Implementation of a Trajectory Library Approach to Controlling Humanoid Standing Balance

Chenggang Liu and Jianbo Su
Department of Automation
Shanghai Jiaotong University
Shanghai, China
{frankliu, jbsu}@sjtu.edu.cn

***Abstract*—This paper presents a nonlinear controller based on a trajectory library. To generate the library, we combine two trajectory optimization methods: a parametric trajectory optimization method that finds coarse initial trajectories and Differential Dynamic Programming (DDP) that further refines these trajectories and generates linear local models of the optimal control laws. To construct a controller from these local models, we maintain the consistency of adjacent trajectories. To keep the resultant library a reasonable size and also satisfy performance requirements, the library is generated based on the controller's predicted performance. It is applied to standing balance control of humanoid robots that explicitly handle pushes. Most previous work assumes that pushes are impulsive. The proposed controller also handles continuous pushes that change with time. We compared our approach with a Linear Quadratic Regulator (LQR) gain scheduling controller using the same optimization criterion. The effectiveness of the proposed method is explored with simulation and experiments.**

***Index Terms*—Humanoid robot, standing balance control, trajectory library**

I. Introduction

Standing balance control keeps balance in the presence of perturbations during upright stance, which is a fundamental problem for humanoid robots. Due to small feet, the ankle torques are quite limited to prevent the feet from tilting and the robot falling [1]. The large range of motion of the upper body makes the system dynamics nonlinear. Limits on joint angles and joint velocities also add to the control challenge. Nonlinear feedback controllers [2], linear feedback controllers [3], intuitive controller designs [4], [5], and controllers using online optimization [6], [7] have been studied.

The standing balance control problem can be formulated as an optimal control problem and it was shown that multiple balance strategies can be generated from one optimization criterion [8], [9]. Dynamic Programming (DP) provides a way to find an optimal feedback control law for a nonlinear system [10]. But when the dimensionality of the state is large, the computation and even the storage of the optimal control law becomes difficult, due to the Curse of Dimensionality [10]. Differential Dynamic Programming is a local version of dynamic programming [11], [12]. It applies the principle of optimality in the neighborhood of a trajectory. This allows the coefficients of quadratic approximations of the value function and linear approximations of the optimal control law to be computed along the trajectory. These coefficients may then be used to compute an improved trajectory. After DDP's convergence to an optimal trajectory, linear local models of the optimal control law and quadratic local models of the value function are available.

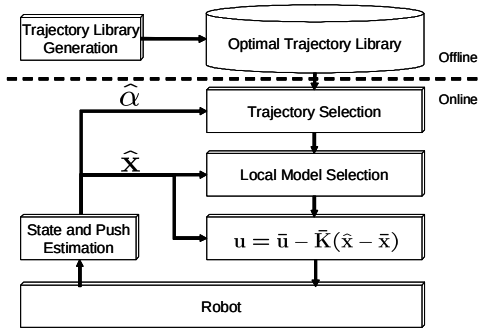

Fig. 1. Architecture of standing balance control using a trajectory library.

For DDP's convergence to an optimum, a good initial trajectory is important especially when a dynamic system is highly nonlinear and constrained. We combine a parametric trajectory optimization method [13] and DDP to generate a library of optimal trajectories and linear local models of the optimal control laws for standing balance control offline. As shown in Fig. 1, we take the push size and the push location as trajectory selection parameters, α. At each time step, appropriate trajectories are selected from the library according to the estimate of the selection parameters, $\hat{\alpha}$. The estimate of the current state, $\hat{\mathbf{x}}$, which consists of the joint angles and the joint angular velocities, is used to select the nearest local model on the selected trajectories. The selected local model is used to calculate the commands online, $\mathbf{u} = \bar{\mathbf{u}} - \bar{\mathbf{K}}(\hat{\mathbf{x}} - \bar{\mathbf{x}})$. The result is a state feedback control law. The idea is partly introduced in our previous work [14]. This paper is the latest result of our continuing effort and its implementation on a real robot. Most previous work assumes that pushes are impulsive and change joint velocities instantaneously [4]–[7], [9]. We also consider pushes that last a while and change with time.

A number of efforts have been made to use collections of trajectories and local models to represent feedback control laws [15]–[18]. In [15], [16], linear local models along optimal trajectories were used to construct a representation of the global control law. In [17], Receding Horizon DDP

was proposed to generate time-invariant local controllers. A trajectory library was used to synthesize a global controller for a simulated multi-link swimming robot. In [18], locally-valid LQR controllers were used to construct a nonlinear feedback policy. To improve the stability that can be obtained from a trajectory tracking control law, online adaptation of the choice of the reference trajectory was proposed for walking control [19].

This article is organized as following: In Section II, the trajectory library generation method and the controller using the optimal trajectory library are described. In Section III, the proposed controller is applied to standing balance control. In Section IV, simulation and experimental results are shown. Conclusions and future work are discussed in Section V.

II. Control Using a Trajectory Library

A. Problem Formulation

In the present paper we are concerned with dynamic systems determined by deterministic time-invariant dynamics equations,

$$\mathbf{x}_{k+1} = F(\mathbf{x}_k, \mathbf{u}_k, \alpha), \tag{1}$$

where subscript k is the time index, $\mathbf{x}_k$ are the state variables, $\mathbf{u}_k$ are the control variables, and α are the control parameters, which are constant. An optimal control problem is to find a control sequence, $\mathbf{u}_{0,N-1}$, that minimizes the cost function

$$J := \phi(\mathbf{x}_N, \alpha) + \sum_{k=0}^{N-1} L(\mathbf{x}_k, \mathbf{u}_k, \alpha), \tag{2}$$

where L is the one step cost function, N is a search horizon, and ϕ is the terminal cost function, which is chosen to approximate the infinite time problem with the same optimization criterion. We choose N so that $\mathbf{x}_N$ is near the goal state and use the quadratic value function of a linear quadratic regulator as ϕ.

Constraints on the state variables of the form

$$\mathbf{x}_L \leq \mathbf{x}_k \leq \mathbf{x}_U, \tag{3}$$

on the control variables

$$\mathbf{u}_L \leq \mathbf{u}_k \leq \mathbf{u}_U, \tag{4}$$

and on the path

$$\mathbf{g}_L \leq g(\mathbf{x}_k, \mathbf{u}_k, \alpha) \leq \mathbf{g}_U, \tag{5}$$

where function g may be scalar or vector-valued functions, may be applied.

B. Parametric Trajectory Optimization

In this section we describe a parametric trajectory optimization method. Let us treat the state and the control variables as a set of NLP (Nonlinear Programming) variables, $\mathbf{y} := [\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \ldots, \mathbf{x}_N]^\mathsf{T}$, and the dynamics equations (1) as constraints. As a result of the transcription, the optimal control constraints (1) and (5) are replaced by the NLP constraints:

$$\mathbf{c}_L \leq \mathbf{c}(\mathbf{y}) \leq \mathbf{c}_U \tag{6}$$

where

$$\mathbf{c}(\mathbf{y}) := \begin{bmatrix} \mathbf{x}_1 - F(\mathbf{x}_0, \mathbf{u}_0, \alpha) \\ \mathbf{x}_2 - F(\mathbf{x}_1, \mathbf{u}_1, \alpha) \\ \cdots \\ \mathbf{x}_N - F(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \alpha) \\ g(\mathbf{x}_0, \mathbf{u}_0, \alpha) \\ g(\mathbf{x}_1, \mathbf{u}_1, \alpha) \\ \cdots \\ g(\mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \alpha) \end{bmatrix} \tag{7}$$

with

$$\mathbf{c}_L := [\underbrace{\mathbf{0}, \ldots, \mathbf{0}}_{N}, \underbrace{\mathbf{g}_L, \ldots, \mathbf{g}_L}_{N}]^\mathsf{T} \tag{8}$$

and a corresponding definition of $\mathbf{c}_U$. The constraints on the state variables (3) and on the control variables (4) are taken as the bounds on the NLP variables. We solve this NLP problem with SNOPT, which uses an SQP algorithm [20].

C. Trajectory Optimization Using DDP

Differential Dynamic Programming (DDP) is a second order gradient-based trajectory optimization method [11], [12]. Given an initial trajectory $\mathbf{x}^j$, DDP integrates the first and second partial derivatives of the value function backward in time. A new updated trajectory $\mathbf{x}^{j+1}$ is generated by integrating the dynamics equations forward in time at each time step using the linear feedback law:

$$\mathbf{u}_k^{j+1} = \mathbf{u}_k^j - \epsilon \Delta \mathbf{u}_k - \mathbf{K}_k(\mathbf{x}_k^{j+1} - \mathbf{x}_k^j) \tag{9}$$

with initial state $\mathbf{x}_0$ and control parameter α specified, where ϵ is a scalar between $(0, 1)$. The definitions of $\mathbf{K}_k$ and $\Delta \mathbf{u}_k$ are given by a standard DDP algorithm [11]. These processes are repeated until convergence to a local optimum is achieved. For constraints on the state variables (3) and on the path (5), we augment the one step cost function with quadratic penalties on the constraint violations.

D. Nearest Neighbor Control

Our approach uses local models of the optimal control policy to interpolate control actions. Byproducts of DDP are linear local models of the optimal control law along the optimal trajectory,

$$\mathbf{u}_k = \bar{\mathbf{u}}_k - \bar{\mathbf{K}}_k(\mathbf{x}_k - \bar{\mathbf{x}}_k), \tag{10}$$

where $\bar{\mathbf{u}}_k$ and $\bar{\mathbf{x}}_k$ are respectively the control variables and the state variables on the optimal trajectory, and $\bar{\mathbf{K}}_k$ is the corresponding gain matrix. We formulate the optimal control problem so that the underlying nonlinear value functions and control laws are time invariant and functions only of state. Because the local models (10) are local approximations to the optimal control law in the neighborhood of an optimal trajectory, they are also time-invariant and functions only of state. Thus, we can construct a controller from these spatially localized local models. The simplest choice is to select the nearest Euclidean neighbor. For different control parameters, the nearest optimal trajectories are selected according to

$$\Delta \alpha^\mathsf{T} \mathbf{D}_\alpha \Delta \alpha, \tag{11}$$

where $\Delta\alpha$ is the deviation of the current control parameters from those of an optimal trajectory and $\mathbf{D}_\alpha$ is a distance metric. Then, the nearest local model on the selected trajectories is selected according to

$$\Delta\mathbf{x}^\mathsf{T}\mathbf{D}_x\Delta\mathbf{x}, \tag{12}$$

where $\Delta\mathbf{x}$ is the deviation of the current state $\mathbf{x}$ from a local model and $\mathbf{D}_x$ is a distance metric. Nearest neighbor control is given by

$$\mathbf{u} = \bar{\mathbf{u}} - \bar{\mathbf{K}}(\mathbf{x} - \bar{\mathbf{x}}) \tag{13}$$

where $\bar{\mathbf{u}}$, $\bar{\mathbf{x}}$, and $\bar{\mathbf{K}}$ are respectively the control variables, the state variables, and the gain matrix of the nearest local model.

E. Trajectory Library Generation

The local controllers from a single trajectory may perform poorly far from the trajectory. To synthesize a controller over a larger portion of state space, a library of optimal trajectories is used.

To limit the size of the resultant library, the optimal trajectories and local models are stored on an adaptive grid of initial states and control parameters according to the simulated performance.

In outline:

Step 1 Use DDP to generate an optimal trajectory with the initial state and control parameters that is easy to optimize. Store the optimal trajectory and local models as an initial library.

Step 2 Initialize the system at nearby initial conditions $(\mathbf{x}, \alpha)$ and let it follow the nearest neighbor control and the current library, which generates a trajectory of total cost J_1.

Step 3 Use DDP or SNOPT+DDP to generate an optimal trajectory with the same initial conditions $(\mathbf{x}, \alpha)$ and total cost J_2.

Step 4 Compare J_1 and J_2. If $J_1 - J_2$ is less than a performance bound, return to Step 2 and continue.

Step 5 Otherwise, store the optimal trajectory and local models generated in Step 3. Return to Step 2 and continue until the portion of state space of interest and all possible control parameters are all considered.

In Step 3, the trajectory generated in Step 2 is used as the initial trajectory for DDP to further refine. If DDP fails to converge to a better trajectory, parametric trajectory optimization with SNOPT is used, whose result or its interpolation is used as the initial trajectory for DDP to further refine. To avoid the same local optimum as DDP, SNOPT can use a different initial trajectory, such as a straight line connecting the initial state to the goal state in the state space.

To choose new initial conditions $(\mathbf{x}, \alpha)$ in Step 2, candidate initial conditions at the beginning of a waiting list are chosen while unsolved neighbors at a grid of initial conditions are appended to the end of the waiting list. This process is repeated until all initial conditions of interest at the grid points are considered. The spacing of the grid has to be chosen in advance.

Both the parametric trajectory optimization method and DDP are local trajectory optimization methods and there is no guarantee of a global optimum. Therefore, local models of the control laws from different trajectories may be inconsistent with each other and they may be generated from different underlying control laws. To construct a controller from many local models, we try to force all these local models to be consistent with each other by 1) starting the optimization process from a goal state and then consider a growing volume of the state space, 2) using the policy of one state of a pair to reoptimize the trajectory of the other state of the pair and vice versa, and 3) adding more local models in between nearest neighbors that continue to fail to predict each other's value until the policy or value function discontinuity is confirmed or eliminated [15]. We also periodically reoptimize each local model using the policies of other local models. If the combination of local value function models generates a global value function that satisfies the Bellman equation everywhere, the resulting policy and value function are globally optimal [10], [21].

III. Standing Balance Control

A. Robot Models

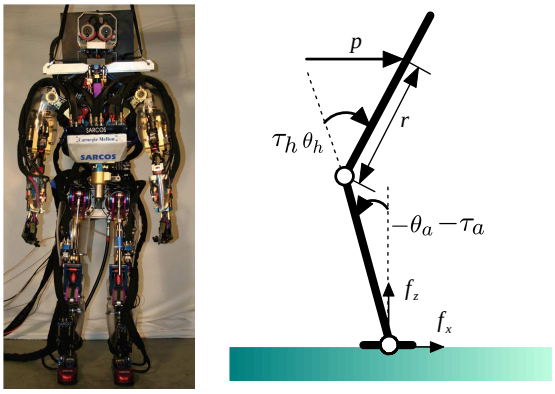

Fig. 2. a) Sarcos Primus System hydraulic humanoid robot b) Two-link inverted pendulum model.

The robot is assumed to be a rigid-body system and planar in the sagittal plane. A torso push is modeled as a horizontal force of size p, applied on the torso at a location, r, which is the distance from the point of action to the hip joint below. The feet are not fixed on the ground, but the robot can be modeled with a two-link inverted pendulum if the ground reaction forces are limited to keep the feet from sliding and flat on the ground. In the two-link inverted pendulum model, the legs are the first link and the upper body is the second link, as shown in Fig. 2(b). The model parameters are based on our Sarcos Primus System hydraulic humanoid robot (shown in Fig. 2(a)). The mass, the length, the moment of inertia of the first link are respectively 56 kg, 0.8 m, and 1.24 kg·m^2. Those of the second link are respectively 37 kg, 0.8 m, and 1.73 kg·m^2. Defining $\mathbf{x} := (\theta_a, \theta_h, \dot{\theta}_a, \dot{\theta}_h)^\mathsf{T}$ and

$\mathbf{u} := (\tau_a, \tau_h)^\mathsf{T}$, the discrete-time dynamics (1) can be derived, where push size p and push location r are taken as control parameters, $\alpha := (p, r)^\mathsf{T}$.

B. State and Push Estimation

We have no sensors for joint velocities, push size p, and push location r, which have to be estimated based on the measurements of the joint angles and the ground reaction forces. Let us define the state to be estimated as $\mathbf{x}^e := (\mathbf{x}^\mathsf{T}, \alpha^\mathsf{T})^\mathsf{T}$, and the observation as $\mathbf{z} := (\theta_a, \theta_h, f_x, f_z)^\mathsf{T}$ (see Fig. 2 for the definitions of f_x and f_z). The state transition model and the observation model are given by:

$$\begin{aligned} \mathbf{x}^e_{k+1} &= \begin{bmatrix} F(\mathbf{x}_k, \mathbf{u}_k, \alpha) \\ \alpha \end{bmatrix} + \mathbf{w} \\ &= G(\mathbf{x}^e_k, \mathbf{u}_k) + \mathbf{w}, \quad \mathbf{w} \sim N(0, \mathbf{S}) \end{aligned} \tag{14}$$

$$\begin{aligned} \mathbf{z}_k &= \begin{bmatrix} \theta_a \\ \theta_h \\ f_x(\mathbf{x}_k, \mathbf{u}_k, \alpha) \\ f_z(\mathbf{x}_k, \mathbf{u}_k, \alpha) \end{bmatrix} + \mathbf{v} \\ &= H(\mathbf{x}^e_k, \mathbf{u}_k) + \mathbf{v}, \quad \mathbf{v} \sim N(0, \mathbf{T}), \end{aligned} \tag{15}$$

where f_x and f_z are calculated by inverse dynamics, the noise terms $\mathbf{w}$ and $\mathbf{v}$ are uncorrelated, and $\mathbf{S}$ and $\mathbf{T}$ are covariance matrices. The state transition model and the observation model are both nonlinear, so the Extended Kalman Filter is employed. During standing balance control, the state $\mathbf{x}$, the push size p, and the push location r are estimated at each time step.

C. Constraints and Optimization Criteria

Any limitations on the joint angles and the joint velocities are taken as constraints on the state variables (3). Limitations on the joint torques are taken as constraints on the control variables (4). To keep the foot from sliding and flat on the ground, the ground reaction forces are limited by the friction cone, $|\frac{f_x}{f_z}| \leq \mu$, where μ is the coefficient of static friction; The CoP (the center of pressure) is limited by the foot support region, $-\frac{\tau_a}{f_z} \in \Omega$, where τ_a is the ankle torque and Ω is the region of the foot support (see Fig. 2 for the definitions of f_x, f_z, and τ_a) [1]. These constraints are taken as constraints on the path:

$$g(\mathbf{x}, \mathbf{u}, p, r) := \begin{bmatrix} f_x/f_z \\ -\tau_a/f_z \end{bmatrix}. \tag{16}$$

The one step cost function we use is in a quadratic form,

$$L(\mathbf{x}, \mathbf{u}, \alpha) := \tilde{\mathbf{x}}^\mathsf{T}\mathbf{Q}\tilde{\mathbf{x}}T + \tilde{\mathbf{u}}^\mathsf{T}\mathbf{R}\tilde{\mathbf{u}}T, \tag{17}$$

where $\tilde{\mathbf{x}} := \mathbf{x} - \mathbf{x}_d(\alpha)$, $\tilde{\mathbf{u}} := \mathbf{u} - \mathbf{u}_d(\alpha)$, T is the time step, $\mathbf{Q}$ and $\mathbf{R}$ are diagonal matrices with appropriate dimensions, $\mathbf{x}_d(\alpha)$ and $\mathbf{u}_d(\alpha)$ are respectively the desired state and the desired control for a specified control parameter vector, α. We use quadratic programming to find the desired state $\mathbf{x}_d$ and the desired control $\mathbf{u}_d$. For specified push size p and push location r, $\mathbf{x}_d$ and $\mathbf{u}_d$ are the state and the control of the robot in static equilibrium that minimize

$$\min_{\mathbf{x}_d, \mathbf{u}_d} (\mathbf{u}_d - \mathbf{u}_{init})^\mathsf{T}(\mathbf{u}_d - \mathbf{u}_{init}), \tag{18}$$

TABLE I
PARAMETERS USED BY TRAJECTORY OPTIMIZATION

parameter	value	parameter	value
$\mathbf{u}_L$	$-(150.0, 150.0)^\mathsf{T}$	$\mathbf{x}_L$	$-(0.4, 0.2, 6.0, 6.0)^\mathsf{T}$
$\mathbf{u}_U$	$(40.0, 150.0)^\mathsf{T}$	$\mathbf{x}_U$	$(0.4, 0.8, 6.0, 6.0)^\mathsf{T}$
$\mathbf{Q}$	$\mathrm{diag}(1.2, 0.1, 0.9, 0.1)$	$\mathbf{R}$	$\mathrm{diag}(0.06, 1.59)$
$\mathbf{g}_U$	$(2.0, 0.16)^\mathsf{T}$	$\mathbf{g}_L$	$-(2.0, 0.04)^\mathsf{T}$

where $\mathbf{u}_{init}$ are the joint torques of the robot during static upright stance without pushes. Constraints (3) and (4) are also applied. We use SNOPT to solve this quadratic programming problem [20].

IV. RESULTS

A. Simulation

To generate optimal trajectories for standing balance control, we take upright with zero velocity as the initial state and generate optimal trajectories on a uniform grid of control parameters (push size p and push location r). The step sizes are 5 Newtons for p and 0.1 meter for r, respectively. The parameters used by trajectory optimization are listed in Table I. To make the proposed controller more applicable to a real robot, we optimize $\mathbf{Q}$ and $\mathbf{R}$ such that the gain matrices generated by DDP are small. Since we use SNOPT to generate coarse initial trajectories for DDP, we use $N = 100$ and $T = 0.05$ for SNOPT and $N = 500$ and $T = 0.01$ for DDP. Because of the resolution difference, trajectories generated by SNOPT are interpolated before they are used as initial trajectories for DDP. According to the ranges of corresponding variables, $\mathbf{D}_x = \mathrm{diag}(1, 1, 0.01, 0.01)$ and $\mathbf{D}_\alpha = \mathrm{diag}(10^2, 10^6)$ are used. The trajectory library is generated on a workstation with Intel(R) Xeon(TM) 3.20GHz dual-core CPU and 2G memory. It takes about 6 seconds for SNOPT to optimize one trajectory and 1 second for DDP to re-optimize it and generate local models. It takes about an hour to generate the trajectory library. For the given performance bound (for example 1000), the resultant library has 82 trajectories and 41000 local models.

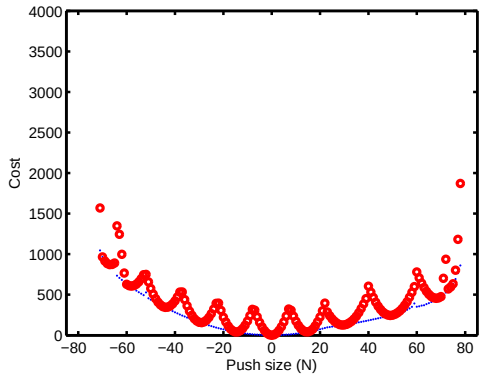

Fig. 3. Total cost comparison given the robot followed the proposed controller (red open circles) and given the robot followed an optimal trajectory (blue dots). Pushes are applied at the middle of the torso (0.4 meters above the hip) and the robot is initialized upright with zero velocity.

With the robot started standing upright with zero velocity, we compare the total costs given the robot followed the proposed controller and followed an optimal trajectory under different pushes. As shown in Fig. 3, they are close to each other if there are optimal trajectories stored in the library for similar pushes. Otherwise, the performance of the proposed controller is sub-optimal. However, the performance degradation is bounded because trajectories are stored in the library according to the performance.

In the following simulations, θ_a, θ_h, $\dot{\theta}_a$, and $\dot{\theta}_h$ denote the true values of ankle angle, hip angle, ankle angular velocity, and hip angular velocity. Their estimates are denoted by $\hat{\theta}_a$, $\hat{\theta}_h$, $\hat{\dot{\theta}}_a$, and $\hat{\dot{\theta}}_h$. τ_a and τ_h are applied torques at the ankle joint and the hip joint. The elements of the closest state and its corresponding controls found in the trajectory library are denoted as $\bar{\theta}_a$, $\bar{\theta}_h$, $\bar{\dot{\theta}}_a$, $\bar{\dot{\theta}}_h$, $\bar{\tau}_a$, and $\bar{\tau}_h$. For push and state estimation, $\text{diag}(0.01^2, 0.01^2, 0.01^2, 0.01^2, 1, 0.01^2)$ and $\text{diag}(0.01^2, 0.01^2, 0.01^2, 0.01^2)$ are used for $\mathbf{S}$ and $\mathbf{T}$, respectively.

A simulation result is shown in Fig. 4. In this simulation, a short push of 100 Newtons lasted for 0.5 seconds is applied at the middle of the torso, which is about 1.2 meters above the ground. The robot leans forward and returns to upright after the push, similar to the 'hip' strategy used by humans [8]. Another simulation result is shown in Fig. 5. In this simulation, a constant push of 70 Newtons is applied at the same place. The robot leans against the push to use the gravity to compensate for the push.

The robustness of the proposed controller is tested with a sequence of random pushes. The test push size sequence is 30, 70, 90, and 10 Newtons. Trajectories for constant pushes of 35, 70, 85, and 10 Newtons at corresponding locations are used. As shown in Figs. 6, 7, 8, and 9, for pushes of sizes and locations not in the library and changing with time, the simulated robot can still keep balance.

We compared the proposed controller with a gain scheduling controller, in which the system is modeled as a two-link inverted pendulum, linearized about the desired states, and linear quadratic regulators are designed using the same optimization criterion (17). According to the estimates of the push size and the push location, an appropriate linear quadratic regulator is used. This gain scheduling controller falls down for constant forward pushes at the middle of the torso of the robot of larger than 64 Newtons. In contrast, the proposed controller is able to handle up to 74 Newtons. The total costs for the gain scheduling controller are 2350, 1428, and 324 for constant pushes of 60, 55, and 50 Newtons. The corresponding costs for the proposed controller are 779, 369, and 247. The proposed controller performs better than the gain scheduling controller

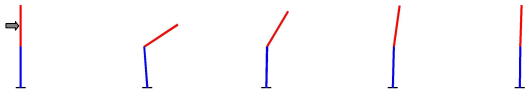

Fig. 4. Our simulated robot responding to a short push at the middle of the torso of 100 Newtons lasting for 0.5 seconds (the configuration of the robot is drawn every 0.5 seconds).

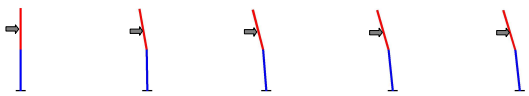

Fig. 5. Our simulated robot responding to a constant push at the middle of the torso of 70 Newtons (the configuration of the robot is drawn every 0.5 seconds).

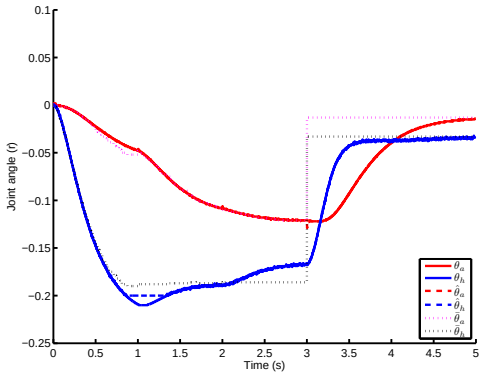

Fig. 6. Joint angles for a random push sequence.

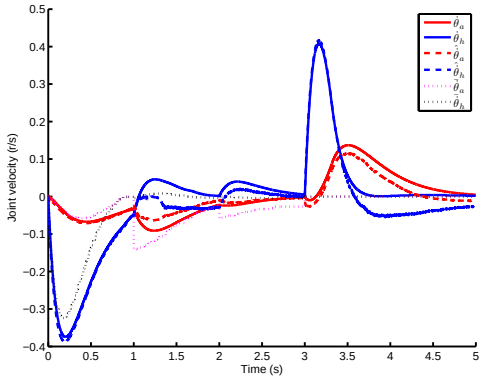

Fig. 7. Joint velocities for a random push sequence.

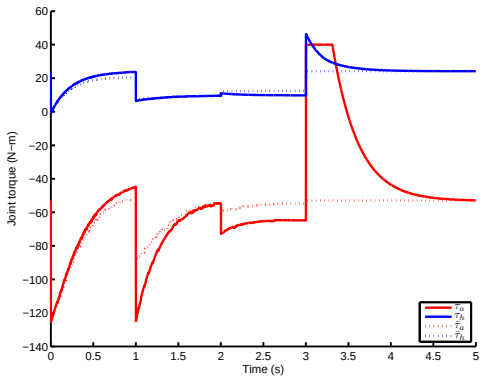

Fig. 8. Joint torques for a random push sequence.

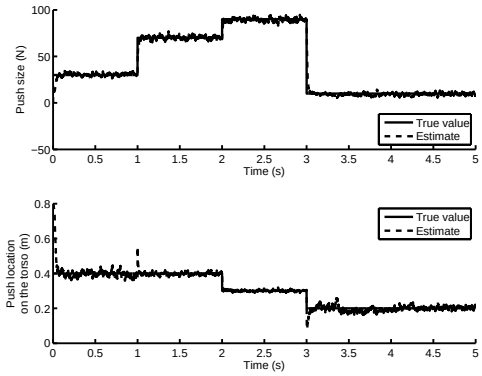

Fig. 9. Push size and location estimates for a random push sequence.

on the nonlinear robot simulation in terms of handing larger pushes and lower cost.

B. Experiment

The proposed standing balance controller was also tested on the Sarcos humanoid robot. Hydraulic actuators and load sensors are used to control the torque of each joint [22]. The following measurements are used during experiments: joint angles measured by potentiometers; the ground reaction forces measured by a force platform on which it stands; the push size measured by the force sensor at the end of the pushing device; and the push location measured by hand. All joints have stiff PD controllers except the two ankle roll joints, the two ankle pitch joints, and the two hip pitch joints. The two ankle roll joints have PD controllers of low gains to keep the feet from rolling and flat on the ground. The legs are coordinated to make the robot a planar two-link inverted pendulum:

$$\tau_{a,l} = \tau_a/2 + K_p^a(\hat{\theta}_a - \theta_{a,l}) \quad (19)$$

$$\tau_{h,l} = \tau_h/2 + K_p^h(\hat{\theta}_h - \theta_{h,l}) \quad (20)$$

$$\tau_{a,r} = \tau_a/2 + K_p^a(\hat{\theta}_a - \theta_{a,r}) \quad (21)$$

$$\tau_{h,r} = \tau_h/2 + K_p^h(\hat{\theta}_h - \theta_{h,r}) \quad (22)$$

where $\tau_{a,l}$, $\tau_{h,l}$, $\tau_{a,r}$, and $\tau_{h,r}$ are the actuation torques of the ankle pitch joint and the hip pitch joint of the left leg and the right leg, respectively. $\theta_{a,l}$, $\theta_{h,l}$, $\theta_{a,r}$, and $\theta_{h,r}$ are the angle measurements of these joints. τ_a and τ_h are the command torques of the ankle joint and the hip joint given by (13). $\hat{\theta}_a$ and $\hat{\theta}_h$ are its angle estimates. K_p^h and K_p^a are the position gains for leg coordination.

In the following experiments, we apply quickly changing and slowly changing pushes at the middle of the torso, which is 1.2 meters above the ground and 0.4 meters above the hip, in both the forward and the backward directions. Example experiment video frames are shown in Fig. 10. The push sizes and the responding joint angles and joint torques are shown in Fig. 11. The measured and estimated push size and push location are shown in Fig. 12. For both quickly changing and slowly changing pushes, the robot can keep balance using the proposed controller.

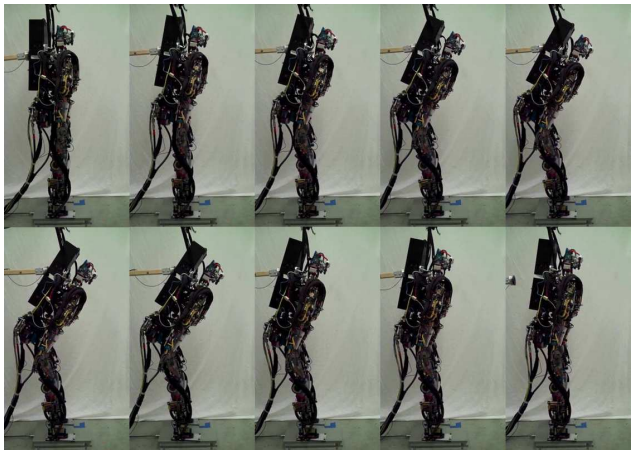

Fig. 10. Our robot responding to a continuous push while standing (video frames are shown every half second).

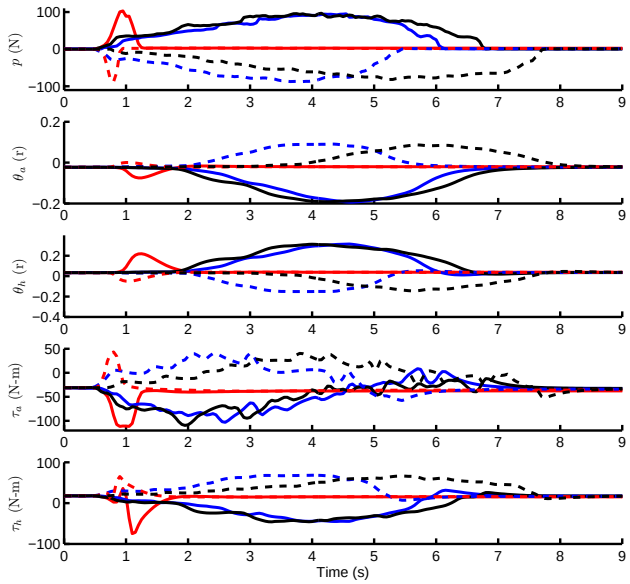

Fig. 11. Our robot responding to pushes while standing. Three pushes are in the forward direction (positive forces) and a similar range of pushes are in the backward direction.

To check our implementation, the measured pushes are given to a simulation, which is compared with the experiments. The corresponding joint angles, joint velocities, and joint torques are shown in Fig. 13. The normalized ground reaction forces are shown in Fig. 14. It is shown that the experimental results match those of our simulation. The constraints on the ground reaction forces are satisfied (as shown in Fig. 14) and the feet remain flat on the ground during the experiment (as shown in Fig. 10).

V. Conclusion and Future Work

In this paper, we use a trajectory library to synthesize a task-level controller for constrained nonlinear systems. We use a parametric trajectory optimization method to generate initial trajectories for Differential Dynamic Programming, which further refines these trajectories and generates local controllers.

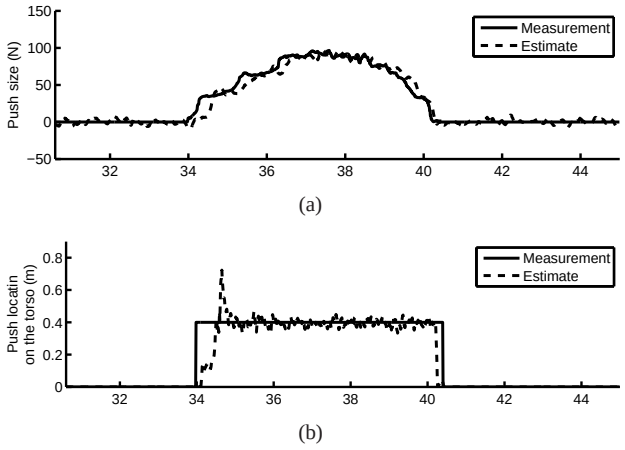

Fig. 12. a) Push size estimation b) Push location estimation

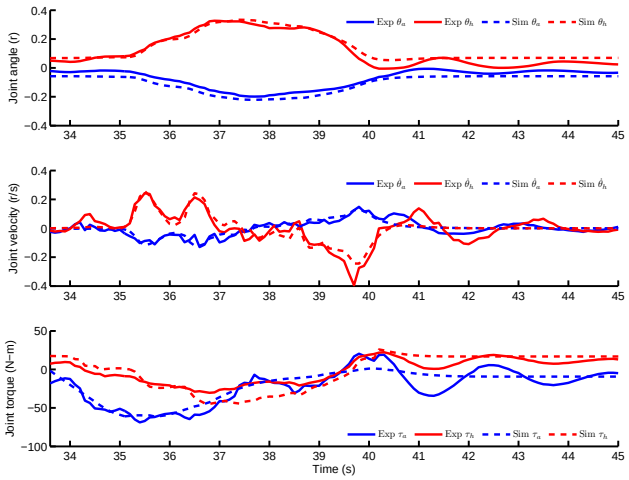

Fig. 13. Joint angles, joint velocities, and joint torques. The blue lines are of the ankle pitch joint and the red lines are of the hip pitch joint. The solid lines are the measurements of the real robot and the dashed lines are those of the simulated robot.

We force the adjacent trajectories to be consistent, which enables us to synthesize a more global controller from these local models. The optimal trajectories and local models are stored based on the simulated performance of the controller, which keeps the resultant library a reasonable size and also satisfies the performance requirements.

In our future work, we will extend our model to include a full 3D robot and use arms and knees. We will also extend the proposed method to periodic tasks like biped walking control, in which there are cyclic trajectories instead of desired states.

ACKNOWLEDGMENT

This material is based upon work supported by National Natural Science Foundation of China Key Project (Grant No. 60935001).

REFERENCES

[1] A. Goswami, "Postural stability of biped robots and the foot-rotation indicator (FRI) point," *Int. J. Rob. Res.*, vol. 18, no. 6, pp. 523–533, 1999.

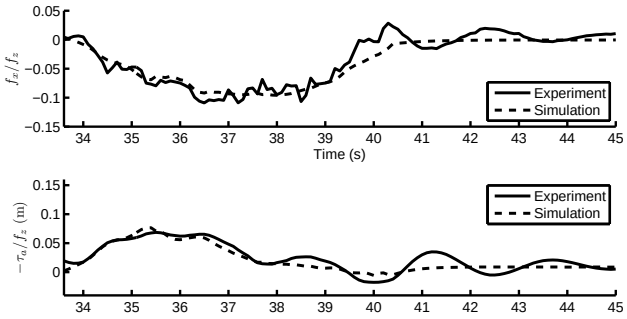

Fig. 14. The normalized ground reaction forces. The solid lines are those of the real robot, while the dashed lines are those of the simulated robot. The foot support region is $(-0.04, 0.16)$ meters.

[2] H. Hemami and P. Camana, "Nonlinear feedback in simple locomotion systems," *IEEE Trans. Autom. Contr.*, vol. 21, no. 6, pp. 855 – 860, 1976.

[3] J. Golliday, C. and H. Hemami, "Postural stability of the two-degree-of-freedom biped by general linear feedback," *IEEE Trans. Autom. Contr.*, vol. 21, no. 1, pp. 74 – 79, 1976.

[4] B. Stephens, "Humanoid push recovery," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Pittsburgh, PA, US, 2007.

[5] ——, "Integral control of humanoid balance," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 4020–4027.

[6] Y. Abe, M. da Silva, and J. Popović, "Multiobjective control with friction contacts," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. San Diego, California: Eurographics Association, 2007, pp. 249 – 258.

[7] A. Macchietto, V. Zordan, and C. R. Shelton, "Momentum control for balance," *ACM Trans. Graph.*, vol. 28, no. 3, pp. 1–8, 2009.

[8] A. Kuo, "An optimal control model for analyzing human postural balance," *IEEE Trans. Biomed. Eng.*, vol. 42, no. 1, pp. 87–101, 1995.

[9] C. G. Atkeson and B. Stephens, "Multiple balance strategies from one optimization criterion," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Pittsburgh, PA, US, 2007.

[10] R. Bellman, *Dynamic Programming*. Dover Publications, 2003.

[11] P. Dyer and S. R. McReynolds, *The Computation and Theory of Optimal Control*. Academic Press, 1970.

[12] D. Jacobson and D. Mayne, *Differential Dynamic Programming*. New York, NY, US: Elsevier, 1970.

[13] J. T. Betts, *Practical Methods for Optimal Control Using Nonlinear Programming*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2001.

[14] C. Liu and C. G. Atkeson, "Standing balance control using a trajectory library," in *Proc. Int. Conf. Intell. Robots Syst.*, St. Louis, 2009, pp. 3031–3036.

[15] C. G. Atkeson, "Using local trajectory optimizers to speed up global optimization in dynamic programming," in *Advances in Neural Information Processing Systems*, J. D. Cowan, G. Tesauro, and J. Alspector, Eds., vol. 6. Morgan Kaufmann Publishers, Inc., 1994, pp. 663–670.

[16] C. G. Atkeson and J. Morimoto, "Nonparametric representation of policies and value functions: a trajectory-based approach," in *Advances in Neural Information Processing Systems*, pp. 1643–1650, 2003.

[17] Y. Tassa, T. Erez, and W. Smart, "Receding horizon differential dynamic programming," in *Advances in Neural Information Processing Systems*, vol. 20, pp. 1465–1472, 2008.

[18] R. Tedrake, "LQR-Trees: Feedback motion planning on sparse randomized trees," in *Proc. the Robotics: Science and Systems*, 2009, p. 8.

[19] P. B. Wieber and C. Chevallereau, "Online adaptation of reference trajectories for the control of walking systems," *Robotics and Autonomous Systems*, vol. 54, no. 7, pp. 559–566, 2006.

[20] P. E. Gill, W. Murray, and M. A. Saunders, "SNOPT: An SQP algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12, pp. 979–1006, 1997.

[21] D. P. Bertsekas, *Dynamic Programming and Optimal Control*. Athena Scientific, 1995.

[22] D. C. Bentivegna, C. G. Atkeson, and J. Y. Kim, "Compliant control of

a hydraulic humanoid joint," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, 2007.