Standing Balance Control Using a Trajectory Library

Chenggang Liu and Christopher G. Atkeson

***Abstract*— This paper presents a standing balance controller. We employ a library of optimal trajectories and the neighboring optimal control method to generate local approximations to the optimal control. We take advantage of a parametric nonlinear optimization method, SNOPT, to generate initial trajectories and then use Differential Dynamic Programming (DDP) to further refine them and get their neighboring optimal control. A library generation method is proposed, which keeps the trajectory library to a reasonable size. We compare the proposed controller with an optimal controller and an LQR based gain scheduling controller using the same optimization criterion. Simulation results demonstrate the performance of the proposed method.**

I. Introduction

Humanoid robots are expected to interact with humans and complex unstructured environments, so unexpected perturbations, such as collisions with people or moving objects, are inevitable. This paper focuses on balance control during upright stance with unexpected pushes.

Bio-mechanically motivated controllers, such as [1], and intuitive controller designs, such [2], [3] have been studied. In [4], [5], optimal control and state estimation is used to explain selection of control strategies used by humans. The system is linearized and Linear Quadratic Regulators are designed for each perturbation. A form of gain scheduling is employed to account for nonlinearities caused by control and bio-mechanical constraints.

In [6], it was shown that multiple balance recovery strategies can be generated by a single optimization criterion, which means human balance control can be interpreted as optimal control. For nonlinear systems, Dynamic Programming (DP) provides a way to find globally optimal control laws. But for high dimensional systems, such as a humanoid robot, the computation and even the storage of nonlinear feedback laws becomes intractable [7]. Parametric nonlinear programming methods, such as SQP (Sequential Quadratic Programming), have been used to solve trajectory optimization for finite dimensional problems [8]. Differential Dynamic Programming (DDP), which is a second order gradient technique for trajectory optimization [9], applies the principle of optimality in the neighborhood of a nominal trajectory. This allows the coefficients of a linear or quadratic expansion of the value function to be computed along the trajectory. These coefficients may then be used to compute an improved trajectory and a local approximation to the optimal control law in its neighborhood, which can be used to get a feedback control law [10]. We take advantage of parametric nonlinear programming methods to generate initial trajectories, which are then refined by DDP to produce local control laws and more optimal trajectories.

Chenggang Liu is with the Department of Automation, Shanghai Jiao Tong University, Shanghai, China 200240 cgliu2008@gmail.com

Christopher G. Atkeson is with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA 15213 cga@cmu.edu

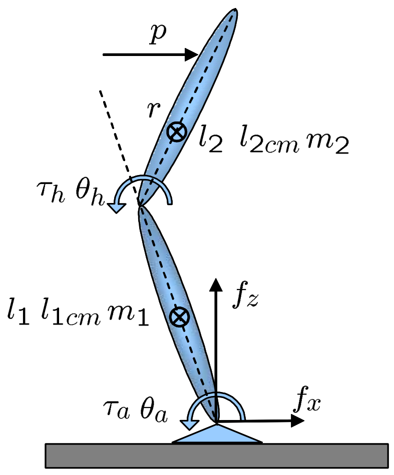

Fig. 1. Two-link robot model.

Most previous work assumes that pushes are instantaneous and change the joint velocities instantaneously. In practice, the pushes may last for a while. The proposed controller can handle instantaneous and continuous pushes.

The rest of the paper is organized as follows. In section II, the robot model and the optimization criterion are proposed. Section III describes the neighboring optimal control method. Section IV and V propose the optimal trajectory library generation method and the balance controller. Simulation results are provided in section VI to demonstrate the validity and the performance of the proposed method. Conclusions and future work are discussed in Section VII.

II. Robot Model

A two-link inverted pendulum model in the sagittal plane is modeled, as shown in Fig. 1. The parameters are listed in Table I, where l_{1cm} and l_{2cm} are the distances from the center of mass (CoM) of each link to the joint below, and MoI$_1$ and MoI$_2$ are the momentum of inertia of each link about its CoM. The ankle angle is bounded by $-0.52 < \theta_a < 0.79$ radians. The hip angle is bounded by $-2.18 < \theta_h < 0.52$ radians. $\theta_a = 0$ and $\theta_h = 0$ is upright. The ankle velocity is bounded by $-4.6 < v_a < 4.6$ radians/second. The hip velocity is bounded by $-7.7 < v_h < 7.7$ radians/second. The maximum hip torque is ± 157 Newton-meters. Ankle torque is limited to prevent the foot from tilting. We use a symmetric foot 0.2 meters long in our model. Assuming that

TABLE I

PARAMETERS OF THE ROBOT MODEL

l_1 (m)	0.661	l_2 (m)	0.653
l_{1cm} (m)	0.430	l_{2cm} (m)	0.141
m_1 (Kg)	19.474	m_2 (Kg)	29.492
MoI$_1$ (Kg.m^2)	0.696	MoI$_2$ (Kg.m^2)	1.03356

in standing the center of pressure is at the center of the foot, then the maximum ankle torque is ± 50 Newton-meters. A horizontal push is applied on some point of a link, where p is the size of push and r is the distance from the point of action to the joint below.

The one step optimization criterion is the weighted sum of the squared deviations of the current state from the desired state and the squared joint torques:

$$L(\mathbf{x}, \mathbf{u}) = T(\mathbf{x} - \mathbf{x}_d)^T \mathbf{Q}(\mathbf{x} - \mathbf{x}_d) + T\mathbf{u}^T \mathbf{R}\mathbf{u}, \qquad (1)$$

where T is the time step of the simulation ($0.01s$), $\mathbf{x}^T = (\theta_a, \theta_h, v_a, v_h)$ is the current state, $\mathbf{u}^T = (\tau_a, \tau_h)$ is the control vector, $\mathbf{x}_d$ is the desired state, which is the static equilibrium state for a specified push, $\mathbf{Q}$ and $\mathbf{R}$ are both currently identity matrices with appropriate dimensions.

III. NEIGHBORING OPTIMAL CONTROL

Given the dynamics of the robot:

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k), p(k), r(k)), \qquad (2)$$

where $p(k)$ is the push size, $r(k)$ is the push location, and the optimal return function

$$V^{opt}(\mathbf{x}^{opt}(k)) = L(\mathbf{x}^{opt}(k), \mathbf{u}^{opt}(k)) + V^{opt}(\mathbf{x}^{opt}(k+1)), \qquad (3)$$

where $\mathbf{u}^{opt}(k)$ is the optimal control for the state, $\mathbf{x}^{opt}(k)$. The neighboring optimal control is given by [11]:

$$\mathbf{u}(k) = \mathbf{u}^{opt}(k) - \mathbf{K}^{opt}(k)(\mathbf{x}(k) - \mathbf{x}^{opt}(k)) \qquad (4)$$

and

$$\mathbf{K}^{opt}(k) = -\frac{\partial \mathbf{u}^{opt}(k)}{\partial \mathbf{x}^{opt}(k)}. \qquad (5)$$

In order to compute $\mathbf{K}^{opt}(k)$, the partial derivatives V_x and V_{xx} have to be computed along the trajectory. Given an optimal trajectory, one can integrate $V(k)$, $V_x(k)$, and $V_{xx}(k)$ backward in time starting from the end of the trajectory [9].

The neighboring optimal control law is a local linear model for the optimal policy in the neighborhood of the optimal trajectory. Therefore, a closed-loop feedback control solution can be given by:

$$\mathbf{u}(\mathbf{x}) = \bar{\mathbf{u}} - \bar{\mathbf{K}}(\mathbf{x} - \bar{\mathbf{x}}), \qquad (6)$$

where $\bar{\mathbf{x}}$ is the closest state on the optimal trajectory to the current state, $\mathbf{x}$, $\bar{\mathbf{u}}$ and $\bar{\mathbf{K}}$ are the optimal control and the feedback gain matrix corresponding to $\bar{\mathbf{x}}$.

IV. TRAJECTORY LIBRARY GENERATION

A. Trajectory Library on a Uniform Grid

One way to generate the initial trajectories is to use motion capture data of humans. The effectiveness will largely depend on how close the dynamic models used by the robot and by humans are. Another way is to use parametric nonlinear programming methods. Parametric nonlinear programming methods have been used to solve trajectory optimization problems [8]. We find they are generally more robust in terms of finding a solution than DDP.

SNOPT is a general-purpose system for constrained optimization using a sparse sequential quadratic programming (SQP) method [12]. We use it to generate initial trajectories for different conditions. For standing balance control, a selection of initial conditions is considered. For constant pushes, the initial joint angles and velocities are all zero. The push size, p, and the push location, r, are not zero. For instantaneous pushes, the initial joint velocities are not zero. The initial joint angles, the push size, and the push location are all zero. For constant pushes, the robot eventually leans into the pushes and attains zero joint torque. In order to balance after the constant pushes are removed, initial conditions with nonzero joint angles should also be considered. For each type of push, initial conditions are generated on a uniform grid. Trajectories are optimized by SNOPT for each initial condition. For example, we use 10 Newtons as the push magnitude step size, 0.3 meters as the push location step size, and generate initial trajectories on a uniform grid for constant pushes on the torso.

B. Optimal Trajectory Generation For the Library

After creating these trajectories, we use DDP to refine them and store the trajectories and their feedback gain matrices in the library. Given a good starting trajectory, DDP can find better solution rapidly. Therefore, in order to generate an optimal trajectory for any specified initial condition, we use the closest trajectory in the library to generate the starting trajectory, which is then refined by DDP. The distant is given by $(\mathbf{x}^T, p, r)\mathbf{D}(\mathbf{x}^T, p, r)^T$. The range of p is about -80 to $+80$ and the range of r is between 0 and 0.653. They both have large effects on the dynamics, so we give large weights to them, which are 10 and 1000, respectively. The range of joint velocity is about 10 times that of joint angle, so the weight 0.1 for velocities and 1 for joint angles is used. The starting trajectory is generated by the neighboring optimal control of the closest library trajectory. DDP is then used to improve it, which not only gives the optimal trajectory, but also its control gains. As the starting trajectories are close to the final solutions, the convergence of DDP is rapid.

C. Trajectory Library on an Adaptive Grid

We also propose an adaptive grid of initial conditions for the trajectories stored in the library. The optimization is started from the zero push, the neighboring optimal control of the closest library trajectory is used for the current push, which gives the total cost, $C'(p)$. Compare it with the total

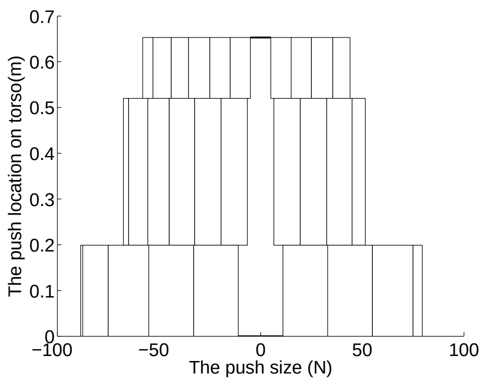

Fig. 2. The lookup table for constant pushes on the torso.

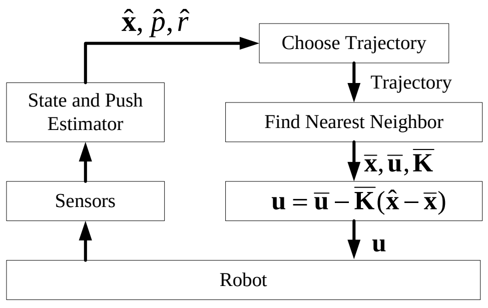

Fig. 3. Standing balance controller architecture.

cost of the optimal control, $C(p)$. Increase the push size gradually until $|C'(p) - C(p)| \geq C_{limit}$. Save the last push size, p_{last}, as the control region's upper boundary of the currently used library trajectory. Then continue to increase the push size, generate the optimal trajectory, and use its neighboring optimal control for the push of p_{last}, which gives the total cost, $C'(p_{last})$. Save the last optimal trajectory into the library before $|C'(p_{last}) - C(p_{last})| \geq C_{limit}$. Repeat the above steps until the control region is too small or DDP can not find a feasible optimal solution.

With the increase of push size, the control region of the neighboring optimal control becomes small. The stop condition prevents it from saving too many trajectories which can only control small regions, and searching infeasible initial conditions. Thus computation and storage can be saved. In addition, only trajectories contribute to the performance are saved, which keeps the final library to a reasonable size. For example, we use 1000 as C_{limit} and generate trajectories to handle constant pushes on the torso. The final library has only 30 trajectories. The result is shown as Fig. 2, in which each block defines a control region of one optimal trajectory. The middle large region uses the optimal trajectory for a zero push, which is LQR controller actually.

V. Balance Controller

A. Controller Architecture

The standing balance controller is shown in Fig. 3. In each time step, the state estimate, $\hat{\mathbf{x}}$, the push size estimate, $\hat{p}$, and the push location estimate, $\hat{r}$ are calculated. Then one trajectory is chosen from the library. Given the optimal trajectory and its neighboring optimal control feedback gains, we get a local linear approximation to the optimal control law in its neighborhood. According to the current state estimate, $\hat{\mathbf{x}}$, the closest state on the optimal trajectory, $\bar{\mathbf{x}}$, along with the corresponding control, $\bar{\mathbf{u}}$, and the feedback gain matrix, $\bar{\mathbf{K}}$ are used. The state distance is given by $\mathbf{x}^T\mathbf{D}'\mathbf{x}$, where $\mathbf{D}'$ is diag(1,1,0.1,0.1) currently. The output of the controller is thus given by:

$$\mathbf{u} = \bar{\mathbf{u}} - \bar{\mathbf{K}}(\mathbf{x} - \bar{\mathbf{x}}). \tag{7}$$

B. Online State and Push Estimation

We have no sensor for the joint velocities, the push size, and the push location, which have to be estimated. We employ a new state variable, $\mathbf{y}^T = (\theta_a, \theta_h, v_a, v_h, p, r)$ and an observation, $\mathbf{z}^T = (\bar{\theta}_a, \bar{\theta}_h, \bar{f}_x, \bar{f}_z)$, where $\bar{\theta}_a$ and $\bar{\theta}_h$ are noisy measurements of the ankle angle and the hip angle, $\bar{f}_x$ and $\bar{f}_z$ are noisy measurements of the ankle forces, as shown in Fig. 1. Therefore, the state transition and the observation model are given by:

$$\mathbf{y}(k+1) = \mathbf{g}(\mathbf{y}(k), \mathbf{u}(k)) + \mathbf{w} \tag{8}$$

$$\mathbf{z}(k) = \mathbf{h}(\mathbf{y}(k), \mathbf{u}(k)) + \mathbf{v} \tag{9}$$

$$\mathbf{w} \sim N(0, \mathbf{S}) \qquad \mathbf{v} \sim N(0, \mathbf{T}) \tag{10}$$

$$\mathbf{g}(\mathbf{y}, \mathbf{u}) = \begin{bmatrix} \mathbf{f}(\mathbf{x}(k), \mathbf{u}(k), p(k), r(k)) \\ p(k) \\ r(k) \end{bmatrix} \tag{11}$$

$$\mathbf{h}(\mathbf{y}, \mathbf{u}) = \begin{bmatrix} \theta_a \\ \theta_h \\ f_x(\mathbf{y}(k), \mathbf{u}(k)) \\ f_z(\mathbf{y}(k), \mathbf{u}(k)) \end{bmatrix}, \tag{12}$$

where $\mathbf{f}(.)$ is the dynamics of the robot, the noise terms $\mathbf{w}$ and $\mathbf{v}$ are uncorrelated, $\mathbf{S}$ and $\mathbf{T}$ are covariance matrices. The state transition model and the observation model are both nonlinear, so the Extended Kalman Filter is employed [13]. The Extended Kalman Filter linearizes the nonlinear state transition model and the observation model as

$$\mathbf{F}(k) = \left.\frac{\partial \mathbf{g}}{\partial \mathbf{y}}\right|_{\hat{\mathbf{y}}(k-1|k-1), \mathbf{u}(k-1)} \tag{13}$$

$$\mathbf{H}(k) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{y}}\right|_{\hat{\mathbf{y}}(k|k-1), \mathbf{u}(k-1)}. \tag{14}$$

To predict the next state before measurements are taken:

$$\hat{\mathbf{y}}(k|k-1) = \mathbf{g}(\hat{\mathbf{y}}(k-1|k-1), \mathbf{u}(k-1)) \tag{15}$$

$$\mathbf{P}(k|k-1) = \mathbf{F}(k)\mathbf{P}(k-1|k-1)\mathbf{F}^T(k) + \mathbf{S} \tag{16}$$

To update the state after measurements are taken:

$$\mathbf{z}_{err} = \mathbf{z}(k) - \mathbf{h}(\hat{\mathbf{y}}(k|k-1), \mathbf{u}(k-1)) \tag{17}$$

$$\mathbf{K}(k) = \mathbf{P}(k|k-1)\mathbf{H}^T(\mathbf{H}\mathbf{P}(k|k-1)\mathbf{H}^T + \mathbf{T})^{-1} \tag{18}$$

$$\hat{\mathbf{y}}(k|k) = \hat{\mathbf{y}}(k|k-1) + \mathbf{K}(k)\mathbf{z}_{err} \tag{19}$$

$$\mathbf{P}(k|k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H})\mathbf{P}(k|k-1), \tag{20}$$

where $\mathbf{K}$ is the Kalman gain matrix and $\mathbf{P}$ is the covariance matrix for the state estimation.

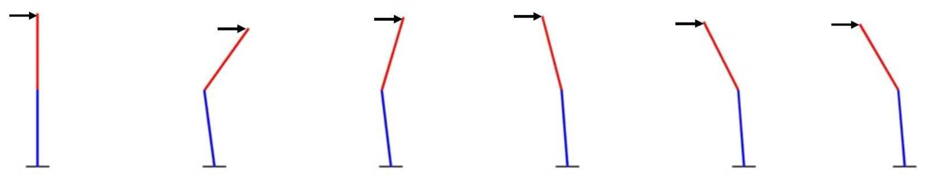

Fig. 4. The robot under the constant forward push at the head of 42 Newtons. The frames are taken in intervals of 0.3 seconds.

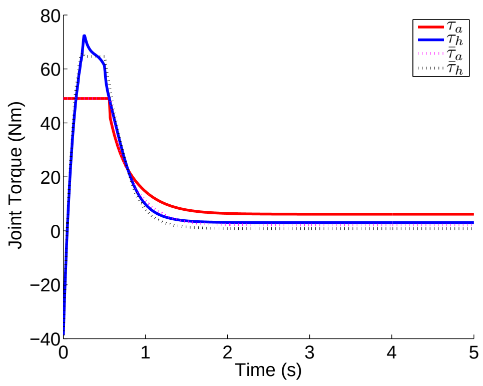

Fig. 5. The joint torques for 42 Newtons forward push at the head.

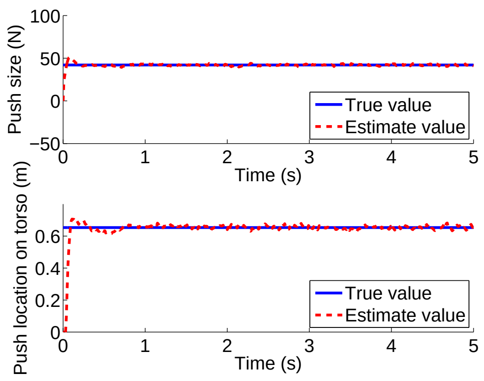

Fig. 8. Push size and location estimates for 42 Newtons forward push at the head.

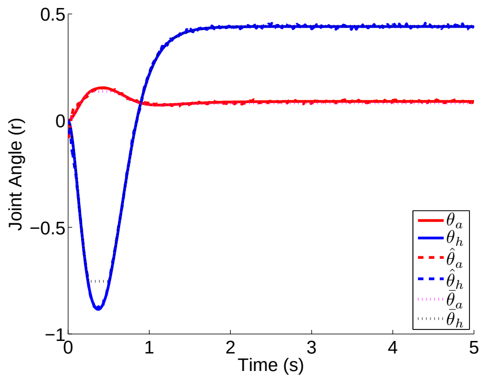

Fig. 6. The joint angles for 42 Newtons forward push at the head.

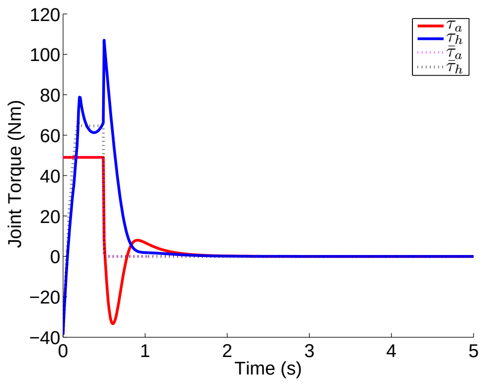

Fig. 9. Joint torques for the short forward push at head of 50 Newtons, lasting 0.5 seconds.

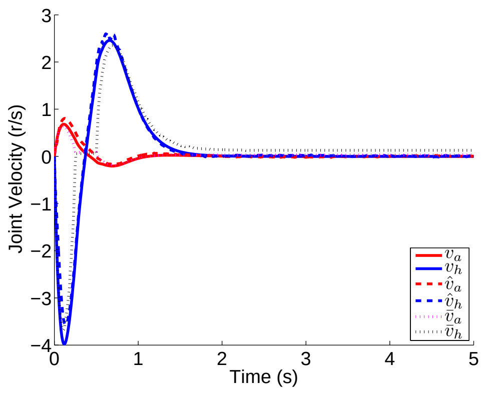

Fig. 7. The joint velocities for 42 Newtons forward push at the head.

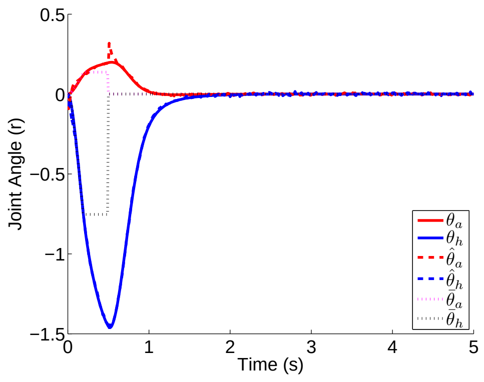

Fig. 10. Joint angles for the short forward push at head of 50 Newtons, lasting 0.5 seconds.

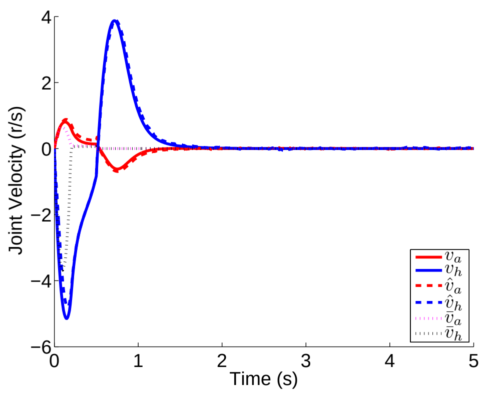

Fig. 11. Joint velocities for the short forward push at head of 50 Newtons, lasting 0.5 seconds.

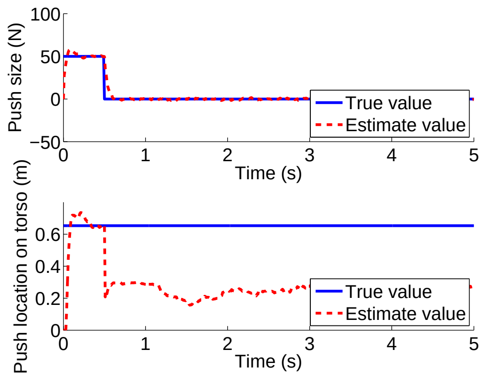

Fig. 12. Push size and location estimates for the short forward push at head of 50 Newtons, lasting 0.5 seconds.

VI. Simulation Results

In the following simulation, the state transition covariance matrix S is diag(0.01^2, 0.01^2, 0.01^2, 0.01^2, 1, 0.01^2) and the observation covariance matrix T is diag(0.01^2, 0.01^2, 0.01^2, 0.01^2). θ_a, θ_h, v_a, and v_h denote the true values of ankle angle, hip angle, ankle velocity, and hip velocity. Their estimates are denoted by $\hat{\theta}_a$, $\hat{\theta}_h$, $\hat{v}_a$, and $\hat{v}_h$. $\bar{\theta}_a$, $\bar{\theta}_h$, $\bar{v}_a$, and $\bar{v}_h$, $\bar{\tau}_a$, and $\bar{\tau}_h$ are elements of the closest state and its corresponding controls in the trajectory library. τ_a and τ_h are applied torques at the ankle joint and the hip joint.

In the first simulation, a constant push of 42 Newtons at the head in the forward direction is applied. There is not trajectory for exactly the same push in the library, so the closest trajectory for constant push of 39.5 Newtons at the head is selected. As shown in Figs. 4 and 5, the robot employs hip torque to accelerate the torso, bends forward quickly, and then it leans backward into the push in order to use gravity to balance the push. Finally, all joint torques tend to zero. As shown in Figs. 6, 7, and 8, the state and push estimates approach to the true values in a very short time.

In the second simulation, a large short push at the head of 50 Newtons in the forward direction lasting 0.5 seconds is tested. As shown in Figs. 9, 10, 11, and 12, the robot uses hip torque to accelerate the torso, bends forward, and finally

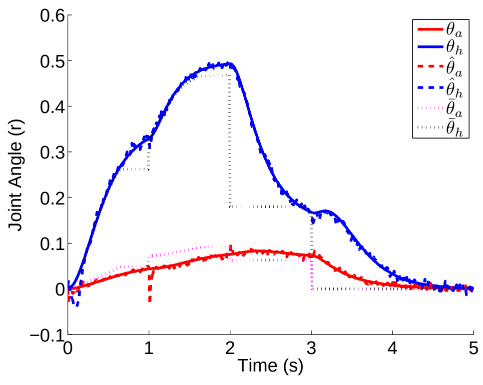

Fig. 13. Joint angles for random pushes sequence.

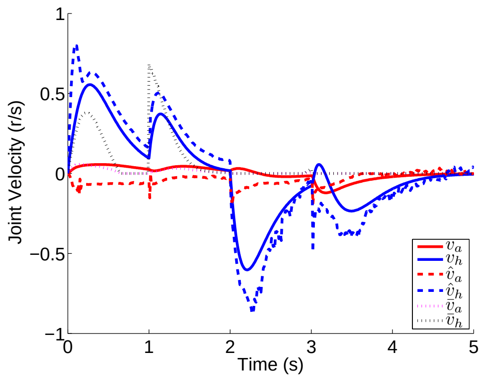

Fig. 14. Joint velocities for random pushes sequence.

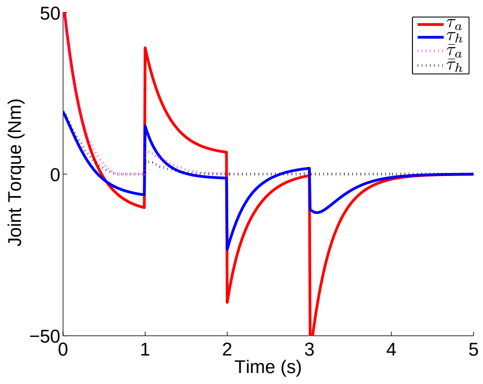

Fig. 15. Joint torques for random pushes sequence.

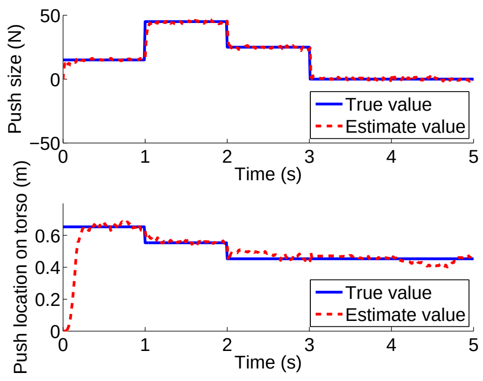

Fig. 16. Push size and location estimates for random pushes sequence.

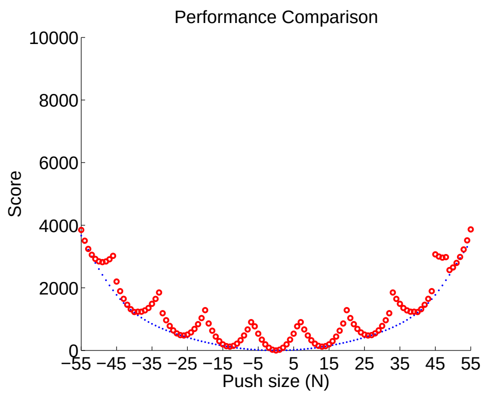

Fig. 17. Performance comparison when pushes are applied at 0.353 m of the torso. The blue dots are total costs under the optimal controller, while the red open circles are total costs under the proposed controller.

recovers its posture to be upright. It is also shown that the state and push estimates approach to the true values in a very short time. In Fig. 12, the push location estimate is wrong when the push size is zero. However, this error can be ignore because of a zero push.

The robustness of the proposed controller is tested with a sequence of random pushes. The test push size sequence is 15, 45, and 25 Newtons. Trajectories for constant pushes of 20, 39.5, and 26 Newtons are used. As shown in Figs. 13, 14, 15, and 16, for pushes of sizes and locations not in library and changing with time, the robot can still keep balance.

For different push sizes and push locations on the torso, the performance of the proposed controller is compared with that of the optimal controller using the same optimization criterion. As shown in Fig. 17, the performance of the proposed controller is close to that of the optimal controller when there are trajectories in the library for the pushes that are close to the pushes applied. It becomes worse when the applied pushes are far from what is in the library. Because the trajectory library is generated according to the performance, the performance degradation is bounded.

We have also designed a gain scheduling controller based on Linear Quadratic Regulators (LQR). It linearizes the system about the equilibrium state for each push size and push location. LQR controllers are then designed according to the same optimization criterion. According to the push size and the push location, an appropriate LQR controller is used. This gain scheduling controller falls down for constant forward pushes at the head of 36 Newtons. In contrast, the controller proposed here is able to handle constant forward pushes up to 55 Newtons.

VII. Conclusion and Future Work

In this paper, a balance controller based on a trajectory library is proposed. We demonstrate that a trajectory library can be used for constrained nonlinear system control, such as a humanoid robot standing balance control. Taking balance control as an optimal control problem, the trajectory library and the neighboring optimal control method are used to generate local linear approximations to the optimal control. Differential Dynamic Programming (DDP) is used to generate the optimal trajectories and the neighboring optimal control. A nonlinear programming method, SNOPT, is used to generate starting trajectories for DDP refinement, which makes the convergence rapid.

The proposed trajectory library generation method saves computation. It makes the final library compact but also satisfy the performance requirements. The trajectories and thus the linear approximation to the optimal control law can be accessed effectively using a lookup table, which make the proposed controller applicable for real-time control.

In our future work, robots with more links will be studied. For example, the 'squat strategy' can be generated if the robot has knee joints. Actually implementing this algorithm on a robot is also expected. This will require dealing with floor compliance and coordinating both legs and feet. Finally, we would like to extend our model to include a full 3D humanoid robot.

References

[1] M. Abdallah and A. Goswami, "A biomechanically motivated two-phase strategy for biped upright balance control," in *Proc. IEEE International Conference on Robotics and Automation (ICRA'2005)*, April 2005, pp. 1996–2001.

[2] B. Stephens, "Integral control of humanoid balance," in *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS'2007)*, October 2007, pp. 4020–4027.

[3] ——, "Humanoid push recovery," in *Proc. The IEEE-RAS 2007 International Conference on Humanoid Robots*, Pittsburgh, PA, US, November 2007.

[4] A. Kuo, "An optimal control model for analyzing human postural balance," *IEEE Transactions on Biomedical Engineering*, vol. 42(1), pp. 87–101, January 1995.

[5] S. Park, F. Horak, and A. Kuo, "Postural feedback responses scale with biomechanical constraints in human standing," *Experimental Brain Research*, vol. 154(4), pp. 417–427, February 2004.

[6] C. Atkeson and B. Stephens, "Multiple balance strategies from one optimization criterion," in *Proc. The IEEE-RAS 2007 International Conference on Humanoid Robots*, Pittsburgh, PA, US, November 2007.

[7] ——, "Random sampling of states in dynamic programming," *IEEE Transactions on Ssytems, Man, and Cybernetics Part B: Cybernetics*, vol. 38(4), pp. 924–929, August 2008.

[8] M. Hardt, "'multibody dynamical algorithms, numerical optimal control, with detailed studies in the control of jet engine compressors and biped walking," Ph.D. dissertation, Univ. California, San Diego, CA, US, 1999.

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

[10] A. Bryson and Y. Ho, *Applied optimal control: Optimization, estimation, and control*. Hemisphere Pub, 1969.

[11] P. Dyer and S. McReynolds, *The computation and theory of optimal control*. New York, Academic Press, 1970.

[12] P. Gill, W. Murray, and M. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM Journal on Optimization*, vol. 12(4), pp. 979–1006, 2002.

[13] F. Orderud, "Comparison of kalman filter estimation approaches for state space models with nonlinear measurements," in *Proc. of Scandinavian Conference on Simulation and Modeling*, 2005.