Continuous Action Reinforcement Learning for Control-Affine Systems with Unknown Dynamics

Aleksandra Faust[*] Peter Ruymgaart[*] Molly Salman[†] Rafael Fierro[‡]
Lydia Tapia[*]

June 24, 2014

Adaptive Motion Planning Research Group Technical Report TR13-002

Abstract

Control of nonlinear systems is challenging in real-time. Decision making, performed many times per second, must ensure system safety. Designing input to perform a task often involves solving a nonlinear system of differential equations, a computationally intensive, if not intractable, problem. This article proposes sampling-based task learning for control-affine nonlinear systems through the combined learning of both state and action-value functions in a model-free approximate value iteration setting with continuous inputs. A quadratic negative definite state-value function implies the existence of a unique maximum of the action-value function at any state. This allows the replacement of the standard greedy policy with a computationally efficient policy approximation that guarantees progression to a goal state without knowledge of the system dynamics. The policy approximation is *consistent*, i.e., it does not depend on the action samples used to calculate it. This method is appropriate for mechanical systems with high-dimensional input spaces and unknown dynamics performing constraint-balancing tasks. We verify it both in simulation and experimentally for a UAV carrying a suspended load, and in simulation, for the rendezvous of heterogeneous robots.

Keywords: Reinforcement learning, policy approximation, approximate value iteration, fitted value iteration, continuous action spaces, control-affine nonlinear systems

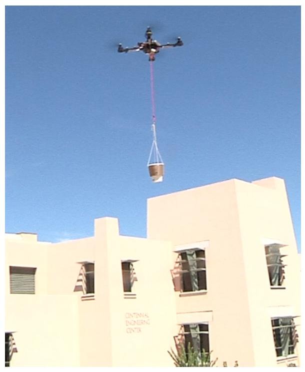
1 Introduction

Humans increasingly rely on robots to perform tasks. A particular class of tasks that interests us are *constraint-balancing tasks.* These tasks have one goal state and opposing constraining preferences on the system. Balancing the speed and the quality of the task are often seen as two opposing

[*]Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, {afaust, apr1248, tapia}@cs.unm.edu

[†]Computer Science Department, Austin College, Sherman, TX 75090, msalman10@austincollege.edu

[‡]Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, 87131-0001, rfierro@ece.unm.edu


(a) Cargo delivery

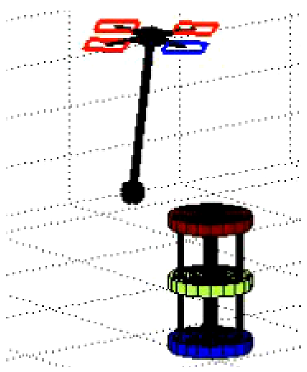

(b) Rendezvous

Figure 1: Evaluated tasks, (a) swing-free cargo delivery and (b) rendezvous.

preferential constraints. For example, the time-sensitive aerial cargo delivery task must deliver suspended load to origin as soon as possible with minimal load displacement along the trajectory (Figure 1a). The rendezvous task (Figure 1b) requires cargo-bearing UAV and a ground robot to meet without a predetermined meeting point. In these tasks the robot must manipulate and interact with its environment, while completing the task in timely manner. This article considers robots as mechanical systems with non-linear control-affine dynamics. Without knowing their exact dynamics, we are interested in producing motions that perform a given *constraint balancing task*.

Control of multi-dimensional nonlinear systems, such as robots and autonomous vehicles must perform decision making many times per second, and must ensure system safety. Yet, designing input to perform a task typically requires system dynamics knowledge. Classical optimal control approaches, use combination of open-loop and closed loop controllers to generate and track trajectories [19]. Another technique, first linearizes the system and then applies LQR methods locally [14]. All classical methods for solving non-linear control problems require knowledge of the system dynamics [14]. On the other hand, we present a solution to an optimal non-linear control problem when the system dynamics is unknown.

Reinforcement learning (RL) solves control of unknown or intractable dynamics by learning from experience and observations. The outcome of the RL is a control policy. Typically the RL learns the value (cost) function and derives a greedy control policy with respect to the value. In continuous spaces, the value function is approximated [5]. When actions are continuous, the greedy policy must be approximated as well. The downside of RL is that its sampling nature renders stability and convergence proofs challenging [5].

We rely on RL, to learn control policy for *constraint-balacing tasks* without knowing the robot's dynamic. Given the continuous state space, *fitted value iteration* (FVI) approximates a value function with a linear map of basis functions [2]. FVI learns the linear map parametrization iteratively in an expectation-maximization manner [5, 9]. The basis function selection is challenging because the learning convergence is sensitive to the selection of the approximation functional space [5]. Here, we select the basis functions to both fit the task and define value function as a Lyapunov candidate function.

We extend FVI, a discrete action RL algorithm, to continuous action space to develop *continuous action fitted value iteration* (CAFVI). The novelty is a joint work with two value functions, state-value and action-value, to learn the control. CAFVI learns, globally to the state space, state-value function, which is negative of the Lyapunov. On the other hand, in the estimation step, it learns an action-value function locally around a state to estimate its maximum. This maximum is found using the newly developed policies that divide-and-conquer the problem by finding the optimal inputs on each axis separately and then combine them. Not only are the policies computationally efficient, scaling linearly with the input's dimensionality, but they produce *consistent* near-optimal input; their outcome does not depend on the input samples used for calculation. Although problem decomposition via individual dimensions is a common technique for dimensionality reduction [25], this article shows that single-component policies lead to a stable system, offers three examples of such policies to turn the equilibrium into an asymptotically stable point, and characterizes systems for which the technique is applicable. The reinforcement learning agent is evaluated on a quadrotor with suspended load and a heterogeneous robot rendezvous task.

From the practical perspective, the article gives methods to implement an FVI with linear map approximation for a constraint-balancing task, on control-affine systems [17] with unknown dynamics and in presence of a bounded drift. These tasks require the system to reach a goal state, while minimizing opposing constraints along the trajectory. The method is fast and easy to implement, rendering an inexpensive tool to attempt before more heavy-handed approaches are attempted.

2 Related Work

Efficient, near-optimal nonlinear system control is an important topic of research both in feedback controls and reinforcement learning. When the system dynamics is known [28] develops adaptive control for interconnected systems. When the system dynamics is not known, optimal [22, 8, 26] and near-optimal [21, 3, 23] control for interconnected nonlinear systems are developed for learning the state-value function using neural networks. This article addresses the same problem, we use linearly parametrized state-value functions with linear regression rather than neural networks for parameter learning. Generalized HJB equation for control-affine systems can be approximately solved with iterative least-squares [**?**]. Our method also learns the value function, which corresponds to the generalized HBJ equation solution, through iterative minimization of the least squares error. However, we learn from samples and linear regression rather than neural networks. For linear unknown systems [13] gives an optimal control using approximate dynamic programming, while we consider nonlinear control-affine systems. Convergence proofs exist for neural network-based approximate value iteration dynamic programming for linear [1] and control-affine systems [7], both with known dynamics. Here, we are concerned with approximate value iteration methods in the reinforcement learning setting - without knowing the system dynamics.

In continuous action RL, the decision making step, which selects an input through a policy, becomes a multivariate optimization. The optimization poses a challenge in the RL setting because the objective function is not known. Robots need to perform input selection many times per second. 50-100 Hz is not unusual [16]. The decision-making challenges brought action selection in continuous spaces to the forefront of current RL research with the main idea that the gradient descent methods find maximums for known, convex value functions [12] and in actor-critic RL [**?**]. Our method is critic-only and because the system dynamics is unknown, the value-function gradient is unavailable [**?**]. Thus, we develop a gradient-free method that divides-and-conquers the problem by finding the optimal input in each direction, and then combines them. Other gradient-free approaches such as Gibbs sampling [15] Monte Carlo methods [18], and sample-averages [**?**] have been tried. Online optimistic sampling planners have been researched [4, 6, 20], [27]. Specifically, *hierarchical optimistic optimization applied to trees* (HOOT) [20], uses hierarchical discretization to progressively narrow the search on the most promising areas of the input space, thus ensuring arbitrary small error.Our methods find a near-optimal action through sample-based interpolation of the objective function and finding the maximum in the closed-form on each axis independently.

Discrete actions FVI has solved the minimal residual oscillations task for a quadrotor with a suspended load and has developed the stability conditions with a discrete action MDP [11]. Empirical validation in [11] shows that the conditions hold. This article characterizes basis vector forms for control-affine systems, defines admissible policies resulting in an asymptotically stable equilibrium, and analytically shows the system stability. The empirical comparison with [11] in Section 4.2 shows that is both faster and performs the task with higher precision. This is because the decision making quality presented here is not limited to the finite action space and is independent of the available samples. We also show wider applicability of the methods developed here by applying them to a multi-agent rendezvous task. Our work currently under submission [**?**], extends [11] to environments with static obstacles specifically for aerial cargo delivery applications, and is concerned with generating trajectories in discrete action spaces along kinematic paths.

3 Methods

This section consists of four parts. First, Section 3.1 specifies the problem formulation for a task on a control-affine system suitable for approximate value iteration with linear basis vectors. Based on the task, the system and the constraints, we develop basis functions and write state-value function in Lyapunov quadratic function form. Second, Section 3.2 develops sample-efficient policies that take the system to the goal and can be used for both planning and learning. Third, Section 3.3 places the policies into FVI setting to present a learning algorithm for the goal-oriented tasks. Together they give practical implementation tools for solving constraint-balancing tasks through reinforcement learning on control-affine systems with unknown dynamics. We discuss these tools in Section 3.4.

3.1 Problem formulation

Consider a discrete time, control-affine system with no disturbances, $\boldsymbol{D} : X \times U \to X$,

$$\boldsymbol{D}: \quad \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k) + \boldsymbol{g}(\boldsymbol{x}_k)\boldsymbol{u}_k. \tag{1}$$

where states are $\boldsymbol{x}_k \in X \subseteq \mathbb{R}^{d_x}$, input is defined on a closed interval around origin, $\boldsymbol{u}_k \in U \subseteq \mathbb{R}^{d_u}$, $d_u \leq d_x$, $\boldsymbol{0} \in U$, and $\boldsymbol{g} : X \to \mathbb{R}^{d_x} \times \mathbb{R}^{d_u}$, $\boldsymbol{g}(\boldsymbol{x}_k)^T = [\boldsymbol{g}_1(\boldsymbol{x}_k) \ ... \ \boldsymbol{g}_{d_u}(\boldsymbol{x}_k)]$ is regular for $\boldsymbol{x}_k \in X \setminus \{\boldsymbol{0}\}$, nonlinear, and Lipschitz continuous. Drift $\boldsymbol{f} : X \to \mathbb{R}^{d_x}$, is nonlinear, and Lipschitz. Assume that the system is controllable [14]. We are interested in autonomously finding control input $\boldsymbol{u}_k$ that takes the system to its origin in a timely-manner while reducing $\|\boldsymbol{A}\boldsymbol{x}\|$ along the trajectory, where $\boldsymbol{A}^T = [\boldsymbol{a}_1, ..., \boldsymbol{a}_{d_g}] \in \mathbb{R}^{d_g} \times \mathbb{R}^{d_x}$, $d_g \leq d_x$ is nonsigular.

A discrete time, deterministic first-order Markov decision process (MDP) with continuous state and action spaces,

$$\mathcal{M} : \ (X, U, \boldsymbol{D}, \rho) \tag{2}$$

defines the problem. $\rho : X \to \mathbb{R}$ is the observed state reward, and the system dynamics $\boldsymbol{D}$ is given in (1). We assume that we have access to its generative model or samples, but that we do not know

$\boldsymbol{D}$. In the remainder of the article, when the time step k is not important, it is dropped from the state notation without the loss of generality.

A solution to MDP is an optimal policy $\boldsymbol{h}^* : X \to U$, that maximizes discounted cumulative state reward. Thus, the objective function to maximize, *state-value cost function* $V : X \to \mathbb{R}$, is

$$V(\boldsymbol{x}) = \sum_{k=0}^{\infty} \gamma^k \rho_k, \tag{3}$$

where ρ_k is immediate reward observed at time step k starting at state x, and $0 \leq \gamma < 1$ a discount constant. RL solves MDP without analytical knowledge of the system dynamics $\boldsymbol{D}$, and reward, ρ. Instead, it interacts with the system and iteratively constructs the value function. Using the Bellman equation [?], the state value function V can be recursively represented as

$$V(\boldsymbol{x}) = \rho(\boldsymbol{x}) + \gamma \max_{\boldsymbol{u}} V(\boldsymbol{D}(\boldsymbol{x}, \boldsymbol{u})).$$

The state value function is an immediate state reward plus discounted value of the state the system transitions following greedy policy. The *action-state function* $Q : X \times U \to \mathbb{R}$ is,

$$Q(\boldsymbol{x}, \boldsymbol{u}) = \rho(\boldsymbol{x}') + \gamma \max_{\boldsymbol{u}'} V(\boldsymbol{D}(\boldsymbol{x}', \boldsymbol{u}')), \text{ and } \boldsymbol{x}' = \boldsymbol{D}(\boldsymbol{x}, \boldsymbol{u}).$$

Action-value function, Q, is the sum of the reward obtained upon performing action $\boldsymbol{u}$ from a state $\boldsymbol{x}$ and the value of the state that follows. Both value functions give an estimate of a value. A *state-value function*, V, is a measure of state's value, while an *action-value function*, Q, assigns a value to a transition from a given state using an input. Note, that RL literature works with either a *state-reward* ρ, or a related *state-action reward* where the reward is a function of both the state and the action. We do not consider a cost of action itself, thus the *state-action reward* is simply the reward of the state that the agent transitions upon applying action $\boldsymbol{u}$ in the state $\boldsymbol{x}$. Therefore, the relation between the V and Q is

$$Q(\boldsymbol{x}, \boldsymbol{u}) = V \circ \boldsymbol{D}(\boldsymbol{x}, \boldsymbol{u}). \tag{4}$$

Both value functions devise a greedy policy $\boldsymbol{h} : X \to U$, at state $\boldsymbol{x}$, as the input that transitions the system to the highest valued reachable state.

$$\boldsymbol{h}^Q(\boldsymbol{x}) = \operatorname*{argmax}_{\boldsymbol{u} \in U} Q(\boldsymbol{x}, \boldsymbol{u}) \tag{5}$$

A greedy policy uses the learned value function to produce trajectories. We learn state-value function, V, because its approximation can be constructed to define a Lyapunov candidate function, and in tandem with the right policy it can help assess system stability. For discrete actions MDPs, (5) is a brute force search over the available samples. When action space is continuous, (5) becomes an optimization problem over unknown function $\boldsymbol{D}$. We consider analytical properties of $Q(\boldsymbol{x}, \boldsymbol{u})$ for a fixed state $\boldsymbol{x}$ and knowing V, but having only knowledge of the structure of the transition function $\boldsymbol{D}$. The key insight we exploit is that existence of a maximum of the action-value function $Q(\boldsymbol{x}, \boldsymbol{u})$, as a function of input $\boldsymbol{u}$, depends only on the learned parametrization of the state-value function V.

Approximate value iteration algorithms with linear map approximators require basis vectors. Given the state constraint minimization, we choose quadratic basis functions

$$\boldsymbol{F_i}(\boldsymbol{x}) = \|\boldsymbol{a}_i^T \boldsymbol{x}\|^2, \quad i = 1, ..., d_g. \tag{6}$$

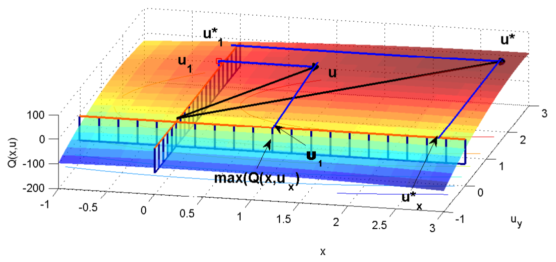


Figure 2: Example of two dimensional input and a quadratic value function. $\boldsymbol{u}^*$ is the optimal input, $\boldsymbol{u}$ is the one selected.

so that state-value function approximation, V, is a Lyapunov candidate function. Consequently, V is,

$$V(\boldsymbol{x}) = \sum_{i=1}^{d_g} \theta_i \boldsymbol{F_i}(\boldsymbol{x}) = (\boldsymbol{A}\boldsymbol{x})^T \boldsymbol{\Theta}(\boldsymbol{A}\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x} \tag{7}$$

for a diagonal matrix $\boldsymbol{\Theta} = \mathrm{diag}(\theta_1, \theta_2, ..., \theta_{d_g})$, and a symmetric matrix $\boldsymbol{\Lambda}$. Let's assume that $\boldsymbol{\Lambda}$ has full rank. Approximate value iteration learns the parametrization $\boldsymbol{\Theta}$ using a linear regression. Let $\boldsymbol{\Gamma} = -\boldsymbol{\Lambda}$. Note, that if $\boldsymbol{\Theta}$ is negative definite, $\boldsymbol{\Lambda}$ is as well, while $\boldsymbol{\Gamma}$ is positive definite, and vice versa. Let also assume that when $\boldsymbol{\Gamma} > 0$ the system drift is bounded with $\boldsymbol{x}$ with respect to $\boldsymbol{\Gamma}$-norm, $\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Gamma} \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{x}^T \boldsymbol{\Gamma} \boldsymbol{x}$. This characterizes system drift, conductive to the task. We empirically demonstrate its sufficiency in the robotic systems we consider.

To summarize the system assumptions used in the reminder of the article:

1. The system is controllable and the equilibrium is reachable. In particular, we use,

$$\exists i, 1 \leq i \leq d_u, \text{ such that } \boldsymbol{f}(\boldsymbol{x}) \boldsymbol{\Gamma} \boldsymbol{g_i}(\boldsymbol{x}) \neq 0, \tag{8}$$

   and that $\boldsymbol{g}(\boldsymbol{x})$ is regular outside of the origin,

$$\boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{\Gamma} \boldsymbol{g}(\boldsymbol{x}) > 0, \, \boldsymbol{x} \in X \setminus \{\boldsymbol{0}\} \tag{9}$$

2. Input is defined on a closed interval around origin,

$$\boldsymbol{0} \in U \tag{10}$$

3. The drift is bounded,

$$\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Gamma} \boldsymbol{f}(\boldsymbol{x}) \leq \boldsymbol{x}^T \boldsymbol{\Gamma} \boldsymbol{x}, \text{ when } \boldsymbol{\Gamma} > 0 \tag{11}$$

Table 1 presents a summary of the key symbols.

3.2 Policy approximation

This section looks into an efficient and a consistent policy approximation for (5) that leads the system (1) to a goal state in the origin. Here, we learn the action-value function Q on the axes, and assume a known estimate of the state-value function approximation V. For the policy to lead the

Table 1: Summary of key symbols and notation.

Symbol	Description
$\mathcal{M}:\ (X, U, \boldsymbol{D}, \rho)$	MDP
$V : X \to \mathbb{R}$, $V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Lambda} x$	State-value function
$Q : X \times U \to \mathbb{R}$	Action-value function
$\boldsymbol{Ax}$	Constraints to minimize
$\boldsymbol{\Lambda} = \boldsymbol{A}^T \boldsymbol{\Theta} \boldsymbol{A}$	Combination of task constraints and value function parametrization
$\boldsymbol{\Gamma} = -\boldsymbol{\Lambda}$	Task-learning matrix
$\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}})$	Policy $\hat{\boldsymbol{h}}^Q$ in state $\boldsymbol{x}$
$\boldsymbol{e_n}$	nth axis unit vector
$\boldsymbol{u} \in U$	Input vector
$u \in \mathbb{R}$	Univariate input variable
$\boldsymbol{u_n} \in \mathbb{R}$	Set of vectors in direction of nth axis
$\hat{u}_n \in \mathbb{R}$	Estimate in direction of the nth axis
$\hat{\boldsymbol{u}}_{\boldsymbol{n}} = \sum_{i=1}^{n} \hat{u}_n \boldsymbol{e_i}$	Estimate over first n axes
$\hat{\boldsymbol{u}}$	Estimate of Q's maximum with a policy
$Q^{(\boldsymbol{p})}_{\boldsymbol{x},n}(u) = Q(\boldsymbol{x}, \boldsymbol{p} + u\boldsymbol{e_n})$	Univariate function in the direction of axis $\boldsymbol{e_n}$, passing through point $\boldsymbol{p}$

system to the origin from an arbitrary state, the origin must be asymptotically stable. Negative of the state-value function V can be a Lyapunov function, and the value function V needs to be increasing in time. That only holds true when the policy approximation makes an improvement, i.e., the policy needs to transition the system to a state of a higher value ($V(\boldsymbol{x}_{n+1}) > V(\boldsymbol{x}_n)$). To ensure the temporal increase of V, the idea is to formulate conditions on the system dynamics and value function V, for which Q, considered as a function only of the input, is concave and has a maximum. In this work, we limit the conditions to a quadratic form Q. When we establish maximum's existence, we approximate it by finding a maximum on the axes and combining them together. Figure 2 illustrates this idea. To reduce the dimensionality of the optimization problem, we propose a divide and conquer approach. Instead of solving one multivariate optimization, we solve d_u univariate optimizations on the axes to find a highest valued point on each axis, u_i. The composition of the axes' action selections is the selection vector $\boldsymbol{u} = [u_1 \ .. \ u_{d_u}]^T$. This section develops the policy approximation following these steps:

1. show that Q is a quadratic form and has a maximum (Proposition 3.1)
2. define admissible policies that ensure the equilibrium's asymptotic stability (Theorem 3.2), and
3. find a sampling-based method for calculating consistent, admissible policies in $O(d_u)$ time with no knowledge of the dynamics (Theorem 3.4).

Since the greedy policy (5) depends on action-value Q, Proposition 3.1 gives the connection between value function (7) and corresponding action-value function Q.

Proposition 3.1. *Action-value function $Q(\boldsymbol{x}, \boldsymbol{u})$ (4), of MDP (2) with state-value function V (7), is a quadratic function of input $\boldsymbol{u}$ for all states $\boldsymbol{x} \in X$. When $\boldsymbol{\Theta}$ is negative definite, the action-value function Q is concave and has a maximum.*

Proof. Evaluating $Q(\boldsymbol{x}, \boldsymbol{u})$ for an arbitrary state $\boldsymbol{x}$, we get

$$\begin{aligned} Q(\boldsymbol{x}, \boldsymbol{u}) &= V(\boldsymbol{D}(\boldsymbol{x}, \boldsymbol{u})) = V(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}), \quad \text{from (1)} \\ &= (\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}))^T \boldsymbol{\Lambda}(\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}) \end{aligned}$$

Thus, Q is a quadratic function of action $\boldsymbol{u}$ at any state $\boldsymbol{x}$. To show that Q has a maximum, we inspect Q's Hessian,

$$HQ(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} \frac{\partial Q(\boldsymbol{x},\boldsymbol{u})}{\partial u_1 \partial u_1} & \dots & \frac{\partial Q(\boldsymbol{x},\boldsymbol{u})}{\partial u_1 \partial u_{d_u}} \\ & \dots & \\ \frac{\partial Q(\boldsymbol{x},\boldsymbol{u})}{\partial u_{d_u} \partial u_1} & \dots & \frac{\partial Q(\boldsymbol{x},\boldsymbol{u})}{\partial u_{d_u} \partial u_{d_u}} \end{bmatrix} = 2\boldsymbol{g}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{g}(\boldsymbol{x}).$$

The Hessian is negative definite because $\boldsymbol{g}(\boldsymbol{x})$ is regular for all states $\boldsymbol{x}$ and $\boldsymbol{\Theta} < 0$, which means that $\boldsymbol{\Lambda} < 0$ as well. Therefore, the function is concave, with a maximum. □

The state-value parametrization $\boldsymbol{\Theta}$ is fixed for the entire state space. Thus, Proposition 3.1 guarantees that when the parametrization $\boldsymbol{\Theta}$ is negative definite, the action-value function Q has a single maximum. Next, we show that the right policy can ensure the progression to the goal, but we first define the acceptable policies.

Definition Policy approximation $\hat{\boldsymbol{u}} = \hat{\boldsymbol{h}}^Q(\boldsymbol{x})$ is *admissible*, if it transitions the system to a state with a higher value when one exists, i.e., when the following holds for *policy's gain* at state $\boldsymbol{x}$, $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}) = Q(\boldsymbol{x}, \hat{\boldsymbol{u}}) - V(\boldsymbol{x})$:

1. $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}) > 0,$ for $\boldsymbol{x} \in X \setminus \{\boldsymbol{0}\}$, and
2. $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}) = 0,$ for $\boldsymbol{x} = \boldsymbol{0}$.

Theorem 3.2 shows that an admissible policy is sufficient for the system to reach the goal.

Theorem 3.2. *Let $\hat{\boldsymbol{u}} = \hat{\boldsymbol{h}}^Q(\boldsymbol{x})$ be an admissible policy approximation. When $\boldsymbol{\Lambda} < 0$, and the drift is bounded with* (11)*, the system* (1) *with value function* (7) *progresses to an asymptotically stable equilibrium under policy $\hat{\boldsymbol{h}}^Q$.*

Proof. Consider $W(\boldsymbol{x}) = -V(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Gamma} \boldsymbol{x}$. W is a Lyapunov candidate function because $\boldsymbol{\Gamma} > 0$.

To show the asymptotic stability, a W needs to be monotonically decreasing in time $W(\boldsymbol{x_{n+1}}) \leq W(\boldsymbol{x_n})$ with equality holding only when the system is in the equilibrium, $\boldsymbol{x_n} = \boldsymbol{0}$. Directly from the definition of the admissible policy, for the state $\boldsymbol{x_n} \neq \boldsymbol{0}$, $W(\boldsymbol{x_{n+1}}) - W(\boldsymbol{x_n}) = -Q(\boldsymbol{x_n}, \hat{\boldsymbol{h}}^Q(\boldsymbol{x_n})) + V(\boldsymbol{x_n}) = V(\boldsymbol{x_n}) - Q(\boldsymbol{x_n}, \hat{\boldsymbol{u}}) < 0$ When $\boldsymbol{x_n} = \boldsymbol{0}$, $\implies$ $\boldsymbol{x_{n+1}} = \boldsymbol{f}(\boldsymbol{0}) = 0$, because of (11) $\implies$ $W(\boldsymbol{x_{n+1}}) = 0$. □

Theorem 3.2 gives the problem formulation conditions for the system to transition to the goal state. Now, we move to finding sample-based admissible policies by finding maximums of Q in the direction parallel to an axis and passing through a point. Because Q has quadratic form, its restriction to a line is a quadratic function of one variable. We use Lagrange interpolation to find the coefficients of Q on a line, and find the maximum in the closed form. We first introduce the notation for Q's restriction in an axial direction, and its samples along the direction.

Definition *Axial restriction* of Q passing through point $\boldsymbol{p}$, is a univariate function $Q^{(\boldsymbol{p})}_{\boldsymbol{x},i}(u) = Q(\boldsymbol{x}, \boldsymbol{p} + u\boldsymbol{e_i})$.

If $\boldsymbol{q_i} = [Q^{\boldsymbol{p}}_{\boldsymbol{x},1}(u_{i1})\ Q^{\boldsymbol{p}}_{\boldsymbol{x},2}(u_{i2})\ Q^{\boldsymbol{p}}_{\boldsymbol{x},3}(u_{i3})]^T$, are three samples of $Q^{(\boldsymbol{p})}_{\boldsymbol{x},i}(u)$ obtained at points $[u_{i1}\ u_{i2}\ u_{i3}]$, then $Q(\boldsymbol{x}, \boldsymbol{p} + u\boldsymbol{e_i})$, is maximized at

$$\begin{aligned} \hat{u}_i &= \min(\max(\hat{u^*}_i, u_i^l), u_i^u), \text{ where} \\ \hat{u}_i^* &= \frac{\boldsymbol{q_i}^T \cdot ([u_{i2}^2 \quad u_{i3}^2 \quad u_{i1}^2] - [u_{i3}^2 \quad u_{i1}^2 \quad u_{i2}^2])^T}{2\boldsymbol{q_i}^T \cdot ([u_{i2} \quad u_{i3} \quad u_{i1}] - [u_{i3} \quad u_{i1} \quad u_{i2}])^T}, \end{aligned} \tag{12}$$

on the interval, $u_i^l \leq u \leq u_i^u$. Equation (12) comes directly from Lagrange interpolation of a univariate second order polynomial to find the coefficients of the quadratic function, and then equating the derivative to zero to find its maximum. In the stochastic case, instead of Lagrange interpolation, linear regression yields the coefficients.

A motivation for this approach is that maximum finding in a single direction is computationally efficient and consistent. A single-component policy is calculated in constant time. In addition, the input selection on an axis calculated with (12) is *consistent*, i.e. it does not depend on the sample points u_{ij} available to calculate it. This is direct consequence of quadratic function being uniquely determined with arbitrary three points. It means that a policy based on (12) produces the same result regardless of the input samples used, which is important in practice where samples are often hard to obtain.

Lemma 3.3 shows single component policy characteristics including that a single-component policy is stable on an interval around zero. Later, we integrate the single-component policies together into admissible policies.

Lemma 3.3. *A single input policy approximation* (12)*, for an input component, i, $1 \leq i \leq d_u$ has the following characteristics:*

1. *There is an input around zero that does not decrease system's state value upon transition, i.e., $\exists u_0 \in [u_l^i, u_u^i]$ such that $Q^{(\boldsymbol{p})}_{\boldsymbol{x},i}(u) \geq Q(\boldsymbol{x}, \boldsymbol{p})$.*
2. *$Q^{(\boldsymbol{0})}_{\boldsymbol{x},i}(\hat{u}_i) - V(\boldsymbol{x}) \geq 0$, when $\boldsymbol{x} \neq \boldsymbol{0}$*
3. *$Q(0, \hat{u}_i\boldsymbol{e_i}) - V(0) = 0$*

The proof for Lemma 3.3 is in Appendix A.

We give three consistent and admissible policies as examples. First, the Manhattan policy finds a point that maximizes Q's restriction on the first axis, then iteratively finds maximums in the direction parallel to the subsequent axes, passing through points that maximize the previous axis. The second policy approximation, convex sum, is a convex combination of the maximums found independently on each axis. Unlike the Manhattan policy that works serially, the convex sum policy parallelizes well. Third, axial sum is the maximum of the convex sum policy approximation and nonconvex axial combinations. This policy is also parallelizable. All three policies scale linearly with the dimensions of the input $O(d_u)$. Next, we show that they are admissible.

Theorem 3.4. *The system* (2) *with value function* (7)*, bounded drift* (11)*, and a negative definite $\boldsymbol{\Theta}$, starting at an arbitrary state $\boldsymbol{x} \in X$, and on a set U* (10)*, progresses to an equilibrium in the origin under any of the following policies:*

1. *Manhattan policy:*

$$\boldsymbol{h}^Q_{\boldsymbol{m}} : \begin{cases} \hat{u}_1 = \operatorname*{argmax}_{u_l^1 \le u \le u_u^1} Q^{(\mathbf{0})}_{\boldsymbol{x},1}(u) \\ \hat{u}_n = \operatorname*{argmax}_{u_l^n \le u \le u_u^n} Q^{(\hat{\boldsymbol{u}}_{\boldsymbol{n-1}})}_{\boldsymbol{x},n}(u), \quad n \in [2, .., d_u], \\ \quad \hat{\boldsymbol{u}}_{\boldsymbol{n-1}} = \sum_{i=1}^{n-1} \hat{u}_i \boldsymbol{e_i}. \end{cases} \tag{13}$$

2. *Convex sum:*

$$\boldsymbol{h}^Q_{\boldsymbol{c}} : \ \hat{\boldsymbol{u}} = \sum_{i=1}^{d_u} \lambda_i \boldsymbol{e_i} \operatorname*{argmax}_{u_l^i \le u \le u_u^i} Q^{(\mathbf{0})}_{\boldsymbol{x},i}(u), \quad \sum_{i=1}^{d_u} \lambda_i = 1 \tag{14}$$

3. *Axial sum:*

$$\boldsymbol{h}^Q_{\boldsymbol{s}} : \ \hat{\boldsymbol{u}} = \begin{cases} \boldsymbol{h}^Q_{\boldsymbol{c}}(\boldsymbol{x}), & Q(\boldsymbol{x}, \boldsymbol{h}^Q_{\boldsymbol{c}}(\boldsymbol{x})) \ge Q(\boldsymbol{x}, \boldsymbol{h}^Q_{\boldsymbol{n}}(\boldsymbol{x})) \\ \boldsymbol{h}^Q_{\boldsymbol{n}}(\boldsymbol{x}), & \textit{otherwise} \end{cases} \tag{15}$$

where

$$\boldsymbol{h}^Q_{\boldsymbol{n}}(\boldsymbol{x}) = \sum_{i=1}^{d_u} \boldsymbol{e_i} \operatorname*{argmax}_{u_l^i \le u \le u_u^i} Q^{(\mathbf{0})}_{\boldsymbol{x},i}(u)$$

The proof for the Theorem 3.4 is in Appendix B.

A consideration in reinforcement learning, applied to robotics and other physical systems, is balancing exploitation and exportation [24]. Exploitation ensures the safety of the system, when the policy is sufficiently good and yields no learning. Exploration forces the agent to perform suboptimal steps, and the most often used ϵ-greedy policy performs a random action with probability ϵ. Although the random action can lead to knowledge discovery and policy improvement, it also poses a risk to the system. The policies presented here fit well in online RL paradigm, because they allow safe exploration. Given that they are not optimal, they produce new knowledge, but because of their admissibility and consistency, their input of choice is safe to the physical system. For systems with independent inputs, axial sum policy is optimal (see Appendix C).

3.3 Continuous action fitted value iteration (CAFVI)

We introduced an admissible, consistent, and efficient decision making method for learning action-value function Q locally, at fixed state $\boldsymbol{x}$, and fixed learning iteration (when $\boldsymbol{\Theta}$ is fixed) without knowing the system dynamics. Now, the decision making policies are integrated into a FVI framework [9, 5] to produce a reinforcement learning agent for continuous state and action MDPs tailored for control-affine nonlinear systems. The algorithm learns the parameterization $\boldsymbol{\Theta}$, and works much like approximate value iteration [9] to learn state-value function approximation $\boldsymbol{\theta}$, but the action selection uses sampling-based policy approximation on the action-value function Q. Algorithm 1 shows an outline of the proposed *continuous action fitted value iteration*, CAFVI. It first initializes $\boldsymbol{\theta}$ with a zero vector. Then, it iteratively estimates Q function values and uses them to make a new estimate of $\boldsymbol{\theta}$. First, we randomly select a state $\boldsymbol{x}_s$ and observe its reward. Line 6 collects the samples. It uniformly samples the state space for $\boldsymbol{x}_{l_s}$. Because we need three data points for Lagrangian interpolation of a quadratic function, three input samples per input dimensions are selected. We also obtain, either through a simulator or an observation, the resulting state $\boldsymbol{x}'_{ij}$ when $\boldsymbol{u}_{ij}$ is applied to $\boldsymbol{x}_{l_s}$. Line 7 estimates the action-value function locally, for $\boldsymbol{x}_{l_s}$ and $\boldsymbol{u}_{ij}$ using the current $\boldsymbol{\theta_l}$ value. Next, the recommended action is calculated, $\hat{\boldsymbol{u}}$. Looking up the available samples or using a simulator, the system makes the transition from $\boldsymbol{x}_{l_s}$ using action $\hat{\boldsymbol{u}}$. The algorithm makes a new estimate of $V(\boldsymbol{x}_{l_s})$. After n_s states are processed, Line 12 finds new $\boldsymbol{\theta}$ that minimizes the least squares error for the new state-value function estimates v_{l_s}. The process repeats until either $\boldsymbol{\theta}$ converges, or a maximum number of iterations is reached.

Algorithm 1 Continuous Action Fitted Value Iteration (CAFVI)

```
Input: $X, U$, discount factor $\gamma$
Input: basis function vector $\boldsymbol{F}$
Output: $\boldsymbol{\theta}$
 1: $\boldsymbol{\theta}_0, \boldsymbol{\theta}_1 \leftarrow$ zero vector
 2: $l \leftarrow 1$
 3: while ($l \leq max\_iterations$) and $\|\boldsymbol{\theta}_l - \boldsymbol{\theta}_{l-1}\| \geq \epsilon$ do
 4:     for $l_s = 1, .., n_s$ do
 5:         sample state $\boldsymbol{x}_{l_s}$ and observe its reward $\rho_{l_s}$
 6:         $\{\boldsymbol{x}_{l_s}, \boldsymbol{u}_{ij}, \boldsymbol{x}'_{ij} | i = 1, .., d_u, j = 1, 2, 3\}$ {obtain system dynamics samples}
 7:         for all i,j, $q_{ij} \leftarrow \boldsymbol{\theta}_l^T \boldsymbol{F}(\boldsymbol{x}'_{ij})$ {estimate action-value function}
 8:         $\hat{\boldsymbol{u}} \leftarrow$ calculated with (12)
 9:         obtain $\{\boldsymbol{x}_{l_s}, \hat{\boldsymbol{u}}, \boldsymbol{x}'_{l_s}, \rho_{l_s}\}$
10:         $v_{l_s} = \rho_{l_s} + \gamma \boldsymbol{\theta}_l^T \boldsymbol{F}(\boldsymbol{x}'_{ls})$ {state-value function new estimate}
11:     end for
12:     $\boldsymbol{\theta_{l+1}} \leftarrow \text{argmin}_{\boldsymbol{\theta}} \sum_{l_s=1}^{n_s} (v_{l_s} - \boldsymbol{\theta}^T \boldsymbol{F}(\boldsymbol{x}_{ls}))^2$
13:     $l \leftarrow l + 1$
14: end while
15: return $\boldsymbol{\theta_l}$
```

The novelties of the Algorithm 1 are continuous input spaces, and the joint work with both state and action-value functions (Lines 6 - 8), while FVI works with discrete, finite input sets and with one of the two functions [5], but not both. Although the outcome of the action-value function learning (Line 8) is independent of the input samples, the state-value function learning (Line 12) depends on the state-samples collected in Line 5, just like discrete action FVI [9].

3.4 Discussion

Considering a *constraint-balancing task*, we proposed quadratic feature vectors, and determined sufficient conditions for which admissible policies presented in Section 3.2 transition the system to the goal state obeying the task requirements. Finally, we presented a learning algorithm that learns the parametrization. There are several points that need to be discussed, convergence of the CAFVI algorithm, usage of the quadratic basis functions, and determination of the conditions from Section 3.1.

Full conditions under which FVI with discrete actions converges is still an active research topic [5]. It is known that it converges when the system dynamics is a contraction [5]. A detailed analysis of the error bounds for FVI algorithms with finite [**?**] and continuous [**?**] actions, finds that the FVI error bounds scale with the difference between the basis functional space and the inherent dynamics of the MDP. The system's dynamics and reward functions determine the MDP's dynamics. We choose quadratic basis functions, because of the nature of the problem we need to solve and for stability. But, basis functions must fit reasonably well into the true objective function (3) determined by the system dynamics and the reward, otherwise CAFVI diverges.

The goal of this article is to present an efficient toolset for solving constraint-balancing tasks on a control-affine system with unknown dynamics. Using quadratic basis functions, Algorithm 1 learns the parametrization $\boldsymbol{\theta}$. Successful learning that converges to a $\boldsymbol{\theta}$ with all negative components, produces a controller based on Section 3.2 policies that is safe for a physical system and completes the task.

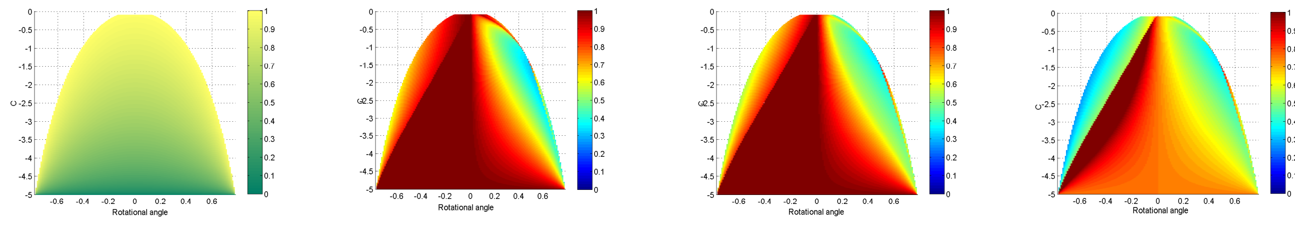
In Section 3.1, we introduced sufficient conditions for successful learning. The conditions are sufficient but not necessary, so the learning could succeed under laxer conditions. Done in simulation prior to a physical system control, the learning can be applied when we are uncertain if the system satisfies the criterion. When the learning fails to succeed, the controller is not viable. Thus, a viable controller is possible under laxer conditions verifiable through learning. so the toolset can be safely and easily attempted first, before more computationally intensive methods are applied. It can be also used to quickly develop an initial value function, to be refined later with another method.

4 Results

This section evaluates the proposed methodology. We first verify the policy approximations' quality and computational efficiency on a known function in Section 4.1, and then we showcase the method's learning capabilities in two case studies: a quadrotor with suspended payload (Section 4.2), and a multi-agent rendezvous task (Section 4.3).

In all evaluations, the Convex Sum was calculated using equal convex coefficients $\lambda_i = d_u^{-1}$. Discrete and HOOT [20] policies are used for comparison. The discrete policy uses an equidistant grid with 13 values per dimension. HOOT uses three hierarchical levels, each covering one tenth of the input size per dimension and maintaining the same number of inputs at each level. All computation was performed using Matlab on a single core Intel Core i7 system with 8GB of RAM, running the Linux operating system.

4.1 Policy approximation evaluation



(a) Eccentricity (b) Manhattan (c) Axial Sum (d) Convex Sum

Figure 3: Eccentricity of the quadratic functions (a) related to policy approximation gain ratio (b-d) as a function of quadratic coefficient (C) and rotation of the semi-axes.

In Section 3.2 we proposed three policy approximations and showed their admissibility. To empirically verify the findings, we examine their behavior on known quadratic functions of two

Table 2: Summary of policy approximation performance. Minimum and maximum of the value gain and the distance from the optimal input.

Method	$\min \Delta Q$	$\max \Delta Q$	$\min \Delta u$	$\max \Delta u$
Manhattan	5.00	168.74	0.00	4.32
Axial Sum	3.40	163.76	0.00	4.37
Convex Sum	3.40	103.42	0.10	4.37

variables, elliptical paraboloids with a maximum. Table 2 depicts maximum and minimum values for $\Delta Q(\boldsymbol{x}, \boldsymbol{h}^Q(\boldsymbol{x}))$ as Q ranges over the class of concave elliptical paraboloids. Since the ΔQ is always positive for all three policies, the empirical results confirm our findings from Proposition 3.4 that the policies are admissible. We also see from $\min \Delta u$ that in some cases Manhattan and Axial Sum make optimal choices, which is expected as well. The maximum distance from the optimal input column shows that the distance from the optimal input is bounded.

To further evaluate the policies' quality we measure the gain ratio between the policy's gain and maximum gain on the action-value function ($\boldsymbol{u}^*$ is optimal input):

$$g_{\boldsymbol{h}^Q}(\boldsymbol{x}) = \frac{Q(\boldsymbol{x}, \boldsymbol{h}^Q(\boldsymbol{x})) - Q(\boldsymbol{x}, \mathbf{0})}{Q(\boldsymbol{x}, \boldsymbol{u}^*) - Q(\boldsymbol{x}, \mathbf{0})}.$$

Non-admissible policies have negative or zero gain ratio for some states, while the gain ratio for admissible policies is strictly positive. The gain ratio of one signifies that policy $\boldsymbol{h}^Q$ is optimal, while a gain ratio of zero means that the selected input transitions the system to an equivalent state from the value function perspective. The elliptic paraboloids', $Q(\boldsymbol{x}, [u_1 u_2]^T) = au_1^2 + bu_1u_2 + cu_2^2 + du_1 + eu_2 + f$, isolines are ellipses, and the approximation error depends on the rotational angle of the ellipse's axes, and its eccentricity. Thus, a policy's quality is assessed as a function of these two parameters: the rotational angle α and range of the parameter c, while parameters a, d, e, and f are fixed. Parameter b is calculated such that $b = (a - c) \tan 2\alpha$. The eccentricity is depicted in Figure 3a, with zero eccentricity representing a circle, and an eccentricity of one representing the ellipse degenerating into a parabola. The white areas in the heat maps are areas where the function is either a hyperbolic paraboloid or a plane, rather than an elliptic paraboloid and has no maximum. Figure 3 displays the heat maps of the gain ratios for the Manhattan (Figure 3b), Axial Sum (Figure 3c), and Convex Sum (Figure 3d) policies. All policies have strictly positive gain ratio, which gives additional empirical evidence to support the finding in Proposition 3.4. Manhattan and Axial Sum perform similarly, with the best results for near-circular paraboloids, and degrading as the eccentricity increases. In contrast, the Convex Sum policy performs best for highly elongated elliptical paraboloids.

Lastly, we consider the computational efficiency of the three policies, and compare the running time of a single decision making with discrete and HOOT [20] policies. Figure 4 depicts the computational time for each of the policies as a function of the input dimensionality. Both discrete and HOOT policies' computational time grows exponentially with the dimensionality, while the three policies that are based on the axial maximums: Manhattan, Axial Sum, and Convex Sum are linear in the input dimensionality, although Manhattan is slightly slower.

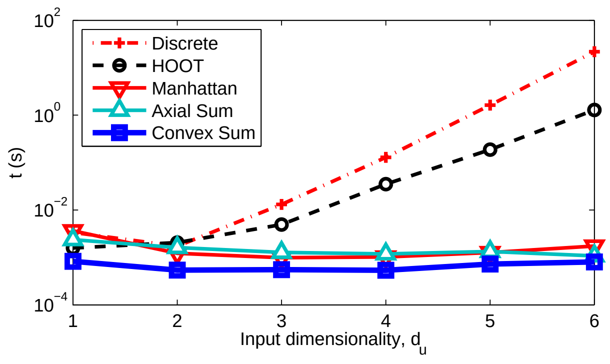


Figure 4: Policy approximation computational time per input dimensionality. Comparison of discrete, HOOT, Manhattan, Axial Sum, and Convex Sum policies. The y-axis is logarithmic.

4.2 Cargo delivery task

This section applies the proposed methods to the aerial cargo delivery task [11]. This task is defined for a UAV carrying a suspended load, and seeks acceleration on the UAV's body, that transports the joint UAV-load system to a goal state with minimal residual oscillations. We show that the system and its MDP satisfy conditions for Theorem 3.2, and will assess the methods though examining the learning quality, the resulting trajectory characteristics, and implementation on the physical system. We compare it to the discrete AVI [11] and HOOT [20], and show that methods presented here solve the task with more precision.

To apply the motion planner to the cargo delivery task for a holonomic UAV carrying a suspended load, we use the following definition of the swing-free trajectory.

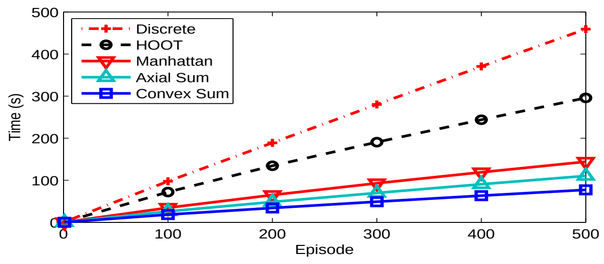
Definition A trajectory of duration t_0 is said to be with *minimal residual oscillations* if for a given constant $\epsilon > 0$ there is a time $0 \leq t_1 \leq t_0$, such that for all $t \geq t_1$, the load displacement is bounded with ϵ ($\rho(t) < \epsilon$).

The MDP state space is the position of the center of the mass of the UAV $\boldsymbol{p} = [x \; y \; z]^T$, its linear velocities $\boldsymbol{v} = [\dot{x} \; \dot{y} \; \dot{z}]^T$, the angular position $\boldsymbol{\eta} = [\psi \; \phi]^T$ of the suspended load in the polar coordinates originating at the quadrotor's center of mass, with the zenith belonging to the axis perpendicular to Earth, and its angular velocities $\dot{\boldsymbol{\eta}} = [\dot{\psi} \; \dot{\phi}]^T$. The actuator is the acceleration on the quadrotor's body, $\boldsymbol{u} = [u_x \; u_y \; u_z]^T$. For the system's generative model, we use a simplified model of the quadrotor-load model described in [11], which satisfies the form (1).

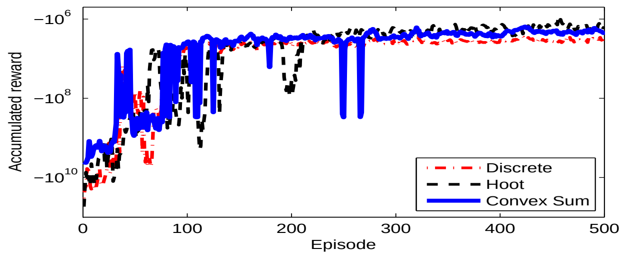
$$\begin{aligned} &\boldsymbol{v} = \boldsymbol{v_0} + \triangle t \boldsymbol{u}; \quad \boldsymbol{p} = \boldsymbol{p_0} + \triangle t \boldsymbol{v_0} + \frac{\triangle t^2}{2}\boldsymbol{u} \\ &\dot{\boldsymbol{\eta}} = \dot{\boldsymbol{\eta}}_0 + \triangle t \ddot{\boldsymbol{\eta}}; \quad \boldsymbol{\eta} = \boldsymbol{\eta_0} + \triangle t \dot{\boldsymbol{\eta}}_0 + \frac{\triangle t^2}{2}\ddot{\boldsymbol{\eta}}, \text{ where} \\ &\ddot{\boldsymbol{\eta}} = \begin{bmatrix} \sin\theta_0 \sin\phi_0 & -\cos\phi_0 & L^{-1}\cos\theta_0\sin\phi_0 \\ -\cos\theta_0\cos\phi_0 & 0 & L^{-1}\cos\phi_0\sin\theta_0 \end{bmatrix} (\boldsymbol{u} - \boldsymbol{g'}) \end{aligned} \tag{16}$$

The system (16) satisfies the form (1). The reward function penalizes the distance from the goal state, the load displacement, and the negative z coordinate. Lastly, the agent is rewarded when it reaches equilibrium.

The value function is approximated as a linear combination of quadratic forms of state subspaces $V(\boldsymbol{x}) = \boldsymbol{\theta}^T * F(\boldsymbol{x}) \; F(\boldsymbol{x}) = [\|\boldsymbol{p}\|^2 \quad \|\boldsymbol{v}\|^2 \quad \|\boldsymbol{\eta}\|^2 \quad \|\dot{\boldsymbol{\eta}}\|^2]^T$ where $\boldsymbol{\theta} \in \mathbb{R}^4$, satisfies the form (7), and because the learning produces $\boldsymbol{\theta}$ with all negative components, all conditions for Theorem 3.2 are satisfied including the drift (11).



(a) Time to learn



(b) Learning curve (logarithmic)

Figure 5: Learning results for Manhattan, and Axial Sum, and Convex Sum, compared to discrete greedy, and HOOT policies averaged over 100 trials. Learning curves for Manhattan and Axial Sum are similar to Convex Sum and are omitted from (b) for better visibility.

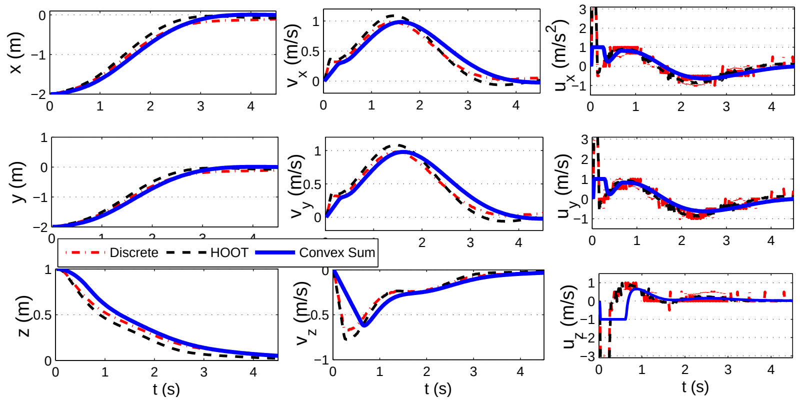
The time-to-learn is presented in Figure 5a. The axial maximum policies perform an order of magnitude faster than the discrete and HOOT policies. To assess learning with Algorithm 1 using Manhattan, Axial Sum, and Convex Sum policies, we compare to learning using the greedy discrete policy and HOOT. Figure 5b shows the learning curve, over number of iterations. After 300 iterations all policies converge to a stable value. All converge to the same value, but discrete learning that converges to a lower value.

Finally, inspection of the learned parametrization vectors confirms that all the components are negative, meeting all needed criteria for Theorem 3.2. This means that the equilibrium is asymptotically stable, for admissible policies, and we can generate trajectories of an arbitrary length.

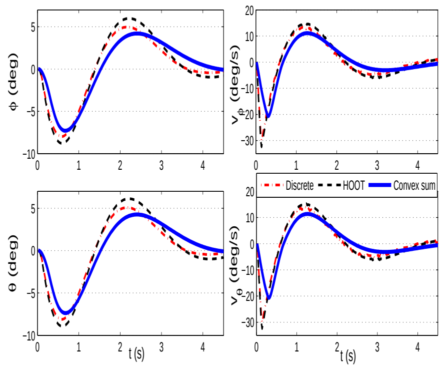
Next, we plan trajectories using the learned parametrizations over the 100 trials for the three proposed policies and compare them to the discrete and HOOT policies. We consider a cargo delivery task complete when $\|\boldsymbol{p}\| \leq 0.010$m, $\|\boldsymbol{v}\| \leq 0.025$ m/s, $\|\boldsymbol{\eta}\| \leq 1°$, and $\|\dot{\boldsymbol{\eta}}\| \leq 5°$/s. This is a stricter terminal set than the one previously used in [11]. The input limits are $-3 \leq u_i \leq 3$, for $i \in 1, 2, 3$. The discrete and HOOT policies use the same setup described in Section 4. The planning occurs at 50Hz. We compare the performance and trajectory characteristics of trajectories originating 3 meters from the goal state. Table 3 presents results of the comparison. Manhattan, Axial Sum, and HOOT produce very similar trajectories, while Convex Sum generates slightly longer trajectories, but with the best load displacement characteristics. This is because the Convex Sum takes a different approach and selects smaller inputs, resulting in smoother trajectories. The Convex Sum method plans the 9 second trajectory in 0.14s, over 5 times faster than the discrete planning, and over 3 times faster than HOOT. Finally, 30% of the discrete trajectories are never able to complete the task. This is because the terminal set is too small for the discretization. In other words, the discretized policy is not admissible. Examining the simulated trajectories in Figure 6 reveals that Convex Sum indeed selects a smaller input, resulting in a smoother trajectory (Figure 6a) and less swing (Figure 6b). HOOT, Manhattan, and Axial Sum, produce virtually identical trajectories, while the discrete trajectory has considerable jerk, absent from the other trajectories.

Table 3: Summary of trajectory characteristics over 100 trials. Means (μ) and standard deviations (σ) of time to reach the goal, final distance to goal, final swing, maximum swing, and time to compute the trajectory. Best results are highlighted.

Method	Percent completed	t (s)		$\parallel p \parallel$ (cm)		$\parallel \eta \parallel$ (°)		max $\parallel \eta \parallel$ (°)		Comp. time (s)	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Discrete	70.00	10.81	3.12	0.98	0.33	0.16	0.14	11.96	1.63	0.81	0.23
HOOT	100.00	**8.49**	**1.33**	**0.83**	0.27	0.18	0.20	12.93	1.49	0.48	0.07
Manhattan	100.00	8.66	1.68	0.89	0.19	0.15	0.16	12.24	1.58	0.24	0.05
Axial Sum	100.00	8.55	1.56	0.85	0.22	0.20	0.18	12.61	1.55	0.17	0.03
Convex Sum	100.00	9.61	1.62	0.97	**0.07**	**0.03**	**0.06**	**9.52**	**1.29**	**0.14**	**0.02**



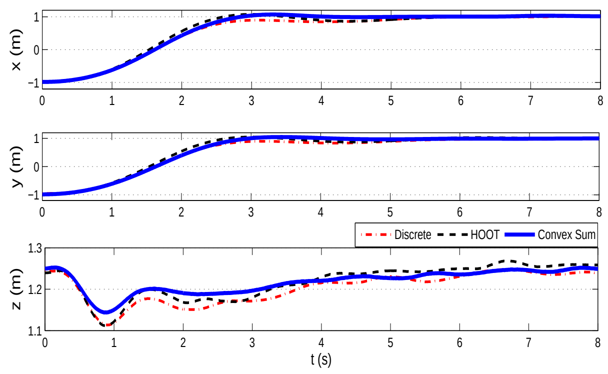
(a) Quadrotor trajectory



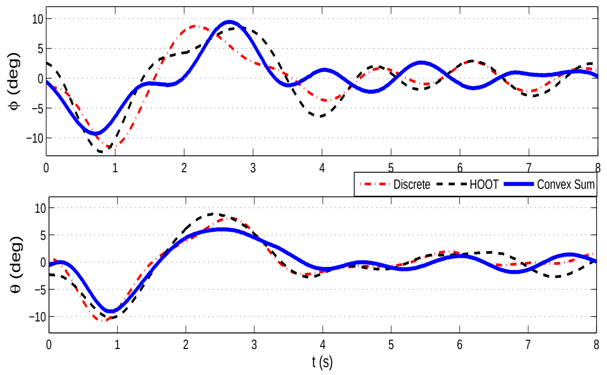
(b) Load trajectory

Figure 6: Comparison of simulated cargo delivery trajectories created with Convex Sum versus trajectories created with discrete greedy and HOOT policies. (Trajectories for Manhattan and Axial Sum are similar to Convex Sum and are omitted for better visibility.)

Lastly, we experimentally compare the learned policies. The experiments were performed on AscTec Hummingbird quadrocopters, carrying a 62-centimeter suspended load weighing 45 grams. The quadrotor and load position were tracked via a Vicon motion capture system at 100 Hz. Experimentally, HOOT and Axial Sum resulted in similar trajectories, while Manhattan's trajectory exhibited the most deviation from the planned trajectory (Figure 7). The Convex Sum trajectory is the smoothest. Table 4 quantifies the maximum load swing and the power required to produce the load's motion from the experimental data. Convex Sum policy generates experimental trajectories with the best load swing performance, and with load motion that requires close to three times less energy to generate. The enclosed video submission contains videos of the experiments.



(a) Quadrotor trajectory



(b) Load trajectory

Figure 7: Comparison of experimental cargo delivery task trajectories created with Convex Sum versus trajectories created with discrete greedy and HOOT policies. (Trajectories for Manhattan and Axial Sum are similar to Convex Sum and are omitted for better visibility.)

Table 4: Summary of experimental trajectory characteristics. Maximum swing and energy needed to produce load oscillations. Best results are highlighted.

Method	$max \parallel \eta \parallel$ (°)	Energy (J)
Discrete	15.21	0.0070
HOOT	15.61	0.0087
Manhattan	15.95	0.0105
Axial Sum	14.20	0.0086
Convex Sum	**12.36**	**0.0031**

4.3 Rendezvous task

The rendezvous cargo delivery task is a multi-agent variant of the time-sensitive cargo delivery task. It requires an UAV carrying a suspended load to rendezvous in swing-free fashion with a ground-bound robot to hand over the cargo. The cargo might be a patient airlifted to a hospital and then taken by a moving ground robot for delivery to an operating room for surgery. The rendezvous location and time are not known a priori, and the two heterogeneous agents must plan jointly to coordinate their speeds and positions. The two robots have no knowledge of the dynamics and each others' constraints. The task requires minimization of the distance between the load's and the ground robot's location, the load swing minimization, and minimization for the agents' velocities, while completing the task as fast as possible.

The quadrotor with the suspended load is modeled as in Section 4.2, while a rigid body constrained to two DOF in a plane models the ground-based robot. The joint state space is a 16-dimensional vector: the quadrotor's 10-dimensional state space (Section 4.2), and the ground robot's position-velocity space. The input is 5-dimensional acceleration to the quadrotor's and ground robot's center of masses. The ground robot's maximum acceleration is lower than quadrotor's.

Applying Algorithm 1 with Convex Sum policy, the system learns the state-value function parametrization $\mathbf{\Theta}$ that is negative definite. Figure 8 shows both robots two seconds in the trajectory. The comparison of simulated trajectories created with the Convex Sum and HOOT policies is depicted in Figure 9. Convex Sum finds an 8.54-second trajectory that solves the task in 0.12

seconds. HOOT policy fails to find a suitable trajectory before reaching the maximum trajectory duration, destabilizes the system, and terminates after 101.44 seconds. The discrete policy yields similar results as HOOT. This is because the input needed to solve the task is smaller than the HOOT's setup, and the system begins to oscillate. The rendezvous point produced with Convex Sum policy is between the robots' initial positions, closer to the slower robot, as expected (Figure 9a). The quadrotor's load swing is minimal (Figure 9b). The absolute accumulated reward collected while performing the task is smooth and steadily making progress, while the accumulated reward along HOOT trajectory remains significantly lower (Figure 9c). Enclosed video submission contains an animation of the simulation. The rendezvous simulation shows that the proposed methods are able to solve tasks that previous methods are unable to because the convex policy is admissible.

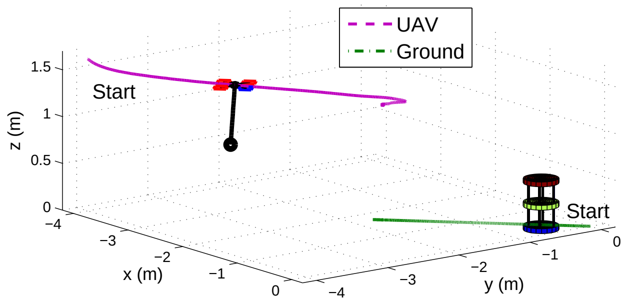


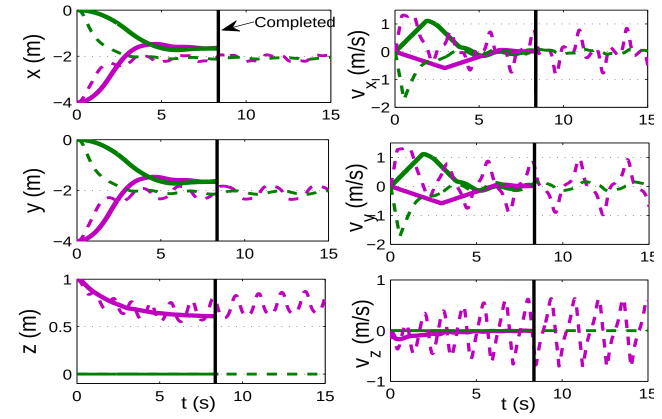
Figure 8: Cargo-bearing UAV and a ground-based robot rendezvous at 2 seconds.

5 Conclusions

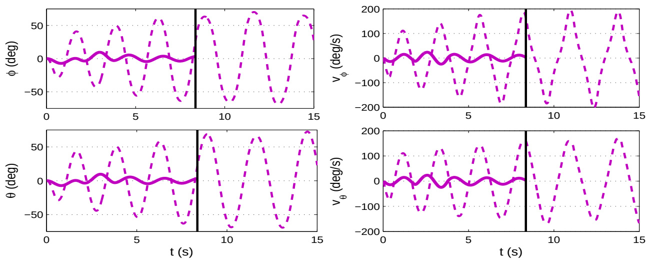
Control of high-dimensional systems with continuous actions is a rapidly developing topic of research. In this paper we proposed a method for learning control of non-linear motion systems through combined learning of state-value and action-value functions. Negative definite quadratic state-value functions imply quadratic, concave action-value functions. That allowed us to approximate policy as a combination of its action-value function maximums on the axes, which we found through interpolation between observed samples. These policies are admissible, consistent, and efficient. Lastly, we showed that a quadratic, negative definite state-value function, in conjunction with admissible policies, are sufficient conditions for the system to progress to the goal while minimizing given constraints.

The verification on known functions confirmed the policies' admissibility. A quadrotor carrying a suspended load assessed the method's applicability to a physical system and a practical problem, and provided a comparison to two other methods demonstrating higher precision of the proposed method as well. The rendezvous task tested the method in higher dimensional input spaces for a multi-agent system, and showed that it finds a solution where other two methods do not. The results confirm that the proposed method outruns current state-of-the-art by an order of magnitude, while the experimental data revealed that the proposed method produces trajectories with better characteristics.

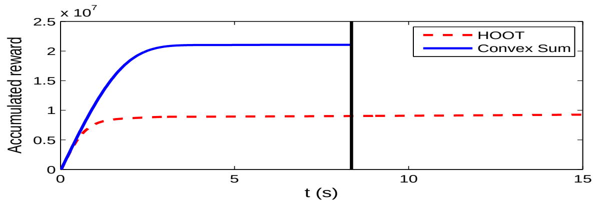
In all, we presented a solid first step for an optimal control framework for unknown control-affine systems for constraint-balancing tasks. Despite the applied method's restrictive condition,



(a) Robot trajectories



(b) Load trajectory



(c) Accumulated reward

Figure 9: Comparison of simulated rendezvous task trajectories created with Convex Sum to trajectories created with discrete greedy and HOOT policies. Green solid - Convex Sum ground; Purple solid - Convex Sum aerial; Green dashed - HOOT ground; Purple dashed - HOOT aerial.

the results demonstrated high accuracy and fast learning times on the practical applications. In future work, the methodology can be extended to stochastic MDPs.

Appendices

A Proof for Lemma 3.3

Proof. First, to show that there is $\exists u_0 \in [u_l^i, u_u^i]$ such that $Q_{\boldsymbol{x},i}^{(\boldsymbol{p})}(u) \geq Q(\boldsymbol{x}, \boldsymbol{p})$, we pick $u = 0$, and directly from the definition, we get $Q_{\boldsymbol{x},i}^{(\boldsymbol{p})}(0) = Q(\boldsymbol{x}, \boldsymbol{p})$. As a consequence

$$Q_{\boldsymbol{x},i}^{(\boldsymbol{p})}(0) \leq Q_{\boldsymbol{x},i}^{(\boldsymbol{p})}(\hat{u}_i) \tag{17}$$

Second, to show that $Q_{\boldsymbol{x},i}^{(\boldsymbol{0})}(\hat{u}_i) - V(\boldsymbol{x}) \geq 0$,

$$\begin{aligned} Q_{\boldsymbol{x},i}^{(\boldsymbol{0})}(\hat{u}_i) &\geq Q_{\boldsymbol{x},i}^{(\boldsymbol{0})}(0), \text{ from (17)} \\ &= \boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{f}(\boldsymbol{x}) \geq \boldsymbol{x} \boldsymbol{\Lambda} \boldsymbol{x}, \text{ due to (11)} \\ &= V(\boldsymbol{x}) \end{aligned}$$

Third, we show $Q(0, \hat{u}_i \boldsymbol{e_i}) - V(0) = 0$. Since, the origin is equilibrium, the dynamics is $D(\boldsymbol{0}, \hat{u}_i \boldsymbol{e_i}) = 0$. Let's evaluate the dynamics at $\hat{u}_i \boldsymbol{e_i}$, when $\boldsymbol{x} = \boldsymbol{0}$,

$$\begin{aligned} D(\boldsymbol{0}, \hat{u}_i \boldsymbol{e_i}) &= \boldsymbol{f}(\boldsymbol{0}) + \boldsymbol{g}(\boldsymbol{0}) \hat{u}_i \boldsymbol{e_i} \\ &= \boldsymbol{f}(\boldsymbol{0}), \text{ because of (9)} \\ &= \boldsymbol{0}, \text{ because of (11)} \end{aligned}$$

Thus, $Q(0, \hat{u}_i \boldsymbol{e_i}) - V(0) = 0$. □

B Proof for Theorem 3.4

Proof. In all three cases, it is sufficient to show that the policy approximations are admissible.

Manhattan policy: To show that the policy approximation (13) is admissible, for $\boldsymbol{x} \neq 0$ we use induction by n, $1 \leq n \leq d_u$, with induction hypothesis,

$$\begin{gathered} \Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}_{\boldsymbol{n}}) \geq 0, \text{ where } \hat{\boldsymbol{u}}_{\boldsymbol{n}} = \sum_{i=1}^{n} \hat{u}_i \boldsymbol{e_i}, \text{ and} \\ \Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}_{\boldsymbol{n}}) = 0 \Leftrightarrow \\ \boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{g}_i(\boldsymbol{x}) = 0, \forall i \leq n, \boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x} \end{gathered} \tag{18}$$

First note that at iteration $1 < n \le d_u$,

$$\begin{aligned}&D(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}} + u\boldsymbol{e_n}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\left(\boldsymbol{\hat{u}_{n-1}} + u\boldsymbol{e_n}\right)\\ &= \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{\hat{u}_{n-1}} + \boldsymbol{g}(\boldsymbol{x})u\boldsymbol{e_n} = \boldsymbol{f_n}(\boldsymbol{x}) + \boldsymbol{g_n}(\boldsymbol{x})u\end{aligned}$$

and

$$\begin{aligned}&Q(\boldsymbol{x}, \boldsymbol{u_n}) = (\boldsymbol{f_n}(\boldsymbol{x}) + \boldsymbol{g_n}(\boldsymbol{x})u)^T\boldsymbol{\Lambda}(\boldsymbol{f_n}(\boldsymbol{x}) + \boldsymbol{g_n}(\boldsymbol{x})u)\\ &= \boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{g_n}(\boldsymbol{x})u^2 + 2\boldsymbol{f_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{g_n}(\boldsymbol{x})u\\ &+ \boldsymbol{f_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f_n}(\boldsymbol{x})\\ &= p_n u^2 + q_n u + r_n, \quad p_n, q_n, r_n \in \mathbb{R}.\end{aligned} \tag{19}$$

Because $\boldsymbol{\Lambda} < 0$, $Q(\boldsymbol{x}, \boldsymbol{u_n})$ is a quadratic function of one variable with a maximum in

$$\hat{u}_n^* = -\frac{\boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f_n}(\boldsymbol{x})}{\boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{g_n}(\boldsymbol{x})} \tag{20}$$

Applying the induction for $n = 1$, and using Lemma 3.3,

$$\begin{aligned}\Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_1}) &= Q(\boldsymbol{x}, \hat{u}_1\boldsymbol{e_1}) - V(\boldsymbol{x})\\ &\ge Q(\boldsymbol{x}, \boldsymbol{0}) - V(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x}\\ &> 0, \text{ when } \boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) > \boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x}.\end{aligned} \tag{21}$$

Given that, $\boldsymbol{\hat{u}_1} \neq 0 \Leftrightarrow \Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_n}) > \Delta Q(\boldsymbol{x}, 0)$, and assuming $\boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x}$, we evaluate $\boldsymbol{\hat{u}_1} = 0$. From (20),

$$\hat{u}_1 = -\frac{\boldsymbol{g_1}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x})}{\boldsymbol{g_1}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{g_1}(\boldsymbol{x})} = 0 \Leftrightarrow \boldsymbol{g_1}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) = 0 \tag{22}$$

So, the induction hypothesis (18) for $n = 1$ holds. Assuming that (18) holds for $1, .., n-1$, and using Lemma 3.3,

$$\begin{aligned}\Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_n}) &= Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}} + \hat{u}_n\boldsymbol{e_n}) - V(\boldsymbol{x})\\ &\ge Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}} + \boldsymbol{0}) - V(\boldsymbol{x})\\ &= \Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}}) \text{ from ind. hyp. (18)}\\ &> 0. \text{ when } \boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) > \boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x}.\end{aligned}$$

Similarly, assuming $\boldsymbol{f}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{\Lambda}\boldsymbol{x}$,

$$\begin{aligned}&\Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_n}) = 0 \Leftrightarrow\\ &\hat{u}_n = -\frac{\boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f_n}(\boldsymbol{x})}{\boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{g_n}(\boldsymbol{x})} = 0, \text{ and } \Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}}) = 0\end{aligned}$$

Since $\Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}}) = 0 \Leftrightarrow \boldsymbol{\hat{u}_{n-1}} = \boldsymbol{0}$, means that $\boldsymbol{f_n}(\boldsymbol{x}) = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{\hat{u}_{n-1}} = \boldsymbol{f}(\boldsymbol{x})$,

$$\begin{aligned}&\Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_n}) = 0 \Leftrightarrow\\ &\boldsymbol{g_n}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) = 0, \text{ and } \Delta Q(\boldsymbol{x}, \boldsymbol{\hat{u}_{n-1}}) = 0 \Leftrightarrow\\ &\boldsymbol{g_i}(\boldsymbol{x})^T\boldsymbol{\Lambda}\boldsymbol{f}(\boldsymbol{x}) = 0, \text{ for } 1 \le i \le n\end{aligned}$$

For $n = d_u$, the policy gain $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}_{\boldsymbol{d_u}}) = 0 \Leftrightarrow \boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x}$, and $\boldsymbol{g_i}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{f}(\boldsymbol{x}) = 0$, for $1 \leq i \leq d_u$. But, that is contradiction with the controllability assumption (8), thus $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}_{\boldsymbol{d_u}}) > 0$, when $\boldsymbol{x} \neq 0$.

When $\boldsymbol{x} = \boldsymbol{0}$, we get directly from Lemma 3.3, $\Delta Q(0, \hat{\boldsymbol{u}}_{\boldsymbol{d_u}}) = 0$. This completes the proof that Manhattan policy (13) is admissible, and therefore the equilibrium is asymptotically stable.

Convex sum (14)*:* Following the same reasoning as for the first step of the Manhattan policy (21) and (22), we get that for all $1 \leq n \leq d_u$, $\Delta Q(\boldsymbol{x}, \hat{u}_n \boldsymbol{e_n}) \geq 0$, where $\hat{u}_n \boldsymbol{e_n} = \operatorname{argmax}_{u_l^n \leq u \leq u_u^n} Q^{(\boldsymbol{0})}_{\boldsymbol{x},n}(u)$ and the equality holds only when

$$\begin{aligned} &\Delta Q(\boldsymbol{x}, \hat{u}_n \boldsymbol{e_n}) = 0 \Leftrightarrow \\ &\boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{g}_n(\boldsymbol{x}) = 0, \boldsymbol{f}(\boldsymbol{x})^T \boldsymbol{\Lambda} \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{\Lambda} \boldsymbol{x} \end{aligned} \tag{23}$$

To simplify the notation, let $Q_i = \Delta Q(\boldsymbol{x}, \hat{u}_n \boldsymbol{e_n})$, and $Q_0 = 0$. Without loss of generality, assume that $Q_0 \leq Q_1 \leq ... \leq Q_{d_u}$, $n = 1, ..., d_u$. The equality only holds when (23) holds for all $n = 1, ..., d_u$ which is contradiction with the (8). Thus, there must be at least one $1 \leq n_0 \leq d_u$, such that $Q_{n_0-1} < Q_{n_0}$, and consequently $0 < Q_{d_u}$.

Lastly, we need to show that the combined input $\hat{\boldsymbol{u}}$ calculated with (14) is admissible, i.e., $\Delta Q(\boldsymbol{x}, \hat{\boldsymbol{u}}) > 0$. It suffices to show that $\hat{\boldsymbol{u}}$ is inside the ellipsoid $\check{Q}_0 = \{\boldsymbol{u} | Q(\boldsymbol{x}, \boldsymbol{u}) \geq Q_0\}$. Similarly, $Q_1, ..., Q_{d_u}$ define a set of concentric ellipsoids $\check{Q}_i = \{\boldsymbol{u} | Q(\boldsymbol{x}, \boldsymbol{u}) \geq Q_i\}, \quad i = 1, ..., d_u$. Since, $\check{Q}_0 \supseteq \check{Q}_1 \supseteq ... \supseteq \check{Q}_{d_u}$, and $\forall i, \hat{\boldsymbol{u}}_{\boldsymbol{i}} \in \check{Q}_i \implies \hat{\boldsymbol{u}}_{\boldsymbol{i}} \in \check{Q}_0$. Because ellipsoid $\check{Q}_0$ is convex, the convex combination of points inside it (14), belongs to it as well. Since, at least one ellipsoid must be a true subset of $\check{Q}_0$, which completes the asymptotic stability proof.

Axial sum policy approximation (15): is admissible because (14) is admissible. Formally, $\Delta Q(\boldsymbol{x}, \boldsymbol{h}^Q_{\boldsymbol{s}}(\boldsymbol{x})) \geq \Delta Q(\boldsymbol{x}, \boldsymbol{h}^Q_{\boldsymbol{c}}(\boldsymbol{x})) \geq 0$. □

C Optimality conditions

Proposition C.1. *When $\boldsymbol{g}(\boldsymbol{x})$ is an independent input matrix, $\boldsymbol{A} = \boldsymbol{I}$, and state-value function parameterization $\boldsymbol{\Theta}$ is negative definite, then Axial Sum policy* (15) *is optimal with respect to the state-value function* (7)*.*

Proof. The optimal input $\boldsymbol{u}^*$ is a solution to $\frac{\partial Q(\boldsymbol{x}, u_i)}{\partial u_i} = 0$, and $\hat{\boldsymbol{u}}$ is a solution to $\frac{\mathrm{d}Q^{(\boldsymbol{0})}_{\boldsymbol{x},i}(u)}{\mathrm{d}u}$ at state $\boldsymbol{x}$ with respect to the state-value function (7). To show that the Axial Sum policy is optimal, $\boldsymbol{u}^* = \hat{\boldsymbol{u}}$, it is enough to show that $\frac{\mathrm{d}Q(\boldsymbol{x}, u_i)}{\mathrm{d}u_i} = \frac{\mathrm{d}Q^{(\boldsymbol{0})}_{\boldsymbol{x},i}(u)}{\mathrm{d}u}$. This is the case when Q has the form of $Q(\boldsymbol{x}, \boldsymbol{u}) = \sum_{i=1}^{d_x} (p_{x_i} u_i^2 + q_{x_i} u_i + r_{x_i})$, for some $p_{x_i}, q_{x_i}, r_{x_i} \in \mathbb{R}$ that depend on the current state $\boldsymbol{x}$. In the Proposition 3.1 we showed that $Q(\boldsymbol{x}, \boldsymbol{u}) = (\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{u}))^T \boldsymbol{\Theta} (\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{u})$
$= \sum_{i=1}^{d_x} \theta_i \left(\sum_{j=1}^{d_u} g_{ij}(\boldsymbol{x}) u_j + f_i(\boldsymbol{x}) \right)^2$. Since there is a single nonzero element j_i in row i of matrix $\boldsymbol{g}$, $Q(\boldsymbol{x}, \boldsymbol{u}) = \sum_{i=1}^{d_x} \left(\theta_i (g_{j_i}(\boldsymbol{x}) u_{j_i} + f_{j_i}(\boldsymbol{x}) \right)^2 = \sum_{i=1}^{d_x} (\theta_i g^2_{j_i}(\boldsymbol{x}) u^2_{j_i} + 2\theta_i f_{j_i}(\boldsymbol{x}) g_{j_i}(\boldsymbol{x}) u_{j_i} + f^2_{j_i}(\boldsymbol{x}))$ After rearranging,$Q(\boldsymbol{x}, \boldsymbol{u}) = \sum_{i=1}^{d_x} (p_{x_i} u_i^2 + q_{x_i} u_i + r_{x_i})$.

□

Acknowledgements

The authors would like to thank Ivana Palunko for animation software, and Patricio Cruz for assisting with experiments. A. Faust is supported in part by New Mexico Space Grant. M. Salman is supported by the Computing Research Association CRA-W Distributed Research Experience for Undergraduates. R. Fierro is supported in part by NSF grant ECCS #1027775, and by the Army Research Laboratory grant #W911NF-08-2-0004. P. Ruymgaart and L. Tapia are supported in part by the National Institutes of Health (NIH) Grant P20GM110907 to the Center for Evolutionary and Theoretical Immunology.

References

[1] A. Al-Tamimi, F. Lewis, and M. Abu-Khalaf. Discrete-time nonlinear hjb solution using approximate dynamic programming: Convergence proof. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 38(4):943–949, 2008.

[2] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, 1st edition, 1996.

[3] S. Bhasin, N. Sharma, P. Patre, and W. Dixon. Asymptotic tracking by a reinforcement learning-based adaptive critic controller. *J of Control Theory and Appl*, 9(3):400–409, 2011.

[4] S. Bubeck, R. Munos, G. Stoltz, and C. Szepesvári. X-armed bandits. *J. Mach. Learn. Res.*, 12:1655–1695, July 2011.

[5] L. Buşoniu, R. Babuška, B. De Schutter, and D. Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators.* CRC Press, Boca Raton, Florida, 2010.

[6] L. Busoniu, A. Daniels, R. Munos, and R. Babuska. Optimistic planning for continuous-action deterministic systems. In *2013 Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, in press 2013.

[7] T. Cheng, F. L. Lewis, and M. Abu-Khalaf. A neural network solution for fixed-final time optimal control of nonlinear systems. *Automatica*, 43(3):482–490, 2007.

[8] T. Dierks and S. Jagannathan. Online optimal control of affine nonlinear discrete-time systems with unknown internal dynamics by using time-based policy update. *IEEE Transactions on Neural Networks and Learning Systems*, 23(7):1118–1129, 2012.

[9] D. Ernst, M. Glavic, P. Geurts, and L. Wehenkel. Approximate value iteration in the reinforcement learning context. application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1):1066.1–1066.37, 2005.

[10] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia. Automated aerial suspended cargo delivery through reinforcement learning. *Artificial Intelligence Journal*, page under submission, 2013.

[11] A. Faust, I. Palunko, P. Cruz, R. Fierro, and L. Tapia. Learning swing-free trajectories for uavs with a suspended load. In *IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany*, pages 4887–4894, May 2013.

[12] H. Hasselt. Reinforcement learning in continuous state and action spaces. In M. Wiering and M. Otterlo, editors, *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 207–251. Springer Berlin Heidelberg, 2012.

[13] Y. Jiang and Z.-P. Jiang. Computational adaptive optimal control for continuous-time linear systems with completely unknown dynamics. *Automatica*, 48(10):2699–2704, Oct. 2012.

[14] H. Khalil. *Nonlinear Systems.* Prentice Hall, 1996.

[15] H. Kimura. Reinforcement learning in multi-dimensional state-action space using random rectangular coarse coding and gibbs sampling. In *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, pages 88–95, 2007.

[16] J. Kober, D. Bagnell, and J. Peters. Reinforcement learning in robotics: A survey. *International Journal of Robotics Research*, 32(11):1236–1272, 2013.

[17] S. M. LaValle. *Planning Algorithms.* Cambridge University Press, Cambridge, U.K., 2006.

[18] A. Lazaric, M. Restelli, and A. Bonarini. Reinforcement learning in continuous action spaces through sequential monte carlo methods. *Advances in neural information processing systems*, 20:833–840, 2008.

[19] J. Levine. *Analysis and Control of Nonlinear Systems: A Flatness-based Approach.* Mathematical Engineering. Springer, 2010.

[20] C. Mansley, A. Weinstein, and M. Littman. Sample-based planning for continuous action markov decision processes. In *Proc. of Int. Conference on Automated Planning and Scheduling*, 2011.

[21] S. Mehraeen and S. Jagannathan. Decentralized nearly optimal control of a class of interconnected nonlinear discrete-time systems by using online Hamilton-Bellman-Jacobi formulation. In *International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2010.

[22] S. Mehraeen and S. Jagannathan. Decentralized optimal control of a class of interconnected nonlinear discrete-time systems by using online Hamilton-Jacobi-Bellman formulation. *IEEE Transactions on Neural Networks*, 22(11):1757–1769, 2011.

[23] H. Modares, M.-B. N. Sistani, and F. L. Lewis. A policy iteration approach to online optimal control of continuous-time constrained-input systems. *ISA Transactions*, 52(5):611–621, 2013.

[24] R. Sutton and A. Barto. *A Reinforcement Learning: an Introduction.* MIT Press, MIT, 1998.

[25] C. Taylor and A. Cowley. Parsing indoor scenes using rgb-d imagery. In *Proc. Robotics: Sci. Sys. (RSS)*, Sydney, Australia, July 2012.

[26] K. G. Vamvoudakis, D. Vrabie, and F. L. Lewis. Online adaptive algorithm for optimal control with integral reinforcement learning. *International Journal of Robust and Nonlinear Control*, 2013.

[27] T. J. Walsh, S. Goschin, and M. L. Littman. Integrating sample-based planning and model-based reinforcement learning. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*, pages 612–617. AAAI Press, 2010.

[28] T. Yucelen, B.-J. Yang, and A. J. Calise. Derivative-free decentralized adaptive control of large-scale interconnected uncertain systems. In *IEEE Conference on Decision and Control and European Control Conference (CDC-ECC)*, pages 1104–1109, 2011.