Learning to Locomote: Action Sequences and Switching Boundaries

Rowland O'Flaherty and Magnus Egerstedt

Abstract— **This paper presents a hybrid control strategy for learning the switching boundaries between primitive controllers that maximize the translational movements of complex locomoting systems. Through this abstraction, the algorithm learns an optimal action for each boundary condition instead of one for each discretized state and action of the system, as is typically in the case of machine learning. This hybridification of the problem mitigates the "curse of dimensionality". The effectiveness of the learning algorithm is demonstrated on both a simulated system and on a physical robotic system. In both cases, the algorithm is able to learn the hybrid control strategy that maximizes the forward translational movement of the system without the need for human involvement.**

I. Introduction

When the control design task is prohibitive due to the complexities of the specifications and the systems themselves, machine learning provides a possible way forward. In fact, learning as a means to produce control strategies has been used on a number of complex systems, such as helicopters [1], humanoid robots [2] [3], robotic arms [4], biological systems [5], and wind turbines [6]. Despite the success associated with these particular applications, a hurdle that almost all learning algorithms face is the "curse of dimensionality"; coined by Richard Bellman in the 1950s. This is the exponential increase of information that must be learned as the number of possible states and actions in the system increases.

In this paper, we present a model-free learning algorithm that overcomes this complexity issue by a particular choice of discretization. The algorithm uses boundary conditions coupled with sets of primitive control laws to create motions for the locomotion of complex robotic systems. In particular, the proposed algorithm learns actions based on boundary states instead of the actual system states, which greatly reduces the amount of learning that must take place.

This paper uses *reinforcement learning*, which can be problematic in high-dimensional continuous state-action systems, as observed in [8]. This scalability problem derives from the fact that, in general, reinforcement learning is attempting to learn the best action to take for each state of the system (a state-action pair) based on a given reward function. In order to facilitate such a formulation, the state-space and action-space must be discretized, partitioned, or parameterized in some way. Unfortunately, the number of possible state-action pairs grows exponentially with the growth of both the state-space and the action-space dimensions.

R. O'Flaherty and M. Egerstedt are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. The work by M. Egerstedt was supported by the DARPA M3 program. rowland.oflaherty@ece.gatech.edu magnus.egerstedt@ece.gatech.edu

Previous work has been done to try to mitigate this problem of feasibly performing a task in high dimensional spaces. For example Kuo et al. [9] discuss different sampling techniques that can be used for numerical integration in high dimensional spaces. We cope with the problem in a similar fashion as Kuo et al. and many other researchers; and that is by sampling the space in an intelligent way. Other methods include using neural approximators [10], clustering [11], or function approximation [12]. Our technique differs in that it hones in on the most important states (switching boundaries of the hybrid system) of the state space and only worries about those while ignoring the rest.

In fact, our proposed method to ameliorate this scalability problem is inspired by nature. It has been shown that animals and insects use a small set of motor primitives to construct and control movements [13]–[16]. Moreover, the transitions between motor primitives do not occur everywhere in the state space and we interpret this in terms of boundaries on which transitions may take place. This suggests that the action-space can be reduced to a finite space where the dimension is equal to the cardinality of the primitive control set. In addition, the state-space used for reinforcement learning can also be reduced to a finite set of boundaries. Therefore, the reinforcement learning algorithm will only need to learn *boundary-controller pairs* instead of state-action pairs. The real strength with this approach is that for highly complex systems—particularly those where it is infeasible to formulate an accurate model of the system dynamics due to imprecise manufacturing, unknown material properties, or complex physical interactions (e.g friction and fluid dynamics)—control for locomotion may still be learned in a computationally feasible manner.

The main contributions of this paper are (i) the introduction of a hybrid system methodology and reinforcement learning algorithm to learn control actions based on boundary conditions to mitigate the "curse of dimensionality", and (ii) the demonstration of the algorithm on both a simulated system and on a real robot shown in Figure 1.

II. System Overview

This paper introduces a learning algorithm for the locomotion of complex robotic systems. The form of the system that our learning algorithm is applicable to is outlined in detail below, but in general it is a continuous time system with the objective of moving in some direction. The algorithm described in this paper learns the appropriate sequence of control laws and the switching protocol that produces a motion that moves the system the "best", relative to a cost function. The switching between the control actions occurs

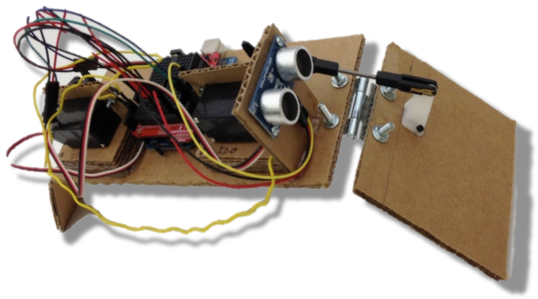

Fig. 1. Physical robotic system used to test the learning algorithm.

at discrete time instants when the state of the system reaches the learned boundary conditions.

In our proposed learning algorithm, the switching between primitive controllers creates a dynamical system that has both continuous time and discrete time dynamics, thus making it a hybrid system. Before we describe this hybrid system, let us begin with the dynamics and constraints of the system.

A. System Dynamics

The system dynamics under consideration in this paper can be written as

$$\dot{x} = f(x,u) = \begin{cases} \dot{x}_I &= f_I(x_I,u) \\ \dot{x}_E &= f_E(x_E,x_I), \end{cases} \tag{1}$$

where the system state, $x \in \mathbb{R}^n$, is composed of two parts: an internal state $x_I \in \mathbb{R}^{n_I}$ and an external state $x_E \in \mathbb{R}^{n_E}$. Thus, $x := [x_I, x_E]^\mathsf{T}$ and $n = n_I + n_E$. The internal state, x_I, describes the configuration of the system in reference to itself (e.g. actuator positions, joint angles, component velocities, etc.). The external state, x_E, describes the configuration of the system in reference to the outside world (e.g. location and velocity of the system in some global reference frame).

For the purpose of this paper, we assume that the internal state is rectangularly bounded.[1] Let these bounds be described by $x_{I\,min} \in \mathbb{R}^{n_I}$ and $x_{I\,max} \in \mathbb{R}^{n_I}$, where

$$x_{I\,min}(j) \leq x_I(j) \leq x_{I\,max}(j),\ \forall j \in \{1,\ldots,n_I\}.$$

The external state is allowed to be unbounded. The input to the system is given by $u \in \mathbb{R}^m$. For this class of systems, the input affects the internal state through $f_I(x_I,u)$ while it only indirectly affects the external state through the coupling with x_I.

B. Primitive Controllers and Decision Conditions

We assume that the primitive controllers have been designed such that they affect the internal state of the system and will always move the internal state until it hits a boundary[2]. In other words, the closed-loop system does not have any equilibrium points in the internal state space. If we let ξ index the controller selection, the primitive controllers are defined as $\kappa_\xi(x_I) : \mathbb{R}^{n_I} \to \mathbb{R}^m$, which determine the input, $u = \kappa_\xi(x_I)$. We let the set of all primitive controllers be given by $\mathcal{E} := \{1,\ldots,k\}$ (with $\xi \in \mathcal{E}$), where k is the number of different primitive controllers. Let us define the system dynamics while a particular controller is being applied as $f_\xi(x) := f(x,\kappa_\xi)$.

Decisions are made on what control law to use when the internal state of the system intersects a decision boundary. These decision boundaries are represented by $n_I - 1$ dimensional hyperplanes in $\mathbb{R}^{n_I}$. Each hyperplane p_i is parameterized by two variables $o_i \in \mathbb{R}^{n_I}$ and $d_i \in \mathbb{R}^{n_I}$, where o_i and d_i describe the origin and unit normal direction, respectively, of the ith hyperplane. Therefore, the hyperplanes are defined as $p_i := \{x_I \mid (x_I - o_i)^\mathsf{T} d_i = 0\}$. The set of all hyperplanes is $P = \{p_1,\ldots,p_\eta\}$, where η is the total number of hyperplanes. The boundary that was last intersected by x_I is encoded with the boundary state variable $\beta \in \mathcal{B}$, where $\mathcal{B} := \{0,1,\ldots,\eta\}$. Initially, when the internal state has not yet intersected a boundary $\beta = 0$.

In order to ensure that the system can indeed learn how to locomote, we need to impose some constraints on the set of controllers and boundaries. In particular, we need to be able to guarantee that a control law is always applicable. This means the system can always move away from a boundary once the boundary has been encountered. Also, we want to ensure the system will always eventually encounter a boundary.

To establish this guarantee, we first assume that the hyperplanes intersect to form a convex polytope. To describe this constraint more formally, let

$$\bar{D} := \left\{ x_I \;\middle|\; (x_I - o_i)^\mathsf{T} d_i < 0\ \forall i \in \{1,\ldots,\eta\} \right\}. \tag{2}$$

The set $\bar{D}$ is the set of all points inside the polytope formed by the intersection of the hyperplanes in P. Thus, the constraint is that $\bar{D}$ must be convex. This constraint also gives a minimum to the number of hyperplanes needed, i.e., $\eta_{min} = n_I + 1$.

The set of primitive controllers move the internal state of the system around in the polytope defined by $\bar{D}$. A set is valid if for each point along all of the hyperplanes in P there is at least one control action that can move the state away from hyperplanes and back into $\bar{D}$. Thus, we assume that the boundary conditions and control laws have been designed such that

$$\forall i \in \{1,\ldots,\eta\},\ \exists j \in \mathcal{E}\ s.t.\ d_i^\mathsf{T} f_j(x) < 0\ \forall x \in p_i. \tag{3}$$

We also impose a non-transversality condition on the primitive controllers and the decision boundaries. This condition restricts the internal state to not move along the decision boundaries. Two important effects are caused by this condition. The first is that the internal state can not return to the same boundary without first encountering another boundary and the second is that the internal state can not stay in the interior of $\bar{D}$ forever.

Verification of these effects can be seen by looking at the trajectories of the primitive controllers when initialized at different points along the boundaries. By continuity, these

[1] This bound can be generalized to any polytopic boundary.

[2] The assumption is valid because this equates to the low level motor controllers that are usually built into the hardware of a robotic system.

trajectories can never cross each other and, therefore, if a controller brings a state back to the same boundary then there must be a stationary, singular point on that boundary. This would violate the non-transversality condition in (3), thus the primitive controllers will never bring the internal state back to a boundary that is has just encountered. The second effect is verified by a similar reasoning as the first. For the internal state to stay in the interior of the boundaries forever with the same primitive controller there must be a point along its trajectory that is tangential to the hyperplanes that make up the boundaries. Again, this violates the non-transversality condition in (3); and as result, the internal state will always eventually encounter a decision boundary.

In addition, in order to keep the notation simple it is assumed that the internal state will not encounter more than one decision boundary at a time. This assumption is reasonable because, for all but contrived systems, it is improbable for the internal state to intersect more than one hyperplane due to the non-transversality condition, which prevents the potentially, non-pathological sliding along a boundary from happening.

C. Hybrid Formulation

Following the notation in [17], our hybrid system is composed of four parts: (i) the flow map, which describes the continuous time evolution of the system; (ii) the flow set, which determines when the flow map takes place; (iii) the jump map, which describes the discrete time updates to the system; and (iv) the jump set, which determines when the jump map takes place. The interpretation is that the system flows during the continuous time evolution and jumps at the discrete time updates. With this we combine all the system information into a generalized state variable $q := [x, \beta, \xi]^\mathsf{T} \in \mathcal{Q}$, where $\mathcal{Q} = \mathbb{R}^n \times \mathcal{B} \times \mathcal{E}$.

The flow set is defined with the bounds on x_I,

$$C = \{q \in \mathcal{Q} \mid x_{I_{min_i}} \leq q_i \leq x_{I_{max_i}},\ i \in \{1, \dots, n_I\}\}. \tag{4}$$

The jump set is thought as the complement to $\bar{D}$,

$$D := \left\{ q \in \mathcal{Q} \;\middle|\; [I^{n_I \times n_I}\ 0^{n_I \times (n_E+2)}] q \notin \bar{D} \right\}. \tag{5}$$

In words, (5) states that the jump set is the set of q's where the first n_I components of q are not elements of $\bar{D}$. An illustration of an example internal state-space with boundaries, flow set, and jump set can be seen in Figure 2.

Before defining the flow map and jump map for the system two other functions must first be introduced. The first is the boundary map $b(x_I) : \mathbb{R}^{n_I} \to \mathcal{B}$, which maps the internal state to a boundary state. Second, there is the controller selector map $e(\beta) : \mathcal{B} \to \mathcal{E}$, which selects which controller to use given a boundary state. The controller selector map for a given boundary must satisfy the condition in (3).

With the above functions the flow map and jump map of the hybrid system can be written as

$$f_\mathcal{H}(q, u) = \begin{bmatrix} f(x, u), & 0, & 0 \end{bmatrix}^\mathsf{T} \tag{6}$$

$$g_\mathcal{H}(q) = \begin{bmatrix} x, & b(x_I), & e(b(x_I)) \end{bmatrix}^\mathsf{T} \tag{7}$$

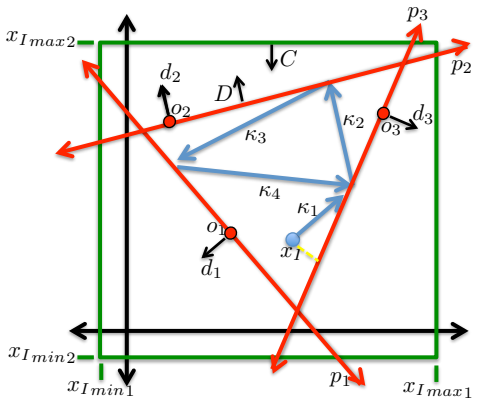

Fig. 2. Illustration of the internal state $x_I \in \mathbb{R}^2$ being controlled into a limit cycle with controllers $\kappa_1, \dots, \kappa_4$ and with the location of the decision boundaries defined by the parameters o_i and d_i. The flow set C is the interior of the green box and the jump set D is the exterior of the red triangle. The interior of the red triangle must be convex.

respectively. Thus, $f_\mathcal{H}(q, u) : C \times \mathbb{R}^m \to \mathcal{Q}$ and $g_\mathcal{H}(q) : D \to \mathcal{Q}$. Finally, the hybrid system is defined as

$$\mathcal{H} : \begin{cases} \dot{q} = f_\mathcal{H}(q, u) & q \in C \\ q^+ \in g_\mathcal{H}(q) & q \in D \end{cases}. \tag{8}$$

D. Reward Function

Learning only makes sense if there is something to learn. To this end, we need to associate a reward function and value function to the system, which is usually determined trivially by what is desirable (e.g. if forward progress is desirable then distance forward is the reward). We let the reward function be $R(x_E, u) : \mathbb{R}^{n_E} \times \mathbb{R}^m \to \mathbb{R}$ and the corresponding value function becomes

$$V(x_E, u) = \int_{t_0}^{t_0 + t_\pi} R(x_E, u) dt, \tag{9}$$

where t_0 is the initial time and t_π is length of time that is being optimized over.

Given the above definitions, the objective is to maximize the value function, $V(x_E, u)$, without explicit knowledge of the system dynamics, $f(x, u)$, and the controllers, $\kappa_i(x_I, \tau)$, by learning the controller selector map, $e(\beta)$, and the set of boundaries, P. Two things to note are (i) the primitive controllers depend on x_I and not on x_E and (ii) that the reward and value functions depend on x_E and not x_I, which is why we refer to this as a locomoting problem.

III. Learning Algorithm

Learning the controller actions and the decision boundaries are the main focus of this section and we primarily use *reinforcement learning* to this end. This type of learning is often done as an online process, which adds the additional caveat that the agent must decide when it has sufficiently learned the environment and start utilizing its knowledge. This is known as "exploratation vs. exploitation" [18].

Reinforcement learning is usually modeled as a Markov Decision Process (MDP) [7] with four components: $\mathcal{S}$, $\mathcal{A}$,

$\mathcal{P}$, and $\mathcal{R}$. $\mathcal{S}$ is the set of states for the agent and the environment. $\mathcal{A}$ is the set of actions or decisions that agent can take. $\mathcal{P}$ is a function that defines the probabilities of transitioning from the current state to the next state given a certain action. $\mathcal{R}$ is the function that determines the reward that is received after choosing a action from a given state.

With reinforcement learning the agent is attempting to learn an optimal policy, π, for the MDP, which is a description of how the agent chooses the actions to perform given a certain state. To do this the agents often learns the value function, which in turn will produce a policy. The value function, V, gives the maximum reward that can be earned from a given state.

For the system framework presented in Section II the learning algorithm components are $\mathcal{S} = \mathcal{B}$, $\mathcal{A} = \mathcal{E}$, $\mathcal{R} = R(x_E, u)$, and $\mathcal{P}$ is not explicitly used.

A. Learning Controller Actions

A type of reinforcement learning known as Q-learning was one of the most important breakthroughs in the field of reinforcement learning [7]. Q-learning is an iterative update algorithm for the value-action function, Q; hence the name. The value-action function is a variant of the value function, which gives the maximum reward that can be earned from a given state after performing a given action. A model that maps from actions to states is not needed for either the learning or the action selection in Q-learning. For this reason, Q-learning is called a *model-free* method. This learning algorithm is guaranteed to converge to the optimal, Q^*, if all state-action pairs continue to be updated. Q-learning is used to learn controller actions given decision boundaries for the hybrid system defined in (8).

Q-learning is a remarkably simple algorithm. The update is as follows:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(R_t + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right), \quad (10)$$

where s_t is the boundary state (β) and a_t is the primitive controller (ξ) at time t. In (10), α is known as the learning rate and γ is known as the discount factor. A smaller α means old information will be trusted more than new information. And a smaller γ means instantaneous rewards are more important than future rewards. From (10) a policy, π, is generated from Q in a "greedy" manner. In other words, the action that is selected for a given state is the one that maximizes the Q value for that state.

B. Learning Decision Boundaries

A different approach is used to find the optimal decision boundary locations. A gradient ascent algorithm is used to iteratively move the boundaries to the optimal locations. The boundaries are moved proportional to the positive of the gradient of the value function with respect to the boundary locations, $\frac{\partial V}{\partial p_i}$. This proportion (or step size) is set by the parameter c.

To estimate the gradient, each time the internal state x_I reaches one of the boundaries in P the position of that boundary is randomly changed by some small amount, Δp. The change in the reward function, ΔR, is calculated over this change in the boundary position. The ratio of ΔR to Δp is used as an approximation for the gradient $\frac{\partial V}{\partial p_i}$. The boundaries are moved by the amount equal to $c\frac{\Delta R}{\Delta p}$. This results in the boundaries moving in a "greedy" direction; in other words, a direction that maximizes the short term reward not necessarily the long term value.

Algorithm 1 Learning To Locomote Algorithm

```
x ← 0 {Initialize state to all zeros}
Q ← 0 {Initialize Q to all zeros}
while x_I is not a stable limit cycle do
  if variance on Q(β, ξ) > v_th then
    ξ ← rand(E) {Pick random action to use}
  else
    ξ ← argmax(Q(β, :)) {Pick maximizing action}
  end if
  x ← simulate(x) {Simulate the system forward until
  boundary is encountered}
  β ← b(x) {Update the boundary state}
  Q ← update(Q, q) {Update Q with (10)}
  p_i ← c ∂R/∂p_i {Update boundary locations}
end while
for each β ∈ B do
  e(β) ← argmax(Q(β, :))
end for
```

C. Exploration vs. Exploitation

Deciding when the agents have learned sufficient information and deciding when to begin executing the learned policy is a current area of research in reinforcement learning. Knowing that Q-learning is the learning process helps determine the transition from exploration to exploitation. In Q-learning the Q values will converge to the optimal value if the state-action pairs continue to be updated. Thus, the variance in the Q values will converge to zero. The variance in the Q values is used to determine when the learning algorithm should explore or exploit.

Exploitation takes place when the maximum variance in the last σ_n^2 updates of Q for a particular state, s, is below a threshold of σ_{th}^2. Otherwise exploration is performed. This variance for a state is denoted as $\sigma^2(s)$. During exploration actions are picked randomly with a distribution that is proportional to $\sigma^2(s)$. If a state-action pair in Q has not been updated more that σ_n^2 times the variance is set to a large number, σ_{inf}^2. This method assures that exploitation will not take place until each state-action pair has been tried σ_n^2 times and that the variance on the estimated Q values for each state-action pair is below σ_{th}^2.

IV. Examples

The efficacy of the learning algorithm from Section III is demonstrate on two systems in this section. One system is a simulated system and one is a real robotic system.

A. Example Simulated System

We demonstrate the ability of our learning algorithm on the nonholonomic integrator [20] known as "Brockett's system" [21]. Brockett's system is an ideal example system to test and demonstrate the learning algorithm outlined in Section III because it is one of the simplest systems that fits the model defined in Section II. In addition, due to the so-called topological obstruction there is no continuous control law to stabilize Brockett's system [21]. The dynamics of Brockett's system are

$$f(x) = [u_1,\ u_2,\ x_{I1}u_2 - x_{I2}u_1]^\mathsf{T}, \tag{11}$$

where $u \in \mathbb{R}^2$, $x_I \in \mathbb{R}^2$, and $x_E \in \mathbb{R}$. Brockett's system is a surprisingly rich system given its innocuous appearance.

For this system, we define the primitive controllers such that they move the internal state, x_I, with unit magnitude. The direction for each controller is random and is drawn from eight uniform random distributions. The domain of these distributions are each equal to one-eighth partitions of the unit circle, which guarantees that the condition in (3) is satisfied. We use 32 controllers and set the bounds on x_I as $x_{I\,min} = [-1, -1]^\mathsf{T}$ and $x_{I\,max} = [1, 1]^\mathsf{T}$.

Lastly, we select four boundaries ($\eta = 4$) for the learning algorithm. The directions of the boundaries are fixed to $d_1 = 0$, $d_2 = \pi/2$, $d_3 = \pi$ and $d_4 = 3\pi/2$. The origin's of the boundaries, o_i, are chosen randomly such the constraint that $\bar{D}$ is convex is satisfied.

From (11) it is seen that for this example system $m = 2$, $n_I = 2$, $n_E = 1$, $n = 3$, $\eta = 4$, and Q is a 4×32 matrix. The reward function is defined as $R(x_E, u) = \frac{dx_E}{dt}$. The parameters used for the learning algorithm were found empirically and are as follows: $\alpha = 0.75$, $\gamma = 0.25$, $c = 0.1$, $\sigma_n^2 = 3$, $\sigma_{th}^2 = 0.025^2$, and $\sigma_{inf}^2 = 10^2$. To simulate the system the algorithm shown in Algorithm 1 is executed in Matlab.

Using this learning algorithm we were able to learn the optimal control sequence and boundary locations given the setup described above. Results of the learning algorithm are shown in Figure 3. It is known that the optimal continuous controller for Brockett's system is sinusoidal of the form

$$\begin{bmatrix} u_1(t) \\ u_2(t) \end{bmatrix} = \begin{bmatrix} \cos(\lambda t) & -\sin(\lambda t) \\ \sin(\lambda t) & \cos(\lambda t) \end{bmatrix} \begin{bmatrix} u_1(0) \\ u_2(0) \end{bmatrix}, \tag{12}$$

where λ and $u(0)$ can be solved for given initial and desired final states of the system [21]. We compared the external state value after running both the optimal control law in (12) and the control law learned with our learning algorithm. After running 100 trials the average value computed from (9) for the learned controller is $90.3\%(\pm 3.8\%)$ of the value computed using optimal controller. If we use 64 primitive controllers and 8 boundaries the average increases to $96.1\%(\pm 2.7\%)$. The results of an experiment using the learning algorithm (described in Section III) on the Brockett's system are shown in Figure 3.

In the experiment shown in Figure 3 the states are all initialized to zero. It can be seen that during the first portion of this experiment (time 0 to 220) the learning algorithm

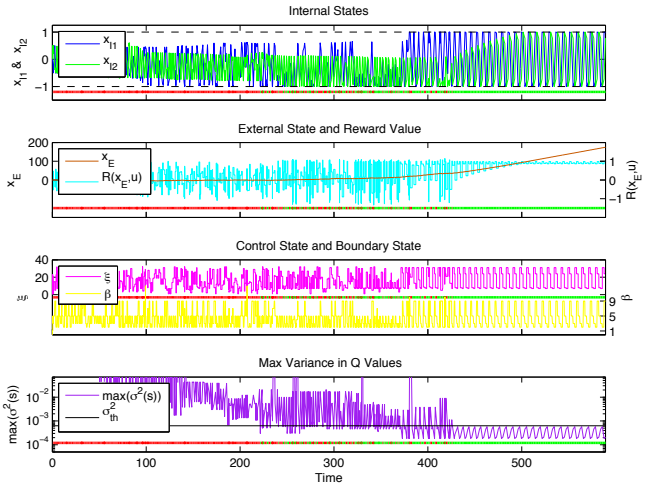

Fig. 3. These plots show the results of running the learning algorithm on Brockett's system. Top: plot of the internal state components, x_{I1} (dark blue line) and x_{I2} (light green line), verses time. The internal state constraints are also shown (black dashed lines). Second: plot of the external state and reward, x_E (dark orange line) and $R(x_E, u)$ (light blue line), respectively, verses time. Third: plot of the control state and boundary state, ξ (dark pink line) and β (light yellow line), respectively, verses time. Bottom: plot of the maximum variance in the Q values for a given state, $max(\sigma^2(s))$ (dark purple line), verses time. The variance threshold for deciding between exploration and exploitation is also shown (solid black line). In all the plots it is shown when the learning algorithm is exploring or exploiting with the dark red and light green marks, respectively. The boundary locations are implicitly shown in the top plot by the envelope of the internal states

is only exploring. During exploration the control state is random and the boundaries locations are moving but not in a consistent direction. In addition, the external state and reward value have an average output of zero. The first time exploitation takes place is when the time equals 220, which can be seen in the bottom plot because $max(\sigma^2(s))$ falls below σ_{th}^2. Exploration and exploitation trade off for the middle portion of the experiment (time 220 to 420). In the last portion of the experiment only exploitation takes place (time 420 to 600). The control state and boundary state settle into a repeating pattern and the value of x_E quickly increases. Note that even after the time when only exploitation is taking place (time of 420) the boundaries are still moving. The boundaries are moving towards positions that give maximum reward. The boundaries settle at the limits of x_I and the average reward value stops increasing.

B. Physical Robotic System

In addition to running our learning algorithm on a simulated system, we tested the algorithm on a physical robotic system. A photograph of this robot is shown in Figure 1. This robot consists of a body and two moveable appendages. Each appendage has only one rotational degree of freedom, which limits the robot to moving along a straight line. The objective of the robot is to move as far along that line as possible. The robot must "scoot" because it is impossible for the appendages to lose contact with the ground. Locomotion is only possible with differences in frictional forces from

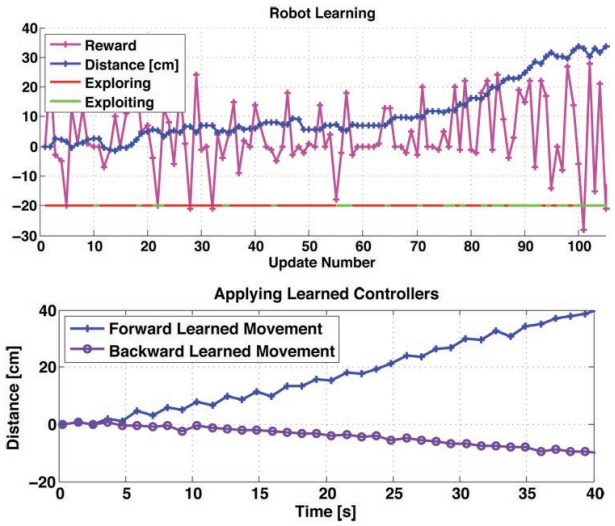

Fig. 4. (Top) The first plot shows the physical robot learning to move forward. The forward progress is shown (dark blue line with plus sign markers) as well as when the robot is exploring (dark red line) and exploiting (light green line). (Bottom) The second plot shows the robots movements after learning how to move both forward (blue line with plus sign marks) and backwards (purple line with circle marks).

the two appendages and the ground. This complexity makes conceptualizing the necessary sequence of movements for forward progress difficult and not intuitive.

Instead of attempting to design the sequence of actuations for the robot, it learns them with our learning algorithm. The internal state, x_I, are the positions of its two appendages and the external state, x_E, is the distance from the origin. The primitive controllers are all possible combinations of moving the appendages up, down, and not at all. However, the case when both are not moving is not included as a primitive controller. This makes eight possibilities, therefore $k = 8$. The boundaries are the same as in the simulated example (Section IV-A), thus $\eta = 4$.

Using the learning algorithm presented in Section III, the robot is able to learn the necessary movements to move forward and backward. After running the learning algorithm several times, we observe that the robot learns to move forward at approximately 1 cm/s and backwards at approximates 0.25 cm/s. Plots of the robot's movements are shown in Figure 4 [3].

V. Conclusion

This paper introduced a new algorithm to learn how to switch between primitive controllers to maximize an objective function for a particular type of hybrid system. The controllers act directly on what is referred to as the internal state, x_I, and the objective function is based only on what is referred to as the external state, x_E. Switching of controllers occurs at boundary conditions, which reside only in x_I's state-space. Learning which controller to use at each boundary is achieved with Q-learning and the boundary locations are learned through gradient ascent. The capacity of this learning algorithm is demonstrated on both a simulated system and on a real robot. By learning boundary-action pairs this algorithm mitigates the "curse of dimensionality".

[3] A movie of the robot learning can be viewed at http://gritslab.gatech.edu/home/2012/11/learning-to-locomote/.

References

[1] J. A. Bagnell and J. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1615–1620, 2001.

[2] N. Kohl and P. Stone, "Policy gradient reinforcement learning for fast quadrupedal locomotion," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2619–2624, 2004.

[3] T. Hester, M. Quinlan, and P. Stone, "Generalized model learning for reinforcement learning on a humanoid robot," in *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2369–2374, IEEE, 2010.

[4] F. Stulp, E. Theodorou, M. Kalakrishnan, P. Pastor, L. Righetti, and S. Schaal, "Learning motion primitive goals for robust manipulation," in *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 325–331, IEEE, 2011.

[5] S. Sastry and L. Crawford, "Learning Controllers for Complex Behavioral Systems," tech. rep., University of California at Berkeley, 1996.

[6] J. Z. Kolter, Z. Jackowski, and R. Tedrake, "Design, analysis and learning control of a fully actuated micro wind turbine," in *Proceedings of the 2012 American Control Conference (ACC)*, June 2012.

[7] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning, The MIT Press, the mit press ed., Mar. 1998.

[8] E. Theodorou, J. Buchli, and S. Schaal, "A Generalized Path Integral Control Approach to Reinforcement Learning," *The Journal of Machine Learning Research*, vol. 11, pp. 3137–3181, Mar. 2010.

[9] F. Kuo and I. Sloan, "Lifting the curse of dimensionality," *Notices of the AMS*, vol. 52, no. 11, pp. 1320–1328, 2005.

[10] R. Zoppoli, M. Sanguineti, and T. Parisini, "Can we cope with the curse of dimensionality in optimal control by using neural approximators?," in *Proceedings of the 40th IEEE Conference on Decision and Control*, pp. 3540–3545, IEEE, 2001.

[11] A. Hinneburg and D. Keim, "Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering," *in Proceedings of 25th International Conference on Very Large Data Bases, VLDB*, pp. 506–517, 1999.

[12] W. McEneaney, "Curse-of-dimensionality free method for Bellman PDEs with Hamiltonian written as maximum of quadratic forms," in *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference (CDC-ECC).*, pp. 42–47, IEEE, 2005.

[13] F. Mussa-Ivaldi, S. Giszter, and E. Bizzi, "Linear combinations of primitives in vertebrate motor control," *Proceedings of the National Academy of Sciences*, vol. 91, no. 16, p. 7534, 1994.

[14] R. Beer, R. Quinn, H. Chiel, and R. Ritzmann, "Biologically inspired approaches to robotics: What can we learn from insects?," *Communications of the ACM*, vol. 40, no. 3, pp. 30–38, 1997.

[15] K. Thoroughman and R. Shadmehr, "Learning of action through adaptive combination of motor primitives," *Nature*, vol. 407, no. 6805, p. 742, 2000.

[16] J. Kober and J. Peters, "Imitation and Reinforcement Learning," *Robotics & Automation Magazine, IEEE*, vol. 17, no. 2, pp. 55–62, 2010.

[17] R. Goebel, R. Sanfelice, and A. Teel, "Hybrid dynamical systems," *IEEE Control Systems Magazine*, vol. 29, pp. 28–93, Apr. 2009.

[18] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *CoRR*, vol. cs.AI/9605103, 1996.

[19] S. Thrun, "Efficient Exploration in Reinforcement Learning," tech. rep., School of Computer Science Carnegie Mellon University, Pittsburgh, Pennsylvania, Jan. 1992.

[20] R. Brockett, "Asymptotic Stability and Feedback Stabilization," *Differential Geometric Control Theory*, pp. 181–191, 1983.

[21] S. Sastry, *Nonlinear systems*. analysis, stability, and control, Springer Verlag, June 1999.