Learning Multi-Modal Control Programs

Tejas R. Mehta and Magnus Egerstedt

{tmehta,magnus}@ece.gatech.edu
Georgia Institute of Technology
School of Electrical and Computer Engineering
Atlanta, GA 30332, USA

Abstract. Multi-modal control is a commonly used design tool for breaking up complex control tasks into sequences of simpler tasks. In this paper, we show that by viewing the control space as a set of such tokenized instructions rather than as real-valued signals, reinforcement learning becomes applicable to continuous-time control systems. In fact, we show how a combination of state-space exploration and multi-modal control converts the original system into a finite state machine, on which Q-learning can be utilized.

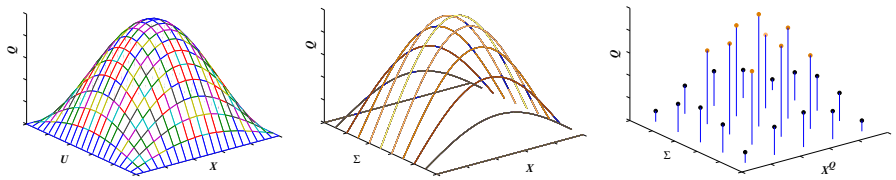
1 Introduction

In this paper we study the problem of controlling complex systems through the decomposition of the control task into a sequence of control modes. Such a divide and conquer approach has proved useful in that it allows the control designer to construct a number of relatively simple control laws, rather than one complex law. Successful examples of this approach include flight mode control in avionics and behavior based control of autonomous robots.

The aim of this paper is to show that such multi-modal control design strategies allow us to use standard reinforcement learning techniques on previously computationally intractable problems, namely for continuous-time control systems, where the states and control signals take on values in uncountably large sets. To see how this can be done, it should be noted that reinforcement learning is readily applicable when the state space and the input set are finite sets and the system is event driven (e.g. finite state machines or Markov decision processes). See for example [9, 11, 16, 17].

However, by considering a finite number of feedback laws $\kappa_i,\ i = 1, \ldots, M$ (i.e. mappings from the state space to the input set), together with interrupts $\xi_j,\ j = 1, \ldots, N$, which are the conditions for the termination of the current mode, a finite quantization of the control space is obtained. Note that the control set itself is not quantized but rather that the quantization acts at a functional level. This observation takes care of the problem of quantizing the control inputs. Moreover, by adopting a Lebesque sampling strategy where a new state is sampled only when the interrupts trigger, the continuous time problem is transformed into an event-driven problem. The final piece of the puzzle is the

observation that, given an initial state x_0 and a finite length multi-modal program, only a finite number of states are reachable. These ideas are illustrated in Figure 1, where the first figure corresponds to a case where the state space $X \sim \mathbb{R}^n$ and the input set $U \sim \mathbb{R}^m$. Depicted as a function of x and u is the so-called Q-function that characterizes the utility of using control input u at state x. In the next figure, U is replaced by Σ, which corresponds to a finite set of control-interrupt pairs. Without discretizing U, a finite control space is obtained by defining a finite set of available control modes. The final figure shows a situation where both the state space and the input space are finite. The input space is again given by Σ, while X^Q is the quantized state space obtained through an exploration of the states that are reachable from x_0 (in N steps) at the distinct times at which the interrupts may trigger.



(a) (b) (c)

Fig. 1. Depicted is the progression from X and U being smooth manifolds (a) to the case when both the state space and the input set are finite (c) through the introduction of multi-modal control procedures and Lebesque sampling.

To go from a continuous time control system to a finite state machine is certainly not a new idea. In particular, discretizations of the space-time domain are routinely used for establishing reachability properties. However, such discretizations do not reflect the underlying dynamics in any meaningful way. Alternatives are given in [2], where tokenized control symbols result in reachable lattices, and in [15], where LTL specifications are defined for a quantized system while guaranteeing that the specifications still hold for the original system. The idea of structured state space explorations was pursued in [1], where the reachable part of the state space was implicitly discretized using rapidly-exploring random trees. Additional results on motion description languages and tokenized control strategies can be found in [3, 6–8]. Moreover, it is not necessary to let the state space and input set be finite in order to apply learning techniques [14]. For example, a set of basis functions can be defines for supporting the Q-function such as sigmoids, wavelets, or Gaussian kernel functions. However, the computational burden associated with these methods of often prohibitive.

In this paper we will make these preliminary, informal observations rigorous, and the outline of the paper is as follows: In Section 2 we will discuss reinforce-

ment learning for discrete event-driven systems and see how these techniques can be modified in order to incorporate multi-modal feedback strategies. In Section 3 we switch our attention to continuous-time control systems, where the state and control spaces are $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively. Contained in this section is moreover a robotics example, that illustrates the potential usefulness of the proposed approach. Additional improvement and refinement issues are treated in Section 4, followed by a brief robustness discussion in Appendix A.

2 Reinforcement Learning

For systems operating in unknown environments and/or with unknown dynamics, reinforcement learning provides the means for systematic trial-and-error interactions with the environment. Although the contribution of this paper is to apply learning techniques to multi-modal hybrid systems, we will here briefly cover the standard reinforcement-learning model.

2.1 Standard Reinforcement Learning

In the standard reinforcement-learning model, at each step (discrete time), the agent chooses an action, $u \in U_F$, based on the current state, $x \in X_F$, of the environment, where U_F and X_F are finite sets (Hence the subscript F). The corresponding result is given by $x_{k+1} = \delta(x_k, u_k)$, where $\delta : X_F \times U_F \to X_F$ is the state transition function that encodes the system dynamics. Moreover, a cost $c : X_F \times U_F \to \mathbb{R}$ is associated with taking action u at state x. The agent should choose actions in order to minimize the overall cost. Given a policy $\pi : X_F \to U_F$, the discounted cost that we wish to minimize is given by

$$V^\pi(x_0) = \sum_{k=0}^{\infty} \gamma^k c(x_k, \pi(x_k)),$$

where $\gamma \in (0, 1)$ is the discount factor and $x_{k+1} = \delta(x_k, \pi(x_k))$, $k = 0, 1, \ldots$

We will use $V^*(x)$ to denote the minimum discounted cost incurred if the agent starts in state x and executes the optimal policy, denoted by π^*. In other words, the optimal value function is defined through the Bellman equation

$$V^*(x) = \min_{u \in U_F} \big[c(x, u) + \gamma V^*(\delta(x, u))\big], \forall x \in X_F.$$

This equation simply states that the optimal value is obtained by taking the action that minimizes the instantaneous cost plus the remaining discounted cost. Once V^* is known, the optimal policy, π^*, follows directly through

$$\pi^*(x) = \min_{u \in U_F} \big[c(x, u) + \gamma V^*(\delta(x, u))\big],$$

which shows why knowing V^* is equivalent to knowing the optimal policy.

If we now let $Q^*(x,u)$ be the discounted cost for taking action u in state x and then continuing to act optimally, we observe that $V^*(x) = min_u Q^*(x,u)$, and therefore

$$Q^*(x,u) = c(x,u) + \gamma \min_{u' \in U_F} Q^*(\delta(x,u),u').$$

To find Q^*, we start by assigning a uniform value to every state-action pair, and then randomly select state-action pairs (x,u) and update the Q-table using the following Q-learning law

$$Q_k(x,u) := Q_{k-1}(x,u) + \alpha_k \Big(c(x,u) + \gamma \min_{u' \in U_F} \Big\{ Q_{k-1}(\delta(x,u),u') - Q_{k-1}(x,u) \Big\} \Big).$$

If each action is selected at each state an infinite number of times on an infinite run and α_k, the learning rate, is decayed appropriately, the Q values will converge to Q^* with probability 1. By appropriate decay of α_k we mean that $\sum_k \alpha_k = \infty$ while $\sum_k \alpha_k^2 < \infty$, hence decreasing the learning rate over time (e.g. $\alpha_k = 1/k$) will guarantee convergence. (For more details regarding reinforcement learning, see for example [9, 11, 14, 16, 17].)

2.2 Learning Control Programs

We now define a new input space that corresponds to tokenized descriptions of feedback laws and interrupts, as prescribed within the motion description language (MDL) framework. Instead of interacting with the environment at each step, the agent takes actions based on a feedback law κ, which is a function of the state x. The agent furthermore continues to act on the feedback control law κ until the interrupt ξ triggers, at which point a scalar cost is incurred.

Formally, let X_F and U_F be finite sets, as defined earlier, and let $\Sigma = K \times \Xi$, where $K \subseteq {U_F}^{X_F}$ (the set of all maps from X_F to U_F) and $\Xi \subseteq \{0,1\}^{X_F}$. Moreover, let $\tilde{\delta} : X_F \times \Sigma \to X_F$ be the state transition mapping, $\tilde{x}_{k+1} = \tilde{\delta}(\tilde{x}_k, (\kappa_k, \xi_k))$, obtained through the following free-running, feedback mechanism [8]: Let $\tilde{x}_0 = x_0$ and evolve x according to $x_{k+1} = \delta(x_k, \kappa_0(x_k))$ until the interrupt triggers, i.e. $\xi_0(x_{k_0}) = 1$ for some index k_0. Now let $\tilde{x}_1 = x(k_0)$ and repeat the process, i.e. $x_{k+1} = \delta(x_k, \kappa_1(x_k))$ until $\xi_1(x_{k_1}) = 1$. Now let $\tilde{x}_2 = x(k_1)$, and so on. Also let $\zeta : X_F \times \Sigma \to \mathbb{R}$ be the cost associated with the transition.

We want to apply reinforcement learning to this model. To accomplish this we must make a few modifications. First, note that $card(\Sigma)$ is potentially much larger than $card(U_F)$, where $card(\cdot)$ denotes the cardinality. This directly affects the number of entries in our Q-table. If all possible feedback laws and interrupts were available, the cardinality of the new input space would be $[2card(U_F)]^{card(X_F)}$ with obvious implications for the numerical tractability of the problem.

Second, in order to find Q^*, we start again by assigning a uniform value to every state-action pair, and then iteratively update the Q values by randomly selecting a state-action pair with the action comprising of one of the possible feedback laws in K and interrupts in Ξ. The consequent Q-learning law is

$$\begin{aligned} Q_k(x,(\kappa,\xi)) &:= Q_{k-1}(x,(\kappa,\xi)) \\ &+ \alpha_k \Big(\zeta(x,(\kappa,\xi)) + \gamma \min_{(\kappa',\xi')} \Big\{ Q_{k-1}(\tilde{\delta}(x,(\kappa,\xi)),(\kappa',\xi')) - Q_{k-1}(x,(\kappa,\xi)) \Big\} \Big). \end{aligned}$$

Since Ξ and K are finite, the set of all possible modes Σ is finite as well. Hence the convergence results still hold, as long as each mode is selected for each state an infinite number of times, and α_k decays appropriately.

2.3 Example: Maze

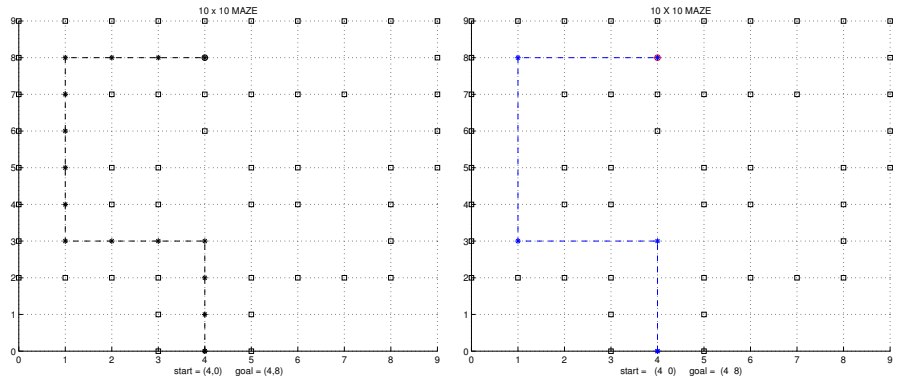
Consider the problem of an agent navigating a $M \times M$ planar grid (we will let $M = 10$) with obstacles. For any of the M^2 possible positions, the agent can move either north (N), south (S), east (E), west (W), or not at all (ϵ). Each such action, except of course ϵ, advances the agent one step, and it is understood that there is a boundary along the perimeter of the grid that the agent can not cross. Moreover the agent can advance through obstacles even though a hefty cost is incurred whenever this happens. Starting from an arbitrary location, the agent needs to find the shortest path to a specified goal, while avoiding obstacles.

We can restate this problem as a reinforcement learning problem, where the agent must learn the optimal policy given the model of the environment. Formally, we have

- $x = (x_1, x_2)$, where $x_1, x_2 \in \{0, 1, 2, \ldots, M-1\}$;
- $u \in \{N, S, E, W, \epsilon\}$;
- $\delta(x,u) = \begin{cases} (x_1, \min\{x_2+1, M-1\}) & \text{if } u = N \\ (x_1, \max\{x_2-1, 0\}) & \text{if } u = S \\ (\min\{x_1+1, M-1\}, x_2) & \text{if } u = E \\ (\max\{x_1-1, 0\}, x_2) & \text{if } u = W \\ (x_1, x_2) & \text{if } u = \epsilon \end{cases}$
- $c(x,u) = \begin{cases} 0 & \text{if } \delta(x,u) = x_{goal} \\ 100 & \text{if } \delta(x,u) \in \mathcal{O} \\ 1 & \text{otherwise} \end{cases}$

Here, x_{goal} is the goal state, while $\mathcal{O} \subset X$ is the set of obstacles. Using standard Q-learning, as previously described, the agent quickly learns the shortest path to the goal and the resulting simulation result is shown in Figure 2(a).

In this example, each input corresponds to one step in the maze. However, one could ask the question about the shortest mode string that makes the agent reach the goal, following the development in [8]. Unfortunately, the total number of feedback laws is $card(K) = card(U_F)^{card(X_F)}$, i.e. in this example we have 5^{100} possible control modes, which is a numerically intractably large number. Hence, we have to reduce the size of K, and our particular choice is the set of constant feedback laws, i.e. $K = \{\kappa_N, \kappa_S, \kappa_E, \kappa_W, \kappa_\epsilon\}$, where $\kappa_N(x) = N,\ \forall x \in X_F$, and so on. Similarly, we need to limit the size of the interrupt set, and we simply let Ξ be set of interrupts that trigger after m steps, $m = 1, 2, \ldots, N$. (We denote these interrupts by $\Xi = \{\xi_1, \ldots, \xi_N\}$.) In this case $card(\Xi) = N$, and for the particular problem we are interested in, we let $N = 9$ (since $M = 10$), so we need $9 \times 5 \times 100 = 4500$ entries in the Q-table. Note that, in order to keep track of the number of steps, the state space has to be augmented in a straightforward manner.



(a) (b)

Fig. 2. Robot navigating through a maze using a standard reinforcement-learning model (left) and using modes with interrupts as the control set (right).

Now in order to find Q^*, and consequently the optimal policy, we start by assigning a uniform value to every state-action pair (recall we have 4500 possible such pairs). We then randomly select a state-action pair and update its Q-value according to the previously discussed, modified Q-learning law. The result of the simulation is shown in Figure 2(b). Note that this may not always be the shortest path in terms of length (even though it happens to be the shortest in this particular case), but it is the optimal path in terms of the length of the mode string.

3 Learning Control Programs for Continuous Systems

Now that the discrete-time case with finite state and input spaces is covered, we shift focus to the main contribution of this paper, namely the solution to the problem of learning multi-modal control programs for continuous-time systems. Suppose we have the following system:

$$\dot{x} = f(x,u), \qquad x \in X = \mathbb{R}^n, \qquad u \in U = \mathbb{R}^m, \qquad \text{where } x(t_0) = x_0 \text{ is given.}$$

If at time t_0, the system receives the input string $\sigma = (\kappa_1, \xi_1), \ldots, (\kappa_q, \xi_q)$, where $\kappa_i : X \to U$ is the feedback control law, and $\xi_i : X \to \{0,1\}$ is the interrupt, then x evolves according to

$$\begin{aligned} \dot{x} &= f(x, \kappa_1(x)); \quad t_0 \le t < \tau_1 \\ &\vdots \qquad\qquad\quad \vdots \\ \dot{x} &= f(x, \kappa_q(x)); \quad \tau_{q-1} \le t < \tau_q, \end{aligned}$$

where τ_i denotes the time when the interrupt ξ_i triggers (i.e. changes from 0 to 1).

We are interested in finding a sequence of control-interrupt pairs that minimizes a given cost for such a system. For example, we might be interested in driving the system to a certain part of the state space (e.g. to the origin), and penalize the final deviation from this target set. Previous work on reinforcement learning for continuous-time control systems can broadly be divided into two different camps. The first camp represents the idea of a direct discretization of the temporal axis as well as the state and input spaces (e.g. [4, 13]). The main criticism of this approach is that if the discretization is overly coarse, the control optimizing the discretized problem may not be very good when applied to the original problem. Of course, this complication can be moderated somewhat by making the discretization more fine. Unfortunately, in this case, the size of the problem very quickly becomes intractable.

The second approach is based on a temporal discretization (sampling) in combination with the use of appropriate basis functions to represent the Q-table (e.g. [5, 12, 14]). Even though this is a theoretically appealing approach, it lacks in numerical tractability. In contrast to both these two approaches, we propose to let the temporal quantization be driven by the interrupts directly (i.e. not by a uniform sampling) and let the control space have finite cardinality through the interpretation of a control symbol as a tokenized control-interrupt pair. In other words, by considering a finite number of feedback laws $\kappa_i : X \to U,\ i = 1, \ldots, M$, together with interrupts $\xi_j,\ j = 1, \ldots, N$, the control space (viewed at a functional level) is finite even though the actual control signals take on values in $\mathbb{R}^m$. Another effect of the finite mode-set assumption is that it provides a natural quantization of the state space. Moreover, if we bound the length of the mode sequences, this quantization is in fact resulting in a finite set of reachable states.

Given an input $\sigma = (\kappa, \xi) \in \Sigma$, where $\Sigma \subseteq U^X \times \{0, 1\}^X$, the flow is given by

$$\phi(x_0, \sigma, t) = x_0 + \int_0^t f(x(s), \kappa(x(s)))ds.$$

If there exists a finite time $T \geq 0$ such that $\xi(\phi(x_0, \sigma, T)) = 1$, then we let the interrupt time be given by

$$\tau(\sigma, x_0) = min\{t \geq 0 \mid \xi(\phi(x_0, \sigma, t)) = 1\}.$$

If no such finite time T exists then we say that $\tau(\sigma, x_0) = \tau_\infty$ for some distinguishable symbol τ_∞. Furthermore, we let the final point on the trajectory generated by σ be

$$\chi(\sigma, x_0) = \phi(x_0, \sigma, \tau(\sigma, x_0))$$

if $\tau(\sigma, x_0) \neq \tau_\infty$ and use the notation $\chi(\sigma, x_0) = \chi_\infty$ otherwise. Moreover let $\chi(\sigma, \chi_\infty) = \chi_\infty, \forall \sigma \in \Sigma$.

This construction allows us to define the Lebesque sampled finite state machine $(X_N^Q, \Sigma, \tilde{\delta}, \tilde{x_0})$, where N is the longest allowable mode string, and where the state transition is given by

$$\tilde{x}_0 = x_0$$

$$\tilde{x}_{k+1} = \tilde{\delta}(\tilde{x}_k, \sigma_k) = \chi(\sigma_k, \tilde{x}_k), k = 0, 1, \ldots$$

The state space X_N^Q is given by the set of all states that are reachable from $\tilde{x}_0$ using mode strings of length less than or equal to N.

Now that we have a finite state machine describing of the dynamics, we can run our learning algorithm, with an appropriate cost function, in order to obtain the optimal control program as discussed earlier. However, in order to preserve computing resources, we run this in parallel with the state exploration, and the general algorithm for accomplishing this is given by

$\mathcal{X} := \{\tilde{x}_0, \tilde{\delta}(\tilde{x}_0, \sigma)\},\ \forall \sigma \in \Sigma$
$step(\tilde{x}_0) := 0$
$step(\tilde{\delta}(\tilde{x}_0, \sigma)) := 1,\ \forall \sigma \in \Sigma$
$p := 1$
$Q_p(\tilde{x}, \sigma) := const\ \forall \tilde{x} \in \mathcal{X}, \sigma \in \Sigma$
`repeat`
  $p := p + 1$
  $\tilde{x} := rand(\chi \in \mathcal{X} \mid step(\chi) < N)$
  $\sigma := rand(\Sigma)$
  $\tilde{x}' := \tilde{\delta}(\tilde{x}, \sigma)$
  `if` $\tilde{x}' \notin \mathcal{X}$ `then`
    $step(\tilde{x}') := step(\tilde{x}) + 1$
    $\mathcal{X} := \mathcal{X} \cup \{\tilde{x}'\}$
    $Q(\tilde{x}', \sigma) := const\ \forall \sigma \in \Sigma$
  `end if`
  $Q_p(\tilde{x}, \sigma) := Q_{p-1}(\tilde{x}, \sigma)$
    $+ \alpha_p\Big(\zeta(\tilde{x}, \sigma) + \gamma \min_{\sigma' \in \Sigma}\Big\{Q_{p-1}(\tilde{x}', \sigma') - Q_{p-1}(\tilde{x}, \sigma)\Big\}\Big)$
`until` $mod(p, L) = 0$ *and* $|Q_p(\tilde{x}, \sigma) - Q_{p-L}(\tilde{x}, \sigma)| < \epsilon,\ \forall\ \tilde{x} \in \mathcal{X}, \sigma \in \Sigma$
$X_N^Q = \mathcal{X}$

Unlike the earlier Q-learning algorithm, the state space is initially unknown for this case, and we thus begin learning/exploring from the states we know (namely $\tilde{x}_0$ and all the states reachable in one step). At each iteration of the learning process, we select a state randomly from the set of known states and we select a mode randomly from the set of modes. In the algorithm, the function $step(\tilde{x})$ represents the length of the shortest control program used so far to reach state $\tilde{x}$ from the initial state $\tilde{x}_0$. This is to ensure we only explore states that are reachable from $\tilde{x}_0$ using mode strings of length less than or equal to N, i.e. $\mathcal{X} \subseteq X_N^Q$. We then calculate the next state and determine if it is a member of our known state space (In practice, it is necessary to check if the next state belongs to a neighborhood of a previously visited state). If not, add this state to the known state space and make the corresponding change in the Q-table. We continue to explore and update the state space and our Q-table (or value function) in this manner until the Q-table is stationary. Note that in the algorithm, $\epsilon > 0$ is a small positive scalar and L is a large number needed to ensure that sufficiently many state-action pairs are visited.

3.1 Example

Consider the following simple planar integrator system:

$$\dot{x} = u, \qquad x, u \in \mathbb{R}^2, \qquad x_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

Moreover, let the modes be given by $\Sigma = \{\sigma_{ij} = (\kappa_i, \xi_{ij}),\ i = 1, 2,\ j = 1, \ldots, 5\}$, where

$$\begin{aligned} \kappa_1(x) &= \begin{pmatrix} 1 & 0.1 \\ 0 & -1 \end{pmatrix} x \\ \kappa_2(x) &= \begin{pmatrix} -1 & 0 \\ -0.2 & 2 \end{pmatrix} x \\ \xi_{1j} &= \begin{cases} 1 \text{ if } x_2^2 < M\delta^j \\ 0 \text{ otherwise} \end{cases} \\ \xi_{2j} &= \begin{cases} 1 \text{ if } x_1^2 < M\delta^j \\ 0 \text{ otherwise} \end{cases} \\ & for\ j = 1, 2, \ldots, 5, \end{aligned}$$

where $M, \delta > 0$. Note that the system is unstable in either mode. We want to learn a mode string that will stabilize the system, i.e. drive it to the origin. Although it may not be possible to drive the system to $x = 0$ with these particular control-interrupt pairs, we want to select a string of modes which bring the system to a neighborhood of $x = 0$.

For the particular choice of modes, the reachable set has cardinality $2\sum_{i=0}^{N} 5^i$, where N is the maximum number of steps (or string length) and as can be expected, the cardinality of the state space increases exponentially with respect to the length of the control program. The resulting plot from solving the learning problem using the combined state space exploration and Q-learning is shown in Figure 3(a) in which $N = 5$ and the cost is given solely by the final distance to the origin. We could of course also change the cost to let it include an additive term that measures the total distance travelled. The corresponding, learned optimal trajectory for this cost is shown in Figure 3(b).

3.2 Example: Maze Revisited

We now apply this strategy for obtaining finite state machine descriptions of continuous time multi-modal control systems to the previously discussed maze problem. In particular, we still use the mode set $\{N, S, E, W, \epsilon\}$, but define it for a planar integrator instead of a finite state machine. We moreover let the interrupts, which previously counted the number of steps taken, correspond to a certain distance travelled. We apply this scheme to the problem of making a robot negotiate a maze and in Figure 4 the experimental setup is shown, where a Magellan Pro Mobile Robot from iRobot is to negotiate the maze. Figure 5 moreover shows final path obtained through the learning algorithm.

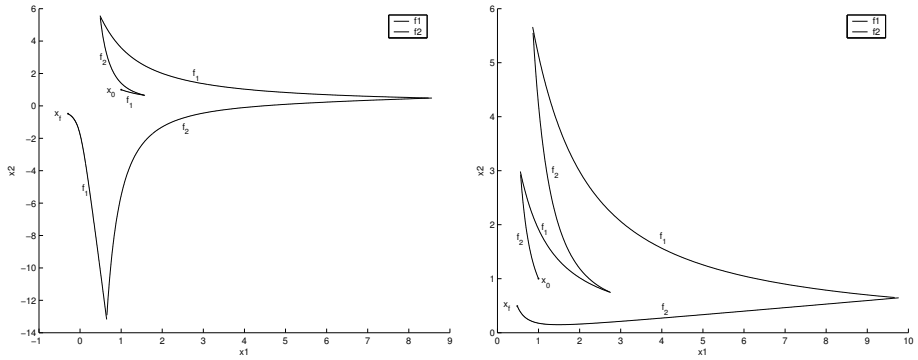


Fig. 3. In this example, $M = 1$ and $\delta = 0.75$ and the resulting optimal mode strings are (cost = final distance to the origin) $\hat{\sigma} = (\kappa_1, \xi_{13}) \cdot (\kappa_2, \xi_{25}) \cdot (\kappa_1, \xi_{15}) \cdot (\kappa_2, \xi_{23}) \cdot (\kappa_1, \xi_{15})$ (left) and (cost = final distance to the origin combined with total distance travelled) $\hat{\sigma} = (\kappa_2, \xi_{24}) \cdot (\kappa_1, \xi_{12}) \cdot (\kappa_1, \xi_{21}) \cdot (\kappa_2, \xi_{13}) \cdot (\kappa_1, \xi_{25})$ (right).

Fig. 4. Experimental setup

4 Refining the Learning Process

In this section we discuss some methods for enhancing the learning process. In particular, for problems with large state and input spaces (basically all interesting problems), the convergence is typically slow when using a purely random exploration strategy. However, it is well-known that one can use knowledge about the problem in order to speed up the learning process. The idea is to start out the learning process completely at random, but as the system gains "experience" the

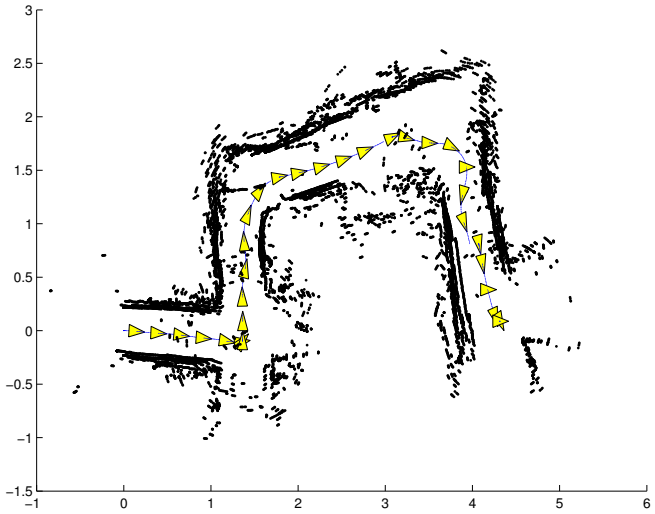


Fig. 5. The final trajectory. Depicted is the path of the robot together with the range sensor readings (IR-based) obtained throughout the final run. Note how the odometric drift makes the maze look somewhat distorted.

state space exploration becomes less and less random. In other words, we bias the selection of the state-action pairs to explore and update based on current values of the Q-table.

In order to formalize this, some notation is needed. We let $P(x, u)$ denote the probability of selecting state-action pair (x, u) from $X_F \times U_F$, with $\sum_{x \in X} \sum_{u \in U} P(x, u) = 1$. Initially we begin with

$$P_0(x, u) = \frac{1}{card(X)card(U)}.$$

In other words, every state-action pair has an equal likelihood of being selected. As we gain experience, we can change these probabilities to bias the selection in favor of state-action pairs with lower Q-values (potentially "good" state-action pairs). There may be many appropriate methods for biasing these probabilities, and one simple approach is to let the probability of selection state-action pair (x, u) be given by

$$P_k(x, u) = \frac{Q_{k-1}(x, u)}{\sum_{x' \in X_F} \sum_{u' \in U_F} Q_{k-1}(x', u')}.$$

Given such a biased probability distribution, we do not want to use it prematurely, for this may lead us to not learn the optimal policy. Instead we want to introduce a confidence value, $c \in [0, 1]$, which is based on the time step k and the past Q-values. With a lower value of c, the exploration strategy should be more random (i.e. use $P_0(x, u)$ when selecting a state-action pair), while higher

value of c suggest using a more biased exploration strategy (i.e. use $P_k(x, u)$). Note that we still want to leave some amount of randomness in the selection process in order to ensure that the entire state and input space is explored. Hence, c should never equal 1. The degree of bias in the selection process and the necessary experience will vary from problem to problem.

Based on our knowledge of the problem we can also start pruning the state-space as we gain experience. This means that we could exclude states that we are certain are not part of the optimal trajectory. This reduction in the size of the state-space enables the learning process to converge faster since all the plausible state-action pairs can be selected more often. However, great caution and high degree of accuracy must be used when pruning the state-space to ensure that the optimal policy is still learned since incorrectly pruning a potentially useful state may mean that only a sub-optimal policy is learned.

5 Conclusions

In this paper we present a method for going from continuous time control systems to finite state machines in a structured manner. In particular, by only considering a finite number of modes, i.e. control-interrupt pairs, the input space is finite and the continuous time dynamics has been replaced by a Lebesque sampled, discrete time system. Moreover, by only allowing mode strings of a certain length, the reachable state space (at the interrupt times) is finite as well. This construction means that previously unavailable computational methods, such as reinforcement learning, are now applicable in a straight forward manner.

Acknowledgements

This work was sponsored by the National Science Foundation through the program ECS NSF-CAREER award (grant # 0237971).

References

1. A. Bhatia, and E. Frazzoli. Incremental Search Methods for Reachability Analysis of Continuous and Hybrid Systems. *Hybrid Systems: Computation and Control. Springer-Verlag*, 2004.
2. A. Bicchi, A. Marigo, and B. Piccoli. On the reachability of quantized control systems. *IEEE Transactions on Automatic Control*, 4(47):546-563, April 2002.
3. A. Bicchi, A. Marigo, and B. Piccoli. Encoding steering control with symbols. *IEEE International Conference on Decision and Control*, pages 3343-3348, 2003.
4. S.J. Bradtke, B.E. Ydstie, and A.G. Barto. Adaptive linear quadratic control using policy iteration. *American Control Conference*, pages 3475-3479, 1994.
5. L. Crawford, and S.S. Sastry. Learning Controllers for Complex Behavioral Systems. *Neural Information Processing Systems Tenth Annual Conference(NIPS 96)*, 1996.

6. M. Egerstedt. On the Specification Complexity of Linguistic Control Procedures. *International Journal of Hybrid Systems*, Vol. 2, No. 1-2, pp. 129-140, March & June, 2002.
7. M. Egerstedt, and D. Hristu-Varsakelis. Observability and Policy Optimization for Mobile Robots. *IEEE Conference on Decision and Control, Las Vegas, NV*, Dec. 2002.
8. M. Egerstedt, and R.W. Brockett. Feedback Can Reduce the Specification Complexity of Motor Programs. *IEEE Transactions on Automatic Control*, Vol. 48, No. 2, pp. 213–223, Feb. 2003
9. T. Jaakkola, M.I. Jordan, and S.P. Singh. On the Convergence of stochastic iterative dynamic programming algorithms. *Neural Computation* 6(6), 1994.
10. L.P. Kaebling, M.L. Littman, and A.R. Cassandra. Learning Policies for Partially Observable Environments: Scaling Up. *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.
11. L.P. Kaebling, M.L. Littman, and A.W. Moore. Reinforcement learning: A survey. *Journal Of Artificial Intelligence Research*, 1996.
12. K. Morgansen, and R.W. Brockett. Optimal Regulation and Reinforcement Learning for the Nonholonomic Integrator. *Proceedings of the American Control Conference*, pp. 462-6, June 2000
13. R.S. Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *Neural Information Processing Systems 8*, 1996.
14. R.S. Sutton, and A.G. Barto. *Reinforcement Learning, An Introduction.* MIT Press, Cambridge, MA, 1998.
15. P. Tabuada and G. Pappas. Model Checking LTL over Controllable Linear Systems is Decidable. *Hybrid Systems: Computation and Control*, Springer-Verlag, Prague, Czech Republic, 2003.
16. J.N. Tsitsiklis. Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3), 1994.
17. C.J.C.H. Watkins, and P. Dayan. Q-learning. *Machine Learning* 8(3/4):257-277, May 1992.

Appendix A Robustness Analysis

Note that the entire argument presented in this paper concerning the finite state space model hinges on the fact that we start from a fixed initial state. In this section we will conduct a sensitivity analysis to show that if the mode string $\hat{\sigma}$ is optimal when starting at x_0, it is in fact still optimal for $\tilde{x}_0 = x_0 + \Delta x_0$, for some small perturbation Δx_0. It is sufficient to show that if x_0 is perturbed a little, then $\tilde{x}_f$, the point obtained after executing $\hat{\sigma}$ from $\tilde{x}_0$, lies within a small neighborhood of x_f, i.e. we need to show that $\Delta x_f = x_f - \tilde{x}_f$ is small.

In order to simplify the notation, we let the interrupt surfaces be encoded by smooth functions $g_i(x) = 0$, i.e. $\xi_i(x) = 1$ when $g_i(x) = 0$ and $\xi_i(x) = 0$ otherwise. Also, the trajectory of x is given by $x(t) = \Phi_1(t, t_0)$ until $g_1(x) = 0$. Then it is given by $x(t) = \Phi_2(t, \tau_1)$ until $g_2(x) = 0$, and so on. Here Φ_i is the state-transition function associated with $\dot{x} = f(x, \kappa_i(x))$, and τ_i is the time that interrupt ξ_i triggers, i.e. $g_i(x(\tau_i)) = 0$. Moreover we will denote this point $x_{h_i} = x(\tau_i)$. So for $t \in [0, \tau_1)$, we get

$$\begin{aligned}\dot{\tilde{x}} &= f_1(\tilde{x}, u) = f_1(x + \Delta x_0, u) \\ &= f_1(x, u) + \frac{\partial f_1}{\partial x}\Delta x_0 + o(\Delta x).\end{aligned}$$

Hence,

$$\dot{\Delta x} = \frac{\partial f_1}{\partial x}\Delta x_0 + o(\Delta x),$$

meaning that for $t \in [0, \tau_1)$, $\Delta x(t) = \Phi_1(t, t_0)\Delta x_0 + o(\Delta x)$. To examine the trajectory after the interrupt, we have to calculate the change in the interrupt time τ_1 and the position at this time, namely x_{h_1}. Again, using the first order approximation, we get

$$\begin{aligned}\tilde{x}(\tau_1 + \Delta\tau_1) &= x(\tau_1 + \Delta\tau_1) + \Delta x(\tau_1 + \Delta\tau_1) \\ &= x(\tau_1) + f_1(x(\tau_1))\Delta\tau_1 + \Delta x(\tau_1) + o(\Delta\tau_1).\end{aligned}$$

Here $t = \tau_1 + \Delta\tau_1$ is the time that the trajectory of $\tilde{x}$ hits the interrupt surface, so we must have

$$g_1(\tilde{x}(\tau_1 + \Delta\tau_1)) = 0,$$

which implies that

$$g_1(x(\tau_1)) + \frac{\partial g_1}{\partial x}(x(\tau_1))\big[f_1(x(\tau_1))\Delta\tau_1\big] + \frac{\partial g_1}{\partial x}(x(\tau_1))\Delta x(\tau_1) + o(\Delta\tau_1) = 0.$$

Letting $L_{f_1}g_1(x(\tau)) := \frac{\partial g_1}{\partial x}(x(\tau_1))\big[f_1(x(\tau_1))\Delta\tau_1\big]$, which is the Lie derivative of g_1 along f_1, and assuming that this quantity is non-zero, we get

$$\Delta\tau_1 = \frac{\frac{\partial g_1}{\partial x}(x(\tau_1))\Phi_1(\tau_1, t_0)\Delta x_0}{L_{f_1}g_1(x(\tau_1))},$$

where we have ignored higher order terms. Hence,

$$\begin{aligned}\Delta x_{h_1} &= \tilde{x}(\tau_1 + \Delta\tau_1) \\ &= \Big[I - \frac{f_1\frac{\partial g_1}{\partial x}(x(\tau_1))}{L_{f_1}g_1(x(\tau_1))}\Big]\Phi_1(\tau_1, t_0)\Delta x_0.\end{aligned}$$

Now, based on the assumption that $L_{f_1}g_1(x(\tau_1)) \neq 0$ (i.e. the interrupt triggers non-tangentially), Δx_{h_1} is small. Similarly we get that Δx_{h_2} is small under the assumption that $L_{f_2}g_2(x(\tau_2)) \neq 0$. Continuing in this manner, we deduce that Δx_f will be small as long as $L_{f_i}g_i(x(\tau_i)) \neq 0$, for $i = 1, \ldots, M$, and the result follows.