Linear Bellman Combination for Control of Character Animation

Marco da Silva[1] Frédo Durand[1] Jovan Popović[1,2,3]

[1]Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

[2]Advanced Technology Labs, Adobe Systems Incorporated

[3]University of Washington

Abstract

Controllers are necessary for physically-based synthesis of character animation. However, creating controllers requires either manual tuning or expensive computer optimization. We introduce linear Bellman combination as a method for reusing existing controllers. Given a set of controllers for related tasks, this combination creates a controller that performs a new task. It naturally weights the contribution of each component controller by its relevance to the current state and goal of the system. We demonstrate that linear Bellman combination outperforms naive combination often succeeding where naive combination fails. Furthermore, this combination is provably optimal for a new task if the component controllers are also optimal for related tasks. We demonstrate the applicability of linear Bellman combination to interactive character control of stepping motions and acrobatic maneuvers.

CR Categories: I.3.7 [Computing Methodologies]: Computer Graphics—Three-Dimensional Graphics and Realism; I.2.8 [Computing Methodologies]: Problem Solving, Control Methods, and Search—Control Theory;

Keywords: Physically Based Animation, Optimal Control

1 Introduction

Physically-based animation of active characters requires controllers that find forces that achieve the goals of the animator. While controllers exist for a variety of human activities [Yin et al. 2007; Coros et al. 2008; Hodgins et al. 1995; Zordan and Hodgins 2002; Sok et al. 2007] and even complex, non-human activities [McNamara et al. 2004; Fattal and Lischinski 2004; Bergou et al. 2007], creating a controller that achieves the animator's goal usually involves either intensive manual tuning or expensive computational optimization.

Given the difficulty of creating individual controllers for given tasks, we seek to reuse controllers for new tasks by combining them. For example, if two controllers can reach different goals, we want to blend them to reach end states in between. Simple linear combination of controllers is not sufficient and can lead to unpredictable results due to the complex relationship between control forces and the resulting animation. In this paper, we present a new method for combining controllers that naturally weights each component controller by its relevance for the current state and goal. Combined controllers can interpolate goals or coordinate between multiple controllers. Coordination between controllers trained for different parts of state space can broaden the domain that can be handled. Coordination between controllers trained for different goals can yield a controller with multiple acceptable outcomes.

Our approach is based on the control formulation where controllers (equivalently, control policies) are measured by an objective function which quantifies how well they achieve the goals of the animator [Bertsekas 2007; Todorov 2006a; Witkin and Kass 1988]. For example, this function can emphasize reaching a particular state at a particular time or tracking a given trajectory for the character. In computer animation, improving a controller with respect to this objective is difficult because there are often many control parameters. Also, the relationship between these parameters and the resulting value of the objective function is non-linear, due to the cost functions used to describe the task and the dynamics that govern the motion of the character. This relationship is encapsulated by a partial differential equation, the Hamilton-Jacobi-Bellman (HJB) equation, whose solution defines the optimal *value* function [Bellman 1957]. The optimal value function maps a character state to the lowest possible cost of reaching the goal. The best policy seeks to follow the gradient of the optimal value function. Non-linearity of the HJB equation is the reason tuning, optimizing, or combining controllers is so difficult.

Our approach, which we call linear Bellman combination, derives from the remarkable fact that the HJB equation can be made linear for certain systems using a change of variables [Fleming 1978; Kappen 2005; Todorov 2006b]. In this paper, we observe that the linear superposition principle allows us to combine temporally aligned optimal policies for similar tasks with different boundary conditions. The resulting controller is provably optimal for the new task under some conditions. In practice, we need to combine controllers that might not be optimal. Still, linear Bellman combination performs accurately and robustly and we provide empirical evidence that it outperforms alternatives. Furthermore, it is simple to use and can be applied to many control schemes as it treats controllers as a black box, using only their outputs to compute combinations.

Evaluating a controller using linear Bellman combination requires representations of each component controller's policy and value function. Often these mappings are lacking and the common approach for low-dimensional systems is to tabulate the policy and value function. In this paper, we also describe a method capable of controlling character motions with many more degrees of freedom. We demonstrate linear Bellman combination on value functions which are approximated around samples from example trajectories.

Control-inspired approaches are used in many areas of graphics, offering many potential applications of the linear Bellman equation. For this paper, we focus on several animation tasks. As an example application, we demonstrate taking steps of varying length and jumps of different heights. Controllers that bring a gymnast to an upright position on a high bar from different initial conditions are combined to work from a range of initial conditions. We use coordination to combine controllers for walking on flat and hilly terrain. We show that linear Bellman combination outperforms simple linear blending and other tracking approaches. The main limitation of the presented formulation is that combined controllers must cover the same duration of time. In the conclusion, we sketch possible extensions to address this limitation.

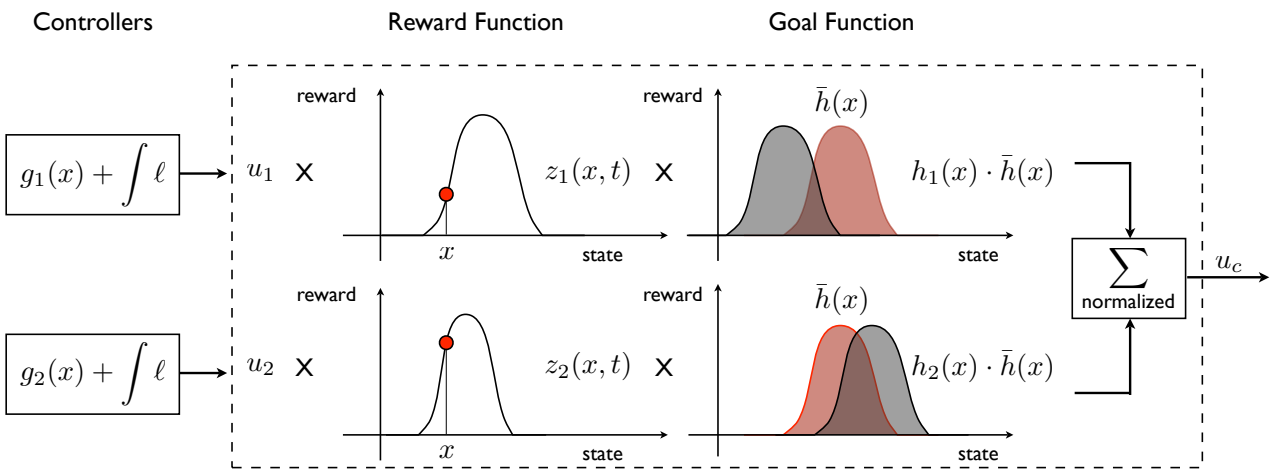

Figure 1: Overview. *Linear Bellman combination creates a new controller using existing controllers optimized for related tasks of finite duration. The tasks share a common* integrated *cost,* $\int_0^T q(\boldsymbol{x}) + \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{u}$ dt*, but have different terminal costs* $g_i(\boldsymbol{x})$*. The output of each controller,* $\boldsymbol{u}_i$*, is scaled by a reward function,* $z_i(\boldsymbol{x},t) = \exp\big(-v(\boldsymbol{x},t)\big)$ *and by the amount the controller's terminal reward function,* $h_i(\boldsymbol{x}) = g_i(\boldsymbol{x})$*, overlaps the desired terminal reward function,* $\bar{h}(\boldsymbol{x})$*. These weighting factors are normalized and the resulting control action is a sum as in Equation 11.*

2 Related Work

The goal of our work is to combine existing controllers to perform new tasks. Others have explored parameter tuning for specific control schemes to achieve new tasks such as adapting running controllers from larger characters to smaller characters [Hodgins and Pollard 1997], modifying jumping controllers [Pollard and Behmaram-Mosavat 2000], or walking under various environmental constraints [Yin et al. 2008]. In contrast, we treat existing controllers as a black box which we do not modify directly. These existing controllers may be hand-crafted [Wooten and Hodgins 2000] or computer optimized [Witkin and Kass 1988]. Rather than manipulate the parameters of the controller, our method achieves new tasks by combining the output of each component controller.

Some prior work has explored interpolation of existing controllers in order to achieve new tasks [van de Panne et al. 1994]. Yin et al [2008] observed that simple linear interpolation of SIMBICON [Yin et al. 2007] parameters can be used to adapt the walking controller to conditions that are "in-between" the conditions expected by example controllers. This observation was expanded upon to build a controller capable of taking different step lengths to avoid holes in the terrain [Coros et al. 2008]. In this work, we show that simple linear interpolation of feedback controllers can be suboptimal both in terms of the goal and the amount of control effort being exerted. In some cases, sub-optimality results in visual imperfections while in others it results in failure to accomplish the task. In practice, this can be overcome by placing optimized examples closer and closer together, but these optimized examples are difficult to obtain.

Controllers can also be sequenced to create composite controllers that achieve new tasks [Burridge et al. 1999; Faloutsos et al. 2001]. In our approach, multiple component controllers may be active at any one time. The total control effort is a weighted sum of the output of each component controller. In contrast to a sequencing approach, combining controllers allows us to achieve tasks that are not executed by any of the component controllers.

Sok et al [2007] and Erez et al [2007] also interpolate example controllers to achieve robust execution of biped locomotion maneuvers. These approaches rely on hand-crafted metrics to determine the blend weights to place on each example controller. In contrast, the metric used to weight component controllers in our approach is automatically computed given a mathematical description of the current goal. This allows our composite controller to look ahead to determine the best course of action to achieve this goal.

Linear Bellman combination builds on the observation that the non-linear HJB equation can be linearized under a change of variables [Fleming 1978]. The same general idea also appears in the study of stochastic diffusion processes [Holland 1977]. Recently, this linearity has been exploited to derive methods for computing low-dimensional optimal value functions using a power method [Todorov 2006b] and importance sampling [Kappen 2005]. In our work, we observe that linearity of the HJB equation allows one to apply linear superposition to combine existing controllers for new tasks for a variety of systems. Concurrent with our work, Todorov also explored linearity for the composition of control laws and applied it to a first-exit control problem of a discrete two-dimensional particle [2009].

For high-dimensional characters, linear Bellman combination uses multiple quadratic expansions of the value function in a non-parametric representation to track multiple trajectories. A related non-parametric representation was proposed by Atkeson et al [2002]. In our case, example trajectories need not be optimized for the same task. In addition, rather than switch between example controllers discretely using whichever example is closest to the current state, controls are smoothly interpolated using the value function. Some recent work [da Silva et al. 2008b; Barbič and Popović 2008] has adapted feed-forward controllers using linear quadratic regulators (LQR) to track a single example trajectory. Earlier work on this topic was pursued by Brotman and Netravali in graphics [1988] and Atkeson in robotics [1994]. However, LQR controllers are only valid near the example trajectory. We show that linear Bellman combination can coordinate multiple controllers to improve controller robustness.

3 Linear Bellman Combination

Linear Bellman combination produces a new controller by combining existing control policies. For example, given two policies for stepping near and far, the combined policy can step in between. A control policy $\boldsymbol{u} = \boldsymbol{\pi}(\boldsymbol{x},t)$ defines the actions (forces) $\boldsymbol{u} \in \mathbb{R}^m$

needed to drive the system from the current state $\boldsymbol{x} \in \mathbb{R}^d$ and time t to some goal in the future. Liner Bellman combination blends a set of control policies $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_n$ to produce a combined policy:

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \alpha_1(\boldsymbol{x}, t)\boldsymbol{\pi}_1(\boldsymbol{x}, t) + \cdots + \alpha_n(\boldsymbol{x}, t)\boldsymbol{\pi}_n(\boldsymbol{x}, t).$$

Setting these coefficients, α_i, such that the resulting controller achieves a well defined goal can be difficult given the non-linear dynamics of the systems typically controlled in physically based animation. Linear Bellman combination enables a principled derivation of these coefficients.

Linear Bellman combination uses carefully computed time-varying coefficients,

$$\alpha_i(\boldsymbol{x}, t) = \frac{w_i z_i(\boldsymbol{x}, t)}{\sum_{i=1}^{n} w_i z_i(\boldsymbol{x}, t)}, \tag{1}$$

that can—under certain conditions—generate *optimal* control actions for a well defined task. The blending formula weights each component policy by how appropriate it is for the current state and goal (see Figure 1). The reward function, $z_i(\boldsymbol{x}, t)$, indicates how appropriate the policy is for the current state, while the fixed (in-time) weights, w_i, determine the goal of the controller. The time-varying reward weights are computed automatically given the controller's value function. The fixed weight parameters are also computed automatically given a new desired goal function. The resulting control can either interpolate goal functions or coordinate the component policies. Coordination combines controllers that share the same goal but were created for different regions of state space. Coordination can also create a goal function with multiple acceptable outcomes.

The blending formula is derived from a linear form of the Bellman equation. In this section, we review the Bellman equation to associate solving this partial differential equation with the computation of optimal control policies. We then state the linear form of this partial differential equation and apply the linear superposition principle to derive Linear Bellman combination. Finally, we discuss the conditions for which this combination yields an optimal control policy.

3.1 Bellman Equation

An optimal control policy minimizes an objective function of the form

$$\min_{\boldsymbol{\pi}} g(\boldsymbol{x}_T) + \int_0^T \ell(\boldsymbol{x}, \boldsymbol{\pi}, t)\, \mathrm{d}t, \tag{2}$$

where ℓ, is a loss function applied at every time instant, and the desired state at the end is described by the terminal cost function, $g(\boldsymbol{x})$. Typical loss functions penalize muscle exertion or deviation from some desired target. The time evolution of the character state is constrained by the equations of motion for the system. We restrict ourselves to controlling systems with control-affine, stochastic dynamics:

$$\mathrm{d}\boldsymbol{x} = [\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{u}]\mathrm{d}t + \boldsymbol{C}(\boldsymbol{x})\mathrm{d}\boldsymbol{\omega}, \tag{3}$$

where $\boldsymbol{a} : \mathbb{R}^d \mapsto \mathbb{R}^d$ and $\boldsymbol{B} : \mathbb{R}^d \mapsto \mathbb{R}^{d\times m}$ are generally non-linear, continuous functions of state, and $\boldsymbol{u} \in \mathbb{R}^m$ is the control action. The passive dynamics of the system are described by $\boldsymbol{a}(\boldsymbol{x})$. The Brownian noise $\boldsymbol{\omega} \in \mathbb{R}^d$ is used to model zero-mean physical disturbances and modeling error with variance $\boldsymbol{C}\boldsymbol{C}^\top$.

The cost-to-go function $v^{\boldsymbol{\pi}}(\boldsymbol{x}, t)$ gives the expected cost of running a given policy $\boldsymbol{\pi}$ starting from any state until the terminal time under the objective defined by Equation 2

$$v^{\boldsymbol{\pi}}(\boldsymbol{x}, t) = E[g(\boldsymbol{x}_T) + \int_t^T \ell(\boldsymbol{x}, \boldsymbol{\pi}, t)\, \mathrm{d}t]$$

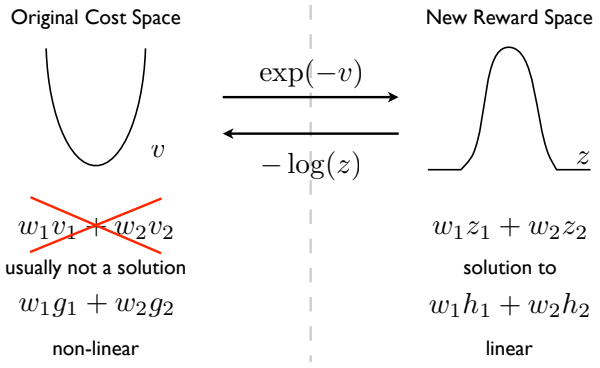

Figure 2: Cost Space-Reward Space. *The HJB equation is a non-linear partial differential equation describing the time derivative of the cost-to-go function. Non-linearity means that linear combinations of optimal value functions are* not *a new optimal value function. However, using a non-linear mapping from cost to rewards space leads to a linear PDE for the time evolution of a reward function. In this space, linear combinations are solutions to an optimal control problem.*

Assuming the stochastic differential equation has a unique solution, stochastic calculus [Oksendal 2002] links the expected cost for a random state trajectory to a solution of a second-order partial differential equation:

$$-\mathrm{D}_t[v^{\boldsymbol{\pi}}(\boldsymbol{x}, t)] = \ell(\boldsymbol{x}, \boldsymbol{\pi}, t) + \mathcal{L}^{\boldsymbol{\pi}}(v^{\boldsymbol{\pi}}(\boldsymbol{x}, t))$$
$$v^{\boldsymbol{\pi}}(\boldsymbol{x}, T) = g(\boldsymbol{x}),$$

where the differential operator $\mathcal{L}^{\boldsymbol{\pi}}$ can be thought of as taking the expectation of the value function an infinitesimal point of time into the future for some policy:

$$\mathcal{L}^{\boldsymbol{\pi}}(v(\boldsymbol{x}, t)) = (\boldsymbol{a}(\boldsymbol{x}) + \boldsymbol{B}(\boldsymbol{x})\boldsymbol{\pi}(\boldsymbol{x}, t))^\top \nabla v(\boldsymbol{x}, t) + \frac{1}{2}\mathrm{tr}[\boldsymbol{C}(\boldsymbol{x})\boldsymbol{C}(\boldsymbol{x})^\top \nabla^2 v(\boldsymbol{x}, t)].$$

The terminal cost function $g(\boldsymbol{x})$ serves as a boundary condition for the PDE, fixing the functional form of the value function at the final time. Linear Bellman combination will blend existing terminal cost functions to create controllers for new terminal costs.

The optimal value function minimizes the cost-to-go function over the space of all allowable policies and satisfies what is known as the Hamilton-Jacobi-Bellman (HJB) equation. For finite-horizon control problems (problems of fixed duration), the HJB equation is

$$-\mathrm{D}_t[v(\boldsymbol{x}, t)] = \min_{\boldsymbol{\pi}}\{\ell(\boldsymbol{x}, \boldsymbol{\pi}, t) + \mathcal{L}^{\boldsymbol{\pi}}(v(\boldsymbol{x}, t))\} \tag{4}$$
$$v(\boldsymbol{x}, T) = g(\boldsymbol{x}), \tag{5}$$

For most systems of interest in computer animation, this partial differential equation is non-linear and high-dimensional, making it very difficult to solve directly. Given the solution however, the optimal policy is obtained by following the gradient of the optimal value function as closely as possible.

3.2 Linear Bellman Equation

Using an exponential mapping that maps costs v into rewards $z = \exp(-v)$, Equation 4 can be transformed into the following form:

$$\mathrm{D}_t[z(\boldsymbol{x}, t)] = q(\boldsymbol{x}, t)z(\boldsymbol{x}, t) - \mathcal{L}(z(\boldsymbol{x}, t)) \tag{6}$$
$$z(\boldsymbol{x}, T) = \exp\big(-g(\boldsymbol{x})\big) = h(\boldsymbol{x}) \tag{7}$$

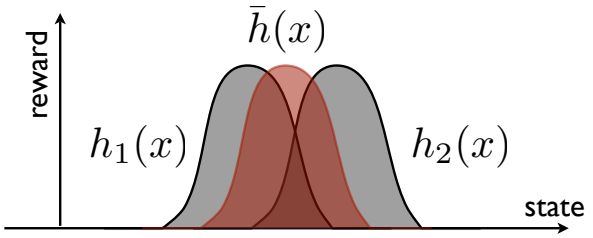

Figure 3: Projecting Terminal Cost Functions. *Since we restrict terminal cost functions, $g(\boldsymbol{x})$, to be quadratic, we have Gaussian reward functions, $h(\boldsymbol{x})$, after exponentiating. To compute the fixed blend weights, w_i, we must project the desired terminal reward function, $\bar{h}(\boldsymbol{x})$ onto the example terminal rewards associated with each component controller, $h_1(\boldsymbol{x})$ and $h_2(\boldsymbol{x})$.*

where the dependence on z is *linear* [Fleming 1978; Kappen 2005; Todorov 2006b; Todorov 2008] and the differential operator no longer depends on control:

$$\mathcal{L}(z(\boldsymbol{x},t)) = \boldsymbol{a}(\boldsymbol{x})^\top \nabla z(\boldsymbol{x},t) + \frac{1}{2}\mathrm{tr}[\boldsymbol{C}(\boldsymbol{x})\boldsymbol{C}(\boldsymbol{x})^\top \nabla^2 z(\boldsymbol{x},t)].$$

Here we have assumed a loss function of the form:

$$\ell(\boldsymbol{x},\boldsymbol{u},t) = q(\boldsymbol{x},t) + \frac{1}{2}\boldsymbol{u}^\top \boldsymbol{u}. \tag{8}$$

The penalty on state q can be any positive function. These loss functions are commonly used in computer graphics (e.g. [Witkin and Kass 1988; Ngo and Marks 1993; Safonova et al. 2004] to minimize applied force or deviation from desired poses. With this particular quadratic penalty on control, linearity of the Bellman equation only holds if $\boldsymbol{B}(\boldsymbol{x}) = \boldsymbol{C}(\boldsymbol{x})$, implying that noise acts in the same subspace as control.

The core observation of this work is that linearity allows us to apply superposition to derive a combined controller that solves a new task (see Figure 2). Suppose we have several solutions to Equation 6 $z_i(\boldsymbol{x},t)$ for different boundary conditions, i.e. terminal reward functions $h_i(\boldsymbol{x})$. Then, by superposition we also have the solution to the same PDE with the terminal reward function

$$h(\boldsymbol{x}) = \sum_{i=1}^{n} w_i h_i(\boldsymbol{x}). \tag{9}$$

The solution is

$$z(\boldsymbol{x},t) = \sum_{i=1}^{n} w_i z_i(\boldsymbol{x},t). \tag{10}$$

To get back to standard value function space, we apply the inverse transformation, $v(\boldsymbol{x},t) = -\log z(\boldsymbol{x},t)$. Applying this transformation, we see that superposition automatically gives a solution to the HJB equation with the terminal condition, $g(\boldsymbol{x}) = -\log\left(w_1 h_1(\boldsymbol{x}) + \cdots + w_n h_n(\boldsymbol{x})\right)$. Later, we will describe how to determine the weights w_i by projecting a desired terminal cost function onto the example terminal cost functions.

3.3 Time-Varying Coefficients

To derive the coefficients of the linear Bellman combination of Equation 1, note that the optimal policy for control problems with control-affine dynamics and loss functions that are quadratic in control actions is $-\boldsymbol{B}(\boldsymbol{x})^\top \nabla v(\boldsymbol{x},t)$ [Bertsekas 2007]. This policy descends the value function in the steepest direction allowed by the current configuration of the system. Using this policy, we see that Equation 10 leads to

$$\boldsymbol{\pi}(\boldsymbol{x},t) = \boldsymbol{B}(\boldsymbol{x})^\top \nabla \log\left(\sum_{i=1}^{n} w_i z_i(\boldsymbol{x},t)\right).$$

Taking the derivative of the log and applying the chain rule leads to:

$$\boldsymbol{\pi}(\boldsymbol{x},t) = \frac{1}{z(\boldsymbol{x},t)}\sum_{i=1}^{n} w_i z_i(\boldsymbol{x},t)\boldsymbol{\pi}_i(\boldsymbol{x},t), \tag{11}$$

3.4 Summary

In summary, given a set of control policies, linear Bellman combination creates a new policy that optimizes a new terminal reward function which is a weighted combination of example terminal reward functions. The combined policy weights each component policy by how inexpensive it is as indicated by its reward function z_i and by its relevance to the goal indicated by the weight w_i.

The optimality of policies described by Equation 11 depends on several factors. First, the form of the linear PDE in Equation 6 must be shared by each solution in order for superposition to hold. Changing the dynamics of the system changes the PDE as does changing the loss function. In addition, the component policies must also be optimal for the combination to be optimal. In practice, it is difficult to compute optimal policies. However, our experiments show that linear Bellman combination is more effective than naive blending approaches even when the component policies are known to be suboptimal.

The presented formulation was derived assuming continuous system dynamics. In many simulation engines, character dynamics can be discontinuous as a result of contact with the environment. In experiments involving contact, we used penalty based contact forces to make our dynamics as smooth as possible. The disadvantage of this is that contact constraints cannot be enforced exactly. Incorporating discrete jumps in character state into the linear Bellman formulation in order to allow for exact constraint enforcement requires further study.

4 Coordination and Interpolation

Given a set of optimal controllers and their associated value functions, linear Bellman combination is optimal for a task with the following terminal cost function:

$$g(\boldsymbol{x}) = -\log\left(\sum_{i=1}^{n} w_i h_i(\boldsymbol{x})\right).$$

This allows us to create controllers which interpolate the terminal cost functions of the component controllers. It also allows us to create goal functions with multiple acceptable outcomes (multiple minima). The fixed blend weights w determine whether the combined controller is interpolating terminal cost functions or coordinating between multiple policies. These weights can either be set automatically given a new terminal cost function or can be set to one to create a terminal cost function with possibly many minima. This section describes these weight settings and the behavior of the resulting control policy.

4.1 Control Interpolation for New Tasks

One application of linear Bellman combination is interpolating existing controllers. For example, this can be used to combine a controller for a long step with a controller for a short step to generate steps of intermediate length. Given a desired terminal cost function $\bar{g}$, we can automatically find the weights w_i such that g approximates $\bar{g}$ as well as possible. Working in reward space, this can be

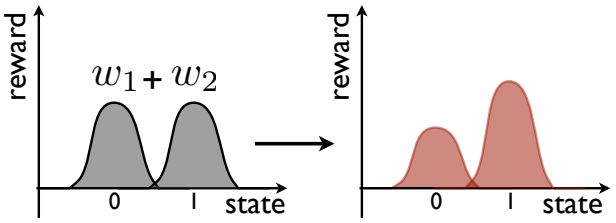

Figure 4: Unintuitive Projection. *Depending on how example terminal cost functions are sampled, weighted combinations may produce unintuitive new terminal cost functions. For example, if the examples are too far apart, a weighted combination will produce a terminal reward function with two acceptable outcomes. Here a terminal cost function centered on zero and another centered on one are scaled such that weighted combinations produce local minima in the reconstructed terminal cost function.*

accomplished by projecting the function $\overline{h} = \exp(-g)$ onto the potentially non-orthonormal basis functions, h_i (see Figure 3). Non-orthonormal projections are carried out by solving a linear system $Aw = b$ where $A_{i,j} = g_i \cdot g_j$ and $b_i = g_i \cdot \overline{g}$. For low-dimensional systems, the functions can be sampled at points in state space. For higher-dimensional systems, we restrict terminal cost functions to be quadratics (which are Gaussians after exponential transformation) and compute their inner product analytically.

The terminal cost functions of the component policies may form a poor basis for representing the desired terminal cost function, $\overline{g}$ as depicted in Figure 4. A good indicator is if the weights computed from projection are negative and do not come close to summing to one. The quadratic cost functions should share the same scale factor which is inversely proportional to the square of the distance between each minimum point. Otherwise, it may not be possible to smoothly interpolate points in state space as potential targets for the combined controller. Note that this scale factor also has an effect on the stiffness of the resulting control policy, as steeper cost functions lead to larger control actions. This can be mitigated by scaling control costs in the objective function.

4.2 Coordinating Multiple Controllers

Linear Bellman combination can also coordinate the operation of multiple controllers. Coordination is useful for two types of tasks as depicted in Figure 5. The first is a task where there is a set of acceptable outcomes. For example, in a subway car, you can balance by holding onto the wall of the car either in front of you or behind you. Coordination is also useful for tasks where controllers share a single desired outcome but are optimized for different parts of state space. As an example, consider continuing a walk after taking a longer or shorter step. In this case, the value function for each component controller would be the same if the component controllers were actually optimal, but this might not be the case due to approximations that are valid in some localized neighborhood of state space. Coordination will smoothly choose the controller with the best chance for success.

Coordination between component controllers is achieved by equally emphasizing the goal of each controller $w_i = 1$. If the assumptions for optimality are satisfied as described above, then the resulting controller $\boldsymbol{\pi}$ is optimal for an objective function with a terminal cost:

$$g(\boldsymbol{x}) = -\log\left(\sum_{i=1}^{n} \exp\left(-g_i(\boldsymbol{x})\right)\right).$$

First, we consider the case where all the terminal cost functions are

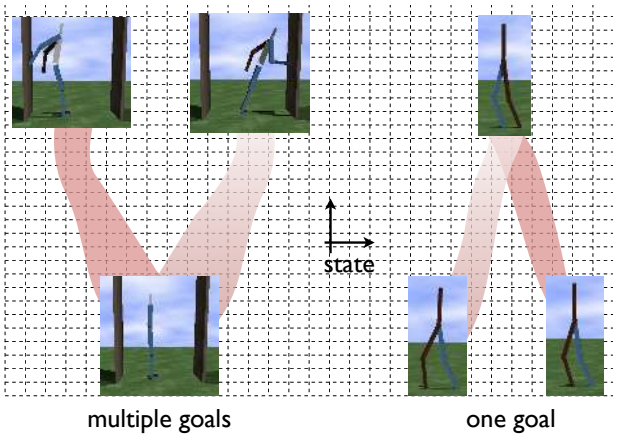

Figure 5: Coordinating Multiple Controllers. *Linear Bellman controllers can track multiple trajectories at once, smoothly choosing whichever is appropriate at the given state and time. For example, we may have multiple controllers that start a step in different configurations but end in the same state. Or we may have controllers that start in the same state but end up in an acceptable outcome.*

equal, i.e. $g_i = g$ for all i. For high-dimensional systems, it is typically only possible to represent policies that cover a portion of state space. For example, we may have feedback controllers optimized to accomplish the swing up task from different initial conditions of a simple gymnast character. Combination can be used to arbitrate between each of these policies.

Next, consider the case where the g_i's are quadratics with minimums at different spots in state space:

$$g_i(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^\top \boldsymbol{Q} (\boldsymbol{x} - \bar{\boldsymbol{x}}_i).$$

Then, the combined terminal cost g has at most n minimum points. This means that the combined policy will try to reach the goal of one of the policies depending on which ever is the most convenient given the current state of the character. As a concrete example, consider the case where a character is pushed forward or backwards and must grab onto something to maintain balance. In this case, the character should activate a controller that reaches forwards or backwards based on the push. The results section describes such an example.

5 Approximating the Value Function

Linear Bellman combination requires policies $\boldsymbol{\pi}_i$ and their associated value functions v_i. In practice, we often start with policies without value functions. To get a value function, we first specify an objective function as in Equation 2. Once the objective is identified, we can represent the value function for each controller in a manner that allows for the interpolation necessary in Equation 11. The details of the representation depend on the dimensionality of the system being animated.

5.1 Defining the Objective

Each component controller is assumed to be optimal with respect to an objective function of the form of Equation 2. Specifying an objective of this form requires making several choices if the objective function is not known. There is the scale of the control cost relative to the other terms, the form of the state penalty q and the form of each terminal cost function g_i.

The control penalty determines the stiffness of the policy. Decreasing the control penalty results in more control effort (i.e. stiffer

feedback control) but gets the character closer to the terminal condition. Overly stiff control has an adverse effect on simulation stability and can lead to undesirable reactions to disturbances. However, placing a large cost on control effort will compromise the accuracy of the controller at achieving the final goal state.

The position penalty $q(\boldsymbol{x})$ is not used for some tasks, i.e. it is simply set to zero. For other tasks, excluding the position cost leads to undesirable feedback strategies that stray far from the example motion. Since the position cost must be consistent for each controller, we set q to be a quadratic penalty centered on some average trajectory or pose. This approach is similar to many other position penalties used in trajectory optimization approaches to character animation [Popović and Witkin 1999; Safonova et al. 2004; Sulejmanpasić and Popović 2005].

The terminal cost term is free to vary for each controller. We use quadratic functions of the form

$$g_i(\boldsymbol{x}) = (\boldsymbol{x} - \bar{\boldsymbol{x}}_i)^\top \boldsymbol{Q}(\boldsymbol{x} - \bar{\boldsymbol{x}}_i),$$

where $\bar{\boldsymbol{x}}_i$ is the state of the system at the end of the ith controller's execution. As described in the previous section, restricting terminal cost functions in this way is important for determining what the composite controller is trying to accomplish and allows us to take inner products analytically. The scale of the terminal cost function $\boldsymbol{Q}$ affects the accuracy of the control and determines how closely examples must be sampled in order to allow for smooth interpolation.

While we specify the objective function manually, there exist algorithms for automatically determining objective functions given example trajectories [Liu et al. 2005; Abbeel and Ng 2004]. A generalization of this process to multiple trajectories would be a nice complement to other applications of linear Bellman combination.

5.2 Sampling the Value Function

Once an objective function is defined for each controller, the construction of each v_i depends on the dimensionality of the problem. There are two cases: low-dimensional systems of dimension less than five, and high-dimensional systems with tens of state variables. We discuss each case in turn.

Low-Dimensional Systems. For low-dimensional systems, we discretize the problem by constructing a regular grid over the area of interest in state space. Then, dynamic programming [Bertsekas 2007] solves for the optimal policy and value function at each point in this grid. At run-time, the value function is evaluated in the grid using barycentric interpolation of the values at grid points[Sutton and Barto 1998]. Low-dimensional optimal control has been used in character animation [McCann and Pollard 2007; Treuille et al. 2007].

High-Dimensional Systems. For high-dimensional systems, memory costs prevent us from building a grid in state space due to exponential growth of storage requirements. Instead, we generate an example trajectory using an execution of the controller from an initial condition. We then optimize this trajectory using an adjoint method [McNamara et al. 2004] or differential dynamic programming (DDP) [Jacobson and Mayne 1970]. The output of the optimization is a state and control trajectory $\bar{\boldsymbol{x}}_t$ and $\bar{\boldsymbol{u}}_t$ that define a value function along the trajectory:

$$v_i(\bar{\boldsymbol{x}}_i, t) = g_i(\bar{\boldsymbol{x}}_i, T) + \int_t^T q(\bar{\boldsymbol{x}}_i, s) + \frac{1}{2}\bar{\boldsymbol{u}}_i(s)^\top \bar{\boldsymbol{u}}_i(s)\,\mathrm{d}s.$$

These samples may not equal the globally optimal value function in general if the control actions are suboptimal.

Given the set of value function samples along the optimized trajectory, we linearize the dynamics about that trajectory and use differential dynamic programming (DDP) [Jacobson and Mayne 1970] to compute a quadratic approximation for each trajectory:

$$\hat{v}_i(\boldsymbol{x}, t) = v_i(\bar{\boldsymbol{x}}_i, t) + \boldsymbol{p}_i(t)^\top \Delta\boldsymbol{x} + \frac{1}{2}\Delta\boldsymbol{x}^\top \boldsymbol{P}_i(t)\Delta\boldsymbol{x} \qquad (12)$$

where the vector $\boldsymbol{p}_i(t)$ and matrix $\boldsymbol{P}_i(t)$ are computed by solving standard Ricatti equations [Jacobson and Mayne 1970], and $\Delta\boldsymbol{x} = \boldsymbol{x} - \bar{\boldsymbol{x}}_i$. This local approximation is valid within some neighborhood of the optimized trajectory. This is an efficient representation as there are large regions of the state space which do not represent realistic states for the character. We considered other interpolation schemes such as radial basis functions but the advantage of this approach is that it avoids setting or determining interpolation parameters.

The approximation in Equation 12 is used with the linear quadratic regulator $\boldsymbol{K}$˙i [Jacobson and Mayne 1970] to compute the combined controller as

$$\boldsymbol{\pi}(\boldsymbol{x}, t) = \sum_{i=1}^{n} \alpha_i(\boldsymbol{x}, t)(\bar{\boldsymbol{u}}_i(t) + \boldsymbol{K}_i(t)\Delta\boldsymbol{x}),$$

where

$$\alpha_i(\boldsymbol{x}, t) = \frac{w_i \exp\big(-\hat{v}_i(\boldsymbol{x}, t)\big)}{\sum_{j=1}^{n} w_j \exp\big(-\hat{v}_j(\boldsymbol{x}, t)\big)}.$$

The exponential remapping used to compute α maps large costs to small rewards which can lead to precision errors with the normalization step. Also, other local approximations to the policy can be used instead of a control tape. For example, the optimization that generates the value function samples may optimize the parameters of a feedback policy rather than a feedforward control tape. These policies can be blended using Equation 11.

6 Results

There are two applications our control formulation supports: controller interpolation and coordination. In this section, we evaluate the performance of both applications in several different simulation scenarios. We also provide empirical evidence that more naive combination schemes can lead to undesirable animation outcomes.

6.1 Control Interpolation for New Tasks

Linear Bellman combination can be used to interpolate control tasks. Here we demonstrate this application using several animation tasks: diving, stepping, and jumping. We also highlight some important factors that go into the construction of an effective combined policy.

Diving. Optimizing any diving motion using trajectory optimization can be a time-consuming task. We can use blending to produce multiple diving examples from just two optimized solutions. In this case, one solution does a reverse dive that under rotates ending up short of the desired vertical position while the other dive over rotates and goes past vertical. Blending the resulting feedback policies creates dives that under rotate more or less or hit vertical.

In Figure 6, we compare linear Bellman combination for interpolation with a naive approach that averages the output of the each component controller using fixed weights. This approach fails to produce a vertical dive. The simple approach can be made to work by tuning a set of time-varying blend weights. Linear Bellman combination produces blend weights automatically given the desired terminal cost function.

An important aspect of linear Bellman combination is that it uses less control than standard proportional derivative control. For graphics, this is important not only for simulation stability, but also for aesthetics. An overly stiff controller does not react to disturbances. As an example, we added random forces to the diver. Using linear Bellman combination, one can see the effects of this added

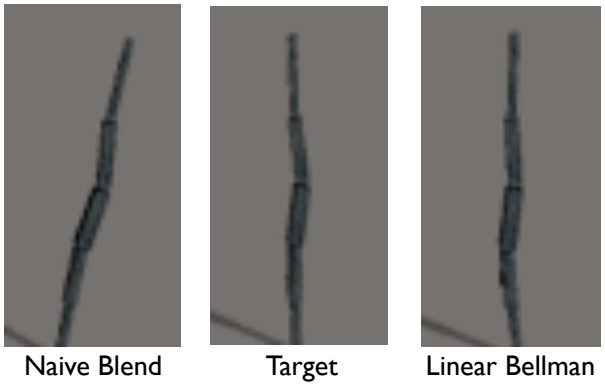

Figure 6: Naive Blend vs Linear Bellman. *Shown are the final frames of two dive simulations. The simulation on the right used linear Bellman combination to compute time-varying weights to combine a dive controller that under rotates and a dive controller that over rotates. The fixed blend weights w_i needed to produce a vertical diver were automatically computed by projecting a goal function centered on the vertical pose onto the goal functions of each component controller. Using these fixed blend weights in a naive weighted average fails to produce a vertical dive.*

force and watch how the control compensates for it. With a standard proportional-derivative dive controller, the effects of the applied disturbances are not immediately visible, although the final outcome is altered. With PD control, disturbances are invisible even when the applied forces are scaled by a factor of four.

Stepping. It is important while walking to be able to avoid stepping on obstacles [Coros et al. 2008]. Linear Bellman combination can create a stepping controller which can take steps of various lengths. For this application, two example steps, a short step and a long step, are combined to interpolate the step lengths. A complete walk can be constructed by sequencing step controllers together.

Jumping. The height of a jump is determined by the speed of the character at the moment of takeoff. Once in the air, the character cannot adjust its momentum to fly higher or further. As such, we use linear Bellman combination to interpolate takeoff controllers to jump over obstacles in the scene or to grab onto objects at different heights. Linear Bellman combination interpolates desired takeoff conditions more accurately than a naive approach. To test this, we created a tracking controller that lands a jump for a given initial condition. We then used this initial condition as the target for the linear Bellman combination. The resulting jump lands successfully. A weighted average controller using the same weights fails to land as it falls outside of the region of competence for the tracking controller.

Objective Function. In each of these cases, we found it important to include a term in the integrated cost of the objective which penalized deviations from some form of average trajectory. In other words, each example minimizes an objective of the form

$$(\boldsymbol{x}(T)-\bar{\boldsymbol{x}}_i(T))^\top \boldsymbol{Q}_T(\boldsymbol{x}(T)-\bar{\boldsymbol{x}}_i(T)) + \int_0^T (\boldsymbol{x}(t)-\bar{\boldsymbol{x}}(t))^\top \boldsymbol{Q}(\boldsymbol{x}(t)-\bar{\boldsymbol{x}}(t)) + \frac{1}{2}\boldsymbol{u}(t)^\top \boldsymbol{R}\boldsymbol{u}(t)\,\mathrm{d}t.$$

If the integrated cost term is not included in the objective, the resulting controller will make significant departures from the example trajectories to try and meet the terminal goal. It may also employ feedback strategies that result in undesirable motions.

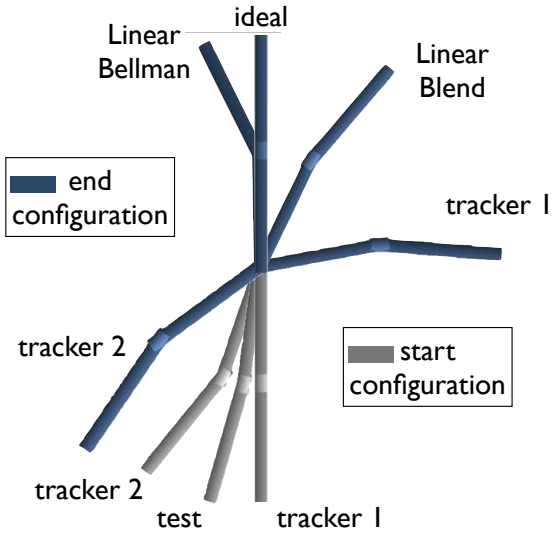

Figure 7: Tracking Multiple Swing Up Trajectories. *Here we compare the final position of the simple gymnast under four policies. Tracker 1 and 2 are LQR policies used to track optimized trajectories for the swing up task starting from the indicated locations. Simulations starting from the test location using these trackers fail to come near the upright position. A simple linear blend fairs better but is not as close as a controller created using linear Bellman combination. The results of all these trackers can be improved by placing examples closer to the test configuration.*

6.2 Coordinating Multiple Controllers

Some recent work has used linear quadratic regulators (LQR) to track a single example trajectory [da Silva et al. 2008b; Barbič and Popović 2008]. However, these tracking controllers are often valid only near the example trajectory. If a perturbation takes the character outside of this neighborhood, the tracking controller will fail to recover. Our solution to this problem is to combine many tracking controllers that cover different parts of the state space.

Simple Gymnast Example. We first demonstrate this idea using a high bar gymnastics task. In this case, the gymnast has two links and can only exert torques at the hip (this model is known as the acrobot model in robotics). Its state space consists of two joint angles and two joint velocities. The goal is to build a controller that brings the gymnast to a hand-stand position on the high bar while minimizing exerted torque. We build such a controller by combining multiple tracking controllers using Equation 11 with fixed weight one to create a combined policy $\boldsymbol{\pi}$. The tracking controllers track example trajectories which solve the swing up task for different initial conditions. These solutions are synthesized using an adjoint method solving an objective of the form

$$g(\boldsymbol{x}) + \int_0^T \frac{1}{2}\boldsymbol{u}(t)^\top \boldsymbol{u}(t)\,\mathrm{d}t,$$

resulting in two feedforward trajectories $\boldsymbol{u}_1$ and $\boldsymbol{u}_2$ which solve the swing up task from two different initial conditions.

Figure 7 compares several approaches to solving the swing up task for an initial condition which does *not* have an associated precomputed policy, $\boldsymbol{x}(0) = (-105, -15, 0, 0)$ where the angles are in degrees. The first attempt uses only a single tracking controller which tracks a solution initialized at $\boldsymbol{x}(0) = (-90, 0, 0, 0)$. The

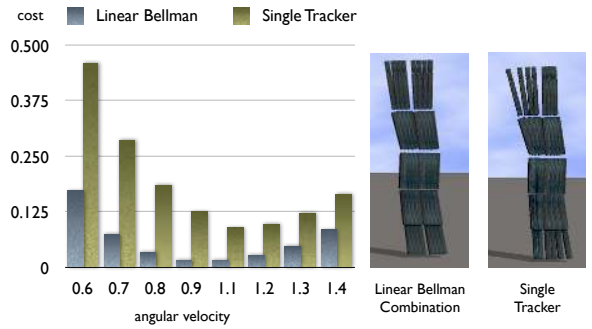

Figure 8: Diving Coordination. *Shown are the final poses of a diver starting from eight different initial angular velocities. The poses on the left were controlled using linear Bellman combination of three dive controllers. The poses on the right were simulated using a single tracking controller. The single tracking controller does not reach a vertical configuration for some initial conditions and has an inferior objective score in each case.*

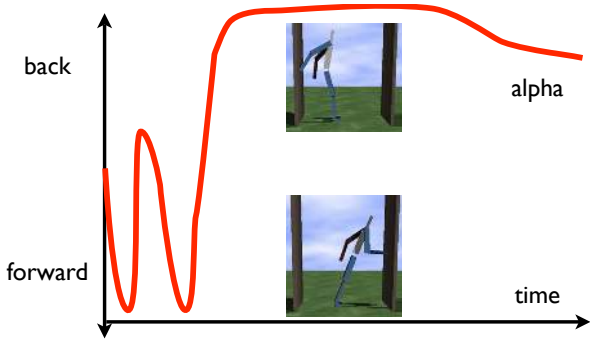

Figure 9: Balance Coordination. *Shown is the time varying blend parameter of a coordinating controller created using linear Bellman combination of a controller which catches itself falling forwards and another falling backwards. In this simulation, the arm of the character was tugged forwards and backwards causing the combined controller to vacillate before settling on the backwards strategy.*

second attempt also uses a single tracking controller but this one is initialized at $\boldsymbol{x}(0) = (-110, -20, 0, 0)$. The third approach simply sums the results of the two tracking controllers. None of these approaches come as close to the ideal handstand as a controller created using linear Bellman combination of the two tracking policies. The hand-stand position is an unstable equilibrium point of the character. It is important to come as close as possible so that a stabilizing controller can easily maintain that position.

Diving. We use a diving example to illustrate that combining multiple controllers can outperform using a single tracking controller. We construct a controller using linear Bellman combination of three components which enter the water vertically for three different initial angular velocities: 0.85, 1, and 1.15, radians per second. A tracking controller is constructed from the dive that starts with an initial angular velocity of 1 radian per second. We sampled a number of different initial angular velocities for the diver. The combined controller does a better job recovering to a vertical pose on entry than a single tracking controller, see Figure 8.

Push Recovery. The idea of coordination can also be used to build simple push recovery controllers. We created a controller which chooses to reach forwards or backwards to prevent a fall based on an initial push. In Figure 9, we plot the blend weight of a coordinating controller indicating which controller has the most influence at each point in time. In the simulation depicted, the arm is tugged back and forth causing changes in the blend weight.

Stepping. Coordination was also used to combine two SIMBICON [Yin et al. 2007] walkers. One walker was tuned to walk on flat ground, while the other was tuned to walk on an incline. When placed on an incline of ten degrees, the flat ground walker does not fall, but fails to advance up the hill. We constructed a combined controller that walks on flat and hilly terrain. As the combined walker encounters a hill, the blend weights shift to weight the hill component more heavily.

6.3 Implementation Details

Simulation. The examples involving linked rigid bodies were simulated using our own minimal coordinates simulation engine. For the examples involving contacts, we used a penalty-based method with parameters as described in Yin et al's SIMBICON simulation [2007].

Physical Models. The two-link gymnast is a point-mass model with unit masses and unit length limbs. The diver is a seven link, planar character with physical parameters as described by Sok et al [2007]. A version of the diver with both sets of extremities was used for the broad jump examples that land. The step example uses the same 2D model as SIMBICON [2007]. The vertical leap examples use the the same 3D model as described in Limit Cycle Control [Laszlo et al. 1996].

Optimization. Linked rigid body examples were optimized using one of two methods. One method was a conjugate gradient descent algorithm where the derivative of the objective function with respect to the control parameters is calculated using the adjoint method. The other method was differential dynamic programming. We found that differential dynamic programming consistently converged in fewer iterations but sometimes settled in a different local minimum than the adjoint method. The optimizations often needed a decent initial guess to converge to good solutions. The diving and jumping examples were initialized using PD controllers similar to those described by Wooten et al [2000]

It is important to note that, as opposed to low-dimensional systems, high-dimensional systems require local optimization methods that avoid the curse of dimensionality by finding the solution for a single initial condition. However, it is not guaranteed that the resulting policy is optimal. This violates the assumptions required for optimality as described in §3, so in practice, all high dimensional examples are using sub-optimal policies.

7 Conclusion

This paper introduces linear Bellman combination as a method for joining control policies to derive new control strategies. Combination can interpolate two policies to obtain a strategy that accomplishes a different goal. Or, it can coordinate two policies to obtain a strategy that combines different functionalities. For example, coordination can combine policies designed for two different preconditions (or outcomes) to derive a composite strategy valid for either precondition (or outcome).

The combined strategies are modular because they only blend the control forces of constituent policies. The individual building blocks could therefore be arbitrary control schemes. Furthermore, the combination amortizes the cost and the difficulty of generating individual control policies. Policies can be painstakingly crafted once and then repurposed many more times in various combina-

tions. With some assumptions, the combination can even combine optimal policies to compute another *optimal* policy for a different terminal cost.

When many related optimal control problems must be solved, there is an advantage to using linear Bellman combination over direct non-linear optimization. Even on simple problems, direct optimization can be too costly to compute controllers in real-time applications. Linear Bellman combination computes a new controller essentially instantly. However, linear Bellman combination is not intended as a replacement for direct non-linear optimization in *offline* applications. An alternative for real-time applications is model predictive control. Model predictive control may be preferable to combination when feasible as in autonomous control of aircraft [Milam 2003] or character models over short time horizons [Tassa et al. 2008; da Silva et al. 2008a].

There are two main limitations of linear Bellman combination. The first limitation is that value function approximations are required to blend. Currently, we use example trajectories from the controller and a manually specified objective function to construct an approximate value function. An interesting area of future work would be to automatically infer value-function approximations given observations of previously successful trajectories. This would allow us to learn control strategies from observations of human performances recorded by motion capture.

The second major limitation of linear Bellman combination is that the component controllers must share the same duration. We would like to devise a strategy for indexing component controllers using non-time based indices in order to blend policies of different duration. One could use any monotonic feature of the motion to reindex time such as the angle between the ankle and hip in a walk. In addition, it should be possible to extend combination to first-exit policies where the policy terminates when a certain state region is first reached.

Optimization and control are essential in many domains of computer graphics. The linearity of the Bellman equation should be explored in those domains as well.

Acknowledgments

This project was inspired by Emo Todorov's work on linearly solvable Markov decision processes. Russ Tedrake provided some early feedback on how combination might be used in practice. Emo provided a MATLAB implementation of his z-iteration algorithm which allowed us to test combination on simple examples. Michiel van de Panne provided a detailed and helpful early review of our paper. This work was supported by grands from the Singapore-MIT Gambit Game Lab, the National Science Foundation (2007043041, CCF-0810888), Adobe Systems, Pixar Animation Studios, and software donations from Autodesk and Adobe systems.

References

ABBEEL, P., AND NG, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine learning (ICML)*, ACM, vol. 69, 1:1–1:8.

ATKESON, C. G., AND MORIMOTO, J. 2002. Nonparametric representation of policies and value functions: A trajectory-based approach. In *Advances in Neural Information Processing Systems (NIPS)*, vol. 15, 1611–1618.

ATKESON, C. G. 1994. Using local trajectory optimizers to speed up global optimization in dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, vol. 6, 663–670.

BARBIČ, J., AND POPOVIĆ, J. 2008. Real-time control of physically based simulations using gentle forces. *ACM Transactions on Graphics 27*, 5, 163:1–163:10.

BELLMAN, R. E. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

BERGOU, M., MATHUR, S., WARDETZKY, M., AND GRINSPUN, E. 2007. Tracks: toward directable thin shells. *ACM Transactions on Graphics 26*, 3, 50:1–50:10.

BERTSEKAS, D. P. 2007. *Dynamic Programming and Optimal Control*, 3 ed., vol. I. Athena Scientific, Nashua, NH.

BROTMAN, L. S., AND NETRAVALI, A. N. 1988. Motion interpolation by optimal control. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, 309–315.

BURRIDGE, R. R., RIZZI, A. A., AND KODITSCHEK, D. E. 1999. Sequential composition of dynamically desterous robot behaviours. *International Journal of Robotics Research 18*, 6, 534–555.

COROS, S., BEAUDOIN, P., YIN, K., AND VAN DE PANNE, M. 2008. Synthesis of constrained walking skills. *ACM Transactions on Graphics 27*, 5, 113:1–113:9.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Simulation of human motion data using short-horizon model-predictive control. *Computer Graphics Forum 27*, 2, 371–380.

DA SILVA, M., ABE, Y., AND POPOVIĆ, J. 2008. Interactive simulation of stylized human locomotion. *ACM Transactions on Graphics 27*, 3, 82:1–82:10.

EREZ, T., AND SMART, W. 2007. Bipedal walking on rough terrain using manifold control. *International Conference on Intelligent Robots and Systems (IROS)*, 1539–1544.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Annual Conference Series, 251–260.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. *ACM Transactions on Graphics 23*, 3, 441–448.

FLEMING, W. H. 1978. Exit probabilities and optimal stochastic control. *Applied Mathematics and Optimization 4*, 329–346.

HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, Annual Conference Series, 153–162.

HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of ACM SIGGRAPH 95*, Annual Conference Series, 71–78.

HOLLAND, C. 1977. A new energy characterization of the smallest eigenvalue fo the schrödinger equation. *Communications on Pure and Applied Mathematics 30*, 755–765.

JACOBSON, D., AND MAYNE, D. 1970. *Differential Dynamic Programming*, 1st ed. Elsevier, New York.

KAPPEN, H. J. 2005. Linear theory for control of nonlinear stochastic systems. *Physical Review Letters 95*, 20, 200–204.

LASZLO, J. F., VAN DE PANNE, M., AND FIUME, E. L. 1996. Limit cycle control and its application to the animation of balancing and walking. In *Proceedings of SIGGRAPH 96*, Annual Conference Series, 155–162.

LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics 24*, 3, 1071–1081.

MCCANN, J., AND POLLARD, N. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics 26*, 3, 6:1–6:7.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Transactions on Graphics 23*, 3, 449–456.

MILAM, M. B. 2003. *Real-Time Optimal Trajectory Generation for Constrained Dynamical Systems*. PhD thesis, Caltech.

NGO, J. T., AND MARKS, J. 1993. Spacetime constraints revisited. In *Proceedings of ACM SIGGRAPH 2000*, Annual Conference Series, 343–350.

OKSENDAL, B. K. 2002. *Stochastic Differential Equations: An Introduction with Applications*. Springer, New York, NY.

POLLARD, N. S., AND BEHMARAM-MOSAVAT, F. 2000. Force-based motion editing for locomotion tasks. In *In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 663–669.

POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. In *Computer Graphics (Proceedings of SIGGRAPH 99)*, ACM SIGGRAPH, Annual Conference Series, 11–20.

SAFONOVA, A., HODGINS, J., AND POLLARD, N. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics 23*, 3, 514–521.

SOK, K. W., KIM, M., AND LEE, J. 2007. Simulating biped behaviors from human motion data. *ACM Transactions on Graphics 26*, 3, 107:1–107:9.

SULEJMANPASIĆ, A., AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Transactions on Graphics 24*, 1, 165–179.

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

TASSA, Y., EREZ, T., AND SMART, W. 2008. Receding horizon differential dynamic programming. In *Advances in Neural Information Processing Systems (NIPS)*, vol. 20, 1465–1472.

TODOROV, E. 2006. *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT Press, Cambridge, MA, ch. 12, 269–298.

TODOROV, E. 2006. Linearly-solvable markov decision problems. *Advances in Neural Information Processing Systems (NIPS) 19*, 1369–1376.

TODOROV, E. 2008. Efficient computation of optimal actions. http://www.cogsci.ucsd.edu/∼todorov/papers/framework.pdf. Unpublished manuscript, March.

TODOROV, E. 2009. Compositionality of optimal control laws. http://www.cogsci.ucsd.edu/∼todorov/papers/primitives.pdf. Unpublished manuscript, January 15.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Near-optimal character animation with continuous control. *ACM Transactions on Graphics 26*, 3, 7:1–7:7.

VAN DE PANNE, M., KIM, R., AND FIUME, E. 1994. Synthesizing parameterized motions. In *Eurographics Workshop on Simulation and Animation*.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, 159–168.

WOOTEN, W. L., AND HODGINS, J. K. 2000. Simulating leaping, tumbling, landing and balancing humans. *International Conference on Robotics and Automation (ICRA)*, 656–662.

YIN, K., LOKEN, K., AND VAN DE PANNE, M. 2007. SIMBICON: Simple biped locomotion control. *ACM Transactions on Graphics 26*, 3, 105:1–105:10.

YIN, K., COROS, S., BEAUDOIN, P., AND VAN DE PANNE, M. 2008. Continuation methods for adapting simulated skills. *ACM Transactions on Graphics 27*, 3, 81:1–81:7.

ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *Symposium on Computer Animation (SCA)*, 89–96.