, where $\mathbb{R}$ is the set of real numbers and $\leq$ the usual ordering, is a totally ordered set, whereas $X = 2^\Sigma$, the power set (the set of all subsets) of set $\Sigma$, equipped with $\leq$, which is defined as for any two elements $x, y \in X, x \leq y$ if and only if $x \subseteq y$, is a poset.
Definition 2.2.3. Let \((X, \leq)\) be a poset, \(Y \subseteq X\), an element \(l \in X\) is called infimal of \(Y\), denoted \(\inf Y\), if and only if

1. \(l\) is a lower bound \(\iff \forall y \in Y, l \leq y\)

2. \(l\) is the largest lower bound \(\iff \forall a \in X, \forall y \in Y, a \leq y \Rightarrow a \leq l\)

Definition 2.2.4. Let \((X, \leq)\) be a poset, \(Y \subseteq X\), an element \(u \in X\) is called supremal of \(Y\), denoted \(\sup Y\), if and only if

1. \(u\) is an upper bound \(\iff \forall y \in Y, y \leq u\)

2. \(u\) is the least upper bound \(\iff \forall b \in X, \forall y \in Y, y \leq b \Rightarrow u \leq b\)

Given \(x, y \in X\), \(x \lor (x \land y)\) is used to denote \(\sup\{x, y\}\)\((\inf\{x, y\})\). As an example, let \(X = 2^\Sigma\) and \(x, y \in X\), then \(x \land y = x \cap y\), i.e., the set intersection, and \(x \lor y = x \cup y\), i.e., set union. It is noted that either sup or inf, if exists, is unique. It follows from the above definitions that \(\sup \emptyset = \inf X\) (if exists), and \(\inf \emptyset = \sup X\) (if exists).

Definition 2.2.5. A lattice is a poset \((L, \leq)\) in which sup and inf of any two elements always exist, i.e.,

\[\land : L \times L \to L, \quad \lor : L \times L \to L\]

If \(x, y, z \in L\) and let \(*\) denotes either \(\land\) or \(\lor\), then

- idempotent: \(x * x = x\)
- commutative: \(x * y = y * x\)
- associative: \((x * y) * z = x * (y * z)\)
- absorption: \(x \land (x \lor y) = x \lor (x \land y) = x\)
- consistency: \(x \leq y \iff x \land y = x \iff x \lor y = y\)

Similarly, an upper (lower) semi-lattice is a poset \((L, \leq)\) in which only sup (inf) of any two elements always exist.

Definition 2.2.6. A lattice \((L, \land, \lor)\) is complete if, for any nonempty subset \(Y\) of \(L\), both \(\inf Y\) and \(\sup Y\) exist as elements of \(L\).

Therefore, one can verify that both \(L = (2^\Sigma, \cap, \cup)\) and \(L = (\mathbb{R}[0, 1], \inf, \sup)\), the real numbers in \([0,1]\), are complete lattice.
Definition 2.2.7. Let \((L, \wedge, \vee)\) be a lattice, where \(L = (X, \leq)\), and \(Y \subseteq X\), \(L' = (Y, \leq)\) is said to be a sublattice of \(L\) if \(Y\) is closed under the \(\wedge\) and \(\vee\) operations of \(L\).

Definition 2.2.8. Let \((L, \wedge, \vee)\) be a complete lattice, a function \(f : L \rightarrow L\) is monotone if, for all \(\alpha, \beta \in L, \alpha \leq \beta\) implies \(f(\alpha) \leq f(\beta)\). It is said to be disjunctive if
\[
\forall Y \subseteq L, f(\lor_{y \in Y} Y) = \lor_{y \in Y} f(y)
\]
It is said to be conjunctive if
\[
\forall Y \subseteq L, f(\land_{y \in Y} Y) = \land_{y \in Y} f(y)
\]
An element \(\lambda \in L\) is a fixpoint of \(f\) if \(\lambda = f(\lambda)\).

One can easily verify that disjunctive and conjunctive functions are also monotone. In addition, note that \(\sup \emptyset = \inf L\) and \(\inf \emptyset = \sup L\), let \(Y = \emptyset\), then \(f(\inf L) = \inf L\) and \(f(\sup L) = \sup L\).

Theorem 2.2.1. Let \((L, \wedge, \vee)\) be a complete lattice and \(f : L \rightarrow L\) be a monotone function. Let \(Y = \{\lambda \in L | f(\lambda) = \lambda\}\) be the set of fixed points of \(f\). Then
1. \(\inf Y \in Y\), and \(\inf Y = \inf \{\lambda \in L | f(\lambda) \leq \lambda\}\)
2. \(\sup Y \in Y\), and \(\sup Y = \sup \{\lambda \in L | \lambda \leq f(\lambda)\}\)

It follows from Theorem 2.2.1 that a monotone function defined over a complete lattice always has an infimal and a supremal fixed point.

2.2.2 Qualitative Design of Discrete Event Supervisors

As mentioned earlier, the behavior of a DES is described as the set of all possible sequences of events that it can generate/execute. The logical behavior of a DES is a subset of \(\Sigma^*\), also represented as the generated language of a finite state automaton. The pair \((2^{\Sigma^*}, \subseteq)\) forms a complete lattice. So far, supervisory control theory of discrete event systems has been extensively studied qualitatively in terms of maximal permissiveness based on lattice theory. Below we briefly review these results on qualitative measure of discrete event supervisor synthesis.
Definition 2.2.9. Given $K \subseteq \Sigma^*$, the extension closure operator $\text{ext} : 2^{\Sigma^*} \rightarrow 2^{\Sigma^*}$ is the set of extensions of strings in $K$, denoted $\text{ext}(K) \subseteq \Sigma^*$, defined as

$$\text{ext}(K) = \{s \in \Sigma^*| \exists t \in K, \text{s.t.}, t \leq s\}$$

where $t \leq s$ denotes the $t$ is a prefix of $s$.

Definition 2.2.10. Quotient operation of $K$ with respect to $H$, denoted $K/H$, is the language obtained by removing suffixes belonging to $H$ from strings in $K$, i.e.,

$$K/H = \{s \in \Sigma^*| \exists t \in H, \text{s.t.}, st \in K\}$$

It is easily verified that for a language $K \subseteq \Sigma^*$, $\text{ext}(K) = K\Sigma^*$. In addition, the supremal prefix closed sublanguage of $K$, denoted $\text{sup} P(K)$, is given by

$$\text{sup} P(K) = \Sigma^* - \text{ext}(\Sigma^* - K)$$

Definition 2.2.11. The observation projection $P : \Sigma^* \rightarrow \Sigma^*$ is defined as

$$P(\epsilon) = \epsilon$$

$$P(\sigma) = \begin{cases} 
\sigma, & \text{if } \sigma \in \Sigma_o \\
\epsilon, & \text{if } \sigma \in \Sigma_{uo}
\end{cases}$$

$$P(\omega\sigma) = P(\omega)P(\sigma), \ \forall \omega \in \Sigma^*, \forall \sigma \in \Sigma$$

where $\Sigma_o$ and $\Sigma_{uo}$ are observable event set and unobservable event set, respectively, and $\Sigma = \Sigma_o \cup \Sigma_{uo}$.

The map $\tilde{P}$ is a modified observation projection that masks all but the last event, i.e.,

$$\tilde{P}(\epsilon) = \epsilon$$

$$\tilde{P}(\omega\sigma) = P(\omega)\sigma, \ \forall \omega \in \Sigma^*, \forall \sigma \in \Sigma$$

Definition 2.2.12. Given a plant $G$ and two supervisors $S_1, S_2 : L(G) \rightarrow 2^{\Sigma - \Sigma_u}$. $S_1$ is said to be less permissive than $S_2$, denoted $S_1 \preceq S_2$, if the following holds

$$\forall s \in L(G), S_2(s) \subseteq S_1(s)$$
In other words, $S_1$ disables more events than $S_2$ following the execution of any trace $s$. The infimum and supremum of $S_1$ and $S_2$ are defined as follows:

$$\forall s \in L(G), \begin{cases} S_1(s) \land S_2(s) = S_1(s) \cup S_2(s) \\ S_1(s) \lor S_2(s) = S_1(s) \cap S_2(s) \end{cases}$$

It is shown in [91] that the set of prefix closed controllable languages, denoted

$$\mathcal{C}(E) = \{ K \subseteq E \mid K = pr(K), pr(K)\Sigma_u \cap L(G) \subseteq pr(K) \}$$

where $E \subseteq \Sigma^*$ is the specification language, is a nonempty class of languages which is closed under arbitrary unions and intersections. While controllability is always preserved under arbitrary unions, it is because of the closedness condition that controllability is also preserved under arbitrary intersections, i.e., $K_\alpha = pr(K_\alpha) \Rightarrow \exists (\bigcap_\alpha K_\alpha) = \bigcap_\alpha pr(K_\alpha)$, for $\alpha$ in some index set. Together with the partial order of Definition 2.2.12, $\mathcal{C}(E)$ forms a complete lattice with union as join operation and intersection as meet operation. Therefore, the infimal prefix closed and controllable superlanguage of the language $E$, denoted $K_c^\uparrow = \inf \mathcal{C}(E)$ [47], and its supremal prefix closed and controllable sublanguage, denoted $K_c^\downarrow = \sup \mathcal{C}(E)$ [66], exist and are effectively computable, as given below.

$$K_c^\uparrow = K \setminus [(L(G) - K)/\Sigma_u]\Sigma^*, \quad K_c^\downarrow = pr(K)\Sigma_u \cap L(G)$$

The infimal prefix closed and observable superlanguage of a given language $K \subseteq L(G)$, denoted $K_o^\uparrow$, is given by

$$K_o^\uparrow = \sup P[\tilde{P}^{-1}(pr(K))] \cap L(G)$$

The supremal prefix closed and normal sublanguage and infimal prefix closed and normal superlanguage of a language $K \subseteq L(G)$, denoted $K_n^\downarrow$ and $K_n^\uparrow$, respectively, are given by

$$K_n^\downarrow = K \setminus [\mathcal{M}(L(G) - K)]\Sigma^*, \quad K_n^\uparrow = \mathcal{M}(pr(K)) \cap L(G)$$
where $\mathcal{M} : 2^{\Sigma^*} \to 2^{\Sigma^*}$ is a map induced by the observation projection $P$

$$\forall L \subseteq \Sigma^*, \mathcal{M}(L) = \{ s \in \Sigma^* | \exists t \in L, s.t., P(t) = P(s) \}.$$ 

### 2.3 Quantitative Measure of Discrete Event Supervisors

It is clear that the safety and liveness are of primary concern in terms of qualitative measure of discrete event systems while preserving the maximal permissiveness [91] [42]. However, there are several issues that can not be addressed in controller synthesis based on the maximal permissiveness measure. Firstly, in many application domains, the purpose of a discrete event supervisor is to make decisions. One and only one controllable event is expected to be issued by a supervisor whenever it needs to make decisions. On the other hand, the conventional supervisor synthesis procedure allows more than two controllable events enabled at a state of a supervisor provided that this is allowed in the specification language, there is no preference by the supervisor on which controllable event will give better performance. Secondly, the importance of either reaching certain states infinitely often or avoiding some other states as much as possible in the discrete event system are not addressed in qualitative design. Thirdly, performance of two DES controllers, not necessarily one is a subset of the other in terms of their generating languages, is not comparable with respect to a given discrete event plant. Finally, it is not necessary to be true that the maximally permissive supervisor gives the best performance. Therefore, this section presents a quantitative measure for the generating languages of discrete event systems, i.e., regular languages.

#### 2.3.1 Problem Formulation

The problem of quantitative measure of DES can be formulated as follows

Given that the relation $\subseteq$ induces a partial ordering on a set of controlled sublanguages $\{ L(S_j/G), j = 1, \ldots, m \}$ of the plant language $L(G)$, the language measure $\mu$ induces a total ordering $\leq$ on the set
\{\mu(L(S_j/G)), j = 1, \ldots, m\}. In other words, the range of the set function \(\mu\) is totally ordered while its domain could be partially ordered.

As well known, the regular language is partitioned into equivalence classes of finite-length event strings based on Myhill-Nerode Theorem. It is also important to note that each marked state of a DES has its own significance in terms of the system operations and control objectives. To characterize this significance, a real-valued weight is assigned to each marked state based on the designer’s perception of the state’s impact on the system performance. Conceptually similar to the conditional probability, each event is assigned a cost based on the state at which it is generated. This procedure permits an event string leading to a good (bad) marked state to have a positive (negative) measure. A supervisor can be designed in this setting such that the controlled sublanguage attempts to disable as many bad strings as possible and as few good strings as possible. Different supervisors may achieve this goal in different ways and generate a partially ordered set of controlled languages. The language measure creates a total ordering on the performance of the controlled languages, which provides a precise quantitative comparison of the controlled plant behavior under different supervisors. Below we first review some basics of measure theory.

### 2.3.2 Basics of Measure Theory

**Definition 2.3.1.** A \(\sigma\)-algebra or \(\sigma\)-field \(\mathcal{F}\) on a set \(\Omega\) is a collection of subsets of \(\Omega\) satisfying the following axioms

1. \(\emptyset \in \mathcal{F}\);
2. If \(A \in \mathcal{F}\), then its complement \(A^c = \{\omega : \omega \notin A\} \in \mathcal{F}\);
3. For all \(i \in I\), a countable index set, if \(A_i \in \mathcal{F}\), then \(\bigcup_{i \in I} A_i \in \mathcal{F}\).

**Definition 2.3.2.** A **measurable space** is a pair \((\Omega, \mathcal{F})\) consisting of a sample space \(\Omega\) together with a \(\sigma\)-algebra \(\mathcal{F}\) of subsets of \(\Omega\) (also called the event space). Each member of the sample space \(\Omega\) is called a measurable set.

A \(\sigma\)-algebra is sometimes referred to as a Borel field in the literature and the resulting measurable space called a Borel space.
Definition 2.3.3. A measure \( \mu \) (probability measure \( P \)) on a measurable space \( (\Omega, \mathcal{F}) \) is a function from \( \mathcal{F} \) (a set function) to \([0, \infty]\) ([0, 1]), such that if \( \{A_i | i \in I\} \) is a countable family of pairwise disjoint sets then

1. \( \mu(\emptyset) = 0 \),
2. \( \mu(\bigcup_{i \in I} A_i) = \sum_{i \in I} \mu(A_i) \).

The triple \( (\Omega, \mathcal{F}, \mu) \) is called a measure space. As a special case, \( (\Omega, \mathcal{F}, P) \) is called a probability space.

2.3.3 Quantitative Measure of Regular Languages

Let \( G_i = (Q, \Sigma, \delta, q_i, Q_m) \) be a DFSA model that represents the discrete-event dynamic behavior of a physical plant. Let \( n \) denote the cardinality of the state set \( Q \), i.e., \( |Q| = n \), and \( I = \{1, \ldots, n\} \) the index of \( Q \); \( \Sigma \) the (finite) alphabet of events; \( \Sigma^* \) is the set of all finite-length strings of events including the empty string \( \epsilon \); \( \delta: Q \times \Sigma \rightarrow Q \) is a (possibly partial) function of state transitions and \( \delta^*: Q \times \Sigma^* \rightarrow Q \) is an extension of \( \delta \); the state \( q_i \) is the initial state; and \( Q_m \) is the set of marked states, \( \emptyset \subset Q_m \subseteq Q \).

Definition 2.3.4. The language \( L(G_i) \) generated by a DFSA \( G \) initialized at the state \( q_i \in Q \) is defined as:

\[
L(G_i) = \{ s \in \Sigma^* \mid \delta^*(q_i, s) \in Q \} \quad (2.1)
\]

The language \( L_m(G_i) \) marked by the DFSA \( G \) initialized at the state \( q_i \in Q \) is defined as:

\[
L_m(G_i) = \{ s \in \Sigma^* \mid \delta^*(q_i, s) \in Q_m \} \quad (2.2)
\]

Definition 2.3.5. For every \( q_j \in Q \), let \( L(q_i, q_j) \) denote the set of all strings that, starting from the state \( q_i \), terminate at the state \( q_j \), i.e.,

\[
L(q_i, q_j) = \{ s \in \Sigma^* \mid \delta^*(q_i, s) = q_j \in Q \} \quad (2.3)
\]

The set \( Q_m \) of marked states is partitioned into \( Q_m^+ \) and \( Q_m^- \), i.e., \( Q_m = Q_m^+ \cup Q_m^- \) and \( Q_m^+ \cap Q_m^- = \emptyset \), where \( Q_m^+ \) contains all good marked states that are desired to
reach, and $Q_m^-$ contains all bad marked states that should be avoided, although it may not always be possible to completely avoid the bad states while attempting to reach the good states. To characterize this, each marked state is assigned a real value based on the designer’s perception of its impact on the system performance.

**Definition 2.3.6.** The characteristic function $\chi : Q \to [-1, 1]$ that assigns a signed real weight to state-based sublanguages $L(q, q)$ is defined as:

$$\forall q \in Q, \; \chi(q) \in \begin{cases} [-1, 0), & q \in Q_m^- \\ \{0\}, & q \notin Q_m \\ (0, 1], & q \in Q_m^+ \end{cases}$$  \hspace{1cm} (2.4)

The state weighting vector, denoted by $X = [\chi_1 \chi_2 \cdots \chi_n]^T$, where $\chi_j \equiv \chi(q_j) \; \forall k$, is called the $X$-vector. The $j$-th element $\chi_j$ of $X$-vector is the weight assigned to the corresponding terminal state $q_j$.

In general, the marked language $L_m(G_i)$ consists of both good and bad event strings that, starting from the initial state $q_i$, lead to $Q_m^+$ and $Q_m^-$ respectively. Any event string belonging to the language $L^0 = L(G_i) - L_m(G_i)$ leads to one of the non-marked states belonging to $Q - Q_m$ and $L^0$ does not contain any one of the good or bad strings. Based on the equivalence classes defined in the Myhill-Nerode Theorem [33], the regular languages $L(G_i)$ and $L_m(G_i)$ can be expressed as:

$$L(G_i) = \bigcup_{q_k \in Q} L(q_i, q_k) = \bigcup_{k=1}^n L(q_i, q_k) \hspace{1cm} (2.5)$$

$$L_m(G_i) = \bigcup_{q_k \in Q_m} L(q_i, q_k) = L_m^+ \cup L_m^- \hspace{1cm} (2.6)$$

where the sublanguage $L(q_i, q_k) \subseteq G_i$ having the initial state $q_i$ is uniquely labeled by the terminal state $q_k$, $k \in I$ and $L(q_i, q_j) \cap L(q_i, q_k) = \emptyset \; \forall j \neq k$; and $L_m^+ \equiv \bigcup_{q \in Q_m^+} L(q, q)$ and $L_m^- \equiv \bigcup_{q \in Q_m^-} L(q, q)$ are good and bad sublanguages of $L_m(G_i)$, respectively. Then, $L^0 = \bigcup_{q \notin Q_m} L(q_i, q)$ and $L(G_i) = L^0 \cup L_m^+ \cup L_m^-$. Following Definition 2.3.3, each element of a measure space for $L(G_i) \subseteq \Sigma^*$ can be defined as follows.

**Definition 2.3.7.** A $\sigma$-algebra $F$ of a nonempty regular language $\Omega \equiv L(G_i) \subseteq \Sigma^*$
is a non-empty collection of subsets of \( L(G_i) \) which satisfies the following three conditions: 1) \( \emptyset \in \mathcal{F} \); 2) \( L' \in \mathcal{F} \), then \( (\Omega - L') \in \mathcal{F} \); 3) \( \forall i \in I \), if \( L_i \in \mathcal{F} \), then \( \bigcup_{i=1}^{\infty} L_i \in \mathcal{F} \).

**Definition 2.3.8.** An at most countable collection \( \{L_k\} \) of members of a \( \sigma \)-algebra \( \mathcal{F} \) is a partition of a member \( L \in \mathcal{F} \) if \( L = \bigcup L_i \) and \( L_i \cap L_j = \emptyset \forall i \neq j \).

**Definition 2.3.9.** Let \( \mathcal{F} \) be a \( \sigma \)-algebra of \( L(G_i) \), the set function \( \mu : \mathcal{F} \to \mathbb{R} \equiv (-\infty, +\infty) \), is called a signed real measure if the following two conditions are satisfied [69]:

(i) \( \mu(\emptyset) = 0 \); 

(ii) \( \mu(\bigcup_{j=1}^{\infty} L_j) = \sum_{k=1}^{\infty} \mu(L_j) \) for every partition \( \{L_k\} \) for any member \( L \in \mathcal{F} \).

Note that, unlike a positive measure (e.g., the Lebesgue measure), \( \mu \) is finite such that the series in part (ii) of Definition 2.3.9 converges absolutely in \( \mathbb{R} \) and the result is independent of any permutation of the terms under union.

**Definition 2.3.10.** Let \( \mathcal{F} \) be a \( \sigma \)-algebra of \( L(G_i) \), For all \( L' \in \mathcal{F} \), a sublanguage \( L \in \mathcal{F} \), with respect to the signed real measure \( \mu \), is defined to be:

(i) null, denoted \( L = 0 \), if \( \mu(L \cap L') = 0 \)

(ii) positive, denoted \( L > 0 \), if \( \mu(L \cap L') > 0 \)

(iii) negative, denoted \( L < 0 \), if \( \mu(L \cap L') < 0 \)

**Definition 2.3.11.** Total variation \( |\mu| \) on a \( \sigma \)-algebra \( \mathcal{F} \) is defined as:

\[
|\mu|(L) = \sup \sum_k |\mu(L_k)|
\]

(2.7)

\( \forall L \in \mathcal{F} \) where the supremum is taken over all partitions \( \{L_k\} \) of \( L \).

**Proposition 2.3.1.** Total variation measure \( |\mu| \) of any regular language \( L \) is non-negative and finite i.e., \( |\mu|(L) \in [0, \infty) \).

The proof follows from standard theorems on complex measures [69].

Total variation can be, in general, defined for complex measures [69] but it is restricted to a signed real measure here. The total variation of a real signed
measure $\mu$, can be represented as $|\mu| = \mu^+ + \mu^-$ where $\mu^+$ and $\mu^-$ are called the positive and negative variation of $\mu$ and are defined as

$$
\mu^+ = \frac{1}{2}(|\mu| + \mu) \quad \text{and} \quad \mu^- = \frac{1}{2}(|\mu| - \mu)
$$

Both $\mu^+$ and $\mu^-$ are positive measures on $\Omega$. It also follows from above equation that $\mu = \mu^+ - \mu^-$. This representation of $\mu$ as the difference of positive measure $\mu^+$ and $\mu^-$ is known as the Jordan Decomposition of $\mu$ [30].

**Proposition 2.3.2.** Every sublanguage $L \in \Omega$ can be partitioned as: $L = L^0 \cup L^+ \cup L^-$ where the mutually exclusive sublanguages $L^0, L^+$ and $L^-$ are called null, positive, and negative, respectively, with respect to a signed real measure $\mu$.

The proof is based on the Hahn Decomposition Theorem [69]. As a consequence of above result, the following relations hold for positive and negative variations:

$$
\forall L \in \Omega, \quad \mu^+(L) = \mu(L \cap L^+), \quad \mu^-(L) = -\mu(L \cap L^-)
$$

A signed real measure $\mu : 2^{L(G_i)} \rightarrow \mathbb{R} \equiv (-\infty, +\infty)$ can therefore be constructed on the $\sigma$-algebra $\Omega = 2^{L(G_i)}$. With the choice of this $\sigma$-algebra, every singleton set made of an event string $\omega \in L(G_i)$ is a measurable set, which qualifies itself to have a numerical quantity based on the above state-based decomposition of $L(G_i)$ into $L^0$ (null), $L^+$ (positive), and $L^-$ (negative) sublanguages.

Conceptually similar to the conditional transition probability, each event is assigned a state-dependent cost.

**Definition 2.3.12.** The event cost of the DFSA $G_i$ is defined as a (possibly partial) function $\tilde{\pi} : \Sigma^* \times Q \rightarrow [0, 1)$ such that $\forall q_i \in Q, \forall \sigma_j \in \Sigma, \forall \omega \in \Sigma^*$,

1. $\tilde{\pi}[\sigma_j|q_i] \equiv \tilde{\pi}_{ij} \in [0, 1); \sum_j \tilde{\pi}_{ij} < 1$;
2. $\tilde{\pi}[\sigma_j|q_i] = 0$ if $\delta(q_i, \sigma_j)$ is undefined; $\tilde{\pi}[\epsilon|q_i] = 1$;
3. $\tilde{\pi}[\sigma_j \omega|q_i] = \tilde{\pi}[\sigma_j|q_i] \tilde{\pi}[\omega|\delta(q_i, \sigma_j)]$.

**Definition 2.3.13.** The state transition cost of the DFSA $G_i$ is defined as a function $\pi : Q \times Q \rightarrow [0, 1)$ such that $\forall q_i, q_j \in Q, \pi[q_j|q_i] = \sum_{\sigma \in \Sigma : \delta(q_i, \sigma) = q_j} \tilde{\pi}[\sigma|q_i] \equiv \pi_{ij}$.
and \( \pi_{ij} = 0 \) if \( \{ \sigma \in \Sigma : \delta(q_i, \sigma) = q_j \} = \emptyset \). The \( n \times n \) state transition cost \( \Pi \)-matrix is defined as:

\[
\Pi = \begin{bmatrix}
\pi_{11} & \pi_{12} & \cdots & \pi_{1n} \\
\pi_{21} & \pi_{22} & \cdots & \pi_{2n} \\
\vdots & \vdots & \ddots & \vdots \\
\pi_{n1} & \pi_{n2} & \cdots & \pi_{nn}
\end{bmatrix}
\]

**Definition 2.3.14.** The signed real measure \( \mu \) of every singleton string set \( \Delta = \{ \omega \} \in 2^{L(G_i)} \), where \( \omega \in L(q_i, q) \), is defined as \( \mu(\Delta) \equiv \tilde{\pi}(\omega, q_i) \chi(q) \) implying that

\[
\forall \omega \in L(q_i, q), \quad \mu(\Delta) = \begin{cases}
0, & q \notin Q_m \\
> 0, & q \in Q^+_m \\
< 0, & q \in Q^-_m
\end{cases}
\]

(2.10)

Therefore an event string terminating at a good (bad) marked state has a positive (negative) measure and zero otherwise. It follows that the signed real measure of the sublanguage \( L(q_i, q) \subseteq L(G_i) \) is defined as

\[
\mu(L(q_i, q)) = \sum_{\omega \in L(q_i, q)} \mu(\{\omega\}) = \left( \sum_{\omega \in L(q_i, q)} \tilde{\pi}[\omega|q_i] \right) \chi(q)
\]

(2.11)

**Definition 2.3.15.** The signed real measure of the language of a DFSA \( G_i \) initialized at a state \( q_i \in Q \), is defined as:

\[
\mu_i \equiv \mu(L(G_i)) = \sum_{q \in Q} \mu(L(q_i, q))
\]

(2.12)

The language measure vector, denoted as \( \mu = [\mu_1 \mu_2 \cdots \mu_n] \), is called the \( \mu \)-vector.

**Definition 2.3.16.** The cost \( \nu \) of a sublanguage \( K \subseteq L(G_i) \) is defined as the sum of the event cost \( \tilde{\pi} \) of all event strings in \( K \).

\[
\nu(K) = \sum_{\omega \in K} \tilde{\pi}[\omega|q_i]
\]

(2.13)

It follows immediately from the above definitions that \( \mu(L(q_i, q)) = \nu(L(q_i, q)) \chi(q) \). It should be noted that while the domain \( 2^{L(G_i)} \) of the language measure \( \mu \)
is partially ordered, its range which is a subset of $\mathbb{R}$ is totally ordered. Therefore $(L(G_i), 2^{L(G_i)}, \mu)$ forms a measure space. In principle, any measure $\mu$ can be defined on the measurable space $(L(G_i), 2^{L(G_i)})$. The choice of the signed language measure as given by Definitions 2.3.14 and 2.3.15 has been motivated by the physical significance of marked states in terms of DES system operations and control objectives. The uncontrolled marked language $L_m(G_i)$ consists of good strings leading to $Q^+_m$ and bad strings leading to $Q^-_m$. A supervisor $S$ may disable some of the bad strings and keep some of the good string enabled, depending on its ability to access to controllable events. Different supervisors $S_j : j \in \{1, 2, \ldots, n_s\}$ for a DFSA $G_i$ achieve this goal in different ways and generate a partially ordered set of controlled sublanguages $\{L_m(S_j/G_i) : j \in \{1, 2, \ldots, n_s\}\}$. The real signed measure $\mu$ provides a precise quantitative comparison of the controlled plant behavior under different supervisors because the set $\{\mu(L_m(S_j/G_i)) : j \in \{1, 2, \ldots, n_s\}\}$ is totally ordered. Therefore, the proposed language measure may serve as a performance measure for DES supervisor synthesis. It will be shown in the next section that defining the measure in this way further facilitates the computation of the measure.

2.3.4 Measure of Closed-loop Sublanguages

The previous section formulates a quantitative measure for a given regular language, equivalently, a deterministic finite state automaton. Given a DES together with $\Pi$-matrix and $X$-vector, its measure can be evaluated by Definition 2.3.15. A DES non-blocking supervisor $S$ restricts the marked behavior of an uncontrolled plant $G_i$ such that $L_m(S/G_i) \subseteq L_m(G_i)$. This section addresses another important issue: how to evaluate measure of different supervisors in a common quantitative basis. It is noted that all supervisors are designed with respect to the same DES plant. Let $L(S)$ be the generating language of a supervisor. The closed-loop DES behavior $L(S/G_i)$ is formally defined by synchronous composition [42] as follows.

**Definition 2.3.17.** Let $G = (Q_1, \Sigma, \delta_1, q_{11}, Q_{m1})$ and $S = (Q_2, \Sigma, \delta_2, q_{21}, Q_{m2})$ be the DFSA$s$ that generate DES plant language and specification language $K$, respectively. The closed-loop system behavior can be represented by a DFSA $C = (Q, \Sigma, \delta, q_1, Q_m)$, where $Q = Q_1 \times Q_2, q_1 = (q_{11}, q_{21}), Q_m = Q_{m1} \times Q_{m2}$, and for
each \( q = (q_1, q_2) \in Q, \sigma \in \Sigma \)

\[
\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))
\]

therefore, \( L(C) = L(G) \cap L(S) \) and \( L_m(C) = L_m(G) \cap L_m(S) \).

One should note that \( C \equiv S \) if \( K \subseteq L(G_i) \). However, Definition 2.3.17 is helpful to understand the DES closed-loop system behavior as well as the language measure of closed-loop sublanguages. As mentioned early, \( L(G) = \bigcup_{k=1}^{n_1} L(q_{11}, q_{1k}) \) where \( n_1 = |Q_1| \) and \( q_{1k} \in Q_1, 1 \leq k \leq n_1 \). To serve the purpose to evaluate the language measure of a closed-loop system, each \( L(q_{11}, q_{1k}) \) can be considered further partitions by \( L(q_{11}, q_{1k}) \cap L(q_{21}, q_{2j}), 1 \leq j \leq n_2 \) where \( n_2 = |Q_2| \), as suggested by Definition 2.3.17. Therefore for every \( q_{1k} \in Q_m^1 \) the set of strings which is retained in \( L_m(G) \cap L_m(S) \) is given by

\[
L(q_{11}, q_{1k}) \cap \left( \bigcup_{q_{2j} \in Q_m^2} L(q_{21}, q_{2j}) \right)
\]

A supervisor which retains maximally possible strings corresponding to \( q_{1k} \in Q_m^1 \) while discarding as many strings as possible corresponding to \( q_{1k} \in Q_m^1 \) would give a higher measure and hence a better performance. The above interpretation on refinement of equivalence classes in closed-loop system shows immediately how the event cost and characteristic function of the uncontrolled supervisors can be used to evaluate and compare the performance of different supervisors in terms of language measure, as stated below.

**Definition 2.3.18.** Let \( G, S \) and \( C \) be defined as in Definition 2.3.17. Let \( \tilde{\Pi}_P \)-matrix be the event cost matrix and \( X_P \)-vector be the characteristic vector for DES plant \( G \). The event cost matrix, denoted \( \tilde{\Pi}_C \), and characteristic vector, denoted \( X_C \), for the DFSA \( C \) are defined as follows

\[
\tilde{\pi}_C[\sigma| (q_{1i}, q_{2j})] = \tilde{\pi}_P[\sigma|q_{1i}] 
\]

(2.14)

for every \( \tilde{\pi}_C \in \tilde{\Pi}_C, \sigma \in \Sigma \) and \( 1 \leq i \leq n_1, 1 \leq j \leq n_2 \).

\[
\chi_C((q_{1i}, q_{2j})) = \chi_P(q_{1i})I(q_{2j})
\]

(2.15)
where $I(\cdot)$ is the indicator function defined as

$$I(q) = \begin{cases} 1, & q \in Q_{m2} \\ 0, & q \notin Q_{m2} \end{cases} \quad (2.16)$$

Let $\omega \in L(q_1, q_{ij})$ where $q_{ij} = (q_{11}, q_{2j})$, and $\Delta = \{\omega\}$. By Definition 2.3.14, the measure of the singleton set $\Delta$ in open loop DES plant $G$ is $\mu_P(\Delta) = \tilde{\pi}_P[\omega|q_{11}] \chi_P(q_{11})$. By Definition 2.3.18, the measure of $\Delta$ in the closed-loop DES system $C$ is given by

$$\mu_C(\Delta) = \tilde{\pi}_C[\omega|q_1] \chi_C(q_{ij}) = \tilde{\pi}_P[\omega|q_{11}] \chi_P(q_{11}) I(q_{2j})$$

Therefore, if no controllable event in the string $\omega$ is disabled by the supervisor $S$, $\mu_C(\Delta) = \mu_P(\Delta)$; otherwise, $\mu_C(\Delta) = 0$. Definition 2.3.14 guarantees that the same strings in different controlled sublanguages of a plant language $L(G)$ are assigned the same measure and thus provides a common basis for quantitative evaluation of different supervisors.

### 2.4 Language Measure Computation

It is impossible to directly compute the measure of every singleton string in a regular language and then sum them up to get the regular language’s measure. It is well known that a regular language can be represented by a regular expression. Various methods of obtaining regular expressions for DFSAs are reported in Hopcroft [33], Drobot [25], and Wonham [91]. It is natural to explore the possibility of computing language measure through a given regular expression. However, while computing the measure of a given DFSA, the same event may have different significance when emanating from different states. This requires assigning (possibly) different values to the same event defined on different states. Therefore, it is necessary to obtain a regular expression which explicitly yields the state-based event sequences. Two methods for language measure computation are presented below.

#### 2.4.1 Method I: Recursive Solution

This section presents a recursive procedure for language measure computation. It is based on concept of Kleene’s theorem [53] which shows that a language accepted
by a DFSA is regular. It also yields an algorithm to recursively construct the
regular expression of its marked language.

**Definition 2.4.1.** Given \( q_i, q_j \in Q \), a non-empty string \( p \) of events (i.e. \( p \neq \epsilon \))
starting from \( q_i \) and terminating at \( q_j \) is called a path. A path \( p \) from \( q_i \) to \( q_j \) is
said to pass through \( q_k \) if there exist \( s \neq \epsilon \) and \( t \neq \epsilon \) such that \( p = st \); \( \delta(q_i, s) = q_k \)
and \( \delta(q_k, t) = q_j \).

**Definition 2.4.2.** Let \( Q = \{ q_1, q_2, \ldots, q_n \} \), a path language \( p_{ij}^k \) is defined to be the
set of all paths from \( q_i \) to \( q_j \), which do not pass through any state \( q_r \), \( r > k \). The
path language \( p_{ij} \) is defined to be the set of all paths from \( q_i \) to \( q_j \), i.e., \( p_{ij} = p_{ij}^n \).
Thus, the language \( L(q_i, q_j) \) can be rewritten in terms of the path language \( p_{ij} \):

\[
L(q_i, q_j) = \begin{cases}
  p_{ii} \cup \{ \epsilon \}, & j = i \\
  p_{ij}, & j \neq i
\end{cases}
\]

and its event cost is given by

\[
\nu(L(q_i, q_j)) = \begin{cases}
  \nu(p_{ii}) + 1 & j = i \\
  \nu(p_{ij}) & j \neq i
\end{cases}
\]

It should be noted that every path language \( p_{ij}^k \) is a regular language and subset
of \( L(G_i) \). As shown in [67], the following recursive relation holds.

**Theorem 2.4.1.** Given a \( G_i \equiv (Q, \Sigma, \delta, q_i, Q_m) \), the following recursive relation
holds for \( 1 \leq j \leq n \)

\[
p_{ij}^0 = \begin{cases}
  \{ \sigma \mid \delta(q_i, \sigma) = q_j \} \cup \{ \epsilon \} & i = j \\
  \{ \sigma \mid \delta(q_i, \sigma) = q_j \} & i \neq j
\end{cases}
\]

\[
p_{ij}^k = p_{ik}^{k-1}(p_{kk}^{k-1})^*p_{kj}^{k-1} \cup p_{ij}^{k-1}
\]

Since the states are numbered from 1 to \( n \) in increasing order, informally, the
definition of \( p_{ij}^k \) above means that paths that go from \( q_i \) to \( q_j \) without passing
through a state higher than \( q_k \) are either

1. in \( p_{ij}^{k-1} \), i.e., never pass through a state as high as \( q_k \); or
2. composed of a string in \( p_{ik}^{k-1} \) (reach \( q_k \) for the first time) followed by zero or more strings in \( p_{kk}^{k-1} \) (from \( q_k \) to \( q_k \) without passing through state higher than \( q_k \)) followed by a string in \( p_{kj}^{k-1} \) (from \( q_k \) to \( q_j \)).

Proof. \( p_{ij}^0 \) is a finite set of strings each of which is either \( \varepsilon \) or a single event \( \sigma \in \Sigma \). Thus \( p_{ij}^0 = \{ \sigma \mid \delta(q_i, \sigma) = q_j \} \cup \{ \varepsilon \} \).

Given \( p_{ij}^{k-1} \subseteq p_{ij}^k \), let us consider the set \( p_{ij}^k - p_{ij}^{k-1} \). It follows that \( p_{ij}^k = p_{ij}^{k-1} + p_{kj}^k \), where \( p_{kj}^k \) can be expanded as \( p_{kj}^k = (p_{kk}^{k-1} p_{kj}^k) \cup p_{kj}^{k-1} \), has a unique solution. Therefore,

\[
p_{ij}^k = p_{ik}^{k-1} (p_{kk}^{k-1})^* p_{kj}^k \cup p_{ij}^{k-1}
\]  

(2.19) \[\square\]

The above relations can be transformed into an algebraic equation according to the following lemmas. Along with the procedure to compute the language measure it can be further shown that \( \sum_{j=1}^{n} \pi_{ij} < 1, \forall i \) is sufficient for finiteness of \( \nu \).

Lemma 2.4.1. \( \nu((p_{kk}^0)^* (\cup_{j \neq k} p_{kj}^0)) \in [0, 1) \)

Proof. Following Definition 2.3.12, \( \nu(p_{kk}^0) \in [0, 1) \). Therefore by convergence of geometric series,

\[
\nu\left( (p_{kk}^0)^* \left( \cup_{j \neq k} p_{kj}^0 \right) \right) = \frac{\sum_{j \neq k} \nu(p_{kj})}{1 - \nu(p_{kk})} \in [0, 1).
\]

since \( \sum_j \nu(p_{kj}^0) < 1 \Rightarrow \sum_{j \neq k} \nu(p_{kj}^0) < 1 - \nu(p_{kk}^0) \). \[\square\]

Lemma 2.4.2. \( \nu(p_{kk}^{k-1}) \in [0, 1) \)

Proof. The path \( p_{kk}^{k-1} \) may contain at most \( k-1 \) loops, one around each of the states \( q_1, q_2, \ldots, q_{k-1} \). If the path \( p_{kk}^{k-1} \) does not contain any loop, then \( \nu(p_{kk}^{k-1}) \in [0, 1) \) because for every \( \omega \in p_{kk}^{k-1}, \nu(\omega) < 1 \) and each of \( \omega \) originates at state \( k \). Next suppose there is a loop around \( q_l \) and that does not contain any other loop; this loop must be followed by one or more events \( \sigma_k \) generated at \( q_l \) and leading to some other states \( q_m \) where \( m \in \{1, \cdots, k\} \) and \( m \neq l \). By lemma 2.4.1, \( \nu(p_{kk}^{k-1}) \in [0, 1) \). Proof follows by starting from the innermost loop and ending with all loops at \( q_{k-1} \). \[\square\]
Lemma 2.4.3.
\[ \nu((p_{kk}^{k-1})^*) = \frac{1}{1 - \nu(p_{kk}^{k-1})} \in [1, \infty) \quad (2.20) \]

Proof. Since \( \nu(p_{kk}^{k-1}) \in [0,1) \) from lemma 2.4.2, \( \nu((p_{kk}^{k-1})^*) = \frac{1}{1 - \nu(p_{kk}^{k-1})} \in [1, \infty) \) \( \square \)

The main result of this section is stated as the following theorem.

Theorem 2.4.2. Given a DFSA \( G_i \equiv (Q, \Sigma, \delta, q_i, Q_m) \) the following recursive result holds for \( 1 \leq k \leq n \):

\[ \nu(p_{ij}^k) = \nu(p_{ij}^{k-1}) + \frac{\nu(p_{ik}^{k-1}) \nu(p_{kj}^{k-1})}{1 - \nu(p_{kk}^{k-1})} \quad (2.21) \]

Proof.

\[ \nu(p_{ij}^k) = \nu(p_{ij}^{k-1} \cup p_{ik}^{k-1}(p_{kk}^{k-1})^* p_{kj}^{k-1}) = \nu(p_{ij}^{k-1}) + \nu(p_{ik}^{k-1}(p_{kk}^{k-1})^* p_{kj}^{k-1}) = \nu(p_{ij}^{k-1}) + \nu(p_{ik}^{k-1}) \nu((p_{kk}^{k-1})^*) \nu(p_{kj}^{k-1}) = \nu(p_{ij}^{k-1}) + \frac{\nu(p_{ik}^{k-1}) \nu(p_{kj}^{k-1})}{1 - \nu(p_{kk}^{k-1})} \]

where the second step follows from fact that \( p_{ij}^{k-1} \cap p_{ik}^{k-1}(p_{kk}^{k-1})^* p_{kj}^{k-1} = \emptyset \). The third step follows from Definition 2.3.16 and the last step is a result of lemma 2.4.3. \( \square \)

Based on the above result, a recursive algorithm to compute a language measure is summarized as:

1. For a given \( G_i \equiv (Q, \Sigma, \delta, q_i, Q_m) \), obtain \( \mathbf{X} \) (characteristic vector) and \( \mathbf{Π} \) (event cost matrix)

2. Compute the \( \mathbf{Π} \) matrix (Definition 2.3.13)

3. \( \nu(p_{ij}^0) \leftarrow \pi_{ij} \) for \( 1 \leq i, j \leq n \)

4. for \( k = 1 \) to \( n \)
   for \( i = 1 \) to \( n \)
     for \( j = 1 \) to \( n \)
       \[ \nu(p_{ij}^k) = \nu(p_{ij}^{k-1}) + \frac{\nu(p_{ik}^{k-1}) \nu(p_{kj}^{k-1})}{1 - \nu(p_{kk}^{k-1})} \]
(5) Calculate $\nu(L(q_i, q_j))$ from $\nu(p_{ij})$ using Definition 2.4.2

(6) $\mu_i \leftarrow \sum_{q \in Q_m} \nu(L(q_i, q)) \chi(q)$ is the measure of marked language of DFSA $G_i$

Since there are only three for loops, the computational complexity of this method is polynomial in number of states of DFSA.

### 2.4.2 Method II: Closed Form Solution

This section presents a closed form method to compute the language measure via inversion of a square operator. The basic idea is to transform symbolic equations of regular expressions to algebraic equations.

**Definition 2.4.3.** Let $L_i \equiv L_m(G_i), i \in \mathcal{I}$, denote the regular expression representing the marked language of an $n$-state DFSA $G_i = (Q, \Sigma, \delta, q_i, Q_m)$ where $q_i$ is the initial state.

**Definition 2.4.4.** Let $\sigma^k_j$ denote the set of event(s) $\sigma \in \Sigma$ that is defined on the state $q_j$ and leads to the state $q_k \in Q$, where $j, k \in \mathcal{I}$, i.e., $\delta(q_j, \sigma) = q_k, \forall \sigma \in \sigma^k_j \subseteq \Sigma$.

Given a DFSA $G_i = (Q, \Sigma, \delta, q_i, Q_m)$ with $|Q| = n$, a procedure to obtain the system equation in terms of a set of regular expressions $L_i$ of the marked language $L_m(G_i), i \in \mathcal{I}$, can be obtained as follows:

$$\forall q_i \in Q, \quad L_i = \sum_j R_{i,j} + \mathcal{E}_i, \quad i \in \mathcal{I} \quad (2.22)$$

where $\forall i, R_{i,j}$ and $\mathcal{E}_i$ are defined as:

1. If $\exists \sigma \in \Sigma$, such that $\delta(q_i, \sigma) = q_j \in Q, j \in \mathcal{I}$, then $R_{i,j} = \sigma^j_i L_j$, otherwise, $R_{i,j} = \emptyset$.

2. If $q_i \in Q_m$, $\mathcal{E}_i = \epsilon$, otherwise, $\mathcal{E}_i = \emptyset$. 
The set of symbolic equations may be written as:

\[ L_i = \sum_j \sigma_i^j L_j + \mathcal{E}_i \]  \hspace{1cm} (2.23)

We note the following special cases.

(1) If \( \mathcal{E}_i = \emptyset \), \( \forall L_i \), then \( L_m(G) = \emptyset \). This implies that the DFSA has no marked state.

(2) If \( \exists q_i \in Q \) such that \( L_i = \epsilon \), then \( q_i \) is marked. Furthermore, \( q_i \) is a deadlock state.

The above system of symbolic equations can be solved using the following lemma, which is illustrated through an example.

**Lemma 2.4.4.** Let \( u, v \) be two known regular expressions and \( r \) be an unknown regular expression that satisfies the following algebraic identity:

\[ r = ur + v \]  \hspace{1cm} (2.24)

Then, the following relations are true:

(1) \( r = u^*v \) is a solution to equation 2.24

(2) If \( \epsilon \notin u \), then \( r = u^*v \) is the unique solution to equation 2.24.

The proof of Lemma 2.4.4 can be found in [25]. Part (2) of it is also known as Arden’s rule [91].

**Example 2.4.1.** Let \( \Sigma = \{a, b\} \), \( Q = \{1, 2, 3\} \), the initial state is 1 the sole marked state is 2 in Figure 2.1. Let the set of linear algebraic equations represent the transitions at each state of the DFSA.

\[
\begin{align*}
L_1 &= a_1^1 L_1 + b_1^2 L_2 \\
L_2 &= a_2^1 L_1 + b_2^3 L_3 + \epsilon \\
L_3 &= a_3^1 L_1 + b_3^2 L_2
\end{align*}
\]  \hspace{1cm} (2.25)
where the ‘forcing’ term $\epsilon$ is introduced on the right side of the $i$-th equation whenever $q_i \in Q_m$, $i \in I$. By application of Lemma 2.4.4, the regular expression for the marked language $L_m(G_1)$ is:

$$L_m(G_1) \equiv L_1 = (a_1^1)^*b_1^2(a_1^1)^*b_2^2 + b_3^1a_3^1(a_1^1)^*b_2^2 + b_3^2b_3^2)^*$$

![Figure 2.1. Example 1](image)

Language measure can be computed by transforming the symbolic equations to linear algebraic equation, rather than from the regular expressions. This is based on the following theorem.

**Theorem 2.4.3.** The language measure of the symbolic equations 2.23 is given by

$$\mu_i = \sum_j \pi_{ij} \mu_j + \chi_i$$

(2.26)

**Proof.** Following Equation 2.22 and Definition 2.3.6:

$$\forall i \in I, \quad \mu(\mathcal{E}_i) = \begin{cases} 
\chi_i & \text{if } \mathcal{E}_i = \epsilon \\
0 & \text{otherwise}
\end{cases}$$

(2.27)

Therefore, $X = [\chi_1 \chi_2 \cdots \chi_n]^T$ is the forcing function in Equation 2.23. Starting from the state $q_i$, the measure

$$\mu_i = \mu(L_i) = \mu \left( \sum_j \sigma_j^i L_j + \mathcal{E}_i \right)$$

$$= \mu \left( \sum_j \sigma_j^i L_j \right) + \mu(\mathcal{E}_i)$$
\[= \sum_j \mu(\sigma^j_i L_j) + \mu(\mathcal{E}_i)\]
\[= \sum_j \mu(\sigma^j_i) \mu(L_j) + \mu(\mathcal{E}_i)\]
\[= \sum_j \pi(\sigma^j_i) \mu(L_j) + \mu(\mathcal{E}_i)\]
\[= \sum_j \pi_{ij} \mu(L_j) + \mu(\mathcal{E}_i)\]
\[= \sum_j \pi_{ij} \mu_j + \chi_i\]

The third equality in the above derivation follows from the fact that \(\mathcal{E}_i \cap \sigma^j_i L_j = \emptyset\). It is also true that
\[\forall j \neq k, \quad \sigma^j_i L_j \cap \sigma^k_i L_k = \emptyset\]
(2.28)
since each string in \(\sigma^j_i L_j\) starts with an event in \(\sigma^j_i\) while each string in \(\sigma^k_i L_k\) starts from an event in \(\sigma^k_i\) for some \(k \neq j\) and thus \(\sigma^j_i \cap \sigma^k_i = \emptyset\) as \(G_i\) is a DFSA. This justifies the fourth equality. Since the DFSA model is modeled to be Markov, \(\mu(\sigma^j_i L_j) = \mu(\sigma^j_i) \mu(L_j)\). Therefore, by Definitions 2.3.12 and 2.4.4, \(\mu(\sigma^j_i L_j) = \pi[q_j|q_i] \mu(L_j) = \pi_{ij} \mu(L_j)\). \(\square\)

In vector notation, Equation 2.26 in Theorem 2.4.3 is expressed as: \(\bm{\mu} = \bm{\Pi} \bm{\mu} + \bm{X}\)
whose solution is given by:
\[\bm{\mu} = (\bm{I} - \bm{\Pi})^{-1} \bm{X}\]
(2.29)
provided that the matrix \(\bm{I} - \bm{\Pi}\) is invertible. This will also guarantee the existence of \(\bm{\mu}\).

**Theorem 2.4.4.** Given DFSAs \(G_i \equiv \langle Q, \Sigma, \delta, q_i, Q_m \rangle, 1 \leq i \leq n\) with the state transition cost matrix \(\bm{\Pi}\). Then

1. \((\bm{I} - \bm{\Pi})\) is invertible;
2. \((\bm{I} - \bm{\Pi})^{-1}\) is a bounded linear operator;
3. \((\bm{I} - \bm{\Pi})^{-1} \geq 0\) where the matrix inequality is implied elementwise.
4. \(\bm{\mu} \in \mathbb{R}^n\).
Proof. It follows from Definitions 2.3.12 and 2.3.13 that the induced norm

$$\|\Pi\|_{\infty} \equiv \max_i \sum_j \pi_{ij} = 1 - \theta$$

where $\theta \in (0, 1)$. Then $(I - \Pi)$ is invertible and is a bounded linear operator with the norm $\|I - \Pi\|_{\infty} \leq \theta^{-1}$. By Taylor series expansion,

$$(I - \Pi)^{-1} = \sum_{k=0}^{\infty} \Pi^k$$

Since each element of $\Pi$ is non-negative, so is each element of $\Pi^k$. Therefore, $(I - \Pi)^{-1} \geq 0$ elementwise. $\square$

It then follows immediately from Equation 2.29 that $\mu \in \mathbb{R}^n$.

Corollary 2.4.1. The language measure vector $\mu$ is bounded as: $\|\mu\|_{\infty} \leq \theta^{-1}$ where $\theta = (1 - \|\Pi\|_{\infty})$.

Proof. The proof follows by applying the norm inequality property and Theorem 2.4.4 to Equation 2.29 and the fact that $\|X\|_{\infty} \leq 1$ by Definition 2.3.6. $\square$

Definitions 2.3.12 and 2.3.13 state that each element in the $\Pi$-matrix is non-negative and each row sum is less than 1. These conditions make the $\Pi$-matrix a contraction operator that is sufficient for the matrix $(I - \Pi)^{-1}$ to be a bounded linear operator, as shown above. Therefore, Definitions 2.3.12 and 2.3.13 provide a sufficient condition for the language measure $\mu$ of the DFSA $G$ to be finite. A closed-form algorithm to compute a language measure based on the above procedure can be summarized as follows:

1. For a given $G_i \equiv \langle Q, \Sigma, \delta, q_i, Q_m \rangle$, obtain the characteristic vector $X$ and the event cost matrix $\tilde{\Pi}$ (Definition 2.3.12).

2. Generate the $\Pi$ matrix (Definition 2.3.13).

3. Compute the language measure vector $\mu \leftarrow (I - \Pi)^{-1}X$ using Gaussian elimination.
(4) $\mu_i$, the $i$th element of $\mu$-vector is the measure of the marked language of the DFSA $G_i$.

The $j$-th element of the $i$-th row of the $(I - \Pi)^{-1}$ matrix, denoted as $\nu_i^j$, is the language measure of the DFSA with the same state transition function $\delta$ as $G_i$ and having the following properties: (i) the initial state is $q_i$; (ii) $q_j$ is the only marked state; and (iii) the $\chi$-value of $q_j$ is equal to 1. Thus, $\mu_i \equiv \mu(L_m(G_i))$ is given by $\mu_i = \sum_j \nu_i^j \chi_j$. Numerical evaluation of the language measure of $G_i$ requires Gaussian elimination of the single variable $\mu_i$ involving the real square matrix $(I - \Pi)$. As such the computational complexity of the language measure algorithm is of polynomial order in the number of states.

### 2.4.3 Convergence of $\mu$

This section establishes necessary and sufficient conditions for finiteness of the measure $\mu$, based on certain properties of non-negative matrices which are stated without proof. The reader is referred to [62] and [27] for details of these results.

**Definition 2.4.5.** Let $A$ and $B$ be two square matrices of the same order $n$ (with real entries), then

- $A \geq B$ if $\forall i, j, a_{ij} \geq b_{ij}$
- $A > B$ if $A \geq B$, $A \neq B$
- $A \gg B$ if $\forall i, j, a_{ij} > b_{ij}$

If a matrix satisfies $A > 0$, it is called a nonnegative matrix and if $A \gg 0$, it is called positive matrix.

**Definition 2.4.6.** A square matrix $A$ of order $n$ is cogradient to a matrix $E$ if for some permutation matrix $P$, $PAP^T = E$. $A$ is reducible if it is cogradient to:

$$E = \begin{bmatrix} B & 0 \\ C & D \end{bmatrix}$$

where $B$ and $C$ are square matrices, or if $n = 1$ and $A = 0$. Otherwise, $A$ is irreducible.

It follows from above definition that a positive matrix is always irreducible.
Proposition 2.4.1. A nonnegative matrix $A$ is irreducible if and only if for every $(i,j)$ there exists a natural number $q$ such that

$$a_{ij}^{(q)} > 0$$

where $a_{ij}^{(q)}$ denote the $(i,j)$ element of $A^q$

Proposition 2.4.2. If $A \geq 0$ is irreducible and $B \geq 0$, then $A + B$ is irreducible.

Another characterization of irreducibility of a nonnegative square matrix has a graph-theoretic interpretation. This relationship can help to determine under what conditions the given finite state automaton $G$, which represents the controlled or uncontrolled plant model is irreducible by looking at the connectivity of its states. To develop this, the following definitions are needed.

Definition 2.4.7. The associated directed graph $G(A)$ of a square matrix $A$ of order $n$ consists of $n$ vertices $P_1, P_2, \ldots, P_n$ and an edge leads from $P_i$ to $P_j$ if and only if $a_{ij} \neq 0$.

Definition 2.4.8. A directed graph $G$ is strongly connected if for any ordered pair $(P_i, P_j)$ of vertices of $G$, there exists a sequence of edges which leads from $P_i$ to $P_j$.

Proposition 2.4.3. Given a matrix $A$, it is irreducible if and only if $G(A)$ is strongly connected.

If $A$ is a nonnegative square matrix, then the following is a result regarding relationship between the spectral radius (i.e., maximum absolute eigenvalue) $\rho$ of nonnegative matrices.

Proposition 2.4.4. If $0 \leq A \leq B$ and $A + B$ is irreducible then $\rho(A) < \rho(B)$

Definition 2.4.9. A square matrix $S$ of order $n$ is called (row) stochastic if it satisfies

$$s_{ij} \geq 0, \quad \sum_{j=1}^{n} s_{ij} = 1 \quad 1 \leq i \leq n$$  \hspace{1cm} (2.30)

Proposition 2.4.5. The maximum eigenvalue of a stochastic matrix $S$ is one, i.e $\rho(S) = 1$. A nonnegative matrix $A$ is stochastic if and only if $e$ is an eigenvector of $A$ corresponding to eigenvalue one, where $e$ is the vector all of whose entries are equal to one.
In order to show that \((I - \Pi)^{-1}\) is invertible it suffices to show that \(\rho(\Pi) < 1\).

**Theorem 2.4.5.** If \(\rho(\Pi) < 1\) then at least for one \(i, 1 \leq i \leq n, \sum_{j=1}^{n} \pi_{ij} < 1\).

**Proof.** Proof follows from the fact that if \(\sum_{j=1}^{n} \pi_{ij} = 1\) \(\forall i\), then \(\Pi\) would be a stochastic matrix by Definition 2.4.9. Hence by Proposition 2.4.5 \(\rho(\Pi) = 1 \Rightarrow (I - \Pi)^{-1}\) is not invertible. □

**Theorem 2.4.6.** If \(\sum_{j=1}^{n} \pi_{ij} < 1, \forall i\) s.t \(1 \leq i \leq n\), then \(\rho(\Pi) < 1\).

**Proof.** Let \(\theta_i = (1 - \sum_{j=1}^{n} \pi_{ij})/n > 0\). Let \(S\) be a matrix of order \(n\) which is defined in the following manner: \(s_{ij} = \theta_i + \pi_{ij}, 1 \leq i, j \leq n\). It is clear that \(S \gg 0\) and hence \(S\) is irreducible. Also \(S\) is a stochastic matrix and by Proposition 2.4.5, \(\rho(S) = 1\). Since \(0 \leq \Pi < S\) and \(\Pi + S\) is irreducible by proposition 2.4.2, it follows that \(\rho(\Pi) < \rho(S) = 1\) from Proposition 2.4.4. □

### 2.5 An Example of Engineering Application

This section presents an engineering example to illustrate the role of the language measure in the design of supervisory control systems for a given plant. A family of supervisors, based on different control specifications, are designed for a twin-engine unmanned aircraft that is used for surveillance and data collection. The measures of the uncontrolled plant language and the controlled sublanguages are compared to quantitatively evaluate the performance of these supervisors.

Engine health and operating conditions, which are monitored in real time based on observed data, are classified into three mutually exclusive and exhaustive categories: good; unhealthy (but operable); and inoperable. In the event of any observed abnormality, the supervisor may decide to continue or abort the mission. The finite state automaton model of the plant in Figure 2.2 has 13 states (excluding the dump state), of which three are marked states, and nine events, of which four are controllable and the remaining five are uncontrollable. All events are assumed to be observable. The states and events of the plant model are listed in Table 2.1 and Table 2.2, respectively. The state transition function \(\delta\) and the state-based event cost \(\tilde{\pi}_{ij}\) (see Definition 2.3.12) are entered simultaneously in Table 2.3. The fraction part in each entry denotes the corresponding state-based event cost \(\tilde{\pi}_{ij}\).
such that each row sum of the event cost matrix $\tilde{\Pi}$ is strictly less than one. The
teger part (within parentheses) in each entry denotes the respective destination
state resulting from the occurrence of the event. The values of $\tilde{\pi}_{ij}$ were selected
by extensive experiments on engine simulation models and were also based on ex-
perience of gas turbine engine operation and maintenance. The dump state and
any transitions to the dumped state are not shown in Table 2.3. The elements of
the characteristic vector (see Definition 2.3.6) were chosen as signed real weights
based on the perception of each marked state’s role on the engine performance.

Table 2.1. Plant Automaton States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
<th>$\chi$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Safe in base</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>Mission executing - two good engines</td>
<td>0.0</td>
</tr>
<tr>
<td>3</td>
<td>One engine unhealthy during mission execution</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>Mission executing - one good and one unhealthy engine</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>Both engines unhealthy during mission execution</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>One engine good and one engine inoperable</td>
<td>0.0</td>
</tr>
<tr>
<td>7</td>
<td>Mission execution with two unhealthy engines</td>
<td>0.0</td>
</tr>
<tr>
<td>8</td>
<td>Mission execution with only one good engine</td>
<td>0.0</td>
</tr>
<tr>
<td>9</td>
<td>One engine unhealthy and one engine inoperable</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>Mission execution with only one unhealthy engine</td>
<td>0.0</td>
</tr>
<tr>
<td>11</td>
<td>Mission aborted /not completed</td>
<td>-0.05</td>
</tr>
<tr>
<td>12</td>
<td>Mission successful</td>
<td>0.25</td>
</tr>
<tr>
<td>13</td>
<td>Aircraft destroyed</td>
<td>-1.0</td>
</tr>
</tbody>
</table>

Table 2.2. Plant Event Alphabet

<table>
<thead>
<tr>
<th>Event</th>
<th>Event Description</th>
<th>Controllable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>start and take-off</td>
<td>√</td>
</tr>
<tr>
<td>$b$</td>
<td>one good engine becoming unhealthy</td>
<td></td>
</tr>
<tr>
<td>$t$</td>
<td>one unhealthy engine becoming inoperable</td>
<td></td>
</tr>
<tr>
<td>$v$</td>
<td>one good engine becoming inoperable</td>
<td></td>
</tr>
<tr>
<td>$k$</td>
<td>keep engine(s) running</td>
<td>√</td>
</tr>
<tr>
<td>$a$</td>
<td>mission abortion</td>
<td>√</td>
</tr>
<tr>
<td>$f$</td>
<td>mission completion</td>
<td></td>
</tr>
<tr>
<td>$d$</td>
<td>destroyed aircraft</td>
<td></td>
</tr>
<tr>
<td>$l$</td>
<td>landing</td>
<td>√</td>
</tr>
</tbody>
</table>

The characteristic values of the 13 states in Table 2.1 is given by the character-
istic vector $X = [0 0 0 0 0 0 0 0 0 0 -0.05 0.25 -1.0]^T$. These parameters
**Figure 2.2.** Finite State Automaton Model of the Plant

**Figure 2.3.** Controller 1 for Specification #1

**Figure 2.4.** Controller 2 for Specification #2
are selected by the designer based on his/her perception of each marked state’s role in the system performance. As the states 1 to 10 are not marked, the first 10 elements of the characteristic vector $X$ are zeros. The implication is that event strings terminating at states 1 to 10 have zero measure. The state 12 is a good marked state having a positive $\chi$ value and the bad marked states 11 and 13 have negative $\chi$ values. Therefore, event strings terminating at state 12 have positive measure and those terminating at states 11 and 13 have negative measure.

Three supervisory controllers were designed independently using a graphical interactive package based on the following control specifications:

1. Specification #1: At least one of the two engines must be in good condition for mission continuation.
2. Specification #2: None of the two engines must be in inoperable condition for mission continuation.
3. Specification #3: Both engines must be in good condition for mission continuation.

Figures 2.3, 2.4, and 2.5 show the finite-state machine diagrams of the supervised plant under control specifications #1, #2, and #3, respectively. The dashed lines in these figures indicate that the transitions under corresponding supervisory controllers have been deleted from the plant model as a result of disabled (controllable) events. The performance measure $\mu_1$ (i.e., with the initial state 1) of the uncontrolled plant is 0.0823 and for three supervised plants under specifications #1, #2,
Table 2.3. State Transition and Event Cost Matrix

<table>
<thead>
<tr>
<th></th>
<th>s</th>
<th>b</th>
<th>t</th>
<th>v</th>
<th>k</th>
<th>a</th>
<th>f</th>
<th>d</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.50</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.05</td>
<td>0.01</td>
<td>0.80</td>
<td>0.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.12</td>
<td>0.16</td>
<td>0.10</td>
<td></td>
<td>0.50</td>
<td>0.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>0.25</td>
<td></td>
<td></td>
<td></td>
<td>0.50</td>
<td>0.20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>0.20</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td>0.3</td>
<td>0.4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.45</td>
<td></td>
<td></td>
<td>0.45</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td>0.35</td>
<td></td>
<td></td>
<td>0.20</td>
<td>0.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.95</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

and #3 were evaluated to be: 0.0807, 0.0822, and 0.0840, respectively. Therefore, the performance of the supervised plant under specifications #1, #2, and #3 is inferior, similar, and superior to that of the unsupervised plant from perspectives of the mission objectives as described by the language measure parameters $\tilde{\Pi} \text{ and } X$. The supervisor #3 yields the best performance among the three supervisors based on language measure parameters in Tables 2.3 and the characteristic vector.

2.6 Related Work

A stochastic discrete event system can be represented by an automaton with probabilities associated with transitions such that the probabilities of all transitions
from any state add up to at most one. There are research works on supervisory control of probabilistic DES within this framework [28] [44]. When the probabilities do not add up to exactly one at each state, it means the systems terminate with the remainder probability which is similar to the interpretation of probability of leading to the dumped state, i.e., unmodeled dynamics. This model is similar to the well-known Markov chain models [17]. The difference of these two models lies in the fact that the transition event symbols in the probabilistic automaton model are significant so that the event traces and the occurrence probabilities can be studied while a pure Markov chain model does not consider the importance of individual event symbol and thus often omitted. There also exists work on stochastic Petri Nets, e.g., [22], that follows the work of Markov chains and hence differs from the probabilistic automaton model.

In Rabin’s probabilistic automata model [65], the probabilities of transitions on each event, rather than all events, from any state add up to one. Thus the system changes its state on each event with probability one. This differs from our model in that the cumulative probability of state change over all events is at most one. Therefore Rabin’s model gives less information about state change on a specific event. Mortzavian [56] considered the supervisory control problem for systems modeled as Rabin’s probabilistic automata and hence differs from our framework. [60] [24] attempts to generalize deterministic automata to the probabilistic setting which also follow Rabin’s model.

Sengupta [70] has studied the optimal control of stochastic discrete event systems, where the behavior of a stochastic DES is given by a probabilistic map over the set of traces, and an optimal controller is obtained by reducing the problem to one of infinite horizon optimal Markov decision problems. Control cost was used in addition to the path cost in optimization of the performance index for trade-off between finding the shortest path and reducing the control cost. Although costs were assigned to the events, no distinction was made for events generated at (lead to) different states that could be “good” or “bad”. Therefore, these optimal control strategies have addressed performance enhancement of discrete event control systems without a quantitative measure of languages.

In [44] [28], a probabilistic language formalism for stochastic DES was introduced bearing similarity to Markov chain models. However, the control of stochas-
tic DES in this formalism is still based on lattice theory. It is easy to see that if \( \chi_i = 1 \), for all \( i \in I \), the model reduces to the one proposed in [28] and therefore most of results therein can be directly applicable for probabilistic supervisor synthesis as well. In this sense, the formalism presented in this chapter is a generalization to the work in [28] by characterizing the significance of difference marked states, which further allows us to quantitatively evaluate the performance of a controlled DES. The motivation for Kumar’s work in [44] was to qualitatively design supervisors for stochastic DES in terms of maximal permissiveness. The motivation here is to compare quantitatively the performance of different supervisors with respect to a given plant. The important contributions of individual marked states to the overall system performance are signified in a numerical manner while work in [44] is more concerned about the legality of execution of event traces.

2.7 Summary and Conclusions

This chapter briefly reviews the qualitative measure of discrete event systems in terms of lattice theory and the qualitative supervisor synthesis in terms of maximally permissiveness. It presents the concept, formulation and validation of a signed real measure for any regular language and its sublanguages based on the principles of automata theory and measure theory. The probabilistic language formalism in [28] is generalized in the sense that the significance of marked states is further characterized numerically, which makes it possible for quantitative performance evaluation of a controlled discrete event system instead of obtaining the supremal or infimal of a complete lattice. While the domain of measure \( \mu \), i.e., \( 2^{L_i(G_i)} \) is partially ordered, its range, a subset of \( \mathbb{R} = (-\infty, \infty) \) becomes totally ordered. Specifically, the relative performance of supervisors can be quantitatively evaluated in terms of the measure of the controlled sublanguages given an open loop plant as a common basis for such comparison. Positive weights are assigned to good marked states and negative weights to bad marked states so that a controllable supervisor is rewarded (penalized) for deleting strings leading to bad (good) marked states. As such the measure of the (open loop) plant language may be less than that of a (proper) controlled sublanguage. Two techniques, a recursive solution and a closed form solution, are presented to compute the language mea-
sure of a DFSA, yielding the same results. The computational complexity of both techniques is of polynomial order in the number of states in the DFSA.

The procedure of controller evaluation in terms of its language measure is validated by an engineering application for three different supervisors. A relatively less permissive supervisor could be more effective than another supervisor that may not adequately delete event strings leading to bad marked states. The computational complexity of the language measure algorithm is of polynomial order in the number of states.

Potential applications of the language measure are model identification, model order reduction, and analysis and synthesis of robust and optimal control and diagnostic systems in the discrete event setting.
Chapter 3

Online Identification of Language Measure Parameters

This chapter focuses on identification of the parameters (i.e., elements) of the events cost matrix $\tilde{\Pi}$, which, in turn, allows computation of the state transition cost matrix $\Pi$ and the language measure $\mu$-vector. A recursive algorithm for identification of the $\tilde{\Pi}$-matrix parameters is presented. It is assumed that the underlying physical process evolves at two different time scales. In the fast-time scale, i.e., over a short time period, the system is assumed to be an ergodic, discrete Markov process. In the slowly-varying time scale, i.e., over a long period, the system (possibly) behaves as a non-stationary process; it might be necessary to redesign the DES control policy in real time. In that case, the $\Pi$-matrix parameters should be periodically updated.

3.1 Event Cost: A Probabilistic Interpretation

The signed real measure (Definition 2.3.14) for a DFSA is based on the assignment of the characteristic vector and the event cost matrix. As stated earlier, the characteristic function is chosen by the designer based on his/her perception of the states’ impact on system performance. On the other hand, the event cost is an intrinsic parameter of the plant $G$. The element $\tilde{\pi}_{ij}$ in event cost matrix $\tilde{\Pi}$ is conceptually similar to the state-based conditional probability as in Markov Chains, except for the fact that it is not allowed to satisfy the equality condition $\sum_j \tilde{\pi}_{ij} = 1$. (Note
that $\sum_{j} \tilde{\pi}_{ij} < 1$ is a requirement for convergence of the language measure.) The rationale for this strict inequality is explained below.

Since the plant model is an inexact representation of the physical plant, there exist unmodeled dynamics to account for. This can manifest itself either as unmodeled events that may occur at each state or as unaccounted states in the model. Let $\Sigma^u_k$ denote the set of all unmodeled events at state $k$ of the DFSA $G_i \equiv (Q, \Sigma, \delta, q_i, Q_m)$. Let us create a new unmarked absorbing state $q_{n+1}$ called the dump state [42] and extend the transition function $\delta$ to $\delta_e : (Q \cup \{q_{n+1}\}) \times (\Sigma \cup (\cup_{k}\Sigma^u_k)) \rightarrow (Q \cup \{q_{n+1}\})$ as follows:

$$
\delta_e(q_k, \sigma) = \begin{cases} 
\delta(q_k, \sigma) & \text{if } q_k \in Q \text{ and } \sigma \in \Sigma \\
q_{n+1} & \text{if } q_k \in Q \text{ and } \sigma \in \Sigma^u_k \\
q_{n+1} & \text{if } k = n + 1 \text{ and } \sigma \in \Sigma \cup \Sigma^u_k 
\end{cases}
$$

Therefore the residue $\theta_k = 1 - \sum_{j} \tilde{\pi}_{kj}$ denotes the probability of the set of unmodeled events $\Sigma^u_k$ conditioned on the state $k$. The $\Pi$ matrix can be similarly augmented to obtain a stochastic matrix $\Pi_{aug}$ as follows:

$$
\Pi_{aug} = \begin{bmatrix}
\pi_{11} & \pi_{12} & \ldots & \pi_{1n} & \theta_1 \\
\pi_{21} & \pi_{22} & \ldots & \pi_{2n} & \theta_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\pi_{n1} & \pi_{n2} & \ldots & \pi_{nn} & \theta_n \\
0 & 0 & \ldots & 0 & 1
\end{bmatrix}
$$

Since the dump state $q_{n+1}$ is not marked, its characteristic value $\chi(q_{n+1}) = 0$. The characteristic vector then augments to $X_{aug} = [X \ 0]^T$. With these extensions the language measure vector $\mu_{aug} = [\mu_1 \ \mu_2 \ \ldots \ \mu_n \ \mu_{n+1}]^T = [\mu \ \mu_{n+1}]^T$ of the augmented DFSA $G_{aug} \equiv (Q \cup \{q_{n+1}\}, \Sigma \cup (\cup_{k}\Sigma^u_k), \delta_e, q_i, Q_m)$ can be expressed as:

$$
\begin{pmatrix}
\mu \\
\mu_{n+1}
\end{pmatrix} = \left( \Pi \mu + \mu_{n+1} [\theta_1 \cdots \theta_n]^T \right) + \begin{pmatrix} X \\ 0 \end{pmatrix}
$$

(3.1)

Since $\chi(q_{n+1}) = 0$ and all transitions from the absorbing state $q_{n+1}$ lead to itself, $\mu_{n+1} = \mu(L_m(G_{n+1})) = 0$. Hence Equation 3.1 reduces to that for the original plant $G_i$. Thus, the event cost can be interpreted as conditional probability, where
the residue $\theta_k = 1 - \sum_j \tilde{\pi}_{kj}$ accounts for the probability of all unmodeled events emanating from the state $q_k$. Moreover, it satisfies the following two constraints.

1. $\forall q \in Q, \tilde{\pi}[\epsilon|q] = 1$;

2. $\forall \omega \in \Sigma^*, \sigma \in \Sigma, q \in Q, \sum_\sigma \tilde{\pi}[\omega\sigma|q] \leq \tilde{\pi}[\omega|q]$.

By Bayesian theorem, the probability of trace $t$ given that trace $s$ has already occurred starting from state $q$ is given by

$$\forall s, t \in \Sigma^*, q \in Q, \delta(q, s) = p, \tilde{\pi}[st|s, p] = \frac{\tilde{\pi}[st|q]}{\tilde{\pi}[s|q]}$$

With this interpretation of event cost, $\tilde{\pi}[\omega|q_k]$ (Definition 2.3.12) denotes the probability of occurrence of the event string $\omega$ in the plant model $G_i$ starting at state $q_k$ and terminating at state $\delta(q_k, \omega)$. Hence, $\nu(L(q, q_i))$ (Definition 2.3.16), which is a non-negative real number, is directly related (but not necessarily equal) to the total probability that state $q_k$ would be reached as the plant operates. The language measure $\mu_i \equiv \mu(L(G_i)) = \sum_{q \in Q} \mu(L(q_i, q)) = \sum_{q \in Q} \nu(L(q_i, q))\chi(q)$ is then directly related (but not necessarily equal) to the expected value of the characteristic function. As mentioned earlier, the choice of the characteristic function (Definition 2.3.6) is solely based on the designer’s perception of the importance assigned to the individual DFSA states. Therefore, in the setting of language measure, a supervisor’s performance is superior if the supervised plant is more likely to terminate at a good marked state and/or less likely to terminate at a bad marked state. However, the event cost matrix $\tilde{\Pi}$ is determined by the dynamics of the underlying continuous process. In the next section, a technique for online estimate of $\tilde{\Pi}$-matrix is presented.

### 3.2 Online $\tilde{\Pi}$-Matrix Parameter Identification

Due to the probabilistic interpretation of the event cost matrix $\tilde{\Pi}$, the idea of online parameter identification is to estimate the frequencies of events observed at corresponding states.
3.2.1 A recursive parameter estimation scheme

Let \( \tilde{\pi}_{ij} \), representing the \( i-j \)-th element of the event cost matrix \( \Pi \) be defined as the transition probability of event \( \sigma_j \) on the state \( q_i \), i.e.,

\[
\tilde{\pi}_{ij} = \begin{cases} 
P[\sigma_j|q_i], & \text{if } \exists q \in Q, \text{ s.t. } q = \delta(q_i, \sigma_j) \\
0, & \text{otherwise}
\end{cases}
\]

The indicator function \( I_t(i, j) \) is defined as the occurrence of event \( \sigma_j \) at time \( t \) if the system was in state \( q_i \) at time \( t - 1 \) where \( t \) represents a generalized time epoch, for example, the experiment number. Formally, \( I_t(i, j) \) is expressed as:

\[
I_t(i, j) = \begin{cases} 
1, & \text{if } \sigma_j \text{ is observed at state } q_i \\
0, & \text{otherwise}
\end{cases}
\]

Let \( N_t(i) \), denoting the number of incidents of reaching the state \( q_i \) up to the time instant \( t \), be a random process mapping the time interval up to the instant \( t \) into the set of nonnegative integers. Similarly, let \( n_t(i, j) \) denote the number of occurrences of the event \( \sigma_j \) at the state \( q_i \) up to the time instant \( t \) and \( J_t(i) \equiv \sum_j I_t(i, j) \). The recursive algorithm of learning \( p_{ij} \) is formulated as a stochastic approximation scheme at the starting time instant \( t = 0 \) with the initial conditions: \( \hat{p}_0(i, j) = 0 \) for all \( q_i \in Q, \sigma_j \in \Sigma \); and \( n_0(i) = 0, J_0(i) = 0 \) for all \( q_i \in Q \). The algorithm runs for \( t \geq 1 \). Upon occurrence of the event \( \sigma_j \) at the state \( q_i \), the recursive algorithm is incremented as:

\[
N_t(i) = N_{t-1}(i) + I_t(i) \\
J_t(i) = J_{t-1}(i) + I_t(i, j) \\
\hat{p}_{N_t(i)}(i, j) = \hat{p}_{N_{t-1}(i)}(i, j) + J_t(i)N_t(i)^{-1}(I_t(i, j) - \hat{p}_{N_{t-1}(i)}(i, j)) \\
n_t(i, j) = n_{t-1}(i, j) + I_t(i, j)
\]

**Proposition 3.2.1.** The recursive stochastic approximation scheme of \( \hat{p}_{N_t(i)}(i, j) \) is the frequency estimator \( \hat{p}_{ij}(t) \) of independent Bernoulli trials at time \( t \), i.e.,

\[
\hat{p}_{N_t(i)}(i, j) = \frac{n_t(i, j)}{N_t(i)}
\]
and \( \lim_{N_t(i) \to \infty} \hat{p}_{N_t(i)}(i, j) = \hat{\pi}_{ij} \).

**Proof.**

**Step 1:** \( J_t(i) = 0 \). \( \Rightarrow N_t(i) = N_{t-1}(i), n_t(i, j) = n_{t-1}(i, j), \hat{p}_{N_t(i)}(i, j) = \hat{p}_{N_{t-1}(i)}(i, j) \).

**Step 2:** \( J_t(i) \geq 1, I_t(i, j) = 0 \). \( \Rightarrow n_t(i, j) = n_{t-1}(i, j), N_t(i) = N_{t-1}(i) + 1 \), and usage of the above identities yields

\[
\hat{p}_{N_t(i)}(i, j) = \frac{n_{t-1}(i, j)}{N_{t-1}(i)} + \frac{1}{N_t(i)N_{t-1}(i)} - \frac{1}{N_t(i)N_{t-1}(i)} - \frac{n_{t-1}(i, j)}{N_t(i)N_{t-1}(i)} = \frac{n_t(i, j)}{N_t(i)}
\]

**Step 3:** \( J_t(i) \geq 1, I_t(i, j) = 1 \). \( \Rightarrow n_t(i, j) = n_{t-1}(i, j) + 1, N_t(i) = N_{t-1}(i) + 1 \) and usage of the above identities yields

\[
\hat{p}_{N_t(i)}(i, j) = \frac{n_{t-1}(i, j)}{N_{t-1}(i)} + \frac{1}{N_t(i)N_{t-1}(i) + 1} - \frac{1}{N_t(i)N_{t-1}(i)} - \frac{n_{t-1}(i, j)}{N_t(i)N_{t-1}(i)} = \frac{(n_{t-1}(i, j) + 1)N_{t-1}(i)}{N_t(i)N_{t-1}(i)} = \frac{n_t(i, j)}{N_t(i)}
\]

\( \square \)

Therefore, \( \pi_{ij} \) is the asymptotic value of the estimated probabilities \( \hat{p}_{N_t(i)}(i, j) \) as if the event \( \sigma_j \) occurs infinitely many times at the state \( q_i \). However, dealing with finite amount of data, the objective is to obtain a good estimate \( \hat{p}_{ij} \) of \( p_{ij} \) from independent Bernoulli trials of generating events. Let \( \Sigma^n = \cup_i \Sigma^n_i, \forall i \equiv \{1, \ldots, n\} \), at each state \( q_i \), \( P[\Sigma^n|q_i] = \theta_i \in (0,1) \) and \( \sum_i \hat{\pi}_{ij} = 1 - \theta_i \). Therefore, an estimate of the \( (i,j) \)-th element in \( \tilde{\Pi} \)-matrix, denoted \( \hat{\pi}_{ij} \), is obtained by

\[
\hat{\pi}_{ij} = \hat{p}_{ij}(1 - \theta_i) \tag{3.2}
\]
Further experiments on a more detailed model are necessary to identify the parameters \( \theta_i, \forall i \in \mathcal{I} \). Since \( \theta_i \ll 1 \), an alternative approximation approach is taken for ease of implementation. Let \( \hat{\theta}_i = \theta, \forall i \in \mathcal{I} \) where the parameter \( 0 < \theta \ll 1 \) is selected from the numerical perspective based on the fact that the sup-norm \( \| \mu \|_\infty \leq \theta^{-1} \). Moreover, the fact that each row sum in the \( \hat{\Pi} \)-matrix being strictly less than 1, i.e., \( \sum_j \hat{\pi}_{ij} < 1 \), is a sufficient condition for finiteness of the language measure.

### 3.2.2 A Stopping Rule for \( \hat{\Pi} \)-Matrix Learning

This section presents a stopping rule to determine a bound on the number of experiments to be conducted for identification of the \( \hat{\Pi} \)-matrix parameters. The objective is to achieve a trade-off between the number of experimental observations and the estimation accuracy. A robust stopping rule is presented below. A bound on the required number of samples is estimated using the Gaussian structure for the binomial distribution that is an approximation of the sum of a large number of independent and identically distributed (i.i.d.) bernoulli trials. Denoting \( \hat{\pi}_{ij} \) as \( p \) and \( \hat{\pi}_{ij} \) as \( \hat{p} \), we have \( \hat{p} \sim \mathcal{N}(p, \frac{p(1-p)}{N}) \), where \( E[\hat{p}] \approx p \) and \( \text{Var}[\hat{p}] \equiv \sigma^2 \approx \frac{p(1-p)}{N} \), provided that the number of samples \( N \) is sufficiently large. Let \( X = \hat{p} - p \), then \( \frac{X}{\sigma} \sim \mathcal{N}(0, 1) \). Given \( 0 < \varepsilon \ll 1 \) and \( 0 < \delta \ll 1 \), the problem is to find a bound \( N_e \) on the number of experiments such that \( P \{|X| \geq \varepsilon \} \leq \delta \). Equivalently,

\[
P \left\{ \frac{|X|}{\sigma} \geq \frac{\varepsilon}{\sigma} \right\} \leq \delta \tag{3.3}
\]

that yields a bound on \( N \) as:

\[
N \geq \left( \frac{\theta^{-1}(\delta)}{\varepsilon} \right)^2 p(1-p) \tag{3.4}
\]

where \( \theta^{-1}(x) \equiv 1 - 2 \int_0^x e^{-\frac{t^2}{2}} dt \). Although the parameter \( p \) is unknown, due to the fact \( p((1-p) \leq 0.25 \forall p \in [0, 1] \), a bound in terms of the specified parameters \( \varepsilon \) and \( \delta \) can be obtained as:

\[
N \geq \left( \frac{\theta^{-1}(\delta)}{2\varepsilon} \right)^2 \tag{3.5}
\]
The above estimate of the bound on the required number of samples is less conservative than that obtained from the Chernoff bound and is significantly less conservative than that obtained from Chebyshev bound [63] that does not require the assumption of any specific distribution of $X$ except for finiteness of the $r^{th}$ ($r = 2$) moment.

3.3 Application of Language Measure – Optimal Control

In [34][35], an unconstrained optimal control policy is made for best trade-off between reaching good states $Q^+_m$ and avoiding bad states $Q^-_m$, and thus optimality is achieved in terms of supervised language measure.

3.3.1 Unconstrained optimal supervisor synthesis

Let $S \equiv \{S^0, S^1, \cdots, S^N\}$ be a set of supervisory control policies for the open loop plant automaton $G$ where $S^0$ is the null controller (i.e., no event is disabled) implying that $L(S^0/G) = L(G)$. Therefore the controller cost matrix $\Pi(S^0) = \Pi^0$ that is the $\Pi$-matrix of the open loop plant automaton $G$. For a supervisor $S^k, k \in \{1, 2, \cdots, N\}$, the control policy is required to selectively disable certain controllable events so that the following (elementwise) inequality holds: $\Pi^k \equiv \Pi(S^k) \leq \Pi^0$ and $L(S^k/G) \subseteq L(G), \forall S^k \in S$. The task is to synthesize an optimal cost matrix $\Pi^* \leq \Pi^0$ that maximizes the performance vector $\mu^* \equiv [I - \Pi^*]^{-1}X$, i.e., $\mu^* \geq \mu^k \equiv [I - \Pi^k]^{-1}X \ \forall \ \Pi^k \leq \Pi^0$ where the inequalities are implied elementwise. The research work in this direction is in progress and some of the results are reported in recent publications [34] [35].

3.4 Summary and conclusions

In this chapter, by introducing a dump state to a discrete event plant model $G$, an augmented plant model $G_{aug}$ can be defined to incorporate unmodeled dynamics. The resulting $\Pi$-matrix forms a stochastic matrix, and hence elements of $\Pi$-matrix
can be considered as the conditional probabilities of events defined at their corresponding states. An online procedure is then given to estimate the $\Pi$-matrix. A stopping rule is presented to characterize the estimation error of the parameter identification procedure.
Chapter 4

DES Control of a Behavior-based Mobile Robotic System

Designing a robotic system interacting with a possibly dynamically changing environment is truly a multi-disciplinary task, which includes the knowledge of control theory, robotics, computer vision, and artificial intelligence. This chapter first briefly reviews several traditional important methods and provide the justification of a discrete event system approach in the design of a mobile robotic system. The design and modeling of the DES behavior based mobile robotic system is then presented in great details. The plant model $G$ of the robotic system is identified based on the available sensing and action capability and a DES supervisor for an experimental scenario is then synthesized. Through experiments, performance of the design of the robotic system is quantitatively evaluated by the proposed language measure $\mu$ for both the open loop and the closed loop system. It is shown that the language measure $\mu$ can indeed be used to guide the design of high performance DES supervisor for high level mission planning.

4.1 System design

4.1.1 Artificial intelligence and behavior-based robotics

Traditional top-down planner-based or deliberative approaches, in which a world model is used by the planner to make the most appropriate sequence of actions
for the agent, have been criticized for poor scalability to real world applications and infeasibility of making real time response to sudden world changes [51] [14] [16] in that these approaches require the frequent replanning due to uncertainty in sensing and action, and changes in the environment.

Purely reactive behavior based bottom-up approaches embed the agent’s control strategy into a collection of preprogrammed parallel condition-action pairs with minimal internal state and no search, and hence real-time performance can be achieved [51]. The modularly designed component behaviors are activated in parallel producing various command to corresponding actuators. While these behaviors are suitable for representing the low-level continuous control and successful in making local incremental decisions, it was difficult to address within this paradigm the higher level tasks as traditionally addressed by artificial intelligence (AI) symbolic planning since no global state of the world is available. To address within subsumption architecture [13], behaviors are represented as finite state machines augmented with a set of input and output channels, providing a communication link to higher and lower layers. The lower level behaviors (e.g., walking, attraction to a goal, repulsion from an obstacle) are active most of the time. They trigger or deactivate the high level behaviors (e.g., wandering around, avoidance), while the higher level behaviors can “subsume” or override the output of the lower level. However, once the layered network is completed it remains fixed since the arbitration scheme is hardwired in the architecture.

The discrete event approach has been proposed to robotics research community by Košecká [40] for navigation of mobile robots with various tasks defined as discrete states, e.g., moving, steeraway, pathfollowing. Similarly, Feddema et al [26] use the discrete states Search, Rotate, Backup, and Track for the line following problem. Discrete event systems are appropriate for modeling of complex systems consisting of many interacting components operating in a dynamically changing environment driven by abrupt changes (also known as discrete events). In this sense, discrete event approach is similar to the AI symbolic planning approaches in which the overall robotic system is hierarchically structured into three levels: symbolic planning, reactive behaviors, and low level servoing. On the other hand, the discrete event approach provides a solid theoretical background and a formal treatment in the design of autonomous mobile robots. Considering the inherently
discrete nature of the robot’s behavior, the interruptions from the environment, and the discrete features extracted from sensors, it is more reasonable to model a mobile robot in the discrete event framework. Therefore the overall control of an autonomous mobile robot is formulated as a supervisory control problem of discrete event systems.

4.1.2 System architecture

The design objectives of the behavior based robotic system are

1. Fast response. All intelligence a robot needs to perform autonomously is local. The DES supervisor runs locally on the robot to interact with the lower layers of the system for decision making, for example, approach an object. Therefore, without communication, the robot should still reliably conduct its own task.

2. Flexible. For different mission objectives, it is only necessary to redesign DES supervisors without changing the robot’s component behaviors.

3. Resource saving. For status monitoring, only high level symbolic representation (e.g., searching, grasping) of the robot is communicated to remote site, thus saving the network bandwidth for other more important needs, for example, coordination information.

4. Failure tolerant. Robustness to component failure is achieved by a simple DES switching mechanism. Upon detecting a failure, a corresponding DES supervisor will be switched to at its proper state.

5. Coordination-oriented. Without any change of the available robot behaviors, the robot can participate with coordination among a set of robots. In other words, the robotic system is scalable.

Figure 4.1 shows the proposed DES behavior based multi-layer robotic system architecture. The system integrates the event driven discrete event dynamics modeled by finite state automaton and the time driven continuous dynamics modeled by ordinary differential equations through the carefully defined continuous/discrete interface. The DES supervisory control module (DCSM) communicates with the
robot’s continuous time-varying control module (CTCM) by sending and receiving of a set of discrete events (see Table 5.1). The DCSM is designed to be independent of the underlying physical process in that it provides a mechanism to interact with the CTCM. A DES controller is allowed to plug and play in DCSM where the DES controller of a special format is loaded. The CTCM consists of three blocks: C/D, D/C, and Continuously-time Controller. The CTCM connects to Player via a standard TCP socket for sending and receiving formatted messages that encode up-to-date sensor readings and continuously varying commands of reference signals, respectively, at 10 Hz. The control strategy is event-driven, in which CTCM (in particular, C/D block) generates discrete events based on the continuous sensor data received from Player. The discrete events are transmitted to DCSM in symbolic form. DCSM reacts immediately by sending out a discrete event command back to CTCM according to the currently loaded supervisor. After receiving a particular event from the DES controller, CTCM sends out a set of continuous reference signals to Player for completion of this behavior to maneuver the robot accordingly. The robot continues executing this behavior until the sensor readings trigger the occurrence of a new event. Under this architecture, DES control strategy for both real robot and a simulator are designed and exercised. The hardware integration of the self-customized real robot is given below, a Pioneer 2 AT\(^1\).

\(^1\)Pioneer 2 AT is a product of ActivMedia Inc.
4.1.3 Hardware integration

The Pioneer 2 AT robot, as shown in Figure 4.2, is equipped with a SICK LMS200 laser range finder for obstacle avoidance and distance measurement. The laser range finder provides depth information for a 180° field of view with an angular resolution of 0.5° and an accuracy of 1cm(±15). The robot uses a SONY EVI-D30 pan-tilt camera in conjunction with a Sensoray 311 PC104 frame grabber for objection recognition and tracking. For communication between the robot and a remote computer for the purpose of monitoring, a Lucent Technologies WaveLan 11 Mbps radio ethernet (2.4GHz) is employed. In practice, a bandwidth of up to 2Mbps is achieved. An Advantech© on-board computer powered by a Transmeta Crusoe Processor TM5400 500MHz CPU performs all the real time computations. It has 310MB memory in total, including 256MB PC133 RAM and 64MB flash memory, and 20GB hard disk. All the devices are powered by three 12v sealed lead acid batteries. DC/DC converters are used to provide appropriate power to various devices. The major actuators available on the robots are motors for wheel drive, camera, and gripper.

4.1.4 Stage simulator and robot kinematics

Stage2 simulates a population of mobile robots, sensors, and environmental objects in a two-dimensional bitmapped environment [29]. Stage enables 1) rapid development of controllers that will eventually drive real robots; 2) robot experiments without access to the real hardware and environments. Stage emulates a set of devices with good enough fidelity, including sonar, laser range finders, visual color segmenters, and a versatile mobile robot base with odometer. A comparison between real sensor readings and simulated sensor readings is shown in Figure 4.4 4.5 4.6 4.7 where the environment set up for both real robot and simulated robot is almost the same. In Figure 4.5 and Figure 4.7, the brown color beams represent sonar readings and light blue colored areas bounded by dark blue dotted lines represent the laser readings. Clearly, real sonar readings are much more noisy than simulated sonar readings. Since the real experiment area is much smaller than that

\footnote{Player/Stage is developed at University of Southern California under GNU Public License. It is available at http://playerstage.sourceforge.net/}
of the simulation, it is observed that in those areas without obstacles, the boundary of the laser readings is a straight line whereas the boundary of the laser readings in simulation forms an arc. All devices in Stage are accessible through Player’s standard interfaces, as if they were real hardware. In design of the robotic system, the identical control software without modification is used on both real robot and Stage simulator. The behaviors of the real robot and the simulated robot are comparable, as explained below. The full configuration of the Pioneer 2 AT robot is described as a directed point, i.e., an ordered pair \( \mathbf{q} = (x, y, \theta) \in \mathcal{C} = \mathbb{R}^2 \times S^1 \), where \((x, y)\) are the coordinates with respect to some global reference frame \(F_g\), \(\theta\) the current orientation with respect to \(x\)-axis, and \(\mathcal{C}\) is the robot configuration space. Motion of the nonholonomic robot is constrained by

\[
-\dot{x} \sin \theta + \dot{y} \cos \theta = 0
\]  

(4.1)
which limits the robot to two degree of freedom (DOF) and hence a motion command of this robot can be fully characterized by two motion parameters only at any given configuration. The kinematics of Pioneer 2 AT can then be expressed by the following equation

$$
\begin{pmatrix}
\dot{x} \\
\dot{y} \\
\dot{\theta}
\end{pmatrix} =
\begin{pmatrix}
\cos \theta & 0 \\
\sin \theta & 0 \\
0 & 1
\end{pmatrix}
\begin{pmatrix}
v \\
w
\end{pmatrix}
$$

(4.2)

where the steering input $w$ controls the angular velocity $\dot{\theta}$ and the driving input $v$ controls the linear velocity along the direction of the wheel. To achieve higher fidelity, continuously varying system identification of the Pioneer 2 AT robot motion dynamics is performed and the resulting model, along with sonar, laser range finder, vision camera, and gripper, is used in the simulation experiments. The experiments were conducted with pseudo-random inputs in both time and speed under the following constraints: (i) $2 \text{ sec} \leq t_d \leq 10 \text{ sec}$; (ii) $-300 \text{ mm/s} \leq v \leq 300 \text{ mm/s}$; (3) $-80 \text{ deg/s} \leq \omega \leq 80 \text{ deg/s}$, where $t_d$ is the time duration in which the robot runs at constant speed; $v$ and $\omega$ are the linear and angular velocities of the robot, respectively. By analyzing the experimental data, a multivariable ARX model was found to be inaccurate to capture the nonlinear dynamics of this nonholonomic robot, therefore the subspace technique for system identification of MIMO systems [58] was applied using the MATLAB toolbox provided in [58]. The resulting state space model, having the state vector $x = [v \ w \ \dot{v} \ \dot{w}]$, is given as:

$$
\begin{align*}
x_{n+1} &= Ax_n + Bu_n \\
y_n &= Cx_n
\end{align*}
$$

(4.3)  

(4.4)

$$
A =
\begin{bmatrix}
0.8579 & 0.0958 & -0.3488 & -0.0118 \\
-0.0012 & 0.6882 & -0.1532 & 0.0540 \\
0.1415 & 0.3104 & 0.6644 & -0.5791 \\
-0.0893 & 0.2350 & 0.3603 & 0.6374
\end{bmatrix}
$$
Figure 4.3 shows that the results of the random walk experiment on the robot system are in good agreement with those predicted by the model. Note that, in Figure 4.3, the variations of the model predictions from the actual measured data are largely due to environmental disturbances (e.g., floor friction and unflatness).

\[
B = \begin{bmatrix}
-0.3081 & -0.0037 \\
-0.0032 & 0.6524 \\
0.4281 & -0.2781 \\
-0.1809 & -0.4719
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
-0.4808 & 0.2368 & -0.5129 & -0.2716 \\
0.0138 & 0.4664 & 0.0547 & 0.2910
\end{bmatrix}
\]
4.1.5 Software design and integration

The robot on-board computer is running Linux Redhat 7.3 kernel 2.4.7-10. The Player robotic development platform is employed for both the real robotic system and the simulator Stage. Player is a robot device server which manages the robot’s sensors and actuators. The robot continuous time control block in Figure 4.1 talks to Player through local port following the publishing/subscribing mechanism. Both the DES supervisor and the robot continuous time controller run locally on the robot on-board computer. If coordination is needed, only the DES supervisor exchanges discrete events possibly attached with certain continuous and/or discrete
parameters with coordinator (or other robots). Table 4.1.5 gives a rough picture of the distributions of CPU time among a set of applications that are directly related to the robot’s operation. ACTS is the program for color segmentation and connected components analysis at run time. It consumes about 30% of CPU time for object detection. Note although sshd is used for remote login to the robot for more detailed monitoring, this is truly not necessary.

<table>
<thead>
<tr>
<th>Applications</th>
<th>Version</th>
<th>CPU usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTS</td>
<td>2.0</td>
<td>30%</td>
</tr>
<tr>
<td>Player</td>
<td>1.3.1</td>
<td>≤1%</td>
</tr>
<tr>
<td>Continuous control</td>
<td>1.3.1</td>
<td>0.4%</td>
</tr>
<tr>
<td>DES supervisor</td>
<td>1.3.1</td>
<td>≤0.2%</td>
</tr>
<tr>
<td>sshd</td>
<td>3.4</td>
<td>20%</td>
</tr>
</tbody>
</table>

Table 4.1. CPU usage of various applications

4.1.6 Design of robot behaviors

A behavior is a set of processes involving sensing and action against the environment. Behaviors can be designed at a variety of levels of abstraction. In general, they are made to be higher than the robot’s atomic actions, e.g., *turn left by 30 degree*, and they extend in time and space. Some commonly implemented behaviors include: *go home, search object, avoid obstacle, pick up object*, etc. In the discrete event setting, a behavior is a controllable event. The union of these behaviors is the controllable event set $\Sigma_c$. The set of uncontrollable events $\Sigma_u$ is the set of all possible response of the robot during the interaction with its environment. It consists of the results of the robot’s behaviors, including a successful or unsuccessful completion of the controllable events (e.g., *approach a target, grab an object*) and an interruption of the robot’s current behaviors (e.g., *find an object, detect obstacle*).

The C/D and D/C blocks in Figure 4.1 are the interface between the discrete event dynamics and continuous time dynamics of the robot system. The D/C block is a map $\phi : \Sigma \to \mathbb{R}^m$ that converts each controllable event into the continuous reference signal as follows.

$$r(t) = \phi(\sigma_k) \quad t_k \leq t < t_{k+1} \quad (4.5)$$
where $\sigma_k \in \Sigma$ is the most recent controllable event and $t_k$ is the time of the $k$-th discrete event occurrence. The C/D block is a map $\psi$ that converts the state space $x$ of the continuous time system into the set of discrete events $\Sigma$. An event $\sigma_\tau$ is generated at time $t = \tau$ if there exist $\epsilon, \delta > 0$, such that $\forall 0 < \epsilon < \delta$

$$h(x, u; \tau) = 0, \text{ and } h(x, u; \tau - \epsilon) \neq 0$$

$$\sigma_\tau = \psi(x)$$  \hspace{1cm} (4.6) \hspace{1cm} (4.7)

Equation 4.6 defines a closed set $\Omega$, when the state trajectory enters $\Omega$ from outside for the first time, an event is generated. Note if there is a segment of trajectory $x$ during $[t_1, t_2]$ that satisfies Equation 4.6, then the event is defined to be occurred at $t_1$. The C/D block (also known as event generator) is fully specified by the pair $(h, \psi)$.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Controllable</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>approach the object</td>
<td>✓</td>
</tr>
<tr>
<td>A</td>
<td>avoid obstacle successfully</td>
<td>✓</td>
</tr>
<tr>
<td>c</td>
<td>reach goal with an object</td>
<td>✓</td>
</tr>
<tr>
<td>C</td>
<td>find an object but gripper full</td>
<td>✓</td>
</tr>
<tr>
<td>d</td>
<td>drop an object</td>
<td>✓</td>
</tr>
<tr>
<td>g</td>
<td>grab an object</td>
<td>✓</td>
</tr>
<tr>
<td>h</td>
<td>return to home</td>
<td>✓</td>
</tr>
<tr>
<td>i</td>
<td>ignore the current observed target</td>
<td>✓</td>
</tr>
<tr>
<td>l</td>
<td>lost the target</td>
<td>✓</td>
</tr>
<tr>
<td>o</td>
<td>obstacle ahead</td>
<td>✓</td>
</tr>
<tr>
<td>p</td>
<td>drop an object successfully</td>
<td>✓</td>
</tr>
<tr>
<td>P</td>
<td>fail to drop an object</td>
<td>✓</td>
</tr>
<tr>
<td>q</td>
<td>grab an object successfully</td>
<td>✓</td>
</tr>
<tr>
<td>Q</td>
<td>fail to grab an object</td>
<td>✓</td>
</tr>
<tr>
<td>T</td>
<td>find goal with an target</td>
<td>✓</td>
</tr>
<tr>
<td>s</td>
<td>search recognizable target</td>
<td>✓</td>
</tr>
<tr>
<td>v</td>
<td>avoid obstacle</td>
<td>✓</td>
</tr>
<tr>
<td>w</td>
<td>reach target without an object</td>
<td>✓</td>
</tr>
<tr>
<td>W</td>
<td>find object 1</td>
<td>✓</td>
</tr>
<tr>
<td>x</td>
<td>lost the goal</td>
<td>✓</td>
</tr>
<tr>
<td>X</td>
<td>find target 2</td>
<td>✓</td>
</tr>
<tr>
<td>y</td>
<td>lost an object</td>
<td>✓</td>
</tr>
</tbody>
</table>

Table 4.2. The discrete event set $\Sigma$ for Pioneer 2 AT robot
Based on all possible robot behaviors $\Sigma_c$ limited by its hardware and all possible response $\Sigma_u$ of the robot during the interaction with its environment in the lab, the set of discrete events $\Sigma$ is identified as listed in Table 4.1.6, where $\Sigma_c$ and $\Sigma_u$ are given by

$$\Sigma_c = \{a, d, g, h, i, s, v\}$$

$$\Sigma_u = \{A, c, C, l, o, p, P, q, Q, T, w, W, x, X, y\}$$

It should be noted that no two events are triggered at the same time instant.

### 4.1.7 Visual servoing

Consider the case where $R_{Fm}(t) \in SE(2)$ is a time-parameterized curve in the Euclidean Group $SE(2)$ representing a trajectory of the robot. The rigid body motion of the mobile frame $F_m$ is attached to the robot and changes over time with respect to a fixed spatial reference frame $F_r$, as shown in Figure 4.8 (a). Let $V_{Fm}(t) = [x, y, z]^T \in \mathbb{R}^3$ be the position vector of the origin of frame $F_m$ from the origin of reference frame $F_r$ and the orientation angle $\theta$ is defined in the counterclockwise sense about the $z$-axis, as shown in Figure 4.8 (a). Now, suppose a camera mounted on the Pioneer 2 AT robot which is facing downward with a tilt angle $\phi > 0$ and it is above the ground plane by distance $h_c$, as shown in Figure 4.8 (b). The $x$-axis of camera coordinate frame $F_c$ is chosen to be the optical axis of

![Figure 4.8](image-url)

*Figure 4.8. (a) Model of Pioneer 2 AT robot; (b) The side-view of P2AT with a camera facing downward at a tilt angle $\phi > 0$*. 
the camera, the \( y \)-axis of \( F_c \) coincides with \( y \)-axis of \( F_m \), and the optical center of the camera coincides with the origins of both \( F_m \) and \( F_c \). Then the kinematics of a point \( p_c = [x, y, z]^T \) attached to the camera frame \( F_c \) is given in the instantaneous camera frame by [50]:

\[
\begin{pmatrix}
\dot{x} \\
\dot{y} \\
\dot{z}
\end{pmatrix} = \begin{pmatrix}
\cos \phi & 0 & -y \cos \phi \\
0 & \sin \phi & z \sin \phi + x \cos \phi \\
\sin \phi & 0 & -y \sin \phi
\end{pmatrix} w
\]

(4.8)

\[\begin{align*}
\text{Figure 4.9.} & \quad \text{The closed-loop vision-guided navigation system for the Pioneer 2 AT robot} \\
\text{It is well-known that the system in Equation 4.2 does not satisfy Brockett’s condition [12]}. & \quad \text{3}\text{. For this reason, visual servoing based switched controllers are developed since these have the potential to provide better overall performance. It is true that as a feedback sensor, vision has a number of significant disadvantages, including a relatively low sample rate, significant latency (transport delay of pixels from camera to framegrabber to memory, image processing algorithms, etc), sensitive to occlusion and lighting, coarse quantization, and constraints to robot motion. However, in practice, visual servoing has been very successful for local navigation in the literature. The advantage of visual feedback control is that the accuracy of the closed-loop performance can be made relatively insensitive to calibration errors and nonlinearities of the open-loop system. In addition, it is the only source of information that the robot can use in a feedback loop to navigate} \\
\text{3A nonholonomic systems with more degrees of freedom than controls can not be stabilized by continuously differentiable, time invariant, state feedback control laws}\end{align*}\]
to an identified target in an unknown environment since it does not process any knowledge of the current positions of itself and the target. Moreover, localization techniques, if used, are in general very noisy, erroneous, and computation intensive, making it very difficult to have robust approach and grab behaviors. Figure 4.9 shows a block diagram of the visual servoing system. In the image plane, a feature parameter space, denoted by $\mathbf{f} \in \mathcal{F}_I$, is adopted to represent those parameters that describe objects being tracked. In the robotic system, a colored destination and a set of same-size different-colored cylindrical objects are being tracked by the real time color blob detection software ACTS. The output of ACTS is further regrouped by Player into a vector of blobs representing the feature parameters of any detected objects. Each blob is parameterized by $B(y_b, z_b, A_b)$, where $(y_b, z_b)$ are the coordinates of the blob $B$ centroid in the image plane and $A_b$ is the area of the blob $B$.

The idea of visual servoing is to regulate a particular error function to zero. In the robotic system, visual servoing attempts to keep a feature staying at the desired image plane coordinate. To do that, the wheel motion controller and camera motion controller are engaged accordingly so that the blob $B$ stays constantly in both $y$ and $z$ direction, respectively. Therefore the $x$ value along the optical axis of the camera in Equation 4.8, i.e., the depth of the blob, is not of particular interest. For steering tasks at low speed and normal driving conditions dynamics effects are not very prevalent, therefore for simplicity, kinematic models are used to derive control laws. Visual servoing starts to take over robot motion control when an object is detected by ACTS. It operates in two distinct stages. The first stage is to steer the robot towards the object by minimizing the angular difference between the robot and the object, i.e., $\Delta \theta = 0$. The second stage is then to move the robot to the desired point while keeping centering the blob in the image plane. The design of robot motion control is described below.

**Robot motion control – first stage**

On the image plane, the centroid of a blob $(y_b, z_b)$ represents the angular offset between the robot and the object. In particular, $-\text{sign}(y_b)|y_b|$ tells the angular difference. Therefore, a proportional control law is applied to steer the robot
accordingly.

\[ w = -k_w \text{sign}(y_b)|y_b| \]  

\hspace{1cm} (4.9)

The linear velocity \( v \) is determined by the blob’s \( z \)-coordinate on the image plane at the time the robot detects an object, denoted as \( z_{b0} \). During search, the camera tilt angle \( \phi \) is set to be zero. Therefore, if \( z_{b0} \) is negative, it means the object is close to the robot; if \( z_{b0} \) is positive, it means the object is far away from the robot. Based on this practical experience, the linear velocity \( v \) is given by

\[
 v = \begin{cases} 
 v_0 + k_{v1}\text{sign}(z_{b0})|z_{b0}|, & z_{b0} > 0 \text{ and } |y_b| > y_{th} \\
 v_0 + k_{v2}\text{sign}(z_{b0})|z_{b0}|, & z_{b0} \leq 0 \text{ and } |y_b| > y_{th} \\
 v_1, & |y_b| \leq y_{th} 
\end{cases}
\] 

\hspace{1cm} (4.10)

To keep tracking the object when the robot is approaching to it, the camera is set to tilt according to \( z_b \) so that the feature (blob) maintains in the camera’s field of view once approaching behavior is triggered. The camera motion control command is sent through a RS232 connection between the on-board computer and the camera by VISCA protocol. A proportional control law is applied to the camera motor.

\[ w_t = -k_c \text{sign}(z_b)|z_b| \hspace{1cm} \phi \leq \phi_{cr} \] 

\hspace{1cm} (4.11)

where \( \phi_{cr} \) is the pre-specified camera tilt angle above which visual tracking of object becomes unstable due to smaller blob and worse illumination of the object.

**Robot motion control – second stage**

During this stage, visual tracking is no longer available, however, the robot is very close to the object and the orientation of the robot with respect to the object is supposed to be proper, therefore, the robot motion control law is given by

\[
 v = \begin{cases} 
 v_2, & t \leq t_d \\
 -v_3, & t > t_d 
\end{cases} 
\] 

\[ w = \begin{cases} 
 0, & t \leq t_d \\
 -w_1, & t > t_d 
\end{cases} 
\] 

\hspace{1cm} (4.12)

where \( t_d \) is the time duration within which the robot should be able to reach the object. The overall performance of visual servoing during robot approaching an object has been validated in experiments, as shown in Figure 4.10. The robot
stops if two laser beams between the gripper are broken, and the event “reach object” is hence generated. However, for fault tolerant design, a time limit \( t_d \) is set for the final stage of approaching the object. If time is expired and the two laser beams are still not broken, the robot backs up, rotates away, resets the camera, and generates the “object lost” event. This is justified due to the imperfections of “real” systems or a dynamically changing environment, for example, the object may be taken away during this period, the robot orientation with respect to the object is not proper due to floor friction, or even the gripper malfunctions and did not detect the breaking of laser beams.

**Figure 4.10.** Experimental validation of robot visual servoing

In practice, the blob detection is sensitive to noisy environment, particularly, the illumination condition of the object that the robot is tracking. Therefore, if the blob under tracking is lost, a “object lost” event is then generated and reported to the DES controller. In addition, the “approaching” behavior may also
be interrupted by “found obstacle” event generated by another sensor module, for example, the laser ranger finder which measures the distance.

Figure 4.11. Object detection. (a) ACTS color blob detection (overlay mode) (b) ACTS color blob detection (background subtraction) (c) Color blobs interpreted by Player

Figure 4.12. DES supervisory control of Pioneer 2 AT (behaviors: approach, reach, grab)

4.1.8 Obstacle avoidance

Obstacle avoidance is an essential behavior that all mobile robots need to acquire for autonomous navigation in an unknown (possibly changing) environment. Generally speaking, there are two types of distance sensor based obstacle avoidance algorithms, namely global and local. Equipped with a global map of the mobile
robot’s operational environment, global obstacle avoidance algorithms can be naturally integrated into the obstacle-free path planning of the mobile robot, such methods can be found in [37] [4] [39] as applications of Potential Field (PF). Local obstacle avoidance algorithms generally do not require any global map since decision is made on sensor readings. There exists a number of algorithms that integrate both global and local information, such as Occupancy Grids and Vector Field Histogram (VFH) family. Vision-based obstacle avoidance algorithms still suffer from poor performance due to real time constraints and sensitivity to environment lighting conditions. In the robotic system design, a procedure that first performs filtering on noisy sensor (LRF) readings and then applies VFH+ algorithm for obstacle avoidance is chosen.

4.1.9 Potential field

In potential field methods, a robot is modeled as a particle with electric charge. Each obstacle is modeled as a number of particles on its boundary with the same electric charge as the robot. The goal is modeled as a particle with the opposite electric charge to the robot. An artificial potential function is first defined at the robot’s position \((x, y)\).

\[
U(x, y) = U_g(x, y) + U_r(x, y)
\]  

(4.13)

where \(U_g(x, y)\) is the attractive potential produced by the goal particle at \((x, y)\), and \(U_r(x, y)\) is the repulsive potential produced by all the obstacle particles. Under this model, the robot’s collision-free path planning at any given location \((x, y)\) is simple: examine the robot neighborhood, and move to the location with the lowest potential field value. In practice, the vector sum of the repulsive force \(\vec{F}_r(x, y)\) and attractive force \(\vec{F}_g(x, y)\) on the robot at \((x, y)\) forms a force \(\vec{F}\) whose direction is then the most appropriate direction that the robot should move on.

The primary advantage of potential field methods is that they are extremely fast to compute. However, since they typically consider only a small subset of obstacles near the robot, as pointed in [39], there are four significant drawbacks that are inherent to potential field methods and independent of the particular implementation.
1. Robots can be trapped due to local minima, as known as cyclic behavior.

2. Robots are generally unable to travel through narrow passages.

3. Robots can oscillate between obstacles.

4. Robots can oscillate in narrow passages.

4.1.10 LRF readings segmentation and noise rejection

Sensor readings from laser range finder could be noisy, as shown in Figure 4.13. The half circles in Figure 4.13 show the boundary of robot’s safe zone beyond which it may collide with an obstacle. A simple filtering method is applied. The main idea is to subdivide the laser scanner readings in each scan into small sets of neighboring points (segments). A segment is a set of measurement data points that are close enough to each other and thus is assumed to belong to the boundary of the same object. Those data points that do not belong to any of the segments are rejected by replacing with neighboring data. Let the set of laser range data measured at \( t = t_k \) be \( \mathcal{R}_k \).

\[
\mathcal{R}_k = \{ l_i \mid l_i = (r_i, \theta_i), i \in [0, N] \}\tag{4.14}
\]

where \( l_i \) is the \( i \)-th data points with distance \( r_i \) and angle \( \theta_i \), and \( N \) is the total number of data points in each reading. The distance between two consecutive points is given by

\[
d(l_i, l_{i+1}) = \sqrt{r_{i+1}^2 + r_i^2 - 2r_{i+1}r_i \cos(\theta_{i+1} - \theta_i)}\tag{4.15}
\]

Let \( l_i \in \mathcal{C}_j \), where \( \mathcal{C}_j \) is the \( j \)-th segment of the reading \( \mathcal{R}_k \). \( l_{i+1} \in \mathcal{C}_j \) if

\[
d(l_i, l_{i+1}) \leq c_0 + c_1 \min\{r_i, r_{i+1}\}\tag{4.16}
\]

where \( c_1 = \sqrt{2(1 - \cos(\theta_{i+1} - \theta_i))} \). The constant \( c_0 \) allows an adjustment of the algorithm to noise level. 0.5m is chosen in the experiments. The constant \( c_1 \) is the coefficient associated with the distance between consecutive points in each scan under the assumption that the two data points have no abrupt change in distance.
\[
\min\{r_i, r_{i+1}\}
\]

is a conservative choice that gives the obstacle avoidance algorithm some flexibility to design its cost function. A minimum size of 5 data points for being a segment is used, i.e., \(|C_j| \geq 5, j = 1, 2, \ldots\). Figure 4.14 shows the samples in Figure 4.13 after filtering.

A simple polar histogram of the filtered laser range data is then developed for the purpose of obstacle avoidance. In implementation, the measurement data points are partitioned into \(n = \frac{360^\circ}{\alpha}\), where \(\alpha = 5^\circ\) is the resolution of an angular sector. In each sector \(\beta_k\), compute

\[
\forall k = 1, \ldots, n, \quad l_k^{\min} = \min_j \{l_j | l_j \in \beta_k\}, \quad w_k = \begin{cases} 
    l_k^{\min}, & l_k^{\min} \leq d_{\min} \\
    0, & l_k^{\min} > d_{\min}
\end{cases} \quad (4.17)
\]

If there are some sectors that have non-zero \(w_k\), one choice for the next direction of motion of the mobile robot is the middle angle of the largest open area on the histogram plot. Figure 4.16 shows an example of such a histogram, and the robot can move toward about the 300\(^\circ\) direction. From experiments, such a simple polar histogram-based obstacle avoidance is effective most of the time and works in real time, however, the robot may be trapped in corners.

### 4.1.11 Vector field histogram

VFH+ [75] is implemented in the robotic system for robust obstacle avoidance. The VFH+ method is an improved version of the Vector Field Histogram (VFH) method originally developed by Borenstein and Koren [9] [8] for mobile robot real-time, local obstacle avoidance. The basic idea of the VFH+ method is that the 2-dimensional map grid is reduced to a 1-dimensional polar histogram \(H\) that comprises \(n\) angular sectors of width \(\alpha\) degrees. The polar histograms constructed over time are always around the robot’s momentary location. Each cell in \(H\) represents the obstacle density in that particular direction. Based on the obstacle density profile, which includes the masked polar histogram and a cost function, a decision is then made to the safest direction for the robot to move on. The VFH+ method consists of four stages for data reduction and computation of the new direction of motion. Details of this method can be found in[75]. During the searching behavior, the robot is running at the speed of 0.2m/s due to the trade-off
4.2 Experimental scenario and parameter identification

As stated early in this chapter, the robot can perform a set of behaviors, including search, home, avoid, ignore, approach, grab, and drop. The experiment scenario is to have the robot collecting colored objects to a pre-specified destination. There are two colored (green and pink) objects randomly located in the field every time after the robot collects one of them back to the destination. Without any discrete event supervisor’s control, the robot starts with random search, upon detecting an object, it has two options: approach or ignore the object. If ignoring the object, it will continue the random search. Otherwise, it will approach the object. If successfully reaching the object, it will grab the object and begin the random search.
to find the destination provided that the object grabbing is successful. Again, if
the destination is found, the robot can either *approach* or *ignore*. Once the robot
decides to approach the destination and succeeds to reach the destination, it will
drop the object. The robot repeats the same procedures until stopped by human
operator.

Given the above experiment scenario, the robot plant model $G$ is designed and
shown in Figure 4.22. A non-blocking \(^4\) supervisor \(S\) is then synthesized according to the following specification language \(K\).

1. Whenever an object is found, the robot approaches it;
2. The robot approaches the destination if it has grabbed an object and found the destination.

The resulting supervisor \(S\) is shown in Figure 4.23. One can easily check that \(S\) is controllable with respect to the plant \(G\) and uncontrollable event set \(\Sigma_u\). Figure 4.21 shows a typical experimental trajectory of the Pioneer 2 AT under the DES supervisor \(S\).

Figure 4.24 and Figure 4.25 show the convergence of some non-zero elements in \(\hat{\Pi}\)-matrix obtained in both real experiments and simulation according to the

\(^4\)A supervisor \(S\) is nonblocking if \(pr(L(S/G)) \cap L_m(G) = L(S/G)\) (See [66]).
Figure 4.21. A trajectory of the real robot in experiment

Figure 4.22. DES plant model $G$ for the experiment scenario

parameter identification procedure proposed in Chapter 3. One can see the probability of finding object 2 at supervisor state 1 in real experiment is higher than then that of simulation. This is because realization of random positioning of the objects in simulation is much easier than in real experiment. In fact, due to lighting condition, the robot has a higher probability of losing an object, as shown in the Figures. So the objects are randomly placed in a much smaller area of the available test field in order for the robot to find them.
Figure 4.23. DES supervisor $S$ for the experiment scenario

Figure 4.24. Convergence of some non-zero $\tilde{\Pi}$ elements in experiment
Table 4.3. The state set $Q$ of the plant model $G$ and its $X$-vector

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
<th>$\chi$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>robot ready for mission</td>
<td>0.0</td>
</tr>
<tr>
<td>1</td>
<td>searching for target</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>found object 2</td>
<td>0.1</td>
</tr>
<tr>
<td>3</td>
<td>found an object but gripper is full</td>
<td>0.0</td>
</tr>
<tr>
<td>4</td>
<td>approaching target</td>
<td>0.0</td>
</tr>
<tr>
<td>5</td>
<td>ready to drop object at destination</td>
<td>0.0</td>
</tr>
<tr>
<td>6</td>
<td>ready to grab object</td>
<td>0.0</td>
</tr>
<tr>
<td>7</td>
<td>drop an object successfully</td>
<td>0.8</td>
</tr>
<tr>
<td>8</td>
<td>grab an object successfully</td>
<td>0.4</td>
</tr>
<tr>
<td>9</td>
<td>avoiding obstacle</td>
<td>0.0</td>
</tr>
<tr>
<td>10</td>
<td>lost target</td>
<td>-0.2</td>
</tr>
<tr>
<td>11</td>
<td>dropping an object at destination</td>
<td>0.0</td>
</tr>
<tr>
<td>12</td>
<td>grabbing an object</td>
<td>0.0</td>
</tr>
<tr>
<td>13</td>
<td>failed grabbing or dropping</td>
<td>-1</td>
</tr>
<tr>
<td>14</td>
<td>found object 1</td>
<td>0.3</td>
</tr>
<tr>
<td>15</td>
<td>found the destination</td>
<td>0.0</td>
</tr>
<tr>
<td>16</td>
<td>ignoring an object</td>
<td>-0.1</td>
</tr>
</tbody>
</table>

Table 4.4. $\Pi$-matrix ($17 \times 17$) for the discrete-event model $G$

4.3 Performance evaluation in measure $\mu$

For purpose of performance evaluation in language measure $\mu$, $X$-vector is needed. The state space of plant model $G$ and their associated $\chi$ values are listed Table 4.2. A value of 0.4 is assigned to state 8 where the robot grabs an object successfully. The state where the robot successfully drops an object at the destination, which also represents the end of the current mission, is assigned a value of 0.8. The two colored object are given different level of importance, therefore, if the robot...
discovers a pink object, it gets a credit of 0.3 whereas a green object has the value of only 0.1. Lost and ignored an object are given a penalty of -0.2 and -0.1, respectively. By Equation 2.29, the final performance comparison between the open loop and closed loop system in both real experiments and simulation is given by

\[
\mu_{\text{exp}}(L(G)) = 0.1640, \quad \mu_{\text{exp}}(L(S/G)) = 0.1987 \\
\mu_{\text{sim}}(L(G)) = 0.0799, \quad \mu_{\text{sim}}(L(S/G)) = 0.1349
\]

One can see that the theoretical results based on real experiments and simulation are consistent, that is, the performance of closed loop system $S/G$ is better than that of the open loop plant $G$. The cumulative performance comparison in simulation is shown in Figure 4.26. In about 5 hours mission of collecting objects,
the robot under DES control performs much better in both language measure and number of successful missions. As expected, whenever the robot finds an object, the DES controller always lets the robot go and grab it, while without DES control, the robot may ignore it and keep searching. In a later chapter, it will be shown that the same DES controller may not have the same performance in coordination with other robots since environment is changed by the participation of other robot(s), which in turn resulting a new $\tilde{\Pi}$-matrix.

### 4.4 Summary

This chapter presents the details of the discrete event behavior based design of a real robotic system and its simulator. Based on an experiment scenario, the discrete event plant model $G$ and a DES controller $S$ are designed. The $\tilde{\Pi}$-matrix
is identified according to the procedure presented in Chapter 3 in both real experiment and simulation. The $X$-vector is given according to the mission objective of the experiment scenario. Through both theoretical language measure computation and empirical cumulative performance, it is numerically shown that the designed supervisor has better performance than the open loop system. While the language measure $\mu$ is proven to be a useful quantitative measure for the performance evaluation of DES controller, the robot simulator demonstrates its high fidelity with respect to the real system, and hence can be used for further study of more complex scenarios.
Chapter 5

Optimal Robot Behavioral Selection Using Language Measure $\mu$

This chapter presents a discrete event supervisory control approach for robot behavior selection in terms of language measure $\mu$ [81][71]. In the discrete-event setting, behaviors of robots are modeled as regular languages that can be realized by deterministic finite state automata (DFSA) [66]. The (regular) sublanguage of a controlled plant could be different under different supervisors that are constrained to satisfy different specifications [66]. Such a partially ordered set of slublanguages requires a quantitative measure for total ordering of their respective performance. The language measure [81] [71] serves as a common quantitative tool to compare the performance of different supervisors and is assigned an event cost $\tilde{\Pi}$-matrix and a state characteristic $X$-vector. Event costs (i.e., elements of the $\tilde{\Pi}$-matrix) based on plant states, where they are generated, are physical phenomena dependent on the plant behavior, and are similar to the conditional probabilities of the respective events. On the other hand, the $X$-vector is chosen based on the designer’s perception of the individual state’s impact on the system performance. In the performance evaluation of both the unsupervised and supervised plant behavior, the critical parameter is the event cost $\tilde{\Pi}$-matrix. Since the plant behavior is often slowly time-varying, there is a need for on-line parameter identification to generate up-to-date values of the $\tilde{\Pi}$-matrix within allowable bounds of errors.
5.1 DES control synthesis in $\mu$-measure

In the conventional discrete event supervisory (DES) control synthesis [66], the qualitative measure of maximum permissiveness plays an important role. For example, under full state observation, if a specification language $K$ is not controllable with respect to the plant automaton $G$ and the set $\Sigma_u$ of uncontrollable events, then a supremal controllable sublanguage $\text{sup}C(K) \subseteq K$ yields maximal permissiveness. However, increased permissiveness of the controlled language $L(S/G)$ may not generate better plant performance from the perspective of mission accomplishment. This section relies on the language measure $\mu$ to quantitatively synthesize a DES control policy. The objective is to design a supervisor such that the closed-loop system $S/G$ maximizes the performance that is chosen as the measure $\mu$ of the controlled plant language $L(S/G)$. The pertinent assumptions for the DES control synthesis are delineated below.

A1 (Cost redistribution) The occurrence probabilities of controllable events in a sublanguage $K = L(S) \subseteq L(G)$ are proportional to those in $L(G)$. For all $q \in Q_S$, where $Q_S$ is the state space of the supervisor $S$, and $\sigma \in \Sigma_S(q)$, where $\Sigma_S(q)$ is the set of events defined at $q \in Q_S$.

$$\tilde{\pi}_S[q, \sigma] = \frac{\tilde{\pi}_G[q, \sigma]}{\sum_{\sigma \in \Sigma_S(q)} \tilde{\pi}_G[q, \sigma]}$$ (5.1)

A2 (Event controllability) For any transition $\delta(q, \sigma)$ defined in the plant $G$, such that $\sigma \in \Sigma_{uc}(G)$ and $q \in Q$, it is kept enabled in a supervisor $S$.

Under assumption A1, the sum of event costs defined at the state $q$ of the supervisor $s$ is equal to that of state $q$ of the plant $G$, i.e.,

$$\sum_{\sigma \in \Sigma_S(q)} \tilde{\pi}_S[q, \sigma] = \sum_{\sigma \in \Sigma_G(q)} \tilde{\pi}_G[q, \sigma]$$ (5.2)

Lemma 5.1.1. (Finiteness) Given a DES plant $G$, there is only a finite number of controllers $S_i, i \in \mathcal{I}_c$, where $\mathcal{I}_c$ is the set of controllers with cardinality $|\mathcal{I}_c| = n_c$, such that for every $i \in \mathcal{I}_c$, $L(S_i) = L(S_i/G) \subseteq L(G)$.

Proof. Under assumption A2, it suffices to show that the number of all possible
permutations of disabling controllable events defined on all states in \( G \) is finite. The worst case is that 1) for every state \( q \in Q \) and every controllable event \( \sigma \in \Sigma_c \), the transition \( \delta(q, \sigma) \) is defined and; 2) every state \( q \) in \( G \) does not depend on any other state to be accessible from initial state \( q_0 \). Then, the number of all possible transitions \( n_t \) is given by

\[
 n_t = |Q| \times |\Sigma_c| \leq mn \tag{5.3}
\]

where \( |Q| = n \) is the number of states and \( |\Sigma_c| = m \) is the number of controllable events in \( G \). And the number of all possible supervisors is given by

\[
 n_c \leq \binom{n_t}{0} + \binom{n_t}{1} + \binom{n_t}{2} + \cdots + \binom{n_t}{n} = \sum_{i=0}^{n} \binom{n_t}{i} < \infty \tag{5.4}
\]

Lemma 5.1.1 shows that there are finitely many supervisors whose generating language is a subset of \( L(G) \) given the fact that the state space and event alphabet are both finite.

**Theorem 5.1.1.** (Existence) \[34\] Given a DES plant \( G = (Q, \Sigma, \delta, q_0, Q_m) \), \( \Pi \)-matrix, and \( \mathbf{X} \)-vector, there exist an optimal supervisor \( S^* \) such that \( \mu(L(S^*/G)) = \max_{i \in I_c} \mu(L(S_i/G)) \).

**Proof.** The proof can be found in [35]. \( \square \)

**Theorem 5.1.2.** Given a DES plant \( G = (Q, \Sigma, \delta, q_0, Q_m) \), \( \Pi \)-matrix, and \( \mathbf{X} \)-vector, Define \( \Sigma_c(G, q) \) and \( Q_c(G) \) as follows

\[
 \Sigma_c(G, q) = \{ \sigma \in \Sigma_c \mid \delta(q, \sigma) \text{ is defined in } G \} \tag{5.5}
\]

\[
 Q_c(G) = \{ q \in Q \mid \Sigma_c(G, q) \neq \emptyset \} \tag{5.6}
\]

For every state \( q \in Q_c(G) \), there is one and only one controllable event left enabled in the optimal supervisor \( S^* \), i.e.,

\[
 \forall q \in Q_c(S^*), \quad |\Sigma_c(S^*, q)| = 1 \tag{5.7}
\]
Proof. Assume there exists a supervisor $S^*$ such that $\mu(L(S^*/G)) > \mu(L(S^*/G))$ and $|\Sigma_c(S^*, q)| > 1$ for some $q \in Q_c(S^*)$.

$$\Delta \mu \triangleq \mu^* - \mu^* = \mu(L(S^*/G)) - \mu(L(S^*/G)) = \mu(L(S^*/G)) - \mu(L(S^*/G)) = \mu^* - \mu^* = \mu^* - \mu^* = \mu^* - \mu^* = \mu^* - \mu^*$$

1) $[I - \Pi(S^*)]^{-1} \geq 0$. By Taylor series expansion,

$$[I - \Pi(S^*)]^{-1} = \sum_{n=0}^{\infty} (\Pi(S^*))^n$$

(5.8)

Since each element of $\Pi(S^*)$ is non-negative, so is each element of $(\Pi(S^*))^n$. Therefore, $[I - \Pi(S^*)]^{-1} \geq 0$ elementwise.

2) Suppose $\Sigma_c(S^*, q_j) = \{\sigma_m, \sigma_n\}$ and $\Sigma_c(S^*, q_j) = \{\sigma_l\}$, for some $j \in \mathcal{I}$, then the $j$-th element of $(\Pi(S^*) - \Pi(S^*)) \mu^*$ is given by

$$\Delta_j = ([0 \cdots \bar{\pi}_{jm} \cdots \bar{\pi}_{jn} \cdots 0] - [0 \cdots \bar{\pi}_{jl} \cdots 0])$$

$$\begin{bmatrix}
\mu^*_m \\
\vdots \\
\mu^*_m \\
\vdots \\
\mu^*_l \\
\vdots \\
\mu^*_n \\
\vdots
\end{bmatrix}$$

$$= \bar{\pi}_{jm} \mu_m + \bar{\pi}_{jn} \mu_n - \bar{\pi}_{jl} \mu_l$$

$$\leq (\bar{\pi}_{jm} + \bar{\pi}_{jn}) \mu_k - \bar{\pi}_{jl} \mu_l$$

(\text{Let } \mu_k = \max\{\mu_m, \mu_n\})

$$< (\bar{\pi}_{jm} + \bar{\pi}_{jn} - \bar{\pi}_{jl}) \mu_l$$

(\text{\because } \mu_m < \mu_l)

$$< 0$$

(\text{\because } \bar{\pi}_{jm} + \bar{\pi}_{jn} = \bar{\pi}_{jl} \text{ by Equation 5.2})
Since $\Delta_j < 0$ for every $j \in I$, $\Delta \mu < 0$. This contradicts the assumption that $\mu(L(S/G)) > \mu(L(S^*/G))$. \qed

Theorem 5.1.1 states that out of all possible supervisors constructed from $G$, there exists a supervisor that maximizes the language measure $\mu$ with respect to $(\Pi, X)$. Theorem 5.1.2 describes a general transition structure of the optimal supervisor $S^*$. In particular, at every state $q \in Q_c(S^*)$ in $S^*$, there is one and only one controllable event enabled.

Intuitively, at a given state, a control synthesis algorithm should attempt to enable only the controllable event that leads to the next state with highest performance measure $\mu$, equivalently, disabling the rest of controllable events defined at that state, if any. A recursive synthesis algorithm is first presented below. It is then shown that $\mu$ is monotonically increasing elementwise on every iteration.

1. Initialization. $\Pi^0 = \Pi_p$, then $\mu^0 = (I - \Pi^0)^{-1}X$.

2. Recursion. $k$-th iteration, where $k \geq 1$.

1) For every $q \in Q_c(G)$, find out the event $\sigma^* \equiv \sigma^*(q) \in \Sigma_c(G,q)$ such that $q^* = \delta(q, \sigma^*)$ and

$$
\mu(L(S^k(\tilde{\Pi}^k), q^*)) = \max_{\sigma \in \Sigma_c(G,q)} \mu(L(S^k(\tilde{\Pi}^k), q'))
$$

where $S^k(\tilde{\Pi}^k)$ is the intermediate supervisor at $k$-th iteration whose transition is determined by $\tilde{\Pi}^k$. Let $\sigma^* = [\sigma^*_1 \sigma^*_2 \cdots \sigma^*_n]^T$, where the $i$-th element in $\sigma^*$ is the controllable event left-enabled at state $q_i$ of the plant $G$ according to Equation 5.9.

2) Disable the event set $\Sigma_c(G,q) - \{ \sigma^*(q) \}$ for every $q \in Q_c(G)$ and redistribute event cost according to Equation 5.1. This results in a new $\Pi^{k+1}$-matrix which consequently produces a new $\Pi^{k+1}$-matrix.

$$
\Pi^{k+1} = \Pi^k + \Delta^k
$$

where $\Delta^k$ records the difference between $\Pi^{k+1}$ and $\Pi^{k+1}$, consisting positive and negative event costs corresponding to those kept and disabled
controllable events, respectively. The resulting intermediate supervisor is \( S^{k+1}(\Pi^{k+1}) \).

3) Compute \( \mu^{k+1} = (I - \Pi^{k+1})^{-1}X \)

3. Termination. If \( \tilde{\Pi}^{k+1} = \tilde{\Pi}^k \), then stop.

It should be noted that at each iteration of the recursive algorithm, neither the number of states is increased, nor any additional transition is added in the resulting supervisor \( S^k(\Pi^k) \) with respect to the plant \( G \). Therefore, \( L(S^k(\Pi^k)) \subseteq L(G) \).

**Theorem 5.1.3.** (Monotonicity) The sequence \( \mu^k \), \( k = 1, 2, \ldots \), generated recursively by the above algorithm is monotonically increasing.

**Proof.** To prove \( \{ \mu^k \}, k = 1, 2, \ldots \) is a monotonic sequence, it is shown that \( \Delta \mu^k = \mu^{k+1} - \mu^k \geq 0 \).

\[
\Delta \mu^k = [I - \Pi^{k+1}]^{-1}X - [I - \Pi^k]^{-1}X \\
= [I - \Pi^{k+1}]^{-1}[[I - \Pi^k] - [I - \Pi^{k+1}]] \\
[I - \Pi^k]^{-1}X \\
= [I - \Pi^{k+1}]^{-1}(\Pi^{k+1} - \Pi^k)\mu^k \quad (\because [I - \Pi^k]^{-1}X = \mu^k) \\
= [I - \Pi^{k+1}]^{-1}\Delta^k \mu^k \quad \text{(by Equation 5.10)}
\]

1) \( [I - \Pi^{k+1}]^{-1} \geq 0 \). By Taylor series expansion,

\[
[I - \Pi^{k+1}]^{-1} = \sum_{n=0}^{\infty} (\Pi^{k+1})^n \quad (5.11)
\]

Since each element of \( \Pi^{k+1} \) is non-negative, so is each element of \( (\Pi^{k+1})^n \). Therefore, \( [I - \Pi^{k+1}]^{-1} \geq 0 \) elementwise.

2) \( \Delta^k \mu^{k+1} > 0 \). For \( k \geq 1 \), by the recursive algorithm, for any state \( q \in Q_c(G) \), there is one and only one left enabled in \( S^k(\Pi^k) \). Let \( \sigma_m \) and \( \sigma_n \) be the only controllable event left enabled on some state \( q \in Q_c(G) \) at the \( k \)-th and \( k + 1 \)-th iteration, respectively.
\[ \Delta^k \mu^{k+1} = \begin{bmatrix} 
\vdots & \vdots & \vdots & \cdots & \vdots \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
0 & \sum_j \tilde{\pi}_i^{k+1} & \cdots & 0 & \vdots \\
\vdots & \vdots & \vdots & \cdots & \vdots \\
\vdots & \vdots & \vdots & \cdots & \vdots 
\end{bmatrix} \begin{bmatrix} 
\mu_1^k \\
\vdots \\
\mu_n^k \\
\vdots \\
\mu_m^k 
\end{bmatrix} 
\]

\[ = \sum_j \tilde{\pi}_{ij} (\mu_n^k - \mu_m^k) \begin{bmatrix} 
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots 
\end{bmatrix} \begin{bmatrix} 
\mu_1^k \\
\vdots \\
\mu_n^k \\
\vdots \\
\mu_m^k 
\end{bmatrix} \]

The last inequality holds because \( \sigma_m \) is disabled and \( \sigma_n \) is enabled in the \((k+1)\)-th iteration only if \( \mu_n \) is greater than \( \mu_m \).

5.1.1 Reset

For the purpose of DES plant parameter identification or statistic study of the performance of DES controllers in the metric of language measure, a recovery mechanism is very useful for automatic continuation of experiments when a blocking state is inevitably reached under certain circumstances. For instance, a robot may lose a battle during its fight with an enemy; under certain aggressive control policy, the robot may run out of battery and die in the middle of a mission. A reset mechanism in the discrete event control system design is introduced. The idea of reset is widely used in engineering practice.

Let \( G = (Q, \Sigma, \delta, q_0, Q_m) \) be the original plant, a reset function \( f_r : Q_r \times \{r\} \rightarrow \{q_0\} \), where \( r \notin \Sigma \) is the reset event and \( Q_r \subseteq Q \) is the set of blocking states at which reset is possible, defines a set of new transitions that reset \( G \) at certain states to its initial state \( q_0 \). Now, the augmented plant \( G_r \) is defined as

\[ G_r = (Q, \Sigma \cup \{r\}, \delta, q_0, Q_m, Q_r) \]  

(5.12)
where $\delta_r : Q \times (\Sigma \cup \{r\}) \rightarrow Q$ is given by

$$
\delta_r(q, \sigma) = \begin{cases} 
\delta(q, \sigma), & \text{if } \sigma \in \Sigma \\
q_0, & \text{if } \sigma = r \text{ and } q \in Q_r
\end{cases}
$$

(5.13)

### 5.2 Experiment validation

The experiment validation of the proposed optimal supervisor synthesis approach in language measure $\mu$ is carried out in the simulator that is verified in Chapter 4. The experimental scenario consists of a single robot performing logistic supply and combat operation in a battle field, as shown in Figure 5.1. There are two friendly units (represented by red and green colored circles) and one enemy (represented by blue circle), which are at stationary locations in the field. The robot does not have prior knowledge of the environment. When a “start mission” signal is received, the robot randomly searches for a red or green unit. When finding a unit, the robot can either proceed to supply or ignore the unit and keep on searching. It may also encounter an enemy during the course of searching. The robot may decide either to avoid or to fight the enemy. In both cases, there are chances that the robot may fail to supply or lose the fight.

A gripper failure is modeled to signify the robot’s failure to complete the task of supplying units. However, fighting with enemy is still possible. In addition, during the mission, the battery consumption rate changes according to the robot’s current actions. For example, the robot’s power consumption is higher during the fight with the enemy than supplying the units. The robot needs to return to its base (represented by a large pink circle) to be recharged before the battery voltage drops below a certain level, or the robot is considered to be lost. After each successful return to the base, the robot is reset to normal conditions including full battery charge and normal gripper status. If the robot is incapacitate either due to battery over-drainage or damage in a battle, the “death toll” is increased by one. In both cases, the mission is automatically restarted with a new robot.

Due to the independence of the robot operation, battery consumption, and occurrence of gripper failures, the entire interaction between robot and environment is modeled by three independent submodels, as shown in Figure 5.2 and Figure 5.3. The blue-colored and red-colored states are good marked states in $Q^+_m$.
and bad marked states in $Q_m^-$, respectively. Then the models are composed by synchronous composition operator defined below to generate the integrated plant model $G = G_1 \parallel G_2 \parallel G_3$.

**Definition 5.2.1.** Given $G_1 = (Q_1, \Sigma_1, \delta_1, q_{0,1}, Q_{m,1})$, $G_2 = (Q_2, \Sigma_2, \delta_2, q_{0,2}, Q_{m,2})$, synchronous composition of $G_1$ and $G_2$, denoted $G_1 \parallel G_2 = (Q, \Sigma, \delta, q_0, Q_m)$, is defined as $Q = Q_1 \times Q_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $q_0 = (q_{0,1}, q_{0,2})$, $Q_m = Q_{m,1} \times Q_{m,2}$ and for every $q = (q_1, q_2) \in Q, \sigma \in \Sigma$

$$\delta(q, \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \sigma \in \Sigma_1 \cap \Sigma_2 \\ (\delta_1(q_1, \sigma), q_2) & \sigma \in \Sigma_1 - \Sigma_2 \\ (q_1, \delta_2(q_2, \sigma)) & \sigma \in \Sigma_2 - \Sigma_1 \\ \text{undefined} & \text{otherwise} \end{cases} \quad (5.14)$$

After eliminating the inaccessible states, the discrete-event model of the plant automaton $G$ consists of 139 states and 21 events, as listed in Table 5.1. The event cost matrix $\Pi$ is then identified by Monte Carlo simulation over 1200 missions according to the parameter identification procedure presented in Section 3 and convergence of selected non-zero elements in $\Pi$-matrix is demonstrated in Figure 5.4. For those states that have more than one controllable events defined, the probabilities of occurrence are assumed to be equally distributed. A vast majority of the plant states are unmarked; consequently, the corresponding elements, $\chi_i$, $i = 1, \ldots$, of the characteristic vector are zero. Negative characteristic values, $\chi_i$ are assigned to the bad marked states. For example, the states in which the robot is dead due to either losing the battle to the enemy or running out of battery is
assigned the most negative value of -1. Similarly, positive values are assigned to good states. For example, the state in which the robot wins a battle is assigned 0.5 and the state of successfully providing supplies is assigned 0.3. Using the recursive synthesis algorithm in Section 4, an optimal supervisor $S^*$ is then synthesized. The optimal DES control algorithm converges at the fourth iteration, as listed in Table 5.2. For the purpose of performance comparison, two supervisors under the following specifications are designed.

**Controller $S_1$ specifications**

1. Avoid enemy when the battery power is not below medium;
2. Abort all operation when the battery power is low;
3. If there is a gripper failure, do not supply a discovered unit or abort supply if the supply is on-going.

**Controller $S_2$ specifications**

1. Abort all operation if battery power is not below medium;
### Table 5.1. List of discrete events

<table>
<thead>
<tr>
<th>$\sigma$</th>
<th>Description</th>
<th>$\sigma$</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sigma_1$</td>
<td>start mission</td>
<td>$\sigma_{12}$</td>
<td>win the fight</td>
</tr>
<tr>
<td>$\sigma_2$</td>
<td>search</td>
<td>$\sigma_{13}$</td>
<td>loose the fight</td>
</tr>
<tr>
<td>$\sigma_3$</td>
<td>find blue unit</td>
<td>$\sigma_{14}$</td>
<td>battery power medium</td>
</tr>
<tr>
<td>$\sigma_4$</td>
<td>find pink unit</td>
<td>$\sigma_{15}$</td>
<td>battery power low</td>
</tr>
<tr>
<td>$\sigma_5$</td>
<td>find enemy</td>
<td>$\sigma_{16}$</td>
<td>battery power dead</td>
</tr>
<tr>
<td>$\sigma_6$</td>
<td>proceed to supply</td>
<td>$\sigma_{17}$</td>
<td>detected gripper fault</td>
</tr>
<tr>
<td>$\sigma_7$</td>
<td>ignore unit</td>
<td>$\sigma_{18}$</td>
<td>abort mission</td>
</tr>
<tr>
<td>$\sigma_8$</td>
<td>fight enemy</td>
<td>$\sigma_{19}$</td>
<td>return</td>
</tr>
<tr>
<td>$\sigma_9$</td>
<td>avoid enemy</td>
<td>$\sigma_{20}$</td>
<td>ignore anomaly</td>
</tr>
<tr>
<td>$\sigma_{10}$</td>
<td>finish supply</td>
<td>$\sigma_{21}$</td>
<td>return successfully</td>
</tr>
<tr>
<td>$\sigma_{11}$</td>
<td>fail supply</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Figure 5.4. Some non-zero elements of $\Pi$-matrix

![Figure 5.4. Some non-zero elements of $\Pi$-matrix](image)

### Table 5.2. Iteration of $\mu$ synthesis

<table>
<thead>
<tr>
<th>$k$-th Iteration</th>
<th>$\mu^k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-1.7834</td>
</tr>
<tr>
<td>1</td>
<td>2.8306</td>
</tr>
<tr>
<td>2</td>
<td>4.5655</td>
</tr>
<tr>
<td>3</td>
<td>4.5696</td>
</tr>
<tr>
<td>4</td>
<td>4.5696</td>
</tr>
</tbody>
</table>
Table 5.3. Simulation statistics of 400 missions

<table>
<thead>
<tr>
<th>Items</th>
<th>$G$</th>
<th>$S_1$</th>
<th>$S_2$</th>
<th>$S^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>proceed to supply</td>
<td>34.56%</td>
<td>32.84%</td>
<td>37.33%</td>
<td>86.69%</td>
</tr>
<tr>
<td># of units found</td>
<td>150</td>
<td>434</td>
<td>134</td>
<td>408</td>
</tr>
<tr>
<td>finish supply</td>
<td>112</td>
<td>88</td>
<td>97</td>
<td>362</td>
</tr>
<tr>
<td>win enemy</td>
<td>37</td>
<td>45</td>
<td>22</td>
<td>69</td>
</tr>
<tr>
<td>fight enemy</td>
<td>55</td>
<td>79</td>
<td>68</td>
<td>167</td>
</tr>
<tr>
<td>unit lost</td>
<td>26</td>
<td>19</td>
<td>14</td>
<td>48</td>
</tr>
<tr>
<td>$\mu$</td>
<td>-1.7834</td>
<td>-1.4428</td>
<td>-1.8365</td>
<td>4.5696</td>
</tr>
<tr>
<td>$\sum_{i=0}^{400} \mu_i$</td>
<td>-20.25</td>
<td>-19.45</td>
<td>-27.3</td>
<td>45.25</td>
</tr>
</tbody>
</table>

2. Abort all operation if a gripper failure is detected.

The (non-optimal) DES controllers, $S_1$ and $S_2$, have 109 and 54 states, respectively, and 400 missions were simulated for the open loop plant and each of the three DES controllers: optimal supervisor $S^*$, $S_1$ and $S_2$. The statistics of the simulation results are summarized in Table 5.3 that shows the plant yields higher performance under $S^*$ than under $S_1$ and $S_2$ or the null supervisor (i.e., the unsupervised plant). During the 400 missions, $S^*$ decides to proceed to supply for 482 times, three times more than $S_1$ or $S_2$. In addition, the probability of deciding to proceed to supply when the robot sees a unit is much higher than $S_1$ or $S_2$. However, the price of this decision is that the robot is likely to drain out the battery energy and therefore may risk being immobile in the middle of the mission. $S^*$ also decides to fight much more times (167) than any other supervisor since the reward to win a battle is large ($\chi = 0.5$). Certainly, the number of robots lost under $S^*$ is also higher (48) than that for other supervisors because $S^*$ has to expose the robot more often to the enemy attack to win the battles. On the average, $S^*$ outperforms $S_1$ and $S_2$ in measure $\mu = (I - \Pi)^{-1}X$ that are the theoretical values. This superior performance of $S^*$ is further justified by the experimental comparison of cumulative performance of the three supervisors and null supervisor over 400 missions, as shown in Figure 5.5. The large oscillations are due to the loss of the robot due to drained battery or loss of the battle.
5.2.1 Optimal control synthesis of scenario in Chapter 4

Chapter 4 presents a simple robot object collection scenario. It is interesting to find out its optimal supervisor in language measure $\mu$ and see if the performance of the supervisor that is manually designed based on experience and described in Chapter 4 can be further improved. Applying the same synthesis procedure, an optimal supervisor can be obtained after 2 iterations, as listed in Table 5.4. It is noted that $\mu$ increases elementwise at each iteration. In addition, both $\mu_3$ and $\mu_{13}$ do not change at each iteration. By observing the corresponding elements of $\hat{\Pi}$-matrix, it can be found that the probabilities of visiting these two states are zero, i.e., they were never visited, consequently, the 3-rd row and 13-th row of $(I - \Pi^k)^{-1}$ contain one and only one none zero element at the 3rd and 13th element, respectively. In other words, the discrete-event plant model could have been further refined by taking out state 3 and state 13 from the plant model $G$. Coincidently, the optimal supervisor is the same supervisor that is given in Chapter 4.
Table 5.4. Iteration of \( \mu \) synthesis with \( (\tilde{\Pi}, X) \) obtained in simulation

<table>
<thead>
<tr>
<th>( \mu ) Elements</th>
<th>( \mu^0 )</th>
<th>( \mu^1 )</th>
<th>( \mu^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_0 )</td>
<td>0.0799</td>
<td>0.1349</td>
<td>0.1349</td>
</tr>
<tr>
<td>( \mu_1 )</td>
<td>0.0840</td>
<td>0.1420</td>
<td>0.1420</td>
</tr>
<tr>
<td>( \mu_2 )</td>
<td>0.1578</td>
<td>0.2781</td>
<td>0.2781</td>
</tr>
<tr>
<td>( \mu_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mu_4 )</td>
<td>0.3667</td>
<td>0.4071</td>
<td>0.4071</td>
</tr>
<tr>
<td>( \mu_5 )</td>
<td>0.7218</td>
<td>0.7670</td>
<td>0.7670</td>
</tr>
<tr>
<td>( \mu_6 )</td>
<td>0.3937</td>
<td>0.4389</td>
<td>0.4389</td>
</tr>
<tr>
<td>( \mu_7 )</td>
<td>0.8798</td>
<td>0.9349</td>
<td>0.9349</td>
</tr>
<tr>
<td>( \mu_8 )</td>
<td>0.4798</td>
<td>0.5349</td>
<td>0.5349</td>
</tr>
<tr>
<td>( \mu_9 )</td>
<td>0.0694</td>
<td>0.1173</td>
<td>0.1173</td>
</tr>
<tr>
<td>( \mu_{10} )</td>
<td>-0.1202</td>
<td>-0.0651</td>
<td>-0.0651</td>
</tr>
<tr>
<td>( \mu_{11} )</td>
<td>0.7598</td>
<td>0.8074</td>
<td>0.8074</td>
</tr>
<tr>
<td>( \mu_{12} )</td>
<td>0.4144</td>
<td>0.4620</td>
<td>0.4620</td>
</tr>
<tr>
<td>( \mu_{13} )</td>
<td>-1.0000</td>
<td>-1.0000</td>
<td>-1.0000</td>
</tr>
<tr>
<td>( \mu_{14} )</td>
<td>0.3514</td>
<td>0.4735</td>
<td>0.4735</td>
</tr>
<tr>
<td>( \mu_{15} )</td>
<td>0.0664</td>
<td>0.1843</td>
<td>0.1843</td>
</tr>
<tr>
<td>( \mu_{16} )</td>
<td>-0.2002</td>
<td>-0.1501</td>
<td>-0.1501</td>
</tr>
</tbody>
</table>

Table 5.5. Iteration of \( \mu \) synthesis with \( (\tilde{\Pi}, X) \) obtained in experiment

<table>
<thead>
<tr>
<th>( \mu ) Elements</th>
<th>( \mu^0 )</th>
<th>( \mu^1 )</th>
<th>( \mu^2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mu_0 )</td>
<td>0.1640</td>
<td>0.1987</td>
<td>0.1987</td>
</tr>
<tr>
<td>( \mu_1 )</td>
<td>0.1726</td>
<td>0.2091</td>
<td>0.2091</td>
</tr>
<tr>
<td>( \mu_2 )</td>
<td>0.2737</td>
<td>0.3382</td>
<td>0.3382</td>
</tr>
<tr>
<td>( \mu_3 )</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( \mu_4 )</td>
<td>0.5114</td>
<td>0.5418</td>
<td>0.5418</td>
</tr>
<tr>
<td>( \mu_5 )</td>
<td>0.8700</td>
<td>0.9013</td>
<td>0.9013</td>
</tr>
<tr>
<td>( \mu_6 )</td>
<td>0.5090</td>
<td>0.5403</td>
<td>0.5403</td>
</tr>
<tr>
<td>( \mu_7 )</td>
<td>0.9640</td>
<td>0.9987</td>
<td>0.9987</td>
</tr>
<tr>
<td>( \mu_8 )</td>
<td>0.5640</td>
<td>0.5987</td>
<td>0.5987</td>
</tr>
<tr>
<td>( \mu_9 )</td>
<td>0.1562</td>
<td>0.1892</td>
<td>0.1892</td>
</tr>
<tr>
<td>( \mu_{10} )</td>
<td>-0.0360</td>
<td>-0.0013</td>
<td>-0.0013</td>
</tr>
<tr>
<td>( \mu_{11} )</td>
<td>0.9158</td>
<td>0.9487</td>
<td>0.9487</td>
</tr>
<tr>
<td>( \mu_{12} )</td>
<td>0.5358</td>
<td>0.5687</td>
<td>0.5687</td>
</tr>
<tr>
<td>( \mu_{13} )</td>
<td>-1.0000</td>
<td>-1.0000</td>
<td>-1.0000</td>
</tr>
<tr>
<td>( \mu_{14} )</td>
<td>0.4798</td>
<td>0.5436</td>
<td>0.5436</td>
</tr>
<tr>
<td>( \mu_{15} )</td>
<td>0.2229</td>
<td>0.2818</td>
<td>0.2818</td>
</tr>
<tr>
<td>( \mu_{16} )</td>
<td>-0.1000</td>
<td>-0.0681</td>
<td>-0.0681</td>
</tr>
</tbody>
</table>
5.2.2 DES $\mu$ synthesis procedure

The procedure for the synthesis of high performance DES supervisor in terms of language measure $\mu$ can be summarized as follows.

1. construct the DES plant model of the real system;
2. build a high fidelity simulator of the real system;
3. perform extensive simulation study to obtain the $\tilde{\Pi}$-matrix;
4. assign the $X$-vector according to the real system;
5. design a supervisor that has high $\mu$ value;
6. deploy the supervisor into real system and update the $\tilde{\Pi}$-matrix through the real experiments;
7. redesign the supervisor according to the optimal $\mu$ synthesis procedure if $\tilde{\Pi}$-matrix changes above a threshold

5.3 Related work

$Q$-learning [86] is widely used for reinforcement learning in behavior-based robotics [5] for its algorithmic simplicity and ease of transformation from a state function to an optimal control policy. In addition, same as the approach presented in this chapter, it does not require a world model. It is best justified for use in stationary, single-agent, fully observable environments modeled by Markov decision processes or (MDPs). However, it often performs well in environments that violate these assumptions. Mahadevan and Connell [52] have used $Q$-learning to teach a behavior-based robot how to push boxes around a room without getting stuck. The mission is heuristically divided into three behaviors: finding a box, pushing a box, and recovering from stalled situations. The results show that the mission decomposition method is capable of learning faster than a method that treats the mission as a single behavior. Martinson et al [54] demonstrated by both simulation and robotic experiments that $Q$-learning can be used for robot behavioral selection
Table 5.6. Comparison of Q-learning and \(\mu\)-selection

<table>
<thead>
<tr>
<th>Items</th>
<th>Q-learning</th>
<th>(\mu)-selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling</td>
<td>(M = (S, A, T, R))</td>
<td>(G = (Q, \Sigma, s_0, Q_m))</td>
</tr>
<tr>
<td>Control policy</td>
<td>MDP</td>
<td>DES</td>
</tr>
<tr>
<td>Objective</td>
<td>(\arg\max_a Q^*(s, a))</td>
<td>(\max_{s\in L(G)} \mu(s))</td>
</tr>
<tr>
<td>Transition probability</td>
<td>(T(s, a, s'))</td>
<td>(\tilde{\pi}(q, \sigma))</td>
</tr>
<tr>
<td>Reward</td>
<td>on transition</td>
<td>on state</td>
</tr>
<tr>
<td>Discount factor (\gamma)</td>
<td>yes(ad hoc)</td>
<td>no</td>
</tr>
<tr>
<td>Learning rate (\alpha)</td>
<td>yes(ad hoc)</td>
<td>no</td>
</tr>
<tr>
<td>Online adaptation to dynamic environment</td>
<td>recursive</td>
<td>((\epsilon, \delta))-threshold, redesign</td>
</tr>
<tr>
<td>Model complexity</td>
<td>exponential in (S) and (A)</td>
<td>polynomial in (Q)</td>
</tr>
</tbody>
</table>

To optimize the overall mission in terms of \(Q\)-value. The basic principles of Q-learning are briefly described below.

A Markov decision process (MDP) is a tuple \(M = (S, A, T, R)\), where \(S\) is the set of states; \(A\) is the set of actions; \(T : S \times A \times S \to [0, 1]\) the transition probability function; and \(R : S \times A \to \mathbb{R}\) is an expected reward function. The objective of MDPs is to find a control policy \(\pi : S \to A\) that maximizes the future reward with a discount factor \(\gamma\), which controls how much effect future rewards have on the optimal decisions with small values of \(\gamma\) emphasizing near-term gain and larger values giving significant weight to later rewards. One of the important assumptions for MDPs is that there exists an optimal stationary and deterministic policy [7]. The optimal value of a state \(s \in S\), denoted as \(V^*(s)\), is the expected discounted (i.e., weighted) infinite sum of rewards if it starts in that state and executes the optimal policy. If \(\pi\) denotes a complete decision policy, for every state \(s \in S\), it is written

\[
V^*(s) = \max_{\pi} E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) = \max_a (R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s'))
\]
where $R(s, a)$ is the expected instantaneous reward by action $a$ at state $s$, $T(s, a, s')$ is the probability of making a transition from state $s$ to state $s'$ by action $a$. To find the optimal control policy, $Q$-learning \cite{86} \cite{87} is a well-known technique. Let $Q^*(s, a)$ be the expected discounted reinforcement of taking action $a$ at state $s$, then $V^*(s) = \max_a Q^*(s, a)$. Therefore

$$Q^*(s, a) = R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') \max_{a'} (s', a')) \tag{5.15}$$

At each state, the best policy is to take the action with the largest $Q$-value, i.e., $\pi^* = \arg \max_a Q^*(s, a)$. The on-line $Q$-learning update rule is

$$Q(s, a) = Q(s, a) + \alpha (r + \gamma \max_{a'} Q(s', a') - Q(s, a)) \tag{5.16}$$

where $\alpha$ is the learning rate and the discount factor $\gamma$ takes the value $0 \leq \gamma < 1$. In \cite{86} \cite{87}, it has been shown that the $Q$ values will converge with probability 1 to $Q^*$ if each action is executed in each state an infinite number of times on an infinite run and $\alpha$ is decayed appropriately.

### 5.4 Summary and Conclusions

This chapter presents optimal supervisory control of robot behavior based on the discrete-event language measure $\mu$ \cite{81} \cite{71}. A recursive supervisor synthesis algorithm is provided as an extension of the earlier work of Fu et al. \cite{34}. A realistic robot simulation scenario has been developed for experimental validation of this new concept of discrete event supervisory (DES) control. The results of simulation experiments validate that a DES controller, designed by the recursive synthesis algorithm, indeed maximizes the $\mu$-measure and has the best performance with respect to two other supervisors, designed in conventional way, and the null supervisor (i.e., the unsupervised plant).

The chapter compares the proposed language measure ($\mu$)-based approach of DES control with the $Q$-learning method, where the reward function $R(s, a)$ is defined for every transition at each state of the Markov decision process based on purely human perception, similar to the characteristic value the $\chi$ value defined.
for every state of the DES $G$ in $\mu_i$ in the formulation of $\mu$. Conceptually, a weight is assigned to each transition in $Q$-learning whereas a weight is assigned to each state in $\mu$-selection. However, while complexity of the $\mu$-based approach of DES control is polynomial in number of plant automaton states, the complexity of the $Q$-learning method increases exponentially with the number of states $n$ and the number of actions(events) $m$. A summary of comparison between $Q$-learning and $\mu$-selection is given in table 5.6.
Chapter 6

Performance of DES Coordinated Multi-robot Systems

Cooperation generally results in higher efficiency. A key feature of multirobot systems is the potential to cooperate: several robots can coordinate to accomplish a task faster or better, and they can even compensate for each other’s weaknesses. Chapter 4 has demonstrated that action-level individual robotic systems perform missions based on reactive behaviors. This chapter will show how task-level systems perform missions at a higher level by decomposing them into subtasks shared among robots. Hierarchical control is an attempt to handle complex problems by decomposing them into smaller subproblems and reassembling their solutions in a hierarchical structure. It is shown in this chapter how performance evaluation in language measure $\mu$ can also be applied to hierarchical DES. In the design of a hierarchical DES control based multi-robot system, the theoretical framework formulated in [92] [89] [90] is adopted. An example of a two-robot scenario under a hierarchical DES controller is presented. By simulation results, it is concluded that language measure $\mu$ can be effectively used as one of the design criteria for high level planning of multi-agent systems.
6.1 Theory of hierarchical DES control

6.1.1 Introduction

Within the DES framework [66], Zhong [92] introduced the concept of hierarchical consistency to ensure that the task of the high-level controller can be realized through the implementation of low-level ones, where the high level system is an abstract model of the low level real systems. Wong [89] developed the concept of observer and showed that the observer property ensures that the low-level implementation of a nonblocking high-level supervisor is also nonblocking in a two-level hierarchy. Caines [15] proposed the dynamic consistency concept for state aggregation of finite machines. The effect of model uncertainty and partial observation of low level systems on the hierarchical consistency were addressed in [59] [38]. Wong [90] developed an effective computation algorithm for the design of an observer in the case where a given causal reporter map can be represented by a finite Mealy automaton.

![Figure 6.1. DES hierarchical control structure](image)

6.1.2 Hierarchical supervisory control of DES

Consider the two-level hierarchical structure [92] shown in Figure 6.1, where $G_{lo}$, the low level system is the actual plant to be controlled by $C_{lo}$, the low level DES supervisor, and $G_{hi}$, the high level system, is an abstract model controlled by $C_{hi}$, the high level DES supervisor. In this hierarchical structure, the control command of $C_{hi}$ is achieved by $C_{lo}$ through $Com_{hilo}$ (command channel) since $G_{hi}$ is an abstract model derived from $G_{lo}$ via $Inf_{lohi}$ (information channel). In other words, the high-level commands sent by $C_{hi}$ are translated by $C_{lo}$ into corresponding low
control sequences for plant $G_{lo}$ to execute. Clearly, the result of hierarchical control which depends on $G_{hi}$ can not be better than that of conventional nonhierarchical control, which depends on $G_{lo}$. To exercise hierarchical control, the deterministic finite automaton $G ≡ G_{lo}$ is extended to a deterministic trim Mealy machine, as defined below.

$$G = (Q, \Sigma, T \cup \{\tau_0\}, \delta, \psi, q_0, Q_m)$$

where $Q, \Sigma, \delta, q_0, Q_m$ take the usual meanings, $T$ is the high level coordinator’s event alphabet, without loss of generality, $\Sigma \cap T = \emptyset$, $\tau_0$ is the “silent output symbol” such that $\tau_0 \notin \Sigma \cup T$. The mapping function $\psi : Q \times \Sigma \to T \cup \{\tau_0\}$ summarizes behaviors in the low-level process and reports relevant information to the high level system through the information channel $inf_{lohi}$. $\psi$ can be extended to $L(G) \to L(G_{hi}) \subseteq T^*$. Formally,

**Definition 6.1.1.** Let $pr(L)$ be the generating language of the deterministic finite state automaton $G = (Q, \Sigma, \delta, q_0, Q_m)$. A function $\psi : pr(L) \to T^*$ is a causal reporter map [92] if it satisfies the following properties:

$$\psi(\epsilon) = \epsilon$$

$$\psi(s\sigma) = \begin{cases} \psi(s), & \text{if } \psi(\delta(q_0, s), \sigma) = \tau_0 \\ \psi(s)\tau, & \text{if } \psi(\delta(q_0, s), \sigma) = \tau \in T \end{cases}$$  

where $s \in pr(L)$, $\sigma \in \Sigma$, and $\tau_0 \notin \Sigma \cup T$ is the “silent output symbol”.

**Proposition 6.1.1.** [64] The causal reporter map $\psi$ is regular.

Therefore, $\psi(L(G))$ can be realized by a finite state automaton, i.e., $G_{hi}$.

**Lemma 6.1.1.** [88] Let $\psi$ be a causal reporter map from $pr(L)$ to $T^*$ and $H \subseteq pr(L)$. Then $\psi$ is an $H$-observer if and only if $(\forall s \in pr(L))(\forall \tau \in T^*)(\psi(s)\tau \in \psi(H)) \Rightarrow (\exists t \in \Sigma^*)st \in H \text{ and } \psi(st) = \psi(s)\tau$.

Thus, if $\psi$ is an $H$-observer, by examining the behavior of the high-level system, if one observes that $\psi(s)$ can be extended to a string $\psi(s)\tau \in \psi(H)$, then there exists a low level implementable controllable event sequence for any of those states, whose $\psi$-image is $\psi(s)$, such that its behavior can be extended to a string $st$ in $H$.
and \( \psi(st) = \psi(s)\tau \). If \( H = pr(L) \), then \( \psi \) is an observer. An effective computation algorithm to obtain \( \psi \) is given in [90].

The design of high level supervisor is the same as its low level counterpart, except the controllability is with respect to high level plant model \( G_{hi} \). Let \( L_{lo} = L(G_{lo}) \) and \( L_{hi} = L(G_{hi}) \). The high level supervisor \( C_{hi} \) is defined by a map \( \gamma_{hi} : L_{hi} \times T \rightarrow \{0, 1\} \) as follows.

\[
\forall w \in L_{hi}, \forall \tau \in T, \quad \gamma_{hi}(w, \tau) = \begin{cases} 
0, & \text{disabled} \\
1, & \text{enabled}
\end{cases}
\]

Let \( T = T_c \cup T_u \), where \( T_c \) and \( T_u \) are the high level controllable event set and high level uncontrollable event set, respectively. As usual, only \( T_c \) can be disabled and \( \gamma_{hi}(w, \tau) = 1 \) for all \( w \in L_{hi} \) and \( \tau \in T_u \) such that \( w\tau \in L_{hi} \). The high level disabled-event map is \( \Delta_{hi} : L_{hi} \rightarrow 2^{T_c} \) such that \( \Delta_{hi}(w) = \{ \tau \in T_c \mid \gamma_{hi}(w, \tau) = 0 \} \).

The corresponding low-level disabled-event map is \( \Delta_{lo} : L_{lo} \times L_{hi} \rightarrow 2^{\Sigma_c} \) where

\[
\Delta_{lo}(s, w) = \{ \sigma \in \Sigma_c \mid \exists s' \in \Sigma_{u}, s\sigma s' \in L_{lo}, \ and \ \psi(s\sigma s') \in \Delta_{hi}(w), \\
\quad \quad \quad \quad \\\ \text{and} \ \forall s'', s'' < s' \Rightarrow \psi(s\sigma s'') = \tau_0 \}
\]

The definition says that \( \Delta_{lo}(s, w) \) is the set of low-level controllable events that must be disabled immediately following the generation of \( s \in L_{lo} \) and \( w = \psi(s) \in L_{hi} \) in order to guarantee that no event in \( \Delta_{hi}(w) \) can be occurred as the next event in \( G_{hi} \). Therefore, the control implemented by \( C_{lo} \) is given by

\[
\gamma_{lo}(s, \sigma) = \begin{cases} 
0, & \text{if } \sigma \in \Delta_{lo}(s, \psi(s)) \\
1, & \text{otherwise}
\end{cases}
\]

### 6.2 Multi-robot coordinated experiment

As described in Chapter 4, each individual robot, under its own DES supervisor, explores an unknown environment by random search; upon detecting an object by its camera, the robot approaches it and grabs the object; then after finding the destination, the robot approaches the destination and releases the object at the destination; the robot will keep repeating this procedures unless asked to stop. If multiple identical robots are exploring the same environment, there is an obvious
Figure 6.2. Hierarchical structure of multi robot system

Figure 6.3. The Mealy machine of DES plant model $G$

conflict without coordination: if two robots see the same object, under its own DES supervisor, both of them will approach the object. There could be two possibilities in the multi-robot system under this circumstance.

1. One of them gets the object. When both robots see the object, one may be far away from the object. By the time it gets closer to the object, the
other robot has already grabbed the object. Due to the close distance to the other robot, it will detect an obstacle around and avoidance behavior is then triggered by the supervisor. Figure 6.5 shows an example of the simulated two robots under this circumstance.

2. Neither of the two robots will be able to grab the object. For the same reason, once they get too close to each other on the way to the object, they both see the other robot as an obstacle and their supervisor will command the robot avoid the obstacle.

However, if some level of coordination exists, one of the two robots will be able to take actions at the earlier stage under the same situation. For instance, one will approach and the other will ignore the current object and search for others. Figure 6.2 shows the hierarchical structure of the multi-robot system. A coordinator is running on either a remote site or on one of the robots (and thus acting as a team leader). Under normal operating condition, the coordinator’s responsibility is to resolve conflict when two robots find and intend to approach the same object. In addition, the coordinator gives command to start or stop robot(s) upon human operator’s instruction. The messages between the robots and the coordinator are discrete events with possible continuous and/or discrete parameters. In addition to normal operating mode, an individual robot can also operate at two additional modes. The faulty mode is that upon an actuator/sensor failure, a robot will be switched to an appropriate controller which mitigates the corresponding failure; the rescuing mode is that it will go search and replace a faulty robot upon the coordinator’s request.
In the design of a DES hierarchical control system for two robots, compared to the experiment scenario in Chapter 4, there are two object 1 and two object 2 in the experiment field. All four objects are random located and will be randomly-relocated individually every time after dropped at the destination. A Mealy machine for a single robot is constructed and shown in Figure 6.3. The high level coordinator’s event alphabet $T$ is thus given by $T = \bigcup_{i=1}^{2} T_i$ where $T_i$ is the $i$-th robot’s high level events, as listed in Table 6.2.

$$T_i = \{A_i, B_i, D_i, G_i, L_i, M_i, N_i, R_i, S_i, T_i, U_i, x_i, y_i\}, \quad i = 1, 2 \quad (6.3)$$

A high level plant model $G_{hi}$ thus constructed contains 169 states. A high level supervisor $S_{hi}$ is then designed according to the following specification “if both robots see the same object, the one robot who reports to the coordinator first will approach the object and the other will ignore the object and keep searching for objects”. The behaviors of robots are compared between real experiments and

Figure 6.5. Two simulated robots without coordination
simulation. Figure 6.6 shows a typical experiment result and Figure 6.7 gives a
simulation result.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Controllable</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>ignored the object</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>repeat the mission</td>
<td>√</td>
</tr>
<tr>
<td>D</td>
<td>drop an object at the destination successfully</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>grab an object successfully</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>lost the target</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>continue current objective</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>approach the object</td>
<td>√</td>
</tr>
<tr>
<td>R</td>
<td>ignore the object</td>
<td>√</td>
</tr>
<tr>
<td>S</td>
<td>start the mission</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>found destination</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>failed current mission</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>found object 1</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>found object 2</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1. The discrete event set Σ for Pioneer 2 AT robot

6.3 Performance evaluation

Simulation results are used in evaluation of performance of open-loop and closed-
loop systems. The same parameter identification procedure used in Chapter 4 is
applied to identify the $\tilde{\Pi}$-matrix of the high plant model $G_{hi}$. Some of the non-zero
elements in $\tilde{\Pi}$-matrix are shown in Figure 6.8. $X$-vector is determined accordingly
to the objective of this scenario as listed in Appendix. By Equation 2.29, the final
performance comparison between the open loop and closed loop system is given by

$$
\mu(L(G_{hi})) = 0.0370, \quad \mu(L(S_{hi}/G_{hi})) = 0.0995
$$

(6.4)

This result is consistent with the simulated cumulative performance as shown in
Figure 6.9. Figure 6.10 shows the simulation result of performance comparison
of individual robots. Because of the second robot’s participation in object collection,
the environment has changed and consequently, the resulting $\tilde{\Pi}$-matrix for
individual robot has changed. As expected, given the same amount of time, the
number of successful missions for robot 1 alone is higher than that of the same
robot 1 with the participation of robot 2. This is because robot 1 may be forced by
the coordinator to ignore an object since robot 2 may discover the object earlier
than robot 1. This result is also consistent with the cumulative performance in
Figure 6.10.

Figure 6.6. Real robot behaviors in coordination

6.4 Related work

In a single robot system, the discounted reward-based $Q$-learning algorithm has
demonstrated the effectiveness of dealing with dynamic environments and changing
task conditions. However, it does not achieve cooperation when applied to cooper-
ative multi-robot systems. The main reason is that the values of rewards fade over
time, causing all robots to prefer actions that have immediate rewards [72]. There
has been some recent work on multi-agent learning in stochastic games [10] [49].
Stochastic games are a natural extension of Markov decision processes to multiple
agents. A stochastic game is a tuple $(n, S, A_1, \ldots, A_n, T, R_1, \ldots, R_n)$, where $n$ is the number
of agents, $S$ is a set of states, $A_i$ is the set of actions available to agent $i$ and
$A$ is the cartesian product action space $A_1 \times \cdots \times A_n$, $T$ is a transition function
$S \times A \times S \to [0, 1]$, and $R_i$ is a reward function for the $i$th agent $S \times A \to \mathbb{R}$. 
This appears very similar to the MDP framework except there are multiple agents selecting actions and the next state and rewards depend on the joint action of the agents. Another important difference is that each agent has its own separate reward function. The goal for each agent is to select actions in order to maximize its discounted future rewards with discount factor $\gamma$.

Littman [49] was the first to examine stochastic games as a framework for multi-agent reinforcement learning. He extended the traditional $Q$-learning algorithm to zero-sum stochastic games. The Minimax-$Q$ algorithm is shown below.

1. Initialize $Q(s \in S, a \in A)$ arbitrarily, e.g., 1, and set $\alpha$ to be the learning rate.

2. Repeat.

   (a) Given the current state $s$, find the equilibrium $\sigma$ of the matrix game $[Q(s, a)_{a \in A}]$. 

---

**Figure 6.7.** Simulated robot behaviors in coordination
(b) Select action $a_i$ according to the distribution $\sigma_i$.

(c) Observing joint-action $a$, reward $r$, and next state $s'$

$$Q(s, a) \leftarrow (1 - \alpha)Q(s, a) + \alpha (r + \gamma V(s')),$$

where

$$V(s) = \text{Value}_i ([Q(s, a)_{a \in A}])$$

The notation of a $Q$ function is extended to maintain the value of joint actions, and the backup operation computes the value of states differently, by replacing the max operator with the Value$_i$ operator, which computes the expected reward for player $i$ if all the players played the unique Nash equilibrium. A stochastic policy $\pi : S \rightarrow PD(A)$, where $PD(A)$ represents the set of discrete probability distributions over the set $S$, is a function that maps states to mixed strategies.
Figure 6.9. Cumulative performance comparison of high level open and closed loop systems

6.5 Summary

This chapter presents the design of a hierarchical DES control system for multiple robots. The simulation of a coordinated robotic scenario is validated through the real experiments. The high level performance of this multi-robot system is evaluated in simulation. The language measure $\mu$ is shown to be an effective way of evaluating the performance of high-level DES controller and thus can be used for the design of high performance DES control systems.
Figure 6.10. Cumulative performance comparison of individual robot with and without coordination
Chapter 7

Conclusions and Recommendations for Future Research

In this final chapter, we first summarize the main contributions of this thesis and then we outline some directions for future research.

7.1 Conclusions of the Dissertation

This dissertation focused on the concept, theoretical development, and application of a signed real measure of regular languages. It has also demonstrated the efficacy of the language measure in evaluation and design of discrete event supervisors through real-world applications. The main contributions of the research reported in the dissertation are:

1. Development of a quantitative language measure theory of regular languages. Unlike the conventional qualitative control synthesis in terms of maximal permissiveness, it is now possible to compare quantitative performance of different supervisory controllers with respect to a given unsupervised plant model. In this way, optimal/robust control can be quantitatively synthesized in the discrete-event setting. Another issue that has not been addressed in conventional DES control synthesis is the importance of either reaching cer-
tain states infinitely often or avoiding some other states as much as possible. In the language measure, this issue is addressed by characterizing states of a plant into different categories (good marked states, bad marked states, and non-marked states) and assigning them different numerical values (positive, negative, and zero). Therefore, a good controller keeps as many as possible the event strings leading to good marked states while deleting as many as possible the strings leading to bad marked states.

2. A parameter identification procedure for the computation of language measure. By this procedure, elements of the $\tilde{\Pi}$-matrix have well-defined physical meanings, i.e., transition probabilities.

3. Design and construction of a behavior-based distributed multi-robot system. The robotic system is supported by a simulator, including the system architecture, individual behavior design (e.g., visual servoing, obstacle avoidance), event generation and detection through continuous-time control, signal processing, and communication, and supervisory control in the discrete event setting.

4. Development of an optimal DES controller synthesis algorithm using the language measure. In many engineering applications, it is expected that a supervisor enables one and only one controllable event at any state where a decision is required. However, the conventional optimal control synthesis procedure in terms of maximal permissiveness does not prevent multiple controllable events enabled in such a state of a supervisor because there is no preference by the supervisor on which a controllable event will give better performance under this qualitative measure. Consequently, it requires the plant to have additional intelligence to decide which controllable event should be executed. In implementation, this complicates the system software structure because additional decision-making blocks have to be embedded into the plant. The optimal control in terms of the quantitative language measure solves this decision ambiguity by enabling a specific controllable event whenever a decision is required. The computational complexity of the language measure based optimal control is polynomial in number of states in the discrete event plant.
model whereas the complexity of $Q$-learning approach solving a Markov Decision Process (MDP) is exponential in the number of states and actions. Through robotic experiments and simulations, it is also demonstrated that the robot supervised by the controller based on our optimal robot behavior selection (called $\mu$-selection) can indeed effectively fulfill its mission and agree well with other heuristic measure. Therefore, $\mu$-selection method can be a complementary approach to the popular $Q$-learning approach widely used in robotics.

5. Validation of the proposed quantitative language measure through real robot experiments and simulations. Robot experiment scenarios have been constructed to validate the quantitative language measure for performance evaluation, comparison, and optimal control synthesis. A more complex scenario is then used to demonstrate how the optimal control synthesis algorithm can automate the controller design that may not be achievable through the conventional approach. The dissertation shows an example of using the language measure to evaluate the design of a higher level supervisor in a hierarchical DES control structure for a coordinated multi-robot experiment.

Part of this thesis work has been published or submitted for publication in [81] [83] [80] [84] [82] [85] [79] [77] [18] [78]. Chapter 2, Chapter 4, and Chapter 5 are also chapters of a forthcoming book [61] to be published by Springer-Verlag.

7.2 Recommendations for Future Research

It is well known that the number of states in a discrete-event system exponentially increases with the number of interacting subsystems. In the single robot system, the robot behavioral selection algorithm has only a few user-selectable parameters for optimization, namely, the elements of $X$-vector, whereas algorithms such as $Q$-learning require the transition costs assigned manually which is itself non-trivial. However, when the state space of the system becomes larger, the difficulty of assigning $\chi$ values to individual states increases and thus exploration of state space reduction techniques is recommended as a topic of future research. In particular, emphasis should be laid on how individual state’s physical meaning in the original
discrete-event system is translated or propagated to some state in the new discrete-event system is critical since the language measure requires the $X$-vector, which is determined based on the physical meaning of the individual states.

As stated in earlier chapters, the applicability of language measure is based on a two-time-scale assumption: a fast time scale and a slow time scale. For a short period of time, elements of $\tilde{\Pi}$-matrix are assumed to be constants. However, over a prolonged period, the $\tilde{\Pi}$-matrix crosses a given threshold, then the DES supervisor may need to be updated. On the other hand, elements in $\tilde{\Pi}$-matrix cannot be updated to capture the effects of abrupt changes. How to adapt the system to abrupt changes is an important area of future research.

It would be challenging to extend the concept of (regular) language measure for languages higher up in the Chomsky Hierarchy [53] such as context free and context sensitive languages. This extension would lead to controller synthesis when the plant dynamics is modeled by non-regular languages such as the Petri-Net.

While this dissertation has used language measures for evaluation of multi-robot systems, no optimal supervisor synthesis procedure is proposed. A vector formed optimal supervisor synthesis approach in language measure might be possible in this case. The idea is that joint actions of robots are treated as single events at the high level, and thus the problem of the optimal high level DES supervisor synthesis can be reduced to the problem of robot behavioral selection for a single robot system.

Most, if not all, of engineering systems are subject to component failures. A classical way of failure adaptation is to reconfigure the control system so that the system can still perform certain level of tasks while the performance is reduced. Online failure adaptation in the proposed quantitative DES control synthesis has not been studied. The difficulty in this problem is how to capture (possibly abrupt) component failure(s) in the parameter identification procedure and subsequently have the corresponding element(s) in $\Pi$-matrix changed accordingly in a timely fashion. Research in this area would also be very useful.

In the robotic experiments, random search has been used for the robot to explore its unknown environment. The robot may lose a detected object along its way to the object due to variation of light condition. Since there was no world map, it would then have to inefficiently continue its random search. To improve the
robot’s performance, mapping and localization techniques can be applied. There are many successful techniques in the mobile robotics literature, e.g., simultaneous localization and mapping (SLAM). There are scan matching based SLAM [73], extended Kalman filter (EKF) based SLAM [23], and particle filtering based SLAM (also known as Fast SLAM) [76] [55].
Bibliography


Vita
Xi Wang

Xi Wang received the B.Sc. and M.Sc. degrees in power Systems Engineering from Xi’an Jiaotong University, Xi’an, China, and Automotive Engineering from Tsinghua University, Beijing, China, in 1993 and 1998, respectively. He received the M.Sc. degree in Electrical Engineering and the Ph.D. degree in Mechanical Engineering from The Pennsylvania State University, University Park, in 2002 and 2003, respectively.

He has been a summer intern at Embedded System and Sensor Group, Xerox Wilson Center for Research and Technology, Webster, NY, in 2002. He is the assistant director of the Networked Robotic System Laboratory in Mechanical Engineering at The Pennsylvania State University, University Park. He is currently leading and coordinating more than 15 researchers of mechanical engineers, electrical engineers, and computer scientists from The Pennsylvania State University, Carnegie Mellon University, Duke University, and Louisiana Tech University on a MURI project sponsored by US Army Research Office.

His research interests include mechatronics, robotics and intelligent machines, computer vision, discrete-event and hybrid systems, design and manufacturing, and industrial automation.