Combining Model-Based Policy Search with Online Model Learning for Control of Physical Humanoids

Igor Mordatch, Nikhil Mishra, Clemens Eppner, Pieter Abbeel
Department of Computer Science and Engineering, University of California, Berkeley, CA, USA.

***Abstract*— We present an automatic method for interactive control of physical humanoid robots based on high-level tasks that does not require manual specification of motion trajectories or specially-designed control policies. The method is based on the combination of a model-based policy that is trained off-line in simulation and sends high-level commands to a model-free controller that executes these commands on the physical robot. This low-level controller simultaneously learns and adapts a local model of dynamics on-line and computes optimal controls under the learned model. The high-level policy is trained using a combination of trajectory optimization and neural network learning, while considering physical limitations such as limited sensors and communication delays. The entire system runs in real-time on the robot's computer and uses only on-board sensors. We demonstrate successful policy execution on a range of tasks such as leaning, hand reaching, and robust balancing behaviors atop a tilting base on the physical robot and in simulation.**

I. Introduction

The ability to control bipedal robots to autonomously perform a wide variety of tasks is one of the standing grand challenges of robotics [24]. Furthermore, if we wish for these robots to be truly autonomous and be widely used, we cannot rely on extensive hand-design and manual engineering specific to each task or robotic platform.

Controlling humanoid robots is particularly difficult because under-actuation can quickly lead to failures such as robot falling down, leading to a region of successful movement strategies that is very narrow and prevents exploration. Additionally, humanoid robot dynamics - especially contact dynamics - are very difficult to model and typically fall outside the class of models provided by physics simulators. We are also interested in developing methods applicable to lower cost robotic platforms using only on-board sensors as opposed to an instrumented laboratory setting. This comes with additional challenges: limited and noisy sensing capabilities, limited communication bandwidth and delays, and limited computational performance.

We tackle these challenges with a combination of model-based and model-free techniques. High-level movement strategies are discovered using a neural network policy trained offline from model-based trajectory optimizations performed in simulation, following [16]. The simulation environment allows for effectively unlimited exploration and can discover complex movement behaviours [18]. The resulting policy can be thought of as an imaginary process that specifies how the movement should progress, but not the details of how to accomplish it. The high-level policy is combined with a low-level model-free controller that specifies the detailed controls sent to the robot. This controller simultaneously learns a local dynamics model on-line and the optimal control signal under that model to accomplish the high-level commands given by the policy. Policy execution, model learning and control all execute in closed-loop and feed back into each other. We also show that for a high-level policy to be successful on a physical robot, it must be trained with consideration of limitations such as sensor availability and communication bandwidth. Thus we explicitly model these limitations in our simulation.

As a result, we are able to stably execute policies on a physical robot that perform tasks such as significant torso orientation changes or hand reaching movements that do not require knowledge about the global position of the hand. We also present robust balancing strategies that adapt to moving and tilting ground and take protective footsteps in simulation (we believe the latter can be done on a physical robot once we acquire foot force sensors).

II. Related Work

To date, a large number of effective approaches to controlling bipedal humanoid robots rely on either execution of pre-created movement trajectories, or manually designed controllers and regulation of control features [2], [5], [11], [20], [21], [24], [25]. However, such controllers require manual design and engineering efforts specific to each task and robot platform and can lack robustness. Some control approaches break up the control problem into a high-level and low-level components, as we do [3], [17], [22], but again rely on manual design in both parts of the hierarchy.

We instead follow an approach of learning control policies based on simulation, following recent approaches [14], [16], [23]. However, transferring these approaches to control physical humanoids is not straightforward. [15] learn time-varying Gaussian mixture dynamics model for a fixed-based PR2 robot based on iteratively updating control policy and dynamics model. A similar approach using Gaussian Process policy and dynamics model is used by [4] for an inverted pendulum and fixed-base robot arm. The issue is that executing such control policies on under-actuated bipeds can quickly lead to falling and destabilization and does not provide useful data to learn dynamics from. Furthermore, contact events on legged robots present discontinuities which make it difficult to learn a model that is globally accurate, especially when it is used with a trajectory optimizer which may exploit the errors in the dynamics model.

Instead of learning a global dynamics model from training motions off-line, it is possible to learn and adapt a locally-accurate model on-line during actual task execution. This is the approach we take. The local models can either be linear [10], [28], or nonlinear [13]. However, such models were again applied to fixed-base systems and used for executing pre-specified motions. We must include a more model-based high-level policy that will perform complex high-level movements that will satisfy the task.

III. Model-Based Policy Search with Trajectory Optimization

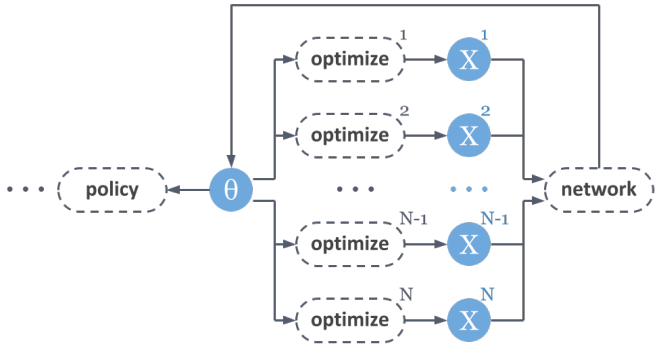

Fig. 1. Overview of our policy learning approach. A neural network training machine (*network*) generates network parameters θ representing the policy learned from a robot's optimal trajectories $\mathbf{X}$. These parameters are used for generating new trajectories close to the policy (*optimize*), and passed as high-level controls for the physical robot.

In this section, we describe how to search for high-level control policies in simulation given a model of the robot's dynamics, following the approach in [16]. Let the full state of the robot be defined as $[\mathbf{r}\ \mathbf{q}\ \mathbf{f}]$, where $\mathbf{r} \in \mathcal{R}^6$ is the root position and orientation of the robot, $\mathbf{q}$ are the joint angles of the robot and $\mathbf{f}$ are the contact forces being applied on the robot by the ground. The motion of the robot is a state trajectory of length T defined by $\mathbf{X} = \left[\mathbf{r}^0\ \mathbf{q}^0\ \mathbf{f}^0\ ...\ \mathbf{r}^T\ \mathbf{q}^T\ \mathbf{f}^T\right]$. Let $\mathbf{X}^1, ..., \mathbf{X}^N$ be a collection of N trajectories, each starting with different initial conditions and executing a different task (such as moving the robot's hand to a particular location).

We introduce a neural network control policy $\boldsymbol{\pi}_\theta : \mathbf{s} \mapsto \mathbf{a}$ parametrized by neural network weights θ that maps a sensory state of the robot $\mathbf{s}$ at each point in time to a high-level control action $\mathbf{a}$ that controls the robot. In general, the sensory state can be designed by the user to include arbitrary informative features, but in this work we only use the on-board sensors available to our robot:

$$\mathbf{s}^t = \left[\mathbf{q}^t\ \dot{\mathbf{q}}^t\ \mathbf{o}^t\ \boldsymbol{\omega}^t\right],$$

where $\dot{\mathbf{q}}^t$ denotes the velocity of $\mathbf{q}$ at time t. $\mathbf{o}$ and $\boldsymbol{\omega}$ are the acceleration due to gravity and angular acceleration of the robot respectively, both typical components of an inertial measurement unit (IMU). Note that additional sensors (such as pressure, motor torque, or motion capture markers) can easily be incorporated into this framework without any special handling.

The high-level control action output by the policy contains the next timestep's joint position and velocity

$$\mathbf{a}^t = \left[\mathbf{q}^{t+1}\ \dot{\mathbf{q}}^{t+1}\right].$$

With this representation of the action, the policy directly commands the desired trajectory of the robot, rather than commanding low-level features such as motor torques. Thus, our network learns both optimal controls and a model of simulator dynamics. Note that the policy does not command root position or orientation, as we cannot directly sense these features on the physical robot and thus cannot regulate them.

Let $C_i(\mathbf{X})$ be the total cost of the trajectory $\mathbf{X}$, which rewards accurate execution of task i and physical correctness of the robot's motion. We want to jointly find a collection of optimal trajectories that each complete a particular task, along with a policy $\boldsymbol{\pi}_\theta$ that is able to reconstruct the sense and action pairs $\mathbf{s}^t(\mathbf{X})$ and $\mathbf{a}^t(\mathbf{X})$ of all trajectories at all timesteps:

$$\begin{aligned} \underset{\theta\ \mathbf{X}^1\ ...\ \mathbf{X}^N}{\text{minimize}} \quad & \sum_i C_i(\mathbf{X}^i) \\ \text{subject to} \quad & \forall\, i, t:\ \mathbf{a}^t(\mathbf{X}^i) = \boldsymbol{\pi}_\theta(\mathbf{s}^t(\mathbf{X}^i)). \end{aligned} \tag{1}$$

The optimized policy parameters θ can then be used to execute policy in real-time and interactively control the robot by the user. Unlike non-parametric methods like motion graphs [12] or Gaussian processes [4], we do not need to keep any trajectory data at policy execution time.

A. Stochastic Policy Inputs

Injecting noise has been shown to produce more robust movement strategies in optimal control [9], [26] and reduce overfitting and prevent feature co-adaptation in neural network training [7]. We inject noise in a principled way to aid in learning policies that do not diverge when rolled out at execution time.

In particular, we inject additive Gaussian noise into the sensory inputs $\mathbf{s}$ given to the neural network. Let the sensory noise be denoted $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\sigma}_\varepsilon^2 I)$, so the resulting noisy policy inputs are $\mathbf{s} + \boldsymbol{\varepsilon}$. This change in input also induces a change in the optimal action to take. If the noise is small enough, the optimal action at nearby noisy states is given by the first order expansion

$$\mathbf{a}(\mathbf{s} + \boldsymbol{\varepsilon}) = \mathbf{a} + \mathbf{a}_\mathbf{s}\boldsymbol{\varepsilon}, \tag{2}$$

where $\mathbf{a}_\mathbf{s}$ (alternatively $\frac{d\mathbf{a}}{d\mathbf{s}}$) is the matrix of optimal feedback gains around $\mathbf{s}$. These gains can be calculated as a byproduct of trajectory optimization as described in section IV. Intuitively, such feedback helps the neural network trainer to learn a policy that can automatically correct for small deviations from the optimal trajectory.

B. Delayed and Rate-Limited Sensory Inputs and Controls

The physical robots we are interested in exhibit significant limitations in the rate at which sensors and controls are communicated to and from the robot. As a consequence,

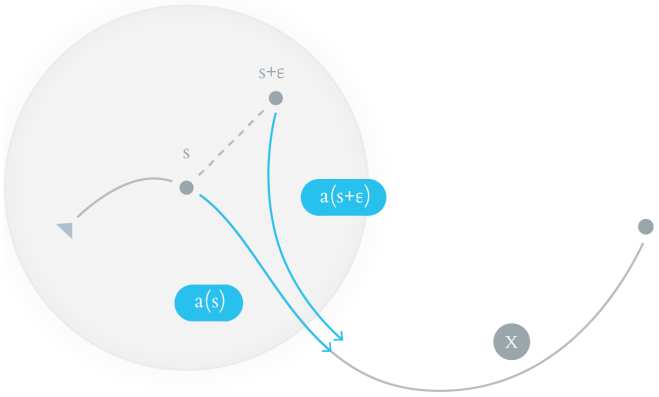

Fig. 2. For a given optimal trajectory $\mathbf{X}$, the optimal actions $\mathbf{a}$ change in the neighborhood of $\mathbf{s}$ to compensate for deviation ε.

the policy cannot finely micromanage the behavior of the robot. Furthermore, the policy must output actions based on sensory information that is significantly delayed. On our experimental robot platform Darwin-OP, these delays are around 40-60 milliseconds. As we will show in section VII, if we ignore these effects during trajectory optimization and policy learning, our policies quickly destabilize when executed on either the physical system or in a simulated environment with delays. Researchers in biomechanics and graphics have found that explicitly incorporating sensory delays has an impact on the resulting movement behaviors [6], [27]

To incorporate communication rate limits and delays into our method, we introduce *communication* timesteps $t_1, \ ..., \ t_K$ which are a subset of $1, ..., T$ that correspond to times when sensors are read and controls are sent (on our robot both of these events happen simultaneously). The constraint in (1) is then adjusted to

$$\forall\, i, t: \ \mathbf{a}^t(\mathbf{X}^i) = \boldsymbol{\pi}_\theta(\mathbf{s}^{t_k}(\mathbf{X}^i)) \tag{3}$$

Where t_k is the closest communication timestep preceding t. In this case, the constraint couples the interaction between two different timesteps (t and t_k), but can still be incorporated into a trajectory optimization problem with efficient solution methods, as we will see in IV. In our experiments, we select t_k to be evenly-spaced 50 milliseconds apart throughout the trajectory.

C. Block-Alternating Stochastic Optimization

The resulting constrained optimization problem (1) is nonconvex and too large to solve directly. Following [16], we replace the hard equality constraint with a quadratic penalty with weight α:

$$R(\mathbf{s}, \mathbf{a}, \theta, \boldsymbol{\varepsilon}) = \frac{\alpha}{2} \left\| (\mathbf{a} + \mathbf{a}_\mathbf{s}\boldsymbol{\varepsilon}) - \boldsymbol{\pi}_\theta(\mathbf{s} + \boldsymbol{\varepsilon}) \right\|^2,$$

leading to the relaxed, unconstrained objective

$$\underset{\theta \ \mathbf{X}^1 \ ... \ \mathbf{X}^N}{\text{minimize}} \ \sum_i C_i(\mathbf{X}^i) + \sum_{i,t} R(\mathbf{s}^{t_k}(\mathbf{X}^i), \mathbf{a}^t(\mathbf{X}^i), \theta, \boldsymbol{\varepsilon}).$$

We then proceed to solve the problem in block-alternating optimization fashion, optimizing for one set of variables while holding others fixed. In particular, we independently optimize for each $\mathbf{X}^i$ (trajectory optimization) and for θ (neural network regression) and resample new noise instantiation $\boldsymbol{\varepsilon}$. As a result, we reduce a complex policy search problem in (1) to an alternating sequence of independent trajectory optimization and neural network regression problems, each of which are well-studied and allow the use of existing implementations. For all experiments, we used neural network policy with 2 hidden layers of 100 units each.

IV. Model-Based Trajectory Optimization

We wish to find trajectories in simulation that start with particular initial conditions and execute the task, while satisfying physical realism of the robot's motion. The existing approach we use is Contact-Invariant Optimization (CIO) [18], which is a direct trajectory optimization method based on an inverse dynamics model. Physical realism is achieved by satisfying equations of motion, non-penetration, and force complementarity conditions at every point in the trajectory [19]:

$$\begin{aligned} H(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) &= \boldsymbol{\tau} + F^\top(\mathbf{q}, \dot{\mathbf{q}})\mathbf{f} \\ \mathbf{d}(\mathbf{q}) &\geq \mathbf{0} \\ \mathbf{d}(\mathbf{q})^\top \mathbf{f} &= \mathbf{0} \\ \mathbf{f} &\in \mathbf{K}(\mathbf{q}), \end{aligned} \tag{4}$$

where $\mathbf{d}(\mathbf{q})$ is the distance of the contact to the ground and $\mathbf{K}$ is the contact friction cone.

The constraints in (4) are implemented as soft constraints, as in [18] and are included in $C(\mathbf{X})$. Initial conditions are also implemented as soft constraints in $C(\mathbf{X})$. Additionally we want to make sure the task is satisfied, such as leaning to a particular orientation or moving a hand to a particular location while minimizing effort. These task costs are the same for all our experiments and are described in section VII.

The trajectory optimization problem consists of finding the optimal trajectory parameters $\mathbf{X}$ that minimize the total cost C:

$$\mathbf{X}^* = \underset{\mathbf{X}}{\text{argmin}}\, C(\mathbf{X}),$$

which is solved using Gauss-Newton method via the following iterative steps (as in [16]):

$$\mathbf{X}^* = \mathbf{X}^* - C_{\mathbf{XX}}^{-1} C_\mathbf{X}.$$

In addition to the optimal trajectory, optimal feedback gains are necessary to incorporate sensory noise in (2). They are calculated as a byproduct of direct trajectory optimization

$$\mathbf{a}_\mathbf{s} = \mathbf{a}_\mathbf{X} C_{\mathbf{XX}}^{-1} \mathbf{s}_\mathbf{X}^\top (\mathbf{s}_\mathbf{X} C_{\mathbf{XX}}^{-1} \mathbf{s}_\mathbf{X}^\top + \frac{1}{\lambda} I)^{-1},$$

where λ is a parameter that controls how aggressive feedback gain corrections are. See [16] for more details and a derivation. Note that $\mathbf{s}_\mathbf{X}$, $\mathbf{a}_\mathbf{X}$ and $C_{\mathbf{XX}}^{-1}$ are already calculated as part of trajectory optimization. Thus, computing optimal feedback gains comes at very little additional cost.

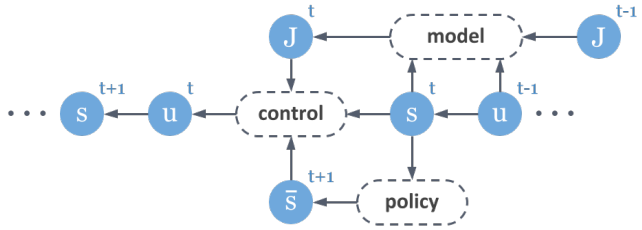

Fig. 3. An overview of our online model learning and control approach, consisting of policy evaluation trained offline in Sec. III, online dynamics model updates (Eq. 5) and online control (Eqs. 6,7)

V. Online Model Learning and Control

After finding an optimal control policy $\boldsymbol{\pi}_\theta$ in a simulated setting, we wish to apply this policy for control of a physical robot. Recall that $\boldsymbol{\pi}_\theta$ commands high-level control features $\mathbf{a}$ such as future joint positions and velocities. It does not directly control the joint torques of the robot, because small differences between simulated and physical model and even small differences between trials can quickly propagate and lead to failure such as robot falling down. For example, if we attach extra weight to the robot in the form of a battery or if ground friction changes slightly, this would change the torques that must be applied. But we wouldn't want to re-train the entire policy for such small changes. Instead, we propose a method to command these features based on online model learning which separates low-level model and trial details from the high-level policy learning.

Assume the physical robot we wish to control is governed by the following forward dynamics equation:

$$\mathbf{s}^{t+1} = \mathbf{g}(\mathbf{s}^t, \mathbf{u}^t),$$

where $\mathbf{s}$ is the available sensory information and $\mathbf{u}$ are the controls of the robot. These dynamics may differ from the simulated dynamics equations (4) due to differences between simulation and the physical world. The policy outputs the desired next state $\bar{\mathbf{s}}^{t+1}$ which can be used with short-horizon optimization to find the physical robot's control as

$$\mathbf{u}^t(\bar{\mathbf{s}}^{t+1}) = \underset{\mathbf{u}}{\text{minimize}}\ \frac{1}{2}\left\|\mathbf{g}(\mathbf{s}^t, \mathbf{u}) - \bar{\mathbf{s}}^{t+1}\right\|^2.$$

A. Learning Linear Dynamics

The class of dynamical systems we restrict ourselves to are linear dynamics

$$\mathbf{g}(\mathbf{s}^t, \mathbf{u}^t) = \mathbf{J}_\mathbf{s}\mathbf{s}^t + \mathbf{J}_\mathbf{u}\mathbf{u}^t,$$

where $[\mathbf{J}_\mathbf{s}\ \mathbf{J}_\mathbf{u}]$ are the system parameters to be learned. Linear dynamics have the advantage of computationally-efficient learning and optimal control calculation, unlike other models, such as neural networks or Gaussian processes. This is important if the controller is executed on-board low-cost robots, such as Darwin-OP, that lack large computational resources. The disadvantage is that linear dynamics are not expressive enough to be valid in all parts of the state space. However, if the linear model parameters are updated and re-learned online, they do act as locally-valid models, as we show in section VII, and are appropriate for short-horizon predictive control.

We update the model parameters based on the previous sensory state and control and the consequent sensory state $(\mathbf{s}^{t-1},\ \mathbf{u}^{t-1},\ \mathbf{s}^t)$ similar to [10], [28]. Because only one such data instance is not enough to define the dynamics, we assume the dynamics parameters change smoothly and keep the parameters close to their previous values. Denoting the concatenated parameter matrix $[\mathbf{J}_\mathbf{s}\ \mathbf{J}_\mathbf{u}]$ as $\mathbf{J}$,

$$\mathbf{J}^t = \underset{\mathbf{J}}{\text{minimize}}\ \frac{1}{2}||\mathbf{J}\,[\,\mathbf{s}\ \mathbf{u}\,]^{t-1} - \mathbf{s}^t||^2 + \frac{\alpha}{2}||\mathbf{J} - \mathbf{J}^{t-1}||^2,$$

Where α is a smoothness weight. The result is a linear system that lends itself to efficient rank one updates [1], [10]

$$\mathbf{J}^t = \mathbf{J}^{t-1} + \frac{\mathbf{s}^t - \mathbf{J}^{t-1}\,[\,\mathbf{s}\ \mathbf{u}\,]^{t-1}}{||\,[\,\mathbf{s}\ \mathbf{u}\,]^{t-1}\,||^2 + \alpha}\,[\,\mathbf{s}\ \mathbf{u}\,]^{t-1}. \tag{5}$$

With this dynamics model, the optimal control sent to the robot can be efficiently calculated as solution to a small linear system:

$$\mathbf{W}(\bar{\mathbf{s}}^{t+1} - \mathbf{J}_\mathbf{s}^t\mathbf{s}^t) = \mathbf{W}\mathbf{J}_\mathbf{u}^t\mathbf{u}^t. \tag{6}$$

$$\lambda\mathbf{u}^{t-1} = \lambda I\mathbf{u}^t \tag{7}$$

where λ is a regularization constant (typically 0.6) and $\mathbf{W}$ is a diagonal scaling matrix. This matrix ensures that joint positions and velocities have roughly the same unit magnitude, and use 1 for position and 0.05 for velocity scaling.

It is important to have some amount of variation in the controls and sensory states to be able to predict the effects of controls on the next state and ensure the linear system solved by (5) is not singular. Thus, we add a small amount of motor noise to $\mathbf{u}^t$ before sending it to the robot. The noise is zero-mean Gaussian with standard deviation 0.01 radians.

VI. System Details

A. Darwin Robot and Simulator Model

The Darwin-OP2 humanoid biped by Robotis Ltd. has 26 DOFs (6D root pose and 20 actuated joints). Each actuator is a MX-28 servo motor with position measurement of 12-bit resolution over 2π radians that is position controlled. An integrated inertial measurement unit in the torso provides 3-axis acceleration and angular velocity. The sensor data output and control input are processed through an x86 1.6 GHz dual core CPU on-board the robot.

While the Darwin-OP2 is a convenient, low-cost biped platform, it exhibits a number of limitations. The control inputs to the servo motors are the set point and P gain, K_P, for a PID controller, which is closed-source. Secondly, while each body part of the Darwin robot can be disassembled and its mass measured, the center of mass of each body part and thus the entire robot is harder to predict. Other potential sources of modelling error include backlash where the motor gear teeth are not in contact with each other when reversing. Estimating and modelling backlash is difficult [8].

The Darwin model we use in simulation includes all 26 degrees of freedom of the physical robot and is based on CAD created specifications to manufacture the robot.

B. Jacobian Initialization

Prior to executing the desired motion policies, we perform a "motor babbling" phase in order to find a good initialization for linear system parameters $\mathbf{J}$. Babbling is done by adding temporally smoothed Gaussian noise to zero pose set points over a period of 30 seconds. In our initial trials, we suspended the robot in the air and initialized the parameters using $\mathbf{J} = \mathbf{0}$. However, we find that initializing from:

$$\mathbf{J}^{(\mathbf{0})} = \begin{bmatrix} I & 0 & 0 \\ -I & I & I \end{bmatrix}$$

produces comparable results, but requires fewer iterations to converge because the Jacobian consistently takes a structure similar to this. Also, to better approximate the dynamics of the leg joints when in contact with the ground, we prefer to execute motor babbling with the robot standing upright. Rather than perform babbling before each policy execution, we save the resulting Jacobian after one session and use it as the initialization for subsequent policy executions.

During the babbling phase, the linear system we learn is exposed to changes in state of a certain magnitude, $||\mathbf{s}^t - \mathbf{s}^{t-1}|| \approx \epsilon$. To prevent the robot from destabilizing and damaging itself should the policy command a far-away desired state, we employ a "trust region" before solving for $\mathbf{u}^t$. We cap the values of $\bar{\mathbf{s}}^{t+1}$ so that no element of $|\mathbf{s}^t - \bar{\mathbf{s}}^{t+1}|$ exceeds the ϵ from the babbling phase.

VII. Experiments

We have tested our method on several tasks both in simulation and on the physical robot. We see complex motions emerging to accomplish the tasks due to high-level policy that greedy controllers do not exhibit.

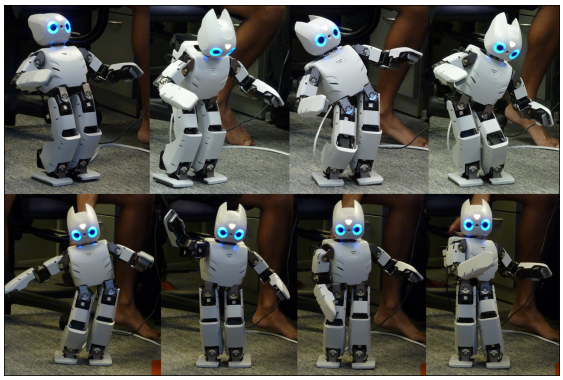

Fig. 4. Examples of leaning and hand reaching interactive policy execution on the physical robot.

A. Leaning Task

In this experiment, we trained a policy to interactively track the global orientation of the robot's torso, as estimated by the on-board IMU accelerometer. In addition to the policy inputs specified in section III, the policy takes a user-specified unit vector representing the desired up-direction of the torso. Examples of applying the leaning policy in simulation and in combination with the linear dynamics model on the physical robot are shown in Fig. 4. Fig. 5 shows how the physical robot is capable of tracking the desired torso orientation despite noise in the accelerometer readings.

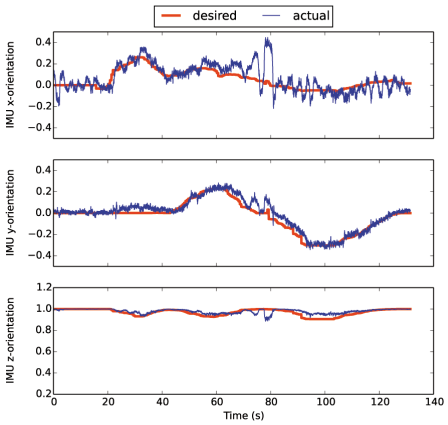

Fig. 5. IMU orientation readings (blue) compared with desired orientation commanded by the user (red) during a leaning task. Orientation is tracked successfully despite the significant amount of noise in the sensors.

B. Hand Reaching Task

In a second experiment, a policy was trained to interactively follow a desired hand position in workspace. In contrast to the first experiment, the controlled quantity is not directly observable. Implicitly the policy learns the forward kinematics and estimates the hand position from joint encoders without the need to provide motion capture information for the hand location. The policy learns the best representation automatically given the task and available sensory information. Examples of executing the learned reaching policy in simulation and on the physical system are shown in Fig. 4

C. Tilting Base Balancing

In the third experiment, the goal is to keep the robot standing upright while the ground the robot stands on is tilting or moving. Successful execution of this policy required the use of force sensors, which our physical robot does not have and which is why we only display results in simulation. We do incorporate sensory noise and delays when executing the policy in simulation.

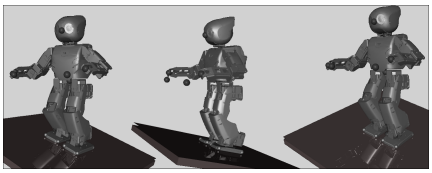

Fig. 6. The tilting base experiment in simulation.

D. Analysis of Linear System Learning

We have investigated the linear system parameters we learn and how these parameters adapt over time. In Fig. 7,

we show how the entries of the Jacobian change during the initialization phase and during policy execution. An example of the matrix we typically see during policy execution on the robot is shown in Fig. 7. The Jacobian consistently converges to a block-diagonal structure during the motor babbling phase, and still continues to adapt during policy execution. We also find that the structure of the learned Jacobian matrix is largely independent of the noise level and P gains used, and we have used values ranging from $\sigma = 0.1$, $K_P = 16$ to $\sigma = 0.5$, $K_P = 4$ with similar results.

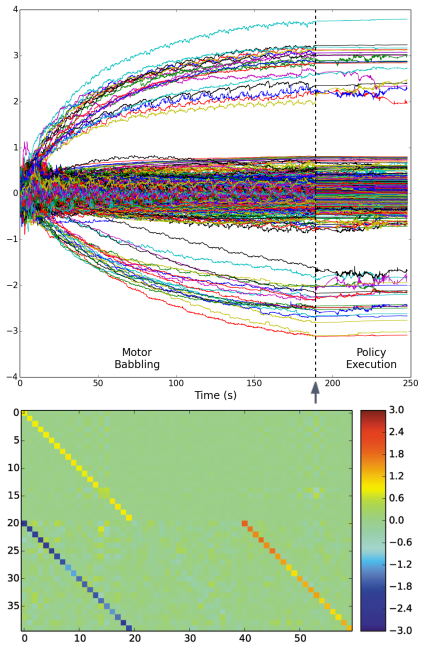

Fig. 7. The entries of the Jacobian matrix evolving over time, and a typical Jacobian matrix $\mathbf{J} = [\mathbf{J_s}\ \mathbf{J_u}]$ at the beginning of policy execution. The Jacobian is initialized from zero. Policy execution begins at $t \approx 190$.

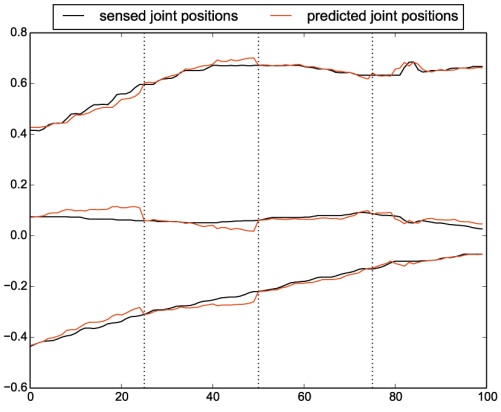

Fig. 8. Predictions of future sensor trajectories according to learned linear dynamics (V-A). The plot shows positions of the three arm joints during a reaching task. Dashed lines indicate timesteps from which the linear dynamics were extrapolated into the future.

We also observe that our dynamics-learning technique can adapt to different values of the P gain parameter. We can vary K_P between 8 and 16 with similar results, and the Jacobian parameters appear to re-converge within the first few seconds of execution. In figure 9, we show that varying K_P results in almost-identical results for an execution of the leaning task. Generally speaking, adjusting the P gain affects how stiff or compliant the robot is during policy execution, but the Jacobian learns to adapt and perform successfully in spite of this variation.

E. Comparison to direct policy control and fixed linear system control

To verify that the online dynamics learning is crucial to successfully execute the policies on the real robot, we evaluate two variations: training a policy that directly outputs low-level motor controls and not updating the linear dynamics parameters online.

We first try to train policies that directly output motor controls (the PID set points). On both the reaching and leaning tasks discussed in VII-A, VII-B, they succeed in some situations, but we notice that these policies require precise fine-tuning of the P gain parameter. If the value of K_P is too large, the policy destabilizes and we see rapid oscillation of the joints. Conversely, if K_P is too small, the robot is unable to keep itself upright.

Next, we try the leaning task with a fixed linear system. We use the Jacobian that results from the motor babbling phase, but do not update it during policy execution. The robot is unable to hold an upright pose, and falls over within 3 seconds (see Fig. 9).

The failures of these experiments suggest that online dynamics learning is necessary to account for discrepancies between the simulation and the physical robot, and to correct for errors that may occur during policy execution. Moreover, as Fig. 7 indicates, the dynamics that we are trying to learn can vary with time, and it is important to update the Jacobian using the most recent sensor measurements.

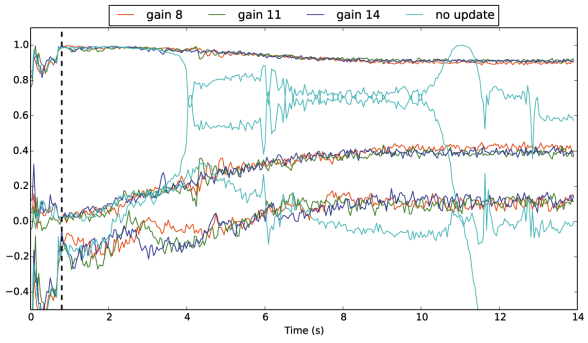

Fig. 9. The IMU accelerometer readings during an execution of the leaning policy. Varying the P gain (red, green, and blue) does not significantly affect the overall behavior. In contrast, the robot quickly falls over (cyan) when we do not update the Jacobian after policy execution begins. The dashed line indicates the end of the "babbling" phase.

VIII. Discussion

In this paper, we presented an automatic method for interactive control of physical humanoid robots and demonstrated

its effectiveness on a range of tasks. In the future, we wish to further increase the range of tasks to those that have been previously demonstrated in simulation, such as walking, getting up, and object manipulation.

Our method is able to operate using only the low-performance computer and sensors available on board the robot. This allows the possibility of the robot being deployed in unstructured outdoor environments in the future. However, the sensors we currently use are quite minimal and noisy, precluding the execution of more sophisticated policies, such as titling base balancing or walking. We are looking to augment on-board sensors with commercially available foot sensors and incorporate the robot's camera information into our policies.

While linear dynamics models have worked successfully as local models in the experiments we attempted, it would be interesting to explore other model classes, such as neural network dynamics models trained on-line.

IX. Acknowledgements

We thank Kendall Lowrey and Emo Todorov for inspiring technical discussions. This research was funded by Defense Advanced Research Projects Agency.

References

[1] C. G. Broyden, "A class of methods for solving nonlinear simultaneous equations," *Mathematics of computation*, pp. 577–593, 1965.

[2] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive dynamic walkers," *Science*, 2005.

[3] S. Coros, P. Beaudoin, and M. van de Panne, "Generalized biped walking control," *ACM Transctions on Graphics*, vol. 29, no. 4, p. Article 130, 2010.

[4] M. P. Deisenroth, D. Fox, and C. E. Rasmussen, "Gaussian processes for data-efficient learning in robotics and control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 2, pp. 408–423, 2015. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/TPAMI.2013.218

[5] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. Hoepflinger, and R. Siegwart, "Control of dynamic gaits for a quadrupedal robot," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 3287–3292.

[6] H. Geyer and H. Herr, "A muscle-reflex model that encodes principles of legged mechanics produces human walking dynamics and muscle activities," *Neural Systems and Rehabilitation Engineering, IEEE Transactions on*, vol. 18, no. 3, pp. 263–273, 2010.

[7] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[8] G. Hovland, S. Hanssen, E. Gallestey, S. Moberg, T. Brogardh, S. Gunnarsson, and M. Isaksson, "Nonlinear identification of backlash in robot transmissions," in *Proceedings of the 33rd ISR (International Symposium on Robotics)*, 2002.

[9] D. Huh and E. Todorov, "Real-time motor control using recurrent neural networks," in *Adaptive Dynamic Programming and Reinforcement Learning, 2009. ADPRL '09. IEEE Symposium on*, March 2009, pp. 42–49.

[10] M. Jgersand, O. Fuentes, and R. Nelson, "Experimental evaluation of uncalibrated visual servoing for precision manipulation," in *, 1997 IEEE International Conference on Robotics and Automation, 1997. Proceedings*, vol. 4. IEEE, Apr. 1997.

[11] T. Koolen, T. De Boer, J. Rebula, A. Goswami, and J. Pratt, "Capturability-based analysis and control of legged locomotion. part 1: Theory and application to three simple gait models," *International Journal Robotics Research*, 2012.

[12] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM transactions on graphics (TOG)*, vol. 21, no. 3, pp. 473–482, 2002.

[13] I. Lenz, R. Knepper, and A. Saxena, "Deepmpc: Learning deep latent features for model predictive control," in *RSS*, 2015.

[14] S. Levine and V. Koltun, "Learning complex neural network policies with trajectory optimization," in *ICML '14: Proceedings of the 31st International Conference on Machine Learning*, 2014.

[15] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search," in *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, 2015, pp. 156–163. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2015.7138994

[16] I. Mordatch, K. Lowrey, G. Andrew, Z. Popovic, and E. Todorov, "Interactive control of diverse complex characters with neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[17] I. Mordatch, M. de Lasa, and A. Hertzmann, "Robust Physics-Based Locomotion Using Low-Dimensional Planning," *ACM Transactions on Graphics*, vol. 29, no. 3, 2010.

[18] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, pp. 43:1–43:8, July 2012. [Online]. Available: http://doi.acm.org/10.1145/2185520.2185539

[19] M. Posa and R. Tedrake, "Direct trajectory optimization of rigid body dynamical systems through contact," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 527–542.

[20] J. Pratt, J. Carff, S. Drakunov, and A. Goswami, "Capture point: A step toward humanoid push recovery," in *Humanoid Robots, 2006 6th IEEE-RAS International Conference on*, Dec 2006, pp. 200–207.

[21] M. H. Raibert, *Legged Robots That Balance*. Cambridge, MA, USA: Massachusetts Institute of Technology, 1986.

[22] J. R. Rebula, P. D. Neuhaus, B. V. Bonnlander, M. J. Johnson, and J. E. Pratt, "A controller for the littledog quadruped walking on rough terrain," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 1467–1473.

[23] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[24] R. Tedrake, M. F. Fallon, S. Karumanchi, S. Kuindersma, M. E. Antone, T. Schneider, T. M. Howard, M. R. Walter, H. Dai, R. Deits, M. Fleder, D. Fourie, R. Hammoud, S. Hemachandra, P. Ilardi, C. Pérez-D'Arpino, S. Pillai, A. Valenzuela, C. Cantu, C. Dolan, I. Evans, S. Jorgensen, J. Kristeller, J. A. Shah, K. Iagnemma, and S. J. Teller, "A summary of team mit's approach to the virtual robotics challenge," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 2014, p. 2087. [Online]. Available: http://dx.doi.org/10.1109/ICRA.2014.6907140

[25] M. Vukobratovic and B. Borovac, "Zero-moment point thirty-five years of its life," *International Journal of Human Robotics*, 2004.

[26] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers for uncertain inputs and environments," *ACM Trans. Graph.*, vol. 29, no. 4, pp. 73:1–73:8, July 2010. [Online]. Available: http://doi.acm.org/10.1145/1778765.1778810

[27] J. M. Wang, S. R. Hamner, S. L. Delp, V. Koltun, and M. Specifically, "Optimizing locomotion controllers using biologically-based actuators and objectives," *ACM Trans. Graph*, 2012.

[28] M. C. Yip and D. B. Camarillo, "Model-Less Feedback Control of Continuum Manipulators in Constrained Environments," *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 880–889, Aug. 2014, 00005.