Low-Dimensional Learning for Complex Robots

Rowland O'Flaherty, *Student Member, IEEE,* and Magnus Egerstedt, *Fellow, IEEE*

***Abstract*—This paper presents an algorithm for learning the switching policy and the boundaries conditions between primitive controllers that maximize the translational movements of a complex locomoting system. The algorithm learns an optimal action for each boundary condition instead of one for each discretized state-action pair of the system, as is typically done in machine learning. The system is model as a hybrid system because it contains both discrete and continuous dynamics. With this hybridification of the system and with this abstraction of learning boundary-action pairs, the "curse of dimensionality" is mitigated. The effectiveness of this 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.**

Note to Practitioners: ***Abstract*—As technological innovation in the field of robotics continues to advance forward at a steady beat, so does the complexity of the robotic systems that are the product of this innovation. Utilizing these robotic systems for a number of automated tasks in diverse situations requires an increased level of deep understanding of the inner workings of the system, which is a burden to the human operator and programmer. To attenuate this burden, the task of programming itself must move towards being automated. This type of automation is the motivation for work presented here. This paper outlines an algorithm for which a complex robotic system can learn the task of locomoting all on its own. The practical application of this is that more complex robotic systems can be incorporated into industry with less effort. This is accomplished with an innovative approach on what the robot needs to learn to achieve its goal. This paper focuses on robot locomotion but future work will focus on high degree of freedom stationary robotic arms that are increasingly seen on factory floors.**

Primary and Secondary Keywords ***Index Terms*—Primary Topics: Learning Control, Hybrid Systems, Robot Motion, Reinforcement Learning, Decision Boundaries**

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 presented algorithm learns actions based on boundary states instead of the actual system states, which greatly reduces the amount of learning that must take place.

To illustrate how this learning algorithm may be used, imagine a situation where a roboticist would like to build a robotic caterpillar, without having to (or even knowing how to) explicitly design the control actions to move the robot forward or backwards. Instead, the roboticist simply "loads" the presented learning algorithm together with a library of primitive feedback controllers onto the robotic caterpillar and sets on its way. On its own, the robotic caterpillar learns to move forward and backwards.

This paper uses *reinforcement learning*, which is a category of machine learning, where an agent learns online how to make a sequence of decisions to maximize some cumulative long term expected reward; typically by interacting with the environment, for example [7]. The scalability of reinforcement learning to high-dimensional continuous state-action systems can be problematic, 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. Reinforcement learning quickly becomes infeasible because its complexity scales linearly in the number of actions and quadratically with the number of states [9].

Previous work has been done to try to mitigate this problem. For example, Kuo et al. [10] discuss different sampling techniques that can be used for numerical integration in high dimensional spaces. In our work, we are not attempting to do numerical integration but face the same problem of feasibly preforming some task in a high dimensional space. Most techniques in dealing with this problem of high dimensionality are tackled by sampling the space in an intelligent fashion. Zoppoli et al. do this with neural approximators [11]. Other techniques use clustering [12] or function approximation [13]. We cope with the problem in a similar fashion as is done in previous work; and that is by sampling the space in an intelligent way. Our technique differs in that it hones in on the most important states of the state space and only worries

R. O'Flaherty (`rowland.oflaherty@gatech.edu`) and M. Egerstedt (`magnus.egerstedt@ece.gatech.edu`) are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA. This work was sponsored by the US National Science Foundation through Grant Number 1329683

about those while ignoring the rest. These important states are the switching boundaries of the hybrid system.

In fact, our 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 [14]–[17]. 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 use of reinforcement learning with the reduced number of boundary-controller pairs presented in this paper closely relates to Iterative Learning Control (ILC). ILC refines the input signal over repeated task iterations so that the output approaches the desired output for all points in the trajectory [18]. ILC in combination with adaptive switching of feedback gains has been used for control of robot manipulators with repetitive tasks [19]–[21]. Our algorithm differs in that it is not switching between different controller gains but completely different controllers. In addition, our algorithm may switch between controllers several times per cycle of the states instead of switching once per cycle or task execution.

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 Fig. 1.

The paper is organized as follows: In Section II, we describe the properties of a hybrid learning systems used for learning. In Section III, we present the learning algorithm. The results for an example simulated system and for the physical robotic system are shown in Section IV. Finally, in Section V we summarize the paper in a brief conclusion.

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

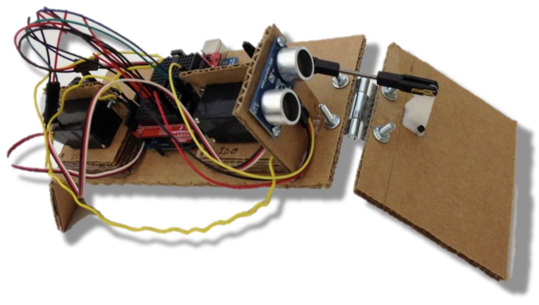


Fig. 1. Physical robotic system used to test the learning algorithm.

In our 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. This notion of a hybrid system differs from other uses of hybrid systems, for example [22]. 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). An alternate way of describing these states is that the internal state needs proprioceptive sensors to measure its value while the external state needs exteroceptive sensors to measure its value.

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

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

In other words, each element in $x_{I_{min}}$ is less than the corresponding element in $x_{I_{max}}$. 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 will always move the internal state of the system

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

until the state encounters 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 $f_\xi(x) := f(x, \kappa_\xi)$. Since, $\kappa_\xi(x_I)$ can be any nonlinear function both the internal state and the external state can be controlled in arbitrary ways with the controller $\kappa_\xi(x_I)$.

Decisions are made on what control law to use when the internal state of the system intersects a decision boundary. This is done to greatly reduce the number of state-action pairs that are used to decide when a new primitive controller needs to be applied. 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

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

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 valid set of primitive controllers is a set such that, for each point along a given hyperplane, there is at least one primitive control action that moves the state away from that hyperplane and back into the convex polytope defined by $\bar{D}$ for all hyperplanes in P. Thus, we assume that the boundary conditions and control laws have been designed such that

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

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 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 it 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 System Formulation

Following the definition and notation in [23], 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,

$$C = \{q \in \mathcal{Q} \mid x_{I_{min\,i}} \le q_i \le x_{I_{max\,i}},\; i \in \{1, \ldots, n_I\}\}. \quad (4)$$

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

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

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 Fig. 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. The second is the controller selector

[2]The assumption is valid because this equates to the motion of low level motor controllers that are usually built into the hardware of a robotic system, which have limited operating range.

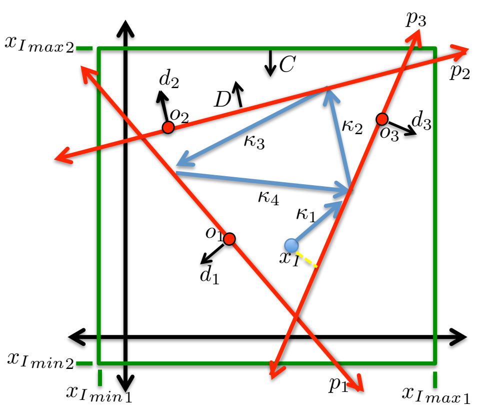


Fig. 2. Illustration of the internal state $x_I \in \mathbb{R}^2$ being controlled into a limit cycle with controllers $\kappa_1, \ \ldots \ , \kappa_4$ and with the location of the decision boundaries defined by the parameters o_i and d_i. The flow set C is the interior of the green box and the jump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

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

$$f_{\mathcal{H}}(q,u) = \left[\begin{array}{ccc} f(x,u), & 0, & 0 \end{array} \right]^{\mathsf{T}} \tag{6}$$

$$g_{\mathcal{H}}(q) = \left[\begin{array}{ccc} x, & b(x_I), & e(b(x_I)) \end{array} \right]^{\mathsf{T}} \tag{7}$$

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.

To relate the above framework to the example given about the robotic caterpillar in Section I, the external state is the position of the caterpillar and the value function is how far forward it has moved. The internal states are the state of its body configurations. These internal states are bounded by how far the body can move back and forth, which defines the flow set. The primitive controllers could be to oscillate the body at different points or with different frequencies. Setting different decision boundaries with different jump maps will cause the body to move fast or slow and in or out of unison with other parts of the body. The goal is to find the setting that makes the robotic caterpillar move forward the "best" (as defined by the value function) by deciding when the given controllers are implemented.

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" [24].

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. A variant of the value function is the value-action function, Q, which gives the maximum reward that can be earned from a given state after performing a given action.

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).

Algorithm 1 Learning To Locomote Algorithm

```
$x \leftarrow 0$ {Initialize state to all zeros}
$Q \leftarrow 0$ {Initialize Q to all zeros}
**while** $x_I$ is not a stable limit cycle **do**
  **if** variance on $Q(\beta, \xi) > v_{th}$ **then**
    $\xi \leftarrow rand(\mathcal{E})$ {Pick random action to use}
  **else**
    $\xi \leftarrow argmax(Q(\beta, :))$ {Pick maximizing action}
  **end if**
  $x \leftarrow simulate(x)$ {Simulate the system forward until boundary is encountered}
  $\beta \leftarrow b(x)$ {Update the boundary state}
  $Q \leftarrow update(Q, q)$ {Update Q with (10)}
  $p_i \leftarrow c\frac{\partial R}{\partial p_i}$ {Update boundary locations}
**end while**
**for** each $\beta \in \mathcal{B}$ **do**
  $e(\beta) \leftarrow argmax(Q(\beta, :))$
**end for**
```

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

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

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. Fundamentally, equation (10) states that the update Q for state s and action a will be the old Q for that pair plus a scaled sum of the instantaneous reward and the discounted maximum value that is currently known for the next state. 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.

The orientation of the boundaries, d_i, are currently chosen in one of two ways. They are either chosen to be aligned with the axes of the state space or chosen randomly but with the constraint that the interior of the boundaries form a convex polytope.

The number of decision boundaries, η, can be no lower than $\eta_{min} = n_i + 1$, but there is no upper limit on η. As η increases a larger number of decisions are made for each cycle of the interior state, thus the better the results, as is shown in the results in IV-A. The improved results come at cost in convergence times, which scale as $O(|\mathcal{B}| \sum_{\beta \in \mathcal{B}} |\mathcal{E}(\beta)|) = O(\eta^2 k)$ [9].

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. Exploration strategies are usually grouped into two categories: *undirected* and *directed* [25]. Undirected techniques use no knowledge of the learning process and ensure exploration by merging randomness into the action selection. Directed techniques utilize knowledge of the learning process to preform exploration in more directed manner.

For the research outlined in this paper the *directed* techniques are utilized to determine when the transition from exploration and exploitation takes place. This is done by knowing that the learning process is Q-learning, which guarantees that 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 attempted 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 three systems in this section. Two system are simulated systems and one is a real robotic system.

A. Example Simulated System

We demonstrate the ability of our learning algorithm on the nonholonomic integrator [26] known as "Brockett's system" [27]. 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

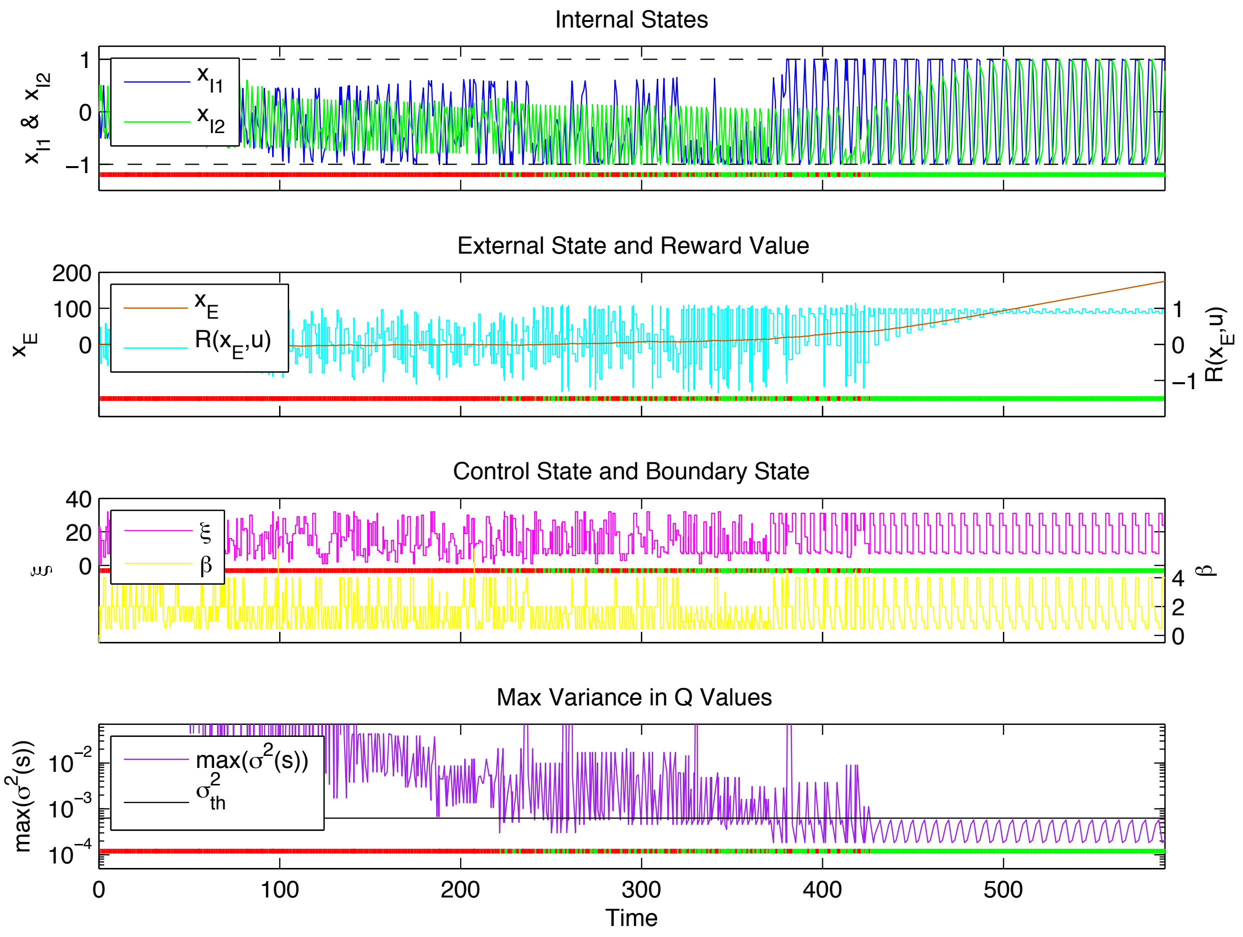


Fig. 3. These plots show the results of running the learning algorithm on Brockett's system. Top: plot of the internal state components, x_{I1} (dark blue line) and x_{I2} (light green line), verses time. The internal state constraints are also shown (black dashed lines). Second: plot of the external 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.

topological obstruction there is no continuous control law to stabilize Brockett's system [27]. 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 Fig. 3 and explained further in Section IV-B. 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 [27]. We compared the external state value after running both the optimal control law in (12) and the control law learned with our learning algorithm. The results after running 100 trials for two different cases are shown in Table I.

TABLE I

External state with different parameters using 100 trials of our learning compared against the optimal state value.

η	k	avg. # iterations	avg. % of optimal
4	32	420	90.3%(±3.8%)
8	64	3377	96.1%(±2.7%)

B. Explanation of Results For Simulated System

The results of an experiment using the learning algorithm (described in Section III) on the Brockett's system (described in Section IV-A) are shown in Fig. 3. The top plot shows the internal state components, x_{I1} (dark blue line) and x_{I2} (light green line), as well as the state constraints (black dashed lines). The second plot shows the external state x_E (dark orange line) and the value of the reward function (light blue line). The third plot shows the controller state ξ (dark pink line) and boundary state β (light yellow line). And the bottom plot shows the maximum variance in the Q values, $max(\sigma^2(s))$ (dark purple line), as well as the threshold value for when exploitation takes place, σ^2_{th} (black line). The times when the learning algorithm is exploring (dark red marks) and exploiting (light green marks) are shown in each of the four plots. The boundary locations are implicitly shown in the top plot by the envelope of the internal states.

In this experiment 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 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 approximately equals 220, which can be seen in the bottom plot because $max(\sigma^2(s))$ falls below σ^2_{th}. 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 (approx. 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.

C. Simulated High Dimensional System

Another simulated system was constructed to demonstrate the learning algorithm's capabilities on a system of high dimension. The dynamics of this system were created to resemble that of an N-jointed serpent swimming through water (see Fig. 4). The serpent is constrained to move in the xy-plane. Its "head" is at the position (x, y) and it is always oriented along the x-axis. The serpent has N movable body parts or links. The ith link is of length l_i and begins with a 1-dimensional rotational joint with angle θ_i and angular velocity $\dot{\theta}_i$. It has a mass, m_i, at the center of the link, and a fin with an associated constant c_i. Rotating the joint causes a force,

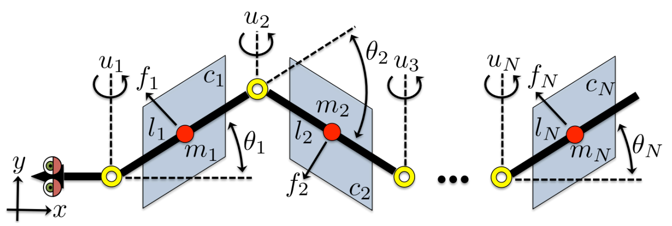


Fig. 4. An illustration of the N-link high dimensional simulate serpent system.

f_i, to be applied to the mass that is proportional to c_i and the square of the angular velocity of joint, $\dot{\theta}_i^2$. The direction of the force f_i is the negative of the direction of the linear velocity of the mass m_i. The acceleration of the serpent is equal to the sum of the forces divided by the sum of the masses. The serpent has direct control of the angular velocity of each of its N joints. Thus, the input, u, to the system is an element of $\mathbb{R}^N$.

The goal of the serpent is to learn to move its joints in such a way that it moves as quickly as possible in the x-direction with no concern for its movement in the y-direction. The serpent has set of k primitive controllers to use that it innately knows and can use accomplish this goal.

Since the serpent resembles a serial link robotic manipulator the kinematics of serpent are solved in a similar fashion to that of an N-link robotic manipulator [28]. In other words, given the serpent's current joint angles and angular velocities the positions, p, and velocities, $\dot{p}$, of the masses of all the links can be computed. To compute the positions and velocities of the masses given the current state of the serpent let us define the forward kinematics function $FK(x, \dot{x}, y, \dot{y}, \theta, u) = [p, \dot{p}]$.

Given the forward kinematics the dynamics of the serpent can be written as

$$\ddot{x} = \frac{\sum_{i=1}^{N} -c_i \dot{p}_{xi}^2}{\sum_{i=1}^{N} m_i}, \ \ddot{y} = \frac{\sum_{i=1}^{N} -c_i \dot{p}_{yi}^2}{\sum_{i=1}^{N} m_i}, \tag{13}$$

where $\dot{p}_{xi}$ and $\dot{p}_{yi}$ are the speed of m_i in the x-direction and y-direction, respectively. From this the internal state is defined as $x_I = \theta$, the external state is defined as $x_E = [x, y]^\mathsf{T}$ (with some abuse of notation for x).

The boundaries for the learning algorithm have fixed orientations such that they form a "hyper rectangular prism" in the internal state space. This means there are $2N$ boundaries. The k primitive controllers for the serpent are chosen randomly from elements in $\mathbb{R}^N$, with the constraint that there is always at least one controller with a positive sign and another controller with a negative sign for each of the N elements. This guarantees that the non-transversality condition in (3) is satisfied. It also means $k \geq 2N$.

Relating the serpent dynamics of (13) to the hybrid system of (8) we have $m = N$, $n_I = N$, $n_E = 2$, $n = 2N + 2$, $\eta = 2N$, and $k \geq \eta$. The bounds on x_I as $x_{Imin} = [-\pi, ..., -\pi]^\mathsf{T}$ and $x_{Imax} = [\pi, ..., \pi]^\mathsf{T}$ and reward function is defined as $R(x_E, u) = \dot{x}$.

Lastly, the authors are well aware that the dynamics as they are written in (13) are not an accurate representation

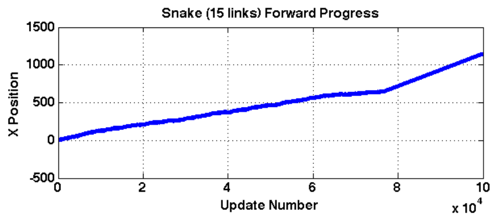


Fig. 5. Forward movement verse update number of a 15-link serpent simulated system learning to move forward. After 8×10^4 updates the system has entered a stable limit cycle and moves forward at the maximum rate.

of the actual dynamics for a N-link swimming serpent. The contribution of this section is not to present high fidelity serpent dynamics but to demonstrate the capabilities of the learning algorithm on a high dimensional system.

The learning algorithm was applied to a 15-link serpent system with 315 primitive controllers. It would be infeasible to discretize the internal state space of this system and learn state-action pairs for each possible combination. Even just discretizing each internal state dimension into 2 parts would create over 1 million state-action pairs to test and learn. With our learning algorithm using 30 boundaries for this system creates 9450 boundary-action pairs to test and learn. The results in Fig. 5 show that the system reaches a steady limit cycle with the external state increasing at a steady maximum rate after approximately 8×10^4 updates.

D. 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 Fig. 1. The fabrication of this robot was for the sole purpose of testing the learning algorithm. This robot consists of a body and two movable appendages. Each appendage has only one rotational degree of freedom. The body of the robot holds an Arduino microcontroller, two servos, ultrasonic distance sensor, XBee wireless transmitter, and two 9V batteries. To maintain a low cost and rapid construction time the entire structure of the robot is made out of cardboard, which is held together with hot glue and two small cabinet hinges. The learning algorithm does not run directly on the hardware of the robot but runs on a separate computer and communicates with the robot over a wireless link. The computer sends actuation commands wirelessly to the robot and the robot sends servo positions and distance readings back to the computer.

This robot is limited to moving along a straight line and the objective of the robot is to move as far along that line as possible. It is impossible for the appendages of the robot to lose contact with the ground, thus for the robot to move it must "scoot". Locomotion is only possible with differences in frictional forces from 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 $\eta = 4$.

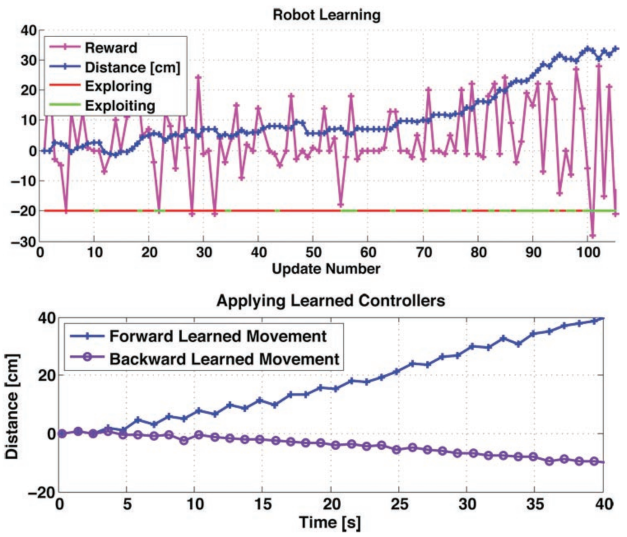


Fig. 6. (Top) The first plot shows the physical robot learning to move forward. The forward progress is shown (dark blue line with plus sign markers) as well as when the robot is exploring (dark red line) and exploiting (light green line). (Bottom) The second plot shows the robots movements after learning how to move both forward (blue line with plus sign marks) and backwards (purple line with circle marks).

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 by positioning the forward appendage at a large angle relative to the ground and proceeding to move the back appendage back-and-forth. This causes the robot to scoot forward at approximately 1 cm/s. To move backward, the robot lays its front appendage down flat and uses the back appendage to pull itself backward. It moves backwards at roughly 0.25 cm/s. Plots of the robot's movements are shown in Fig. 6 [3].

V. Conclusion

This paper has introduced a new algorithm to learn how to maximize an objective function of a complex locomoting system by learning to switch between primitive controllers. The system is described as hybrid system because it contains both continuous time and discrete time dynamics due to the switching between primitive controller actions. The hybrid system formulation allows for a concise representation of the system and the switch boundaries. The primitive controllers act directly on what is referred to as the internal state and the objective function is based only on what is referred to as the

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

external state. The switching boundaries and primitive controllers are used as state-action pairs for Q-learning algorithm to learn which controller to apply at each boundary encountered by the internal state. The locations of the boundaries are adjusted using a gradient ascent algorithm. The capacity of this learning algorithm is demonstrated on both a simulated system with a known analytical solution, a simulated system of high dimension, and on a real robot complex dynamics due to fictional forces. By learning boundary-action pairs this algorithm mitigates the "curse of dimensionality".

Future work will include an in-depth study of the primitive controllers. Particularly, if there are characteristics of the controllers that will produce better results; measured in terms of convergence rates. In addition, how to improve on the greedy gradient ascent of boundary locations and the arbitrary picking of boundary orientations. Lastly, future work will include applying the algorithm to a larger variety of robotic system that have applications to the real world, particularly manufacturing.

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, "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] S. Koenig and R. G. Simmons, "Complexity Analysis of Real-Time Reinforcement Learning," in *Conference on Artificial Intelligence*, pp. 99–105, 1993.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

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

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

[18] K. L. Moore, Y. Q. Chen, and H.-S. Ahn, "Iterative Learning Control: A Tutorial and Big Picture View," in *Decision and Control, 2006 45th IEEE Conference on*, pp. 2352–2357, 2006.

[19] P. R. Ouyang, W. J. Zhang, and M. M. Gupta, "An adaptive switching learning control method for trajectory tracking of robot manipulators," *Mechatronics*, vol. 16, no. 1, pp. 51–61, 2006.

[20] S. Islam and P. X. Liu, "Adaptive iterative learning control for robot manipulators without using velocity signals," in *Advanced Intelligent Mechatronics (AIM), 2010 IEEE/ASME International Conference on*, pp. 1293–1298, IEEE, July 2010.

[21] P. R. Ouyang and P.-i. Pipatpaibul, "Iterative Learning Control: A Comparison Study," *ASME 2010 International Mechanical Engineering Congress and Exposition*, pp. 939–945, Nov. 2010.

[22] W. J. Zhang, P. R. Ouyang, and Z. H. Sun, "A novel hybridization design principle for intelligent mechatronics systems," in *5th International Conference on Advanced Mechatronics*, (Osaka, Japan), 2010.

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

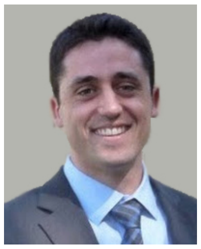
[24] L. Kaelbling, M. Littman, and A. Moore, "Reinforcement learning: A survey," *CoRR*, vol. cs.AI/9605103, 1996.

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

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

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

[28] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot modeling and control*. John Wiley & Sons, Inc., 2006.



Rowland O'Flaherty is a robotics Ph.D. candidate at Georgia Institute of Technology. His research focuses on the intersection of control theory and machine learning with applications to robotics. He received a BS and MS degree in Electrical Engineering in 2007 and 2008, respectively, from the University of California, Santa Barbara.

Magnus Egerstedt is the Schlumberger Professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, where he serves as Associate Chair for Research. He conducts research in the areas of control theory and robotics, with particular focus on control and coordination of complex networks, such as multi-robot systems, mobile sensor networks, and cyber-physical systems. Magnus Egerstedt is the director of the Georgia Robotics and Intelligent Systems Laboratory (GRITS Lab), a Fellow of the IEEE, and a recipient of the ECE/GT Outstanding Junior Faculty Member Award, and the U.S. National Science Foundation CAREER Award. He serves as the Deputy Editor-in-Chief for the IEEE Transactions on Network Control Systems.