2013 European Control Conference (ECC)
July 17-19, 2013, Zürich, Switzerland.

Approximate Dynamic Programming via Sum of Squares Programming

Tyler H. Summers, Konstantin Kunz, Nikolaos Kariotoglou,
Maryam Kamgarpour, Sean Summers, and John Lygeros

Abstract— **We describe an approximate dynamic programming method for stochastic control problems on infinite state and input spaces. The optimal value function is approximated by a linear combination of basis functions with coefficients as decision variables. By relaxing the Bellman equation to an inequality, one obtains a linear program in the basis coefficients with an infinite set of constraints. We show that a recently introduced method, which obtains convex quadratic value function approximations, can be extended to higher order polynomial approximations via sum of squares programming techniques. An approximate value function can then be computed offline by solving a semidefinite program, without having to sample the infinite constraint. The policy is evaluated online by solving a polynomial optimization problem, which also turns out to be convex in some cases. We experimentally validate the method on an autonomous helicopter testbed using a 10-dimensional helicopter model.**

I. Introduction

Many problems in engineering and finance can be modeled as stochastic control problems on infinite state and input spaces, in which a control policy is sought to optimize the behavior of a stochastic dynamical system over a finite or infinite time horizon. While such models are quite general and expressive, the resulting optimization problems are extremely difficult because the decision variable (the control policy) is a *function*, which is generally infinite-dimensional and thus not amenable to computation or even storage on a computer. One general solution method is dynamic programming, which was developed in the seminal work of Bellman in the 1950s [1]. The solution utilizes the Bellman equation, which relates the problem data to the optimal value function and policy. However, solutions of the Bellman equation can be tractably obtained only in a few special cases, when the state and input spaces have very small dimension (hence, can be gridded) or when very strong assumptions are made on the problem data.

Approximate dynamic programming (ADP) is a collection of heuristic methods for solving stochastic control problems for cases that are intractable with standard dynamic programming methods [2, Ch. 6], [3]. The methods can be classified into three broad categories, all of which involve some kind of function approximation: (1) lookahead/rollout/receding horizon/model predictive control policies, (2) direct policy function approximation, (3) policies based on value function approximation. Here, we will focus on an approach in the last category in which the value function is approximated by a linear combination of pre-specified basis functions.

A method recently introduced by Wang and Boyd in [4] involves computing an approximate value function by relaxing the Bellman equation to an inequality. One can then obtain a linear program in the basis function coefficients with an infinite set of constraints, one for each state and input pair. In [4], the value function is approximated with quadratic basis functions, and the S-procedure (see e.g. [5, Appendix B.2]) is used to transform the infinite constraint set into a linear matrix inequality. The quadratic approximation is used to compute the control policy online, which for certain cases requires solving a small quadratic program. Furthermore, it can be shown that the approximation is a lower bound on the optimal value function and therefore can be used to obtain performance bounds for the system. Finally, the approximation minimizes a norm to the optimal value function, i.e. is in some sense an optimal projection onto the given basis set. The method of Wang and Boyd is based on a similar method from De Farias and Van Roy for finite state and input spaces in [6], in which various results on properties of the approximations and constraint sampling methods for finite spaces can be found. Kveton et al also obtain similar results for hybrid spaces [7].

In this paper, we approximate the value function with more general polynomial basis functions. Polynomial approximations are not new; in fact, Bellman himself studied such approximations in the early 1960s [8]. However, the study of polynomial approximations is now particularly interesting in light of recent developments in sum of squares programming [9] and efficient numerical solvers for semidefinite programming [10], [11]. In particular, it was shown in [9] that positivity of a polynomial can be ensured by testing for a sum of squares decomposition, which can then be expressed as a semidefinite program. In stochastic control problems, when the problem data are polynomials, the infinite constraint set in the linear program described above can be expressed as the positivity of a polynomial, which can then be expressed as a semidefinite program. Today, there are widely available numerical solvers for computing solutions to semidefinite programs, making polynomial approximations potentially attractive options for obtaining suboptimal but high-performing solutions to stochastic control problems on high-dimensional spaces.

The main contribution of the present paper is to demonstrate that higher order polynomial value function approximations can be obtained using sum of squares program-

T. H. Summers is supported by an ETH Zurich Postdoctoral Fellowship.
This research is partially supported by the European Commission under the project MoVeS, FP7-ICT-257005.

ming techniques. An approximate value function can then be computed offline by solving a semidefinite program, without having to sample the infinite constraint. The policy is evaluated online by solving a polynomial optimization problem; an additional sum of squares constraint can be added to ensure convexity of the approximate value function, making the policy evaluation efficiently computable in certain cases. We also experimentally validate the method using a quadratic value function approximation on an autonomous helicopter testbed. Hover and tracking controllers show good performance, and the policy can be computed online in a few tens of microseconds using recent code generation techniques [12], [13]. To the best of our knowledge, this is the first experimental implementation of this particular approximate dynamic programming method.

The paper is organized as follows. Section II formulates an infinite horizon stochastic control problem, summarizes the dynamic programming solution, and describes an approximate dynamic programming method based on approximating the value function with a linear combination of basis functions. Section III shows how sum of squares techniques can be used to obtain polynomial value function approximations when the problem data are polynomials. Section IV describes an implementation of an approximate dynamic programming controller on an experimental autonomous helicopter testbed. Section V gives concluding remarks and an outlook for future research.

II. Dynamic Programming and Approximate Dynamic Programming

In this section, we formulate an infinite horizon stochastic control problem. We then describe the dynamic programming solution and the approximate dynamic programming approach based on value function approximation. We use a somewhat informal mathematical style and deliberately set aside non-trivial (but manageable) measurability issues and questions associated with existence of superior non-Markovian or randomized policies to simplify the exposition; see [14] for a detailed and formal treatment.

A. An Infinite Horizon Stochastic Control Problem

Consider a discrete-time stochastic dynamical system

$$x^+ = f(x, u, w) \tag{1}$$

where $x \in \mathcal{X} \subseteq \mathbf{R}^n$ is the state, $u \in \mathcal{U} \subseteq \mathbf{R}^m$ is the input, and $w \in \mathcal{W} \subseteq \mathbf{R}^p$ is the stochastic process noise with given distribution. We assume that $\mathcal{X}$, $\mathcal{U}$, and $\mathcal{W}$ are closed sets and that w_t, $t = 0, \ldots$ are independent and identically distributed.

The goal is to find an admissible[1] state feedback policy $\psi : \mathcal{X} \to \mathcal{U}$ with $u_t = \psi(x_t)$ that minimizes the expected infinite horizon discounted objective function

$$J(x_0, \psi) = \mathbf{E}_w \sum_{t=0}^{\infty} \gamma^t \ell(x_t, u_t) \tag{2}$$

where $\gamma \in (0, 1)$ is a discount factor and $\ell : \mathcal{X} \times \mathcal{U} \to \mathbf{R}$ is the stage cost function. This is in general an extremely difficult infinite-dimensional non-convex optimization problem.

The solution can be expressed in principle via dynamic programming. The optimal value function $V^* : \mathcal{X} \to \mathbf{R}$ is defined as

$$V^*(x) = \inf_{\psi \in \Psi} J(x, \psi), \tag{3}$$

where Ψ is the set of admissible state feedback policies. The function V^* satisfies the Bellman equation

$$V^*(x) = \inf_{u \in \mathcal{U}} \left\{ \ell(x, u) + \gamma \mathbf{E}_w V^*(f(x, u, w)) \right\}. \tag{4}$$

The right-hand-side can be written as an operator on V^*

$$V^* = TV^*, \tag{5}$$

which has the following properties:

- monotonicity: $f \leq g \Rightarrow Tf \leq Tg$
- value iteration convergence:

$$V^*(x) = \lim_{k \to \infty} (T^k f)(x) \quad \forall f : \mathcal{X} \to \mathbf{R}.$$

The optimal policy is then given by

$$\psi^*(x) = \arg\min_{u \in \mathcal{U}} \left\{ \ell(x, u) + \gamma \mathbf{E}_w V^*(f(x, u, w)) \right\}. \tag{6}$$

The optimal value function and policy can only be efficiently computed and stored in a few special cases. When the state, control, and noise spaces are finite and small (approximately $|\mathcal{X}||\mathcal{U}||\mathcal{W}| \leq 10^8$), standard dynamic programming methods, e.g. value iteration, policy iteration, or linear programming, can be used to compute the optimal value function and policy. If the state, control, or noise spaces are infinite but have very small dimension (a continuous state space dimension of no more than 4 or 5), the optimal value function and policy can be approximated by gridding the spaces, approximating the dynamics by a finite state Markov chain, and computing the functions at the grid points. For these cases, the dynamic programming methods work for complicated problems but are limited by the so-called curse of dimensionality: computation and storage requirements grow exponentially with the problem dimensions.

For continuous spaces, there is one known case which is computationally tractable in high dimensions. If $\mathcal{X} = \mathbf{R}^n$, $\mathcal{U} = \mathbf{R}^m$, $\mathcal{W} = \mathbf{R}^p$, the dynamics are linear, the noise is Gaussian, and the stage costs are quadratic, then the optimal value function is quadratic, the optimal policy is affine, and both can be computed efficiently from the problem data (either by solving Riccati equations or semidefinite programs). The slightest complication however (e.g. any one of: input/state constraints, non-Gaussian noise, non-quadratic costs, nonlinear dynamics) makes computing the optimal value function extremely difficult. In such cases, we require systematic methods for approximating the optimal value function and policy.

[1] By admissible, we mean that the policy is causal and always satisfies the input constraints. One can formulate problems with state constraints that must be satisfied almost surely or in a probabilistic sense. However, the approximations described below are not guaranteed to satisfy these; this remains a significant open problem for using these methods. In this paper, state constraints are used to restrict the region of the state space in which the approximation is relevant.

B. Approximate Dynamic Programming

We now describe an approximate dynamic programming method that involves approximating the value function with a linear combination of pre-specified basis functions [4], [6]:

$$\hat{V}(x) = \sum_{i=1}^{k} \alpha_i \hat{V}_i(x) \tag{7}$$

where $\alpha \in \mathcal{A} \subseteq \mathbf{R}^k$. The coefficients α are computed offline by solving an optimization problem described below. Then the policy is evaluated online by substituting the approximate value function into the right-hand-side of (6) (leading to what is sometimes called a greedy policy):

$$\psi^{adp}(x) = \arg\min_{u \in \mathcal{U}} \left\{ \ell(x,u) + \gamma \mathbf{E}_w \hat{V}(f(x,u,w)) \right\}. \tag{8}$$

The idea is to find an optimal projection of the optimal value function onto the given set of basis functions and hope that the suboptimal policy (8) yields good performance.

Finding the Approximation via Bellman Inequalities: One method to obtain a value function approximation is to relax the Bellman equation into an inequality [4], [6]. The set of functions that satisfy the Bellman inequality are underestimators of the optimal value function. To see this, suppose a function $\hat{V}$ satisfies $\hat{V} \leq T\hat{V}$. Then by monotonicity of T and value iteration convergence we have

$$\hat{V} \leq T\hat{V} \leq T(T\hat{V}) \leq \cdots \leq \lim_{k\to\infty} T^k \hat{V} = V^*. \tag{9}$$

The Bellman inequality is only a sufficient condition for underestimation, i.e. the set of functions that satisfy the Bellman inequality may not include *all* underestimators. In [4], it is shown how "iterated" Bellman inequalities reduce this conservatism.

This suggests an optimization problem for finding the best value function underestimator in the span of the basis function set:

$$\begin{array}{ll} \text{maximize} & \displaystyle\int_{\mathcal{X}} c(x)\hat{V}(x)dx, \quad c(x) > 0 \\ \text{subject to} & \hat{V}(x) \leq T\hat{V}(x), \quad \forall x \in \mathcal{X} \end{array} \tag{10}$$

where $c(x)$ is any positive weighting function. The Bellman inequality constraint is convex in α and can be converted into an infinite set of linear constraints: note that the expression inside the $\inf$ in $T\hat{V}$ is affine in α for each u, since expectation is a linear operator, and that the infimum over a family of affine functions is concave and on the right side of the inequality

$$\underbrace{\hat{V}(x)}_{\text{affine in } \alpha} \leq \underbrace{\inf_{u\in\mathcal{U}} \overbrace{\left\{ \ell(x,u) + \gamma \mathbf{E}_w \hat{V}(f(x,u,w)) \right\}}^{\text{affine in } \alpha}}_{\text{concave in } \alpha}.$$

By simply removing the $\inf$ and enforcing the affine constraints for every u, we end up with a linear program with an infinite set of constraints, one for every state and input pair:

$$\begin{array}{ll} \text{maximize} & \displaystyle\int_{\mathcal{X}} c(x)\hat{V}(x) \\ \text{subject to} & \hat{V}(x) \leq \ell(x,u) + \gamma \mathbf{E}_w \hat{V}(f(x,u,w)), \\ & \forall x \in \mathcal{X},\ \forall u \in \mathcal{U} \end{array} \tag{11}$$

The solution to (11) can be shown to be an optimal projection onto the span of the basis functions that satisfy the Bellman inequality in that it minimizes a c-weighted 1-norm to the optimal value function [4], [6]. This gives an interpretation of $c(x)$ as a "state-relevance weight", which can be thought of as a distribution over the state space; higher weight can be given to regions of the state space in which we would like better approximation.

In addition to suboptimal policies, value function underestimators also provide performance bounds. For each $x \in \mathcal{X}$ we obtain a number that bounds the performance achievable with *any* admissible state feedback policy. Actual performance of suboptimal policies can be evaluated by Monte Carlo simulation. If the actual performance is close to the bound, the suboptimal policy is close to optimal. Otherwise, either the bound is bad or the policy is bad.

To apply this method in practice, one needs to resolve three main difficulties: (1) evaluating the high-dimensional expectations/integrals, (2) handling the infinite constraint set, and (3) evaluating the policy. Closely related to these issues is the choice of basis functions. We focus on polynomial basis functions. The difficulty in evaluating the expectations/integrals depends on whether or not there are state constraints. When there are no state constraints (or when state constraints are not explicitly taken into account in the integration, as in [4]), all that is needed are the moments of the noise distribution, which we assume to be given or computable from the given distribution. If state constraints are to be explicitly accounted for, certain assumptions must be made on the noise distribution and constraints. If the constraints are boxes and the noise distribution is (approximated by) a polynomial over the constraint set, the integrals can be calculated analytically. The infinite constraint set can be handled in some cases through an appropriate transformation (discussed below); otherwise, one must resort to constraint sampling, incurring more error and losing any theoretical guarantees. To evaluate the policy, an optimization problem (8) must be solved online, which itself is difficult in general even after a good approximate value function is computed. In certain cases (discussed below), this can be made a convex optimization problem and thus amenable to be solved online.

Wang and Boyd [4] address these issues by using quadratic basis functions. They do not explicitly take into account state constraints in evaluating the integrals, using only the moments of the noise distribution. To handle the infinite constraint, the S-procedure is used to convert it to a linear matrix inequality. By restricting the quadratic approximation to be convex, the policy can be evaluated online by solving a small quadratic program. In the next section, we show how

to extend this approach and approximate value functions by polynomials by utilizing sum or squares programming.

III. Polynomial Value Function Approximations via Sum of Squares Programming

In this section, we show how sum of squares programming techniques can be used to find higher-order polynomial value function approximations. Polynomial value functions were studied by Bellman himself in the early 1960s [8]. However, recent developments in sum of squares programming and semidefinite programming motivate further study.

A. Sum of Squares and Semidefinite Programming

In general, determining whether a given multivariate polynomial is everywhere positive is decidable but NP-hard [9]. An obvious sufficient condition for a polynomial to be positive is for it to be a sum of squares, i.e. be expressible as a finite sum of squared polynomials. We denote the set of all polynomials in $x \in \mathbf{R}^n$ with real coefficients by $\mathbf{R}[x]$. The set of sum of squares polynomials is denoted by

$$SOS = \left\{ F(x) \in \mathbf{R}[x] | F(x) = \sum_i f_i^2(x),\ f_i(x) \in \mathbf{R}[x] \right\}.$$

and forms a cone in $\mathbf{R}[x]$.

Let $F(x) \in \mathbf{R}[x]$ be a polynomial of degree $2d$. Let z be a vector of monomials of degree less that or equal to d; in particular, $z(x) = [1, x_1, x_2, ..., x_n, x_1x_2, ..., x_n^d]^T$. The following result from [9] leads to an efficient method to test for a sum of squares decomposition.

Theorem 1: The polynomial $F(x)$ is a sum of squares if and only if there exists a symmetric positive semidefinite matrix Q such that $F(x) = z^T(x)Qz(x)$.

By choosing a positive semidefinite matrix subject to affine constraints defined by matching coefficients of $F(x)$ with an expansion of a quadratic form in the monomial vector z, one can ensure that $F(x)$ is a sum of squares. This is a semidefinite programming feasibility problem in primal form, which is a convex optimization problem and can be solved efficiently using various well-developed software packages. This method can be used if the coefficients of $F(x)$ must be chosen to satisfy some additional affine constraints.

Sum of Squares S-procedure [9]: Now suppose that we want to choose the coefficients of a polynomial so that it is positive over a given semialgebraic set defined by polynomial inequalities, i.e. choose coefficients of $p(x)$ such that $p(x) \geq 0,\ \forall x : g(x) \geq 0$, where $g(x) \in \mathbf{R}[x]$. A sufficient condition for this is the existence of a positive polynomial multiplier $\lambda(x) \geq 0$ such that $p(x) - \lambda(x)g(x) \geq 0$. This can then be restricted to a sum of squares program in which we choose the coefficients of λ and p subject to $\lambda(x) \in SOS$ and $p(x) - \lambda(x)g(x) \in SOS$, which is again a semidefinite program.

B. Polynomial Approximate Dynamic Programming

Now suppose that all the problem data are polynomial: the dynamics f is polynomial in x, u, and w, the stage cost ℓ is polynomial in x and u, and the constraint sets $\mathcal{X}$ and $\mathcal{U}$ can be described by polynomial inequalities, i.e. $x \in \mathcal{X}, u \in \mathcal{U} \Leftrightarrow g(x,u) \geq 0$. Suppose also that the value function approximation is a d degree polynomial in x with coefficients as decision variables. Note that none of these polynomials need to be convex.

The optimization problem (11) can be written

$$\begin{array}{ll} \text{maximize} & b^T\alpha \\ \text{subject to} & p(x,u,\alpha) \geq 0, \quad \forall g(x,u) \geq 0 \end{array} \tag{12}$$

where $p(x,u,\alpha)$ is a polynomial in x and u and affine in α. Here, we have assumed that the expectations/integrals are either computed analytically or computed based on the moments of the weight and distribution over the state and disturbance space; these computations enter in the entries of b and in the coefficients of p.

We have now a linear optimization problem subject to a constraint that a polynomial is positive on a semialgebraic set, in which all decision variables appear affinely. We can restrict the positivity constraint to a sum of squares constraint using the sum of squares S-procedure to obtain

$$\begin{array}{ll} \text{maximize} & b^T\alpha \\ \text{subject to} & p(x,u,\alpha) - \lambda(x,u)g(x,u) \in SOS \\ & \lambda(x,u) \in SOS. \end{array} \tag{13}$$

Via Theorem 1, this can be directly transformed into a semidefinite program with variables α and coefficients of λ, which can be solved offline.

C. Numerical Example

To illustrate that higher-order polynomials give improved approximations, we compute quadratic and quartic value function approximations for a 1D problem with linear dynamics, quadratic stage cost, unit-variance Gaussian noise distribution, and state and input constraints:

$$x^+ = x - 0.5u + w, \quad \ell(x,u) = x^2 + u^2, \quad \gamma = 0.99$$

$$\mathcal{X} = [-20, 20],\ \mathcal{U} = [-1, 1] \Leftrightarrow x^2 \leq 400,\ u^2 \leq 1.$$

Figure 1 shows families of quadratic (blue) and quartic (green) approximations for various weighting functions $c(x)$. The families give similar approximations close to the origin, but the quartic approximations are clearly better near the state constraint boundaries, since all approximations are guaranteed to be underestimators of the optimal value function.

D. Evaluating Policies

Once we have computed the coefficients of the approximate value function via (13), the policy is evaluated online by solving (8) For a given state x, this is a polynomial optimization problem in variable u (assuming again the integrals are analytically computable). In general, this is difficult, but we now discuss tractable cases and general techniques.

The policy evaluation (8) is a convex optimization problem whenever $\mathcal{U}$ is convex and the right-hand-side of (8) is a convex function of u, which is true if $\hat{V}$ is convex, ℓ is convex in u, and the dynamics are input affine, i.e. $f(x,u,w) = f_1(x,w) + f_2(x,w)u$, since the composition

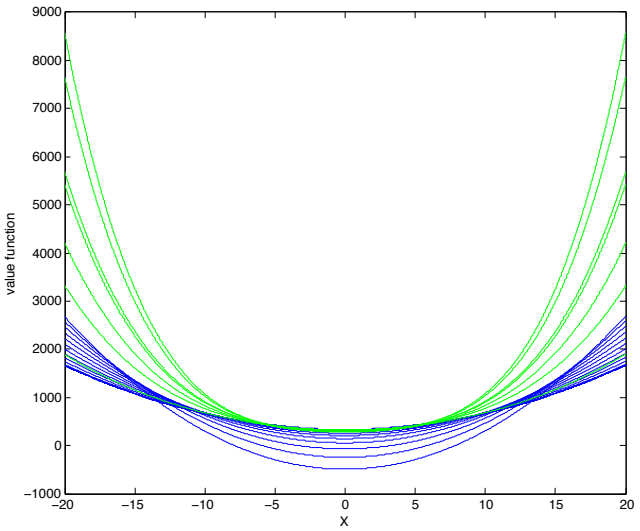

Fig. 1. Families of quadratic (blue) and quartic (green) value function approximations.

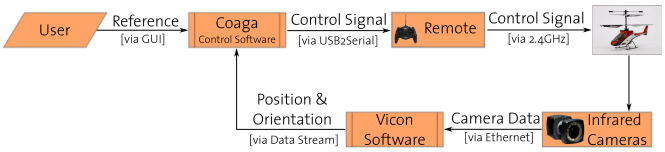

Fig. 2. The helicopter testbed setup.

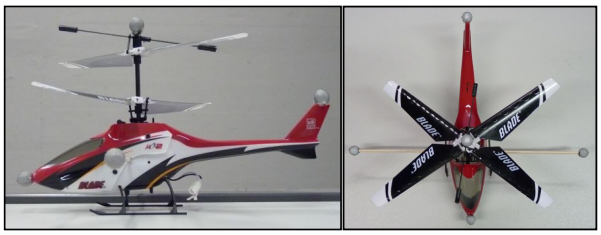

Fig. 3. Blade mCX2 micro-coaxial helicopters

of a convex function with an affine function is convex, and the sum of convex functions is convex. It turns out that we can enforce convexity of the value function approximation by adding an additional sum of squares constraint on the Hessian of $\hat{V}$ [15], [16]. A polynomial $\hat{V}(x)$ is convex if and only if $y^T \nabla^2 \hat{V}(x) y \geq 0, \ \forall x, y \in \mathcal{X}$. Since this is another polynomial inequality, we can ensure convexity of $\hat{V}$ simply by adding the sum of squares constraint $y^T \nabla^2 \hat{V}(x) y \in SOS$ to the semidefinite program (13). We have a tradeoff: adding another constraint can only make the approximation worse, even if the optimal value function is convex. However, even if the optimal value function is not known to be convex, it may be advantageous to make the policy evaluation (8) tractable. In these cases, the policy can be evaluated with gradient or interior point methods. Furthermore, since (8) is relatively small (the number of decision variables is the dimension of the input), recent code generation techniques [12], [13] can be used to make the online policy evaluation extremely fast (on the order of microseconds with cheap, embedded processors).

For cases in which a convex approximation is undesirable, there are other techniques that can be considered from a growing literature on general polynomial optimization [17]–[19]. For example, sum of squares relaxations have been recently explored and observed to recover globally optimal solutions, even in non-convex cases [18], [20], [21].

IV. Experimental Validation

We experimentally validated the approximate dynamic programming method on an autonomous helicopter testbed, which we now briefly describe. Further details can be found in the related paper [22].

A. System Architecture

The helicopter navigates in an indoor space equipped with four Vicon Bonita infrared cameras (Vicon product specifications can be found at http://www.vicon.com), which emit infrared light via diodes and record the reflections of markers placed on the helicopter. The camera data is processed by the Vicon Tracker software, which calculates the position and orientation of the helicopter in space. The Coaga [23] custom helicopter control software framework developed by the rCopterX project at ETH Zurich (http://www.rcopterx.ethz.ch) is used to compute the control input from a user-defined desired trajectory and the position and orientation measurements. Coaga is an object-oriented control framework written in C++ to enable fast computations and real-time applications and runs on a stand-alone desktop computer. It also includes a Kalman filter for the estimation of translational and rotational velocities of the helicopter and a feasible trajectory generator based on differential flatness of the model. The computed control input is sent to a remote control via USB. The remote control then transmits the control input directly to the helicopter on a 2.4 GHz frequency. Figure 2 shows a block diagram of the experimental setup.

The helicopter used in the experiments is a 28-gram Blade mCX2 miniature coaxial helicopter shown in Figure 3. The helicopter is augmented by infrared reflectors so that it can be tracked with the Vicon camera system. It has four control inputs: pitch (for forward flight), roll (for sideways flight), thrust (for vertical flight) and yaw (for heading change). Pitch and roll are actuated with a swashplate mechanism connected to two servos. The thrust is set by the rotation speed of the main rotor motors and yaw is actuated by a rotational speed difference between the two main rotors.

The software used in the experimantal setup consists of Coaga and the Vicon Tracker Software. A convex quadratic value function approximation was computed offline using the semidefinite programming solver SeDuMi [11] in the convex optimization modeling framework CVX [24]. We chose a quadratic approximation for validation because this is the simplest type of polynomial that yields a good approximation, and because the resulting quadratic program to be solved online could be done easily with available software

and hardware. The ADP online control policy evaluation described above was implemented in C++ and integrated into the Coaga software. The quadratic program was solved with code generated by the CVXGEN code generator [12] and called in Coaga.

B. Simplified Helicopter Model

We used a simplified grey-box model whose parameters were identified based on a least-squares criterion between experimental step-response data and the model prediction. The states used to describe the system are the position and heading in an inertial reference frame X_I, Y_I, Z_I, Ψ, the velocity in body-frame $\dot{X}_B$, $\dot{Y}_B$, $\dot{Z}_B$, $\dot{\Psi}$ and two integral states X_{int}, Y_{int} in x and y to reduce steady-state error. The pitch and roll angles of the helicopter are neglected due to the small size of the helicopter. The yaw angle Ψ is defined from an inertial-frame x-axis to the body-frame x-axis.

The continuous-time state-space representation of the simplified model are

$$
\begin{aligned}
\dot{x} &= Ax + Bu + c \\
x &= \begin{bmatrix} X_I & Y_I & Z_I & \Psi & \dot{X}_B & \dot{Y}_B & \dot{Z}_B & \dot{\Psi} & X_{int} & Y_{int} \end{bmatrix}^T \\
u &= \begin{bmatrix} u_x & u_y & u_z & u_\Psi \end{bmatrix}^T \\
c &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & -k_i x_{ref} & -k_i y_{ref} \end{bmatrix}^T
\end{aligned}
$$

$$
A = \begin{bmatrix}
0 & 0 & 0 & 0 & \cos(\Psi^*) & -\sin(\Psi^*) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \sin(\Psi^*) & \cos(\Psi^*) & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & k_x & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & k_y & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & k_\Psi & 0 & 0 \\
k_i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & k_i & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

$$
B = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
b_x & 0 & 0 & 0 \\
0 & b_y & 0 & 0 \\
0 & 0 & b_z & 0 \\
0 & 0 & 0 & b_\Psi \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0
\end{bmatrix}, \quad
\begin{bmatrix} b_x \\ k_x \\ b_y \\ k_y \\ b_\Psi \\ k_\Psi \\ b_z \end{bmatrix} = \begin{bmatrix} 2.0 \\ -0.5 \\ 2.1 \\ -0.5 \\ 111.0 \\ -5.0 \\ 18 \end{bmatrix}
$$

where Ψ^* is the current yaw state around which the system is linearized, the parameters k_x, k_y , k_Ψ represent fuselage drag, b_x, b_y , b_z, b_Ψ represent the influence of the inputs u on the states x, and $g = 9.81m/s^2$ is gravitational acceleration. These dynamics are time-discretized for implementation. We assumed that unit-variance Gaussian noise acts on accelerations, and we used a quadratic stage cost function.

C. Results

Both hover and trajectory tracking controllers were developed and tested. The hover performance in x and y can be seen in Figure 4. The maximum deviations in X_I and Y_I are 3.2cm and 6.8cm, respectively. Trajectory tracking was assessed by having the helicopter fly a box-shaped trajectory with a side length of 1m. A feasible trajectory is precomputed based on the identified helicopter model. The controller tracks both the states and inputs. The results are shown in

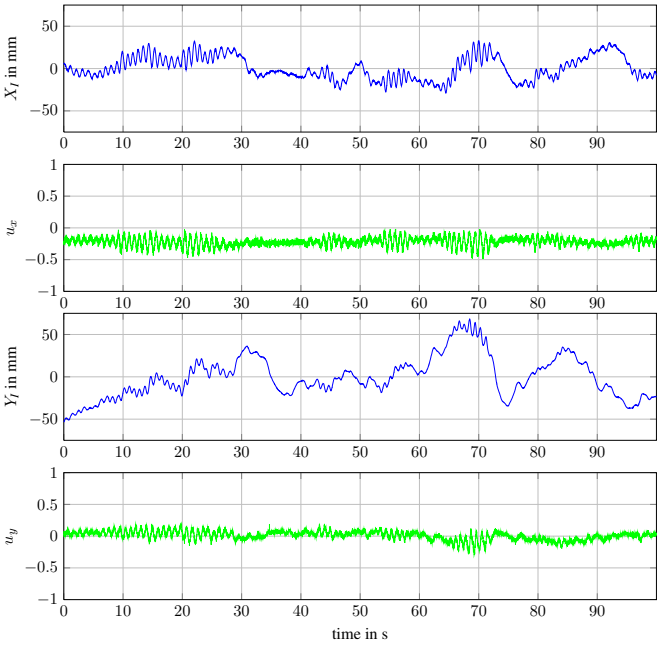

Fig. 4. ADP controlled helicopter hovering

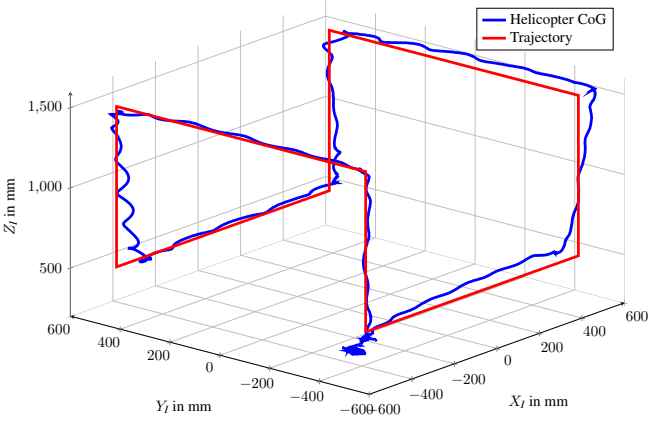

Fig. 5. ADP controlled helicopter flying a box trajectory

Figure 5. The results in both hover and trajectory tracking compare favorably with more standard PID and MPC-based controllers reported in [22]. Furthermore, the policy can be evaluated online in a few tens of microseconds, allowing for kilohertz sampling rates (though here we were limited by communication hardware to 50 hertz).

V. Concluding Remarks

We have described an approximate dynamic programming method on infinite state and control spaces. We showed how sum of squares techniques can be used to compute polynomial value function approximations offline via semidefinite programming. The policy is computed online by solving a polynomial optimization problem, which can be made convex in certain cases. Future work will include exploring various application domains, focusing in particular on what can be gained by using higher-order polynomial approximations.

Also, the methods can be applied in stochastic reachability problems, which is explored in a companion paper [25] via radial basis functions and constraint sampling techniques.

REFERENCES

[1] R. Bellman, "Bottleneck problems and dynamic programming," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 9, p. 947, 1953.

[2] D. Bertsekas, *Dynamic programming and optimal control*, 4th ed. Athena Scientific Belmont, MA, 2012, vol. 2.

[3] W. Powell, *Approximate dynamic programming: Solving the curses of dimensionality*. Wiley-Interscience, 2007, vol. 703.

[4] Y. Wang and S. Boyd, "Approximate dynamic programming via iterated Bellman inequalities," *Manuscript preprint*, 2010.

[5] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[6] D. de Farias and B. Van Roy, "The linear programming approach to approximate dynamic programming," *Operations Research*, vol. 51, no. 6, pp. 850–865, 2003.

[7] B. Kveton, M. Hauskrecht, and C. Guestrin, "Solving factored MDPs with hybrid state and action variables," *Journal of Artificial Intelligence Research*, vol. 27, no. 1, pp. 153–201, 2006.

[8] R. Bellman, R. Kalaba, and B. Kotkin, "Polynomial approximation–a new computational technique in dynamic programming: Allocation processes," *Mathematics of Computation*, vol. 17, no. 82, pp. 155–161, 1963.

[9] P. Parrilo, "Semidefinite programming relaxations for semialgebraic problems," *Mathematical Programming*, vol. 96, no. 2, pp. 293–320, 2003.

[10] L. Vandenberghe and S. Boyd, "Semidefinite programming," *SIAM review*, vol. 38, no. 1, pp. 49–95, 1996.

[11] J. Sturm, "Using sedumi 1.02, a matlab toolbox for optimization over symmetric cones," *Optimization methods and software*, vol. 11, no. 1-4, pp. 625–653, 1999.

[12] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, pp. 1–27, 2012.

[13] A. Domahidi, M. Zeilinger, C. Jones, and M. Morari, "Efficient interior point methods for multistage problems arising in receding horizon control," in *to appear in Proceedings of the 51st IEEE Conference on Decision and Control*. IEEE, 2012.

[14] D. Bertsekas and S. Shreve, *Stochastic optimal control: The discrete time case*. Academic Press NY, 1978, vol. 139.

[15] A. Magnani, S. Lall, and S. Boyd, "Tractable fitting with convex polynomials via sum-of-squares," in *44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 1672–1677.

[16] A. Ahmadi and P. Parrilo, "A complete characterization of the gap between convexity and SOS-convexity," *arXiv preprint arXiv:1111.4587*, 2011.

[17] J. Lasserre, "Global optimization with polynomials and the problem of moments," *SIAM Journal on Optimization*, vol. 11, no. 3, pp. 796–817, 2001.

[18] P. Parrilo and B. Sturmfels, "Minimizing polynomial functions," *Algorithmic and quantitative real algebraic geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, vol. 60, pp. 83–99, 2003.

[19] J. Nie, "Global optimization of polynomial functions and applications," Ph.D. dissertation, University of California, Berkeley, 2006.

[20] J. Lasserre, "Convergent SDP-relaxations in polynomial optimization with sparsity," *SIAM Journal on Optimization*, vol. 17, no. 3, pp. 822–843, 2006.

[21] J. Nie, J. Demmel, and B. Sturmfels, "Minimizing polynomials via sum of squares over the gradient ideal," *Mathematical programming*, vol. 106, no. 3, pp. 587–606, 2006.

[22] K. Kunz, S. Huck, T. Summers, and J. Lygeros, "Fast model predictive control of miniature helicopters," in *Submitted to the European Control Conference*. IEEE, 2013.

[23] J. Amstuz, R. Bernhard, T. Wellerdieck, and F. Wermelinger, "Coaga wiki," Website, 2012, available online at https://bitbucket.org/raffber/coaga/wiki/Home; visited on September 2th 2012.

[24] M. Grant, S. Boyd, and Y. Ye, "CVX: Matlab software for disciplined convex programming," *Online accessiable: http://stanford. edu/˜ boyd/cvx*, 2008.

[25] N. Kariotoglou, S. Summers, T. Summers, M. Kamgarpour, and J. Lygeros, "Approximate dynamic programming for stochastic reachability," in *Submitted to the European Control Conference*. IEEE, 2013.