Robot Behavioral Selection Using Discrete Event Language Measure

Xi Wang xxw117@psu.edu
Jinbo Fu jbf@psu.edu
Peter Lee cfl106@psu.edu
Asok Ray axr2@psu.edu
Department of Mechanical Engineering
The Pennsylvania State University
University Park, PA 16802

Keywords: *Discrete-Event Supervisory Control; Formal Languages; Performance Measure; Behavioral Robotics; Learning*

***Abstract*— This paper proposes a robot behavioral μ-selection method that maximizes a quantitative measure of languages in the discrete-event setting. This approach complements Q-learning (also called reinforcement learning) that has been widely used in behavioral robotics to learn primitive behaviors. While μ-selection assigns positive and negative weights to the marked states of a deterministic finite-state automaton (DFSA) model of robot operations, Q-learning assigns reward/penalty on each transition. While the complexity of Q-learning increases exponentially in the number of states and actions, complexity of μ-selection is polynomial in the number of DFSA states. The paper also presents results of simulation experiments for a robotic scenario to demonstrate efficacy of the μ-selection method.**

I. Introduction

Q-learning [7] is widely used for reinforcement learning in behavior-based robotics [1] for algorithmic simplicity and ease of transformation from a state function to an optimal control policy. Mahadevan and Connell [3] 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.

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 γ. One of the important assumptions for MDPs is that there exists an optimal stationary and deterministic policy. 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 π denotes a complete decision policy, for every state $s \in S$, it is written

$$V^*(s) = \max_a (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') V^*(s')) \qquad (1)$$

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 [7] 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

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') \max_{a'} *(s', a')) \quad (2)$$

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

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s,a)) \quad (3)$$

where α is the learning rate and the discount factor γ takes the value $0 \leq \gamma < 1$. In [7], 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 α is decayed appropriately.

In this paper, we propose a discrete event supervisory control approach for robot behavior selection in terms of language measure μ [5]. In the discrete-event setting, behavior of physical plants is often modelled as regular languages that can be realized by deterministic finite state automata (DFSA) [4]. The (regular) sublanguage of a controlled plant could be different under different supervisors that are constrained to satisfy different specifications [4]. Such a partially ordered set of sublanguages requires a quantitative measure for total ordering of their respective performance. The language measure [5] serves as a common quantitative tool to compare the performance of different supervisors and is assigned an event cost $\tilde{\mathbf{\Pi}}$-matrix and a state characteristic $\mathbf{X}$-vector. Event costs (i.e., elements of the $\tilde{\mathbf{\Pi}}$-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 $\mathbf{X}$-vector is chosen based on the designer's perception of the individual state's impact on the system performance.

The paper is organized in five sections including the present section. The language measure is briefly reviewed in Section 2. Section 3 presents the language measure parameter identification. Section 4 proposes a discrete event supervisory (DES) control synthesis strategy. Section 5 validates the synthesis algorithm through simulation on a mobile robotic system. The paper is concluded in Section 6 with a brief discussion on comparison of Q-learning method and μ-selection method.

II. Review of the Language Measure Concept

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 $\mathcal{I} \equiv \{1, \ldots, n\}$ the index of Q; Σ the (finite) alphabet of events; Σ^* is the set of all finite-length strings of events including the empty string ϵ; $\delta : Q \times \Sigma \to Q$ is a (possibly partial) function of state transitions and $\delta^* : Q \times \Sigma^* \to Q$ is an extension of δ; the state q_i is the initial state; and Q_m is the set of marked states, $\emptyset \subset Q_m \subseteq Q$.

Definition 2.1: The language $L(G_i)$ generated by a DFSA G initialized at the state $q_i \in Q$ is defined as:

$$L(G_i) = \{s \in \Sigma^* \mid \delta^*(q_i, s) \in Q\} \tag{4}$$

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

$$L_m(G_i) = \{s \in \Sigma^* \mid \delta^*(q_i, s) \in Q_m\} \tag{5}$$

Definition 2.2: 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.,

$$L(q_i, q_j) = \{s \in \Sigma^* \mid \delta^*(q_i, s) = q_j \in Q\} \tag{6}$$

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 we desire to reach, and Q_m^- contains all *bad* marked states that we want to avoid, 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: The characteristic function $\chi : Q \to [-1, 1]$ that assigns a signed real weight to state-based sublanguages $L(q_i, q)$ is defined as:

$$\forall q \in Q, \quad \chi(q) \in \begin{cases} [-1, 0), & q \in Q_m^- \\ \{0\}, & q \notin Q_m \\ (0, 1], & q \in Q_m^+ \end{cases} \tag{7}$$

The state weighting vector, denoted by $\mathbf{X} = [\chi_1 \ \chi_2 \ \cdots \ \chi_n]^T$, where $\chi_j \equiv \chi(q_j) \ \forall k$, is called the $\mathbf{X}$-vector. The j-th element χ_j of $\mathbf{X}$-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, the regular languages $L(G_i)$ and $L_m(G_i)$ can be expressed as:

$$L(G_i) = \bigcup_{q_k \in Q} L(q_i, q_k) = \bigcup_{k=1}^{n} L(q_i, q_k) \tag{8}$$

$$L_m(G_i) = \bigcup_{q_k \in Q_m} L(q_i, q_k) = L_m^+ \cup L_m^- \tag{9}$$

where the sublanguage $L(q_i, q_k) \subseteq G_i$ having the initial state q_i is uniquely labeled by the terminal state $q_k, k \in \mathcal{I}$ and $L(q_i, q_j) \cap L(q_i, q_k) = \emptyset \ \forall j \neq k$; and $L_m^+ \equiv \bigcup_{q \in Q_m^+} L(q_i, q)$ and $L_m^- \equiv \bigcup_{q \in Q_m^-} L(q_i, q)$ are good and bad sublanguages of $L_m(G_i)$, respectively. Then, $L^0 = \bigcup_{q \notin Q_m} L(q_i, q)$ and $L(G_i) = L^0 \cup L_m^+ \cup L_m^-$.

Now we construct a signed real measure $\mu : 2^{L(G_i)} \to \mathbf{R} \equiv (-\infty, +\infty)$ on the σ-algebra $K = 2^{L(G_i)}$. Interested readers are referred to [5] for the details of measure-theoretic definitions and results. With the choice of this σ-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 2.4: The event cost of the DFSA G_i is defined as a (possibly partial) function $\tilde{\pi} : Q \times \Sigma^* \to [0, 1)$ such that $\forall q_i \in Q$, $\forall \sigma_j \in \Sigma$, $\forall \omega \in \Sigma^*$,

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

Definition 2.5: The state transition cost of the DFSA G_i is defined as a function $\pi : Q \times Q \to [0, 1)$ such that $\forall q_i, q_j \in Q$, $\pi[q_i, q_j] = \sum_{\sigma \in \Sigma : \delta(q_i, \sigma) = q_j} \tilde{\pi}[q_i, \sigma] \equiv \pi_{ij}$ and $\pi_{ij} = 0$ if $\{\sigma \in \Sigma : \delta(q_i, \sigma) = q_j\} = \emptyset$. The $n \times n$ state transition cost $\mathbf{\Pi}$-matrix is defined accordingly.

Definition 2.6: The signed real measure μ of every singleton string set $\Omega = \{\omega\} \in 2^{L(G_i)}$ where $\omega \in L(q_i, q)$ is defined as $\mu(\Omega) \equiv \tilde{\pi}[q_i, \omega]\chi(q)$. It follows that the signed real measure of the sublanguage $L(q_i, q) \subseteq L(G_i)$ is defined as

$$\mu(L(q_i, q)) = \left(\sum_{\omega \in L(q_i, q)} \tilde{\pi}[q_i, \omega] \right) \chi(q) \tag{10}$$

And the signed real measure of the language of a DFSA G_i initialized at a state $q_i \in Q$, is defined as:

$$\mu_i \equiv \mu(L(G_i)) = \sum_{q \in Q} \mu(L(q, q_i)) \tag{11}$$

The language measure vector, denoted as $\boldsymbol{\mu} = [\mu_1 \ \mu_2 \ \cdots \ \mu_n]$, is called the $\boldsymbol{\mu}$-vector. It is shown in [5] that the signed real measure μ_i can be written as

$$\mu_i = \sum_j \pi_{ij} \mu_j + \chi_i \tag{12}$$

In vector form, $\boldsymbol{\mu} = \mathbf{\Pi}\boldsymbol{\mu} + \mathbf{X}$ whose solution is given by

$$\boldsymbol{\mu} = (\mathbf{I} - \mathbf{\Pi})^{-1} \mathbf{X} \tag{13}$$

III. Language Measure Parameters

This section focuses on identification of the parameters (i.e., elements) of the event cost matrix $\tilde{\mathbf{\Pi}}$ (see Definition 2.4) which, in turn, allows computation of the state transition cost matrix $\mathbf{\Pi}$ (see Definition 2.5) and the language

measure $\boldsymbol{\mu}$-vector (see Definition 2.6). We propose a recursive algorithm for identification of the $\tilde{\mathbf{\Pi}}$-matrix parameters under the following assumptions.

1) All states of the DFSA plant model are visited *infinitely* often;
2) The underlying physical process is evolving at two different time scales. In the fast-time scale, i.e., over a short time period, the system is 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 stochastic process.

For a slowly-varying non-stationary process, the DES control policy can be redesigned in real time. In that case, the $\tilde{\mathbf{\Pi}}$-matrix parameters should be periodically updated.

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 Σ_k^u denote the set of all unmodeled events at state k of the DFSA $G_i \equiv \langle Q, \Sigma, \delta, q_i, Q_m \rangle$. Let us create a new unmarked absorbing state q_{n+1} called the dump state [4] and extend the transition function δ to $\delta_e : (Q \cup \{q_{n+1}\}) \times (\Sigma \cup (\cup_k \Sigma_k^u)) \rightarrow (Q \cup \{q_{n+1}\})$ as follows:

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

Therefore the residue $\theta_k = 1 - \sum_j \tilde{\pi}_{kj}$ denotes the probability of the set of unmodeled events Σ_k^u conditioned on the state q_k. The $\mathbf{\Pi}$-matrix is accordingly augmented to obtain a stochastic matrix. Since the dump state q_{n+1} is not marked, its characteristic value $\chi(q_{n+1}) = 0$. The characteristic vector then augments to $\mathbf{X}^{aug} \equiv [\mathbf{X}\ 0]^T$. With these extensions the language measure vector $\boldsymbol{\mu}^{aug}$ = $[\mu_1\ \mu_2\ \cdots\ \mu_n\ \mu_{n+1}]^T$ =$[\boldsymbol{\mu}\ \mu_{n+1}]^T$ of the augmented DFSA $G_i^{aug} \equiv (Q \cup \{q_{n+1}\}, \Sigma \cup (\cup_k \Sigma_k^u), \delta_e, q_i, Q_m)$ can be expressed as:

$$\begin{pmatrix} \boldsymbol{\mu} \\ \mu_{n+1} \end{pmatrix} = \begin{pmatrix} \mathbf{\Pi}\boldsymbol{\mu} + \mu_{n+1} [\theta_1 \cdots \theta_n]^T \\ \mu_{n+1} \end{pmatrix} + \begin{pmatrix} \mathbf{X} \\ 0 \end{pmatrix} \quad (14)$$

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 (14) reduces to that for the original plant DFSA G_i. Thus, the event cost can be interpreted as the 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. In our earlier work [6], a recursive parameter estimation scheme has been proposed.

IV. DES CONTROL SYNTHESIS IN μ-MEASURE

In the conventional discrete event supervisory (DES) control synthesis [4], 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 Σ_u of uncontrollable events, then a supremal controllable sublanguage 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 μ 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 μ 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 $q \in Q_S$.

$$\tilde{\pi}_S[q, \sigma] = \frac{\tilde{\pi}_G[q, \sigma]}{\sum_{\sigma \in \Sigma_S(q)} \tilde{\pi}_G[q, \sigma]} \quad (15)$$

Under this assumption, 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.,

$$\sum_{\sigma \in \Sigma_S(q)} \tilde{\pi}_S[q, \sigma] = \sum_{\sigma \in \Sigma_G(q)} \tilde{\pi}_G[q, \sigma] \quad (16)$$

Lemma 4.1: (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 $i \in \mathcal{I}_c$, $L(S_i) = L(S_i/G) \subseteq L(G)$.

Lemma 4.1 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 4.1: (Existence) [2] Given a DES plant $G = (Q, \Sigma, \delta, q_0, Q_m)$, $\mathbf{\Pi}$-matrix, and $\mathbf{X}$-vector, there exist an optimal supervisor S^* such that $\mu(L(S^*/G)) = \max_{i \in \mathcal{I}_c} \mu(L(S_i/G))$.

Theorem 4.2: Given a DES plant $G = (Q, \Sigma, \delta, q_0, Q_m)$, $\mathbf{\Pi}$-matrix, and $\mathbf{X}$-vector. Define $\Sigma_c(q)$ as follows

$$\Sigma_c(q) = \{\sigma \in \Sigma_c \mid \delta(q, \sigma) \text{ is defined in } G\} \quad (17)$$

If supervisor S is the optimal supervisor S^*, then for every state $q \in Q$, such that $\Sigma_c(q) \neq \emptyset$, there is one and only one controllable event enabled.

Theorem 4.1 states that out of all possible supervisors constructed from G, there exists a supervisor that maximizes the language measure μ with respect to $(\mathbf{\Pi}, \mathbf{X})$. Theorem 4.2 discribes a general transition structure of the optimal supervisor S^*. In particular, at every state in S^*, there is at most one controllable event defined.

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 μ, equivalently, disabling the rest of controllable events defined at that state, if any. Below we first give a recursive synthesis

algorithm and then show that $\boldsymbol{\mu}$ is monotonically increasing elementwise on every iteration.

1) Initialization. $\mathbf{\Pi}^0 = \mathbf{\Pi}_p$, then $\boldsymbol{\mu}^{\mathbf{0}}$=$(I - \mathbf{\Pi}^0)^{-1}\mathbf{X}$.
2) Recursion. k-th iteration, where $k \geq 1$.
   1) For every $q \in Q$, find out the event $\sigma^* \equiv \sigma^*(q) \in \Sigma_c(q)$ such that $q^* = \delta(q, \sigma^*)$ and

$$\mu(L(S^k(\tilde{\mathbf{\Pi}}^k), q^*) = \max_{\substack{\sigma \in \Sigma_c(q) \\ q' = \delta(q,\sigma)}} \mu(L(S^k(\tilde{\mathbf{\Pi}}^k), q') \qquad (18)$$

   where $S^k(\tilde{\mathbf{\Pi}}^k)$ is the intermediate supervisor at k-th iteration whose transition is determined by $\tilde{\mathbf{\Pi}}^k$. Let $\boldsymbol{\sigma}^* = [\sigma_1^* \ \sigma_2^* \ \cdots \ \sigma_n^*]^T$, where the i-th element in $\boldsymbol{\sigma}^*$ is the controllable event left-enabled at state q_i of the plant G according to Equation 18.
   2) Disable the event set $\Sigma_c(q) - \{\ \boldsymbol{\sigma}^*(q)\}$ for every $q \in Q$ and redistribute event cost according to Equation 15. This results in a new $\tilde{\mathbf{\Pi}}^{k+1}$-matrix which consequently produces a new $\mathbf{\Pi}^{k+1}$-matrix.

$$\mathbf{\Pi}^{k+1} = \mathbf{\Pi}^k + \Delta^k \qquad (19)$$

   where Δ^k records the difference between $\mathbf{\Pi}^{k+1}$ and $\mathbf{\Pi}^{k+1}$, consisting positive and negative event costs corresponding to those kepted and disabled controllable events, respectively.
   3) Compute $\boldsymbol{\mu}^{k+1}$ =$(I - \mathbf{\Pi}^{k+1})^{-1}\mathbf{X}$
3. Termination. If $\tilde{\mathbf{\Pi}}^{k+1} = \tilde{\mathbf{\Pi}}^k$, 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 $S^k(\tilde{\mathbf{\Pi}}^k)$ with respect to the plant G. Therefore, $L(S^k(\tilde{\mathbf{\Pi}}^k)) \subseteq L(G)$.

Theorem 4.3: (Monotonicity) The sequence $\boldsymbol{\mu}^k$, $k = 1, 2, \ldots$, generated recursively by the above algorithm is monotonically increasing.

Given the fact that every iteration in the above recursive synthesis algorithm gives an intermediate supervisor S^k in the form of $\tilde{\mathbf{\Pi}}^k$ and its $\boldsymbol{\mu}$ is monotonically increasing. Moreover, as there are only finitely many supervisors, the algorithm should stop in a finite number of iterations and yield an optimal supervisor S^* in terms of the measure $\boldsymbol{\mu}$.

V. Experiment validation

This section presents a robotic simulation test bed that has been validated by real experiments. The control system architecture is shown in Figure 1. Player/Stage[1] is employed in the test bed. A Pioneer 2 AT robot model equipped with sonar, laser range finder, vision camera, gripper, and battery is used in the simulation experiments. The DES control module (DCM) communicates with the robot's continuous varying control module(CVCM) by sending and receiving a set of events (see Table I). The DCM is designed to be independent of the underlying physical process such that it provides a mechanism to interact with the CVCM. A DES controller is allowed to plug and play in DCM. The CVCM consists of three blocks as shown in Figure 1: C/D, D/C, and Continuous time control $C(t)$. The CVCM connects to Player via a TCP socket for sending and receiving formatted messages that encode sensor readings and continuous time commands of reference signals, respectively, at 10 Hz. The control strategy is event-driven, in which CVCM generates discrete events based on the continuous sensor data received from Player. The discrete events are transmitted to DCM in symbolic form. By receiving a particular event from the DES controller, CVCM 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.

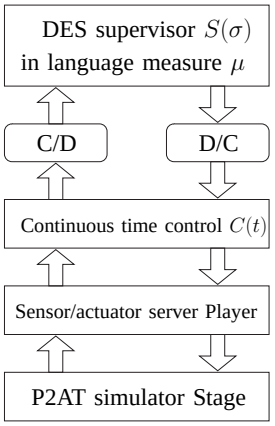


Fig. 1. Pioneer 2 AT DES simulation block diagram

The experimental scenario consists of a single robot performing logistic supply and combat operation in a battle field. There are two friendly units and one enemy, 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.

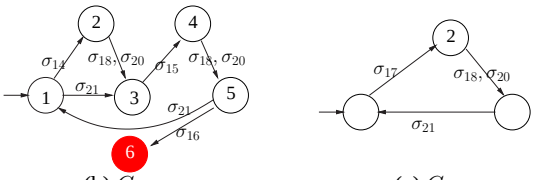
A gripper failure is modelled to signify the robot's failure to complete the task of supplying units. However, fighting with enemy is still possible. In addition, during the mission, the battery consumption rate changes according to the robot's current actions. For example, the robot's power consumption is higher during the fight with the enemy than supplying the units. The robot needs to return to its base to be recharged before the battery voltage drops below a certain level, or the robot is considered to be lost. After each

[1]Player/Stage is developed at University of Southern California under GNU Public License. It is available at `http://playerstage.sourceforge.net/`

TABLE I

LIST OF DISCRETE EVENTS

Σ	Description	Σ	Description
σ_1	start mission	σ_{12}	win the fight
σ_2	search	σ_{13}	loose the fight
σ_3	find blue unit	σ_{14}	battery power medium
σ_4	find pink unit	σ_{15}	battery power low
σ_5	find enemy	σ_{16}	battery power dead
σ_6	proceed to supply	σ_{17}	detected gripper fault
σ_7	ignore unit	σ_{18}	abort mission
σ_8	fight enemy	σ_{19}	return
σ_9	avoid enemy	σ_{20}	ignore anormly
σ_{10}	finish supply	σ_{21}	return successfully
σ_{11}	fail supply		



(b) G_2 (c) G_3

Fig. 3. Battery failure model and gripper failure model

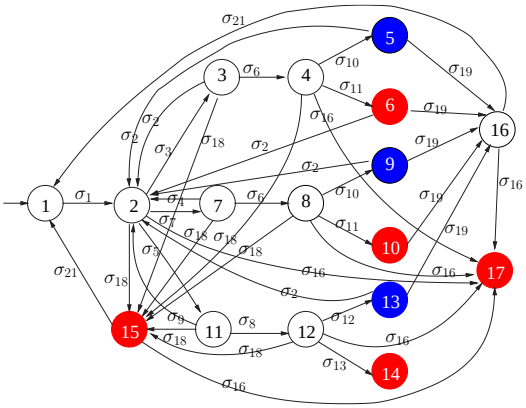
TABLE II

ITERATION OF μ SYNTHESIS

μ^0	μ^1	μ^2	μ^3	μ^4
0.5466	0.8635	0.8657	0.8662	0.8662

successful return to the base, the robot is reset to normal conditions including full battery charge and normal gripper status. If the robot is incapacitated 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, we model the entire interaction between robot and environment by three independent submodels, as shown in Figure 2 and Figure 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 to generate the integrated plant model $G = G_1 \| G_2 \| G_3$.



(a) G_1

Fig. 2. Robot behavioral model

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 I. The event cost matrix $\tilde{\Pi}$ 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 $\tilde{\Pi}$-matrix is demonstrated in Figure 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,χ_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 algorithm presented in Section 4, an optimal supervisor S^* is synthesized. The optimal DES control algorithm converges at the fourth iteration, as listed in Table II. For the purpose of performance comparison, two supervisors under the following specifications are designed.

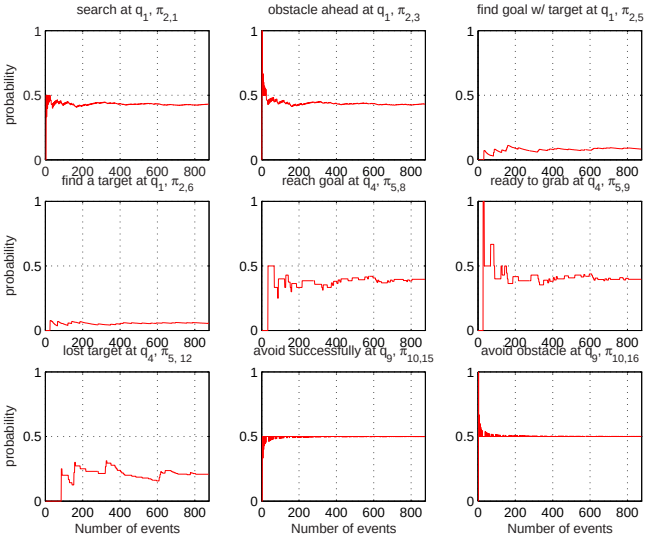


Fig. 4. Some non-zero elements of $\tilde{\Pi}$-matrix

Controller S_1 specifications

1) Avoid enemy if battery 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.

TABLE III

SIMULATION STATISTICS OF 400 MISSIONS

Items	G	S_1	S_2	S^*
$\frac{\text{proceed to supply}}{\#\text{ of units found}}$	$\frac{150}{434}$	$\frac{134}{408}$	$\frac{137}{367}$	$\frac{482}{556}$
	34.56%	32.84%	37.33%	**86.69%**
finish supply	112	88	97	362
$\frac{\text{win enemy}}{\text{fight enemy}}$	$\frac{37}{65}$	$\frac{33}{70}$	$\frac{22}{68}$	$\frac{69}{167}$
unit lost	26	19	14	48
μ	-0.6751	-0.6646	-1.1126	**2.8560**
$\sum_{i=0}^{400} \mu_i$	-20.25	-19.45	-27.3	**45.25**

Controller S_2 specifications

1) Abort operation if battery is not below medium;
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 III 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 robot loss 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 $\mu = (I - \mathbf{\Pi})^{-1}\mathbf{X}$ 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. The large oscillations are due to the loss of the robot due to drained battery or loss of the battle.

VI. SUMMARY AND CONCLUSIONS

This paper presents optimal supervisory control of robot behavior based on the discrete-event language measure μ [5]. A recursive supervisor synthesis algorithm is provided as an extension of the earlier work of Fu et al. [2]. The results of simulation experiments validates that a DES controller, designed by the recursive synthesis algorithm, indeed maximizes the μ-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 paper compares the proposed language measure (μ)-based approach of DES control with the Q-learning method. Conceptually, a weight is assigned to each transition in Q-learning whereas a weight is assigned to each state in μ-selection. A summary of comparison between Q-learning and μ-selection is given in table IV.

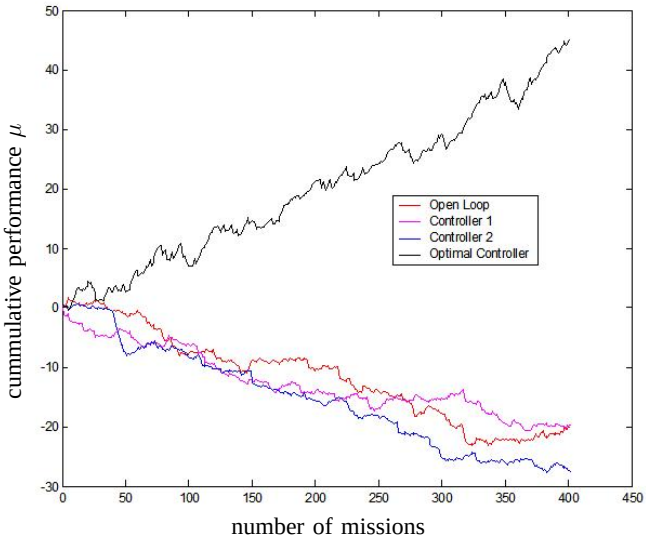


Fig. 5. Cummulative performance comparison of all controllers

TABLE IV

COMPARISON OF Q-LEARNING AND μ-SELECTION

Items	Q-learning	μ-selection
Modeling	$M = (S, A, T, R)$	$G = (Q, \Sigma, \delta, q_0, Q_m)$
Control policy	MDP	DES
Objective	$\arg\max_a Q^*(s, a)$	$\max_{s \in L(G)} \mu(s)$
Transition probability	required $T(s, a, s')$	required $\tilde{\pi}(q, \sigma)$
Reward	on transition $r(s, a)$	on state $\chi(q)$
Discount factor γ	yes(ad hoc)	no
Learning rate α	yes(ad hoc)	no
Online adaption to dynamic environment	recursive	(ϵ, δ)-threshold, redesign
Model complexity	exponential in S and A	polynomial in Q

REFERENCES

[1] R. C. Arkin, *Behavior-based robotics*, MIT Press, 1998.

[2] A. Ray J. Fu and C.M. Lagoa, *Unconstrained optimal control of regular languages*, IEEE Conference on Decision and Control, Las Vegas, Nevada (2002).

[3] S. Mahadevan and J. Connell, *Automatic programming of behavior-based robots using reinforcement learning*.

[4] P. J. Ramadge and W. M. Wonham, *Supervisory control of a class of discrete event processes*, SIAM J. Control and Optimization **25** (1987), no. 1, 206–230.

[5] X. Wang and A. Ray, *Signed real measure of regular languages*, Proceedings of the 2002 American Control Conference **5** (2002), 3937–3942.

[6] X. Wang, A. Ray, and K. Amol, *Online identification of language measure parameters for discrete event supervisory control*, Proc. of 42th IEEE Conf. on Decision and Control (2003).

[7] Christopher J. C. H. Watkins and Peter Dayan, *Q-learning*, Machine Learning **8** (1992), no. 3, 279–292.