# Ensemble-CIO: Full-Body Dynamic Motion Planning that Transfers to Physical Humanoids

Igor Mordatch, Kendall Lowrey, Emanuel Todorov
Department of Computer Science & Engineering, University of Washington

***Abstract*— A lot of progress has recently been made in dynamic motion planning for humanoid robots. However this work has remained limited to simulation. Here we show that executing the resulting trajectories on a Darwin-OP robot, even with local feedback derived from the optimizer, does not result in stable movements. We then develop a new trajectory optimization method, adapting our earlier CIO algorithm to plan through ensembles of perturbed models. This makes the plan robust to model uncertainty, and leads to successful execution on the robot. We obtain 90% success rate in (dynamic) forward walking, 95% success rate in sideways walking, 80% success rate in turning, and a similarly high success rate in getting up from the floor (the robot broke before we quantified the latter). Even though the planning is still done offline, this work represents a significant step towards automating the tedious scripting of complex movements.**

## I. Introduction

The mechanical and actuation capabilities of modern robots are quickly surpassing our ability to control them in ways that do justice to their hardware. Especially challenging are under-actuated systems such as humanoids where some form of dynamic planning appears to be unavoidable. We and others have recently developed optimization methods that can generate complex and dynamically-consistent movements in simulation [4]–[7], [9]. However this work has not yet transfered to physical robots. Instead the controllers deployed on physical systems tend to rely on task-specific simplifications and reduced models. Examples are ZMP and a variety of inverted pendulum models – which have been very successful in walking [1]–[3], [8], [11]. But humanoids are mechanically capable of much more than walking, and yet walking appears to be one of the very few complex task where adequate model reductions are available. Another pragmatic approach is for a team of engineers to spend weeks designing and fine-tuning a specialized controller (usually a reference trajectory with PD gains around it), but ideally this effort will be automated.

Our goal is to transfer full-body dynamic plans from simulation to the real world. The obvious place to start is to build the most accurate model that we can with reasonable effort, optimize a movement trajectory, play it on the robot and see what happens. Not surprisingly, the robot falls. We are working with Darwin-OP, and indeed using an inexpensive and easy-to-handle robot may be key to making progress in this direction. The next obvious step is to use the locally-optimal time-varying linear feedback control law which most trajectory optimizers generate, and apply it to the physical system with the hope that it will suppress deviations from the planned motion. This improves performance somewhat as we will see later, but in a nutshell it still doesn't work. Now this could be reason for desperation. If we cannot stabilize a single dynamic trajectory on the physical system, even though it works well in simulation, what hope is there for model-based control? One answer could be that we need more accurate models and state estimators – and if we had them performance would certainly improve. Indeed this field is approaching a point where we can make our simulations accomplish most tasks of interest, and so the next step may be not only about control, but about system identification and state estimation coupled with control. Yet developing more accurate modeling techniques, especially for complex robots where multiple poorly-understood factors can mask each other and complicate the interpretation of sensor measurements, will take time.

The main contribution of this paper is a new method for transferring model-based controllers to physical systems, that can tolerate the errors in our existing models. The idea is conceptually simple: instead of optimizing through one (nominal) model, we optimize through an ensemble of models obtained by perturbing the parameters of the nominal model. The ensemble cannot be too large for computational reasons (presently around 10), and the space of model parameters is too high-dimensional to be explored by such a small number of samples. Thus the samples in model space have to be placed with consideration of which aspects of the nominal model are likely to be inaccurate, and also which types of errors are likely to affect control performance. Our new method is based on contact-invariant optimization (CIO) [5] which we adapt to the ensemble case. We show that the resulting trajectories can indeed be executed on the physical system with high success rate, for diverse tasks including forward and sideways walking, turning, and getting up from the floor. Using the local feedback generated by the trajectory optimizer further improves performance.

We chose CIO as the basis of the ensemble approach developed here because of all available dynamic planning methods it appears to be the most capable of discovering complex long-horizon movements automatically. CIO is particularly good at planning through multiple contacts, without requiring the user to specify contact events in advance. Nevertheless it is possible that other trajectory-planning algorithms as well as more generic policy-gradient algorithms can be used successfully in the ensemble setting. Indeed some algorithms have previously been adapted to optimization under uncertainty [12] but only in simulation.

Our ensemble-CIO method does not work in realtime. Presently it takes around 5 minutes of CPU time for walking, and up to 20 minutes of CPU time for getting-up movements (which involve more contacts), on a powerful workstation. So in this sense our work is related to the above-mentioned team of engineers designing a reference trajectory with suitable local feedback. The difference is that here this process is automated. We still have to design suitable cost functions, but that takes much less effort and is quite intuitive as described below. There are opportunities for speeding-up the new method which we have to yet explored, and it remains to be seen how well it will perform with warmstarts in model-predictive control mode. But even in its present form, Ensemble-CIO already provides an automated way to plan complex dynamic movements that can be successfully executed on humanoid robots.

## II. Preliminaries

### A. Model and Actuation Dynamics

We focus on second-order servo-controlled systems – which is what most existing humanoids are, even if they have force feedback mimicking compliance to some extent. Let $\mathbf{q}$ denote the vector of positions including the root position and orientation (the latter is represented as a unit quaternion). Physically consistent motions must satisfy the traditional equations of motion for an articulated rigid body system:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) = J(\mathbf{q})^T \mathbf{f} + \boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{r}) \tag{1}$$

Here $M$ is the joint-space inertia matrix, $C$ is the vector of Coriolis, centrifugal, and gravitational terms, $J$ is the contact Jacobian, $\mathbf{f}$ is the contact force vector, and $\boldsymbol{\tau}$ is the force applied by the controller. We use the Mujoco physics engine [10] to calculate these quantities for any given state. MuJoCo has a built-in model of PD control, so the entire dynamical system together with the low-level actuation can be handled in a uniform way.

The actuation model is based on PD control, with reference positions/setpoints $\mathbf{r}$ that need to be specified by the high-level controller we seek to design. The control force is

$$\boldsymbol{\tau}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{r}) = B\left(k_s(\mathbf{q}_{act} - \mathbf{r}) + k_d \dot{\mathbf{q}}_{act}\right) \tag{2}$$

where $k_s$ and $k_d$ are stiffness and damping coefficients and $\mathbf{q}_{act}$ are the degrees of freedom corresponding to the actuators. The constant matrix $B = [0; I]$ inserts the actuation torques in the overall torque vector.

### B. CIO trajectory optimization

The motions we are interested in involve complex contact interaction between the robot and the environment. We include the contact forces as explicit optimization variables so as to help the optimizer. This is the key idea behind the CIO method [6], which we know briefly recall.

We predefine a number of potential points $\mathbf{p}$ on the robot which will be making contact with the environment, each of which can result in a force being applied by the environment on the robot. Contact forces and states must be consistent with each other. Contact forces must lie within the friction cone $\mathbf{f} \in C(\mathbf{q})$. If a contact is not active, the contact force should be zero. If a contact is active, the contact point should be touching the environment and not be sliding. The former can be written as a cone constraint, while the latter can be expressed as the following soft complementarity constraint for each contact $i$:

$$\lambda_i(\|\mathbf{p}_i - \text{proj}(\mathbf{p}_i)\|^2 + \|\dot{\mathbf{p}}_i\|^2) = 0 \tag{3}$$

where proj() is the projection of a point onto the environment and $\lambda$ is a smooth contact indicator variable calculated from contact normal force $\mathbf{f}^N$ as

$$\lambda = \frac{1}{2}\tanh(k_c \mathbf{f}^N) + \frac{1}{2} \tag{4}$$

For the purposes of CIO optimization, any state and its temporal derivatives satisfying the above constraints is considered dynamically consistent. This may not be fully consistent with the forward dynamics. However in the present work we post-process the CIO trajectories by applying a single Newton step using MuJoCo's forward dynamics model.

The unique aspect of the CIO method is the above soft cost. We add this cost to any other costs needed to encode a specific task (see below), resulting in the total cost

$$\frac{1}{T}\sum_t c^t(\mathbf{q}^t, \mathbf{f}^t) \tag{5}$$

This cost is optimized with respect to the trajectory of system configurations and contact forces $[\mathbf{q}\ \mathbf{f}]^{1:T}$, using a direct collocation approach described in more detail later.

### C. Model Uncertainty

Even after system identification, we have uncertainty in a number of model parameters including individual limb masses or their centers of mass. In addition we rely on approximations such as discrete number of contact points (instead of a contact surface). Therefore trajectories that are dynamically-consistent with respect to the model may not be feasible on the robot. To address this issue, we take model uncertainty into account when performing trajectory optimization, as described in the next section. The sources of uncertainty we consider are:

**Contact Point Location:** We use predefined spherical contact points instead of the meshes of the robot. Thus one source of uncertainty is the location of these points in local body coordinates. Considering this source of uncertainty not only increases robustness to poor sphere placement – which is a type of modelling error – but also ensures that contacts can be made in unexpected ways and the trajectory can still succeed. For example, instead of a robot's foot needing to contact the ground specifically at one point on the heel, it will plan for contacts around the heel and even tolerate a flat foot. Contact points are sampled from a Gaussian distribution centered around the nominal contact point location with a variance of 5mm.

**Limb Mass:** Although we made an effort during system identification to measure the mass of the robot's body parts, these measurements may not be perfect. By optimizing under different mass distributions, the resulting trajectories become more conservative. We vary masses by 20% of their nominal estimated values.

**Limb Center of Mass:** We also have uncertainty in the location of the center of mass for each body part. Again, varying these parameters can yield trajectories that are more cautious and conservative. Center of mass locations are sampled from a Gaussian distribution centered around the nominal location with a variance of 20% of the limb's bounding volume.

We denote all model parameters by a vector $\theta$. These model parameters affect the terms $M$, $C$, $J$ in equation (1).

In this work we assume that inaccuracies in motor-related parameters will result in short-horizon errors, which will be overcome at execution time either by built-in PD controllers, or by state-feedback control as described in section III-C. Furthermore the Darwin-OP robot uses identical motors at all joints, and we carefully identified one isolated motor. Thus we do not explore uncertainty in the motor parameters here.

## III. The new method: Ensemble-CIO

We wish to find a robust behavior which performs well under a variety of different model parameters $\theta$. One approach would be to optimize a collection of $N$ dynamically-consistent state and contact force trajectories, one for each model instance. However if there is no dependency between the trajectories corresponding to the different instantiations of $\theta$, we will end up with plans that are specific to the individual perturbed models and have no reason to be robust on the physical system. Thus the key is to couple the optimizations somehow. Two obvious choices are coupling at the trajectory level or at the open-loop control level. However neither of these is ideal: there is no reason why a single trajectory will be physically-consistent for all the model instances, or that one sequence of open-loop controls will make all models perform the task.

What we really need is a single feedback control policy, which may give rise to different state and control trajectories for each model instance, but will nevertheless accomplish the task for all instances. For a servo-controlled robot, a natural way to parameterize this control policy is a reference trajectory and a set of PD gains – which in this case are fixed, to values that are not too large so that the policy can adapt to the model instance at runtime. We also consider a more elaborate form of feedback later, where the deviations from the reference trajectory are used at runtime to adjust the reference positions for the remainder of the movement.

But first we describe how we find the reference trajectory. This is done by optimizing a cost in the form

$$\min_{[\mathbf{q}\ \mathbf{f}]_{1:N}^{1:T}} \frac{1}{N}\sum_{i,t} c^t(\mathbf{q}_i^t, \mathbf{f}_i^t) + \frac{1}{2}\left\|\mathbf{r}_i^t - \bar{\mathbf{r}}^t\right\|^2 \tag{6}$$

where $\bar{\mathbf{r}} = \frac{1}{N}\sum_i \mathbf{r}_i$ is the average PD setpoint trajectory. The quantities $\mathbf{r}_i^t$ can be recovered from equations (2, 1) as:

$$\mathbf{r}(\mathbf{q},\mathbf{f}) = \mathbf{q}_{act} + \frac{k_d}{k_s}\dot{\mathbf{q}}_{act} - \frac{1}{k_s}(M\ddot{\mathbf{q}} + C - J^T\mathbf{f}) \tag{7}$$

If $\mathbf{q}_i$ and $\mathbf{f}_i$ are interpreted as samples from their respective random variables, one way of interpreting (6) is to reduce the expected cost under random model parameters, while reducing the variance of the resulting PD setpoint trajectory.

### A. Objective Function

Our objective is a weighted combination of terms, each depending only on state variables and its temporal derivatives at a particular point in time.

$$c^t(\mathbf{q},\mathbf{f}) = \sum_k w_k c_k^t(\mathbf{q},\mathbf{f}) \tag{8}$$

The terms we include are:

**Equations of Motion and CIO Constraints:** We require state and contact forces to satisfy dynamic consistency constraints as described in section II-A. Equations (1) and (3) are enforced as soft constraints with quadratic costs.

**Kinematic Constraints:** We require that states $\mathbf{q}$ be kinematically consistent. Contact points should not be penetrating the environment, body parts should not be self-intersecting and joint angles must be within model limits. These bound constraints are enforced softly as half-quadratic costs.

**Regularization:** To get more natural trajectories and to improve problem conditioning, we prefer state and contact force trajectories to be smooth. Additionally, we prefer PD setpoint to not be too far from the state, which implicitly minimizes velocity and control effort.

$$c_{\text{reg}} = \|\ddot{\mathbf{q}}\|^2 + \|\ddot{\mathbf{f}}\|^2 + \|\mathbf{q}_{\text{act}} - \mathbf{r}\|^2 \tag{9}$$

**User Task:** While the above terms put us in a space of plausible trajectories, the trajectory must also satisfy tasks specificed by the user. These objectives can be open-ended, and are described in section IV-A.

| Cost Term | eom | cio | kin | reg | task |
|---|---|---|---|---|---|
| Weight | $10^1$ | $10^1$ | $10^0$ | $10^{-5}$ | $10^0$ |

TABLE I
THE WEIGHTS FOR THE TERMS CONTRIBUTING TO THE OBJECTIVE FUNCTION (8)

### B. Optimization Algorithm

To solve the above optimization problem we use the Levenberg-Marquardt method. It requires inversion of the (approximate) Hessian of the total cost, but in our case each $c^t$ only depends on $\mathbf{q}^t$, $\mathbf{f}^t$ and its time derivatives. Thus, the Hessian is block-diagonal and can be inverted efficiently using Cholesky factorization.

We use $N = 10$ model samples, which is fairly small and allows us to perform optimization in (6) without additional approximations such as stochastic or distributed optimization methods. We found the method to converge between 100 to 500 iterations, depending on complexity of the problem.

### C. State-Feedback Control

The simplest way to apply our optimized trajectories to the robot is to set $\bar{\mathbf{r}}$ as the reference trajectory and let the built-in PD controller deal with any errors online. But as discussed in section IV, there are various modelling and system deficiencies that must be overcome for successful trajectory playback. Ensembles of certain parameters can overcome modelling errors, but feedback is necessary to overcome execution-related issues such as limitations of the servo motors. For this reason we implemented an additional feedback loop, mapping deviations from planned positions and velocities to adjustments in reference position.

This state-feedback controller is implemented as follows. Let $\mathbf{x} = [\mathbf{q}\ \dot{\mathbf{q}}]$ be the state of the system in forward dynamics simulation. We first execute our PD setpoint trajectory $\bar{\mathbf{r}}$ in forward simulation under the average model parameters to get a rollout $\bar{\mathbf{x}}^{1:T}$. We linearize about this rollout and perform an LQG backward pass to compute the optimal time-varying feedback gains $A^t$. Then the state-feedback controller modifies the reference trajectory online as:

$$\mathbf{r}^t_{\text{FB}}(\mathbf{x}) = \bar{\mathbf{r}}^t + A^t(\mathbf{x} - \bar{\mathbf{x}}^t) \tag{10}$$

Here $\mathbf{x}$ contains the joint angles and joint velocities measured by the robot's potentiometers. We are not yet using feedback based on the root position and velocity. Adding such feedback is likely to increase performance further, and we will explore it in the near future.

## IV. Darwin Robot

The Darwin-OP humanoid from Robotis Ltd. is a 26 degree-of-freedom (DOF) robot, including 3 translational and 3 rotational DOFs at the root, and 20 actuated hinge joints. Each actuator is a MX-28 servo motor with position measurement of 12-bit resolution over $2\pi$ radians that is position controlled. An integrated inertial measurement unit in the robot's torso provides 3-axis acceleration and 3-axis angular velocity. Each foot also contains four force resistive sensors to measure contact forces. The sensor data output and control input are processed by an onboard Atom CPU.

While the Darwin-OP is a convenient, low-cost biped platform, there are a number of limitations that must be overcome in one way or another. First, the control inputs to the servo motors are the set point for a PD controller. This means that the actual forces are generated by the built-in controller which we have no access to (the motor's firmware 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. Although our dynamical model was initially derived from the CAD models of the robot which include inertia and center of mass measurements provided by Robotis, it is unknown if their model included all mass contributions such as the electronics boards, wiring, and hollow spaces in addition to the obvious metal frame and plastic body. While these are both sources of modelling error, we have tried our best to perform accurate system identification of these properties offline before trajectory optimization.

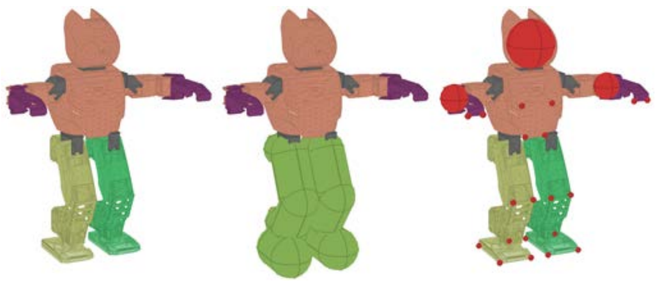


Fig. 1. Simulated Darwin model (a), self-collision proxy (b), and environment contact points (c)

However, even with a perfect model (which ours is not), there are real world phenomena that our simulator does not take into account or cannot be identified. These include mechanical backlash in the gears, as well as programmed deadband in the control logic itself, whereby the servo will deliberately allow some small error so as to reduce gear wearing out. Together, these phenomena mean that idealized position controlled motors in a simulation can act differently from those in the real world unless the model is heavily augmented or specifically designed to compensate for these sources of error.

External factors also contribute to trajectory playback failure, namely contacts with the external environment. While they can be predicted and planned through, hard contacts have discontinuous nature. This means that if a contact does not occur at the expected time, there is no assurance as to the rest of trajectory completing satisfactorily. Trajectories must then be planned in such a way to make up for missed or mistimed contacts, while still positioning the robot in a way that avoids falling. Said another way, a biped robot needs to be deliberate in its planning to stay balanced. Indeed this is the primary motivation behing the ensemble method developed here.

### A. Simulation Model

Our Darwin model includes all 26 degrees of freedom of the physical robot, denoted by $\mathbf{q} \in \mathbb{R}^{26}$. The model is based on CAD created specifications to manufacture Darwin. We have identified each actuator to have dry friction coefficient of 0.16 and damping of 1.13. The total mass of the robot is 2.835 kilograms. We assume contact between the robot and the ground to have friction coefficient of 0.75. For contact and actuator parameters, we use $k_c = 10$, $k_s = 36$ and $k_d = -12$ for all our simulations.

The Darwin robot geometry is a composition of complex, non-convex shapes which can make self-collision resolution a costly process and cause poor conditioning when used in trajectory optimization. To avoid these issues, we approximate the Darwin body with bounding capsules shown in Figure 1b for the purposes of self-collision detection and resolution.

Our tasks (especially the robot lying on the ground or getting up) involve complex interaction between Darwin and the environment. Feet, palms, knees, elbows, torso and head may be contacting the ground at any point in time. We predefine 23 potential contact points shown in Figure 1c, each of which can result in a 3-dimensional linear force being applied by the environment on the robot. We denote the total force vector to be $\mathbf{f} \in \mathbb{R}^{69}$.

The tasks we consider in this paper are centered around locomotion and getting up, and require the robot standing and facing a particular orientation at the final time. In other words we plan and execute a couple of steps of walking as opposed to a limit cycle with infinite duration, and so our trajectories include not only walking but also starting and stopping. The practical reason we do this is because it makes experiments easier to perform, but it is interesting that our method can indeed solve these more complex planning problems. The goal state is encoded as a quadratic cost $c^T_{\text{task}}$ on the final torso position and orientation, as well as the final position and orientation of the feet.

## V. Experiments

### A. Trajectory Preparation

After an optimal trajectory is calculated, the first step is to test it in simulation. We use the MuJoCo physics engine to play back the calculated controls in forward dynamics. This serves two purposes. First, the we can quickly examine the optimized trajectory and its intuitive potential of real world success. If the trajectory causes the robot to fall in the simulation, it is very likely that the physical robot will also fall. The second purpose is that the forward dynamics simulation is used to find the reference trajectory used with state feedback, as described earlier.

### B. Trajectory Execution

To save computation time, the optimization and forward dynamics happen at a slower time-step than the robot's control loop. In order to control the Darwin with a trajectory planned at a larger timestep, we linearly interpolate between the optimization's timesteps at runtime. This is important for feedback especially. At the start of every trajectory playback, the Darwin loads the trajectory data from file into memory. Then, it assumes the initial position specified in the trajectory and waits for a user command to begin execution.

### C. Experiment Design

Rather than showing anectodal movies illustrating the performance of the new algorithm, we decided to perform a systematic test. The experiment had 4 conditions with a 2-by-2 design. The conditions varied but the type of optimization used to generate the reference trajectory (nominal model only vs. ensemble method) and by the type of feedback (PD only vs. state feedback plus PD). We pertormed 20 consequtive trials in each condition, for a total of 80 trials per task. There were 3 tasks: walking forward, walking sideways, and turing 90 def – resulting in 240 trials in the entire experiment. This of course is in addition to many informal tests we have done, including getting-up which we were not able to quantify because the control board of the robot died just before the submission deadline. Here we focus on the results from the systematic experiment.

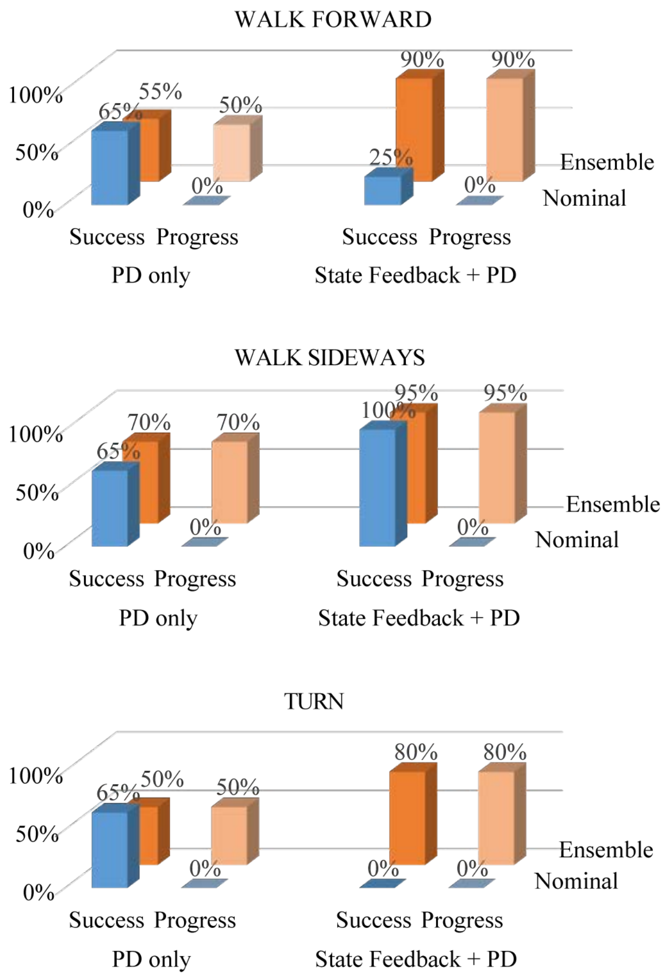


Fig. 2. Each subplot shows the results for one task: walking forward, walking sideways, and turning. We repeated each task and each condition for 20 trials. There were 4 conditions per task, which differed by the type of feedback (PD only vs. State Feedback + PD) and the type of optimization (Ensemble vs Nominal). In each condition we show the percent successful trials (defined as the robot not falling), and the percent trials on which the robot made progress (as opposed to just stepping in place).

## VI. Results

The quantitative results are presented in Figure 2 as percentages of the 20 trials in each condition and task. The outcomes were determined by the experimenter watching the robot. This was rather straightforward. We used two criteria: Success and Progress (although these names may not be ideal). Success means simply that the robot did not fall, or rather, we did not have rescue it by pulling on the safety string shown in later figures. Progress means that it actually made progress in the task – as opposed to shuffling in place or otherwise getting stuck without falling. Thus every trial scored as Progress is also a Success trial, but not vice versa.

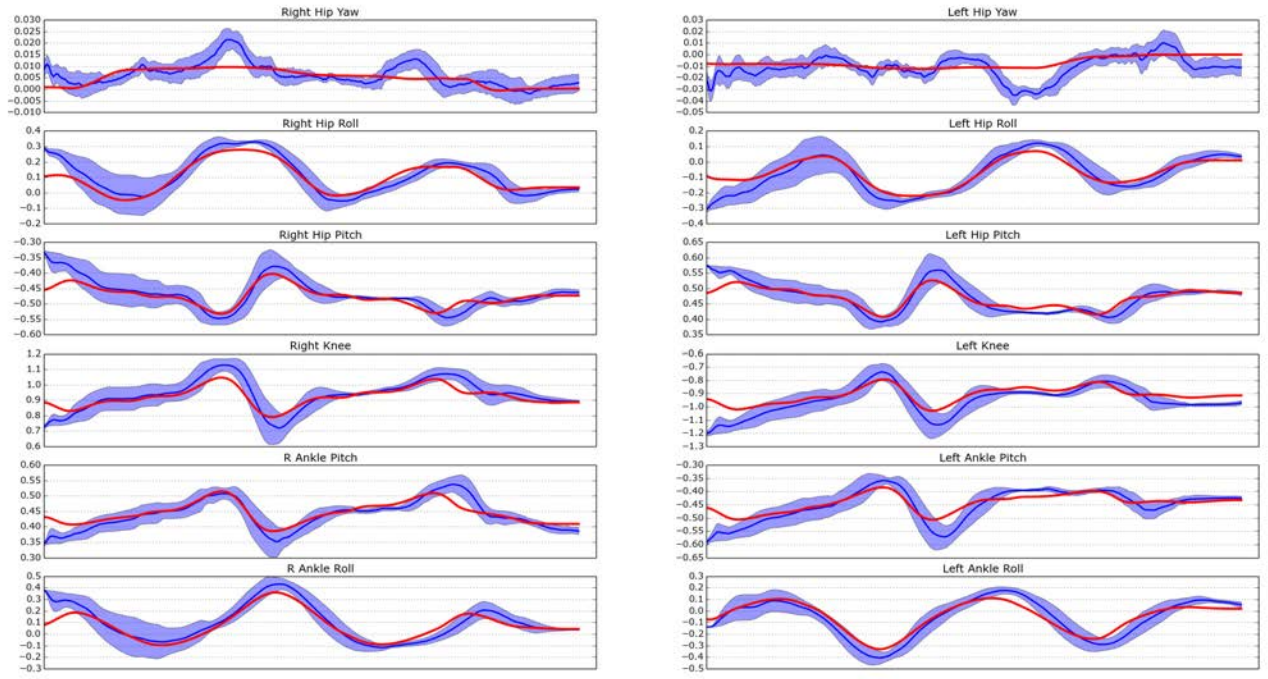


Fig. 3. Mean and variance (in blue) compared to the reference control signal (in red) for side stepping with feedback on an ensembled trajectory

The results are quite clear. The Ensemble-CIO method, with state feedback (adjusting the reference positions at runtime) has the best performance. We should point out that it exceeded the expectations we had at the beginning of this project. There are several more subtle findings illustrated in Figure 2. Trajectories optimized in the traditional way, using only the nominal model, are often stabilized by PD control alone and in fact state feedback hurts in that case. However the robot tends to get stuck and does not make progress. Ensemble-optimized trajectories on the other hand benefit from more intelligent feedback.

One might ask how well does the robot actually follow the specified trajectories. Figure 3 compares the reference and actual joint trajectories in side walking for the Ensemble-optimized trajectories with state feedback. Also shows in the figure is the variance of the physical trajectories. The tracking is quite good, but note that these data are somewhat meaningless because they do not include the root position and orientation (because we did not record it). The problem with an under-actuated system such as a humanoid is that it can fall and still continue to waive its arms and legs in the air following the initial plan, and creating the impressing that it performs well on plots such as Figure 3. We will soon add root tracking as well as feedback and will be able to address this issue better.

The performance of our controller is best appreciated by watching the accompanying movie. Figure 5 shows frame sequences from successful execution of each task. Figure 4. shows one example of failure without safety harness.

We see the qualitative differences between walking motions optimized with and without the ensemble method at the top of Figure 5. In particular, note that non-ensemble method lands on heel and transfers weight from heel to toe during double support phase. This is a natural, but precarious strategy. In contrast, the ensemble method lands with the foot flat on the ground. The non-ensemble method also swings the foot really close to the ground surface, whereas the ensemble method raises to foot more during swing phase. Also, the non-ensemble method takes a smaller number of large steps, whereas the ensemble method takes a larger number of smaller steps. In general, the movement strategies found by ensemble method are more conservative.

## VII. Additional Observations

The results demonstrate that while trajectories optimized for a single nominal model might be stable enough to not fall, they do not make progress appropriate for the given task. This means that for the walking, turning, and side-step tasks on a flat ground surface, the contacts with the ground did not produce the appropriate forces to propel the robot in the correct direction. In effect, the robot was slipping and sliding in place. This suggests that either the expected friction of the ground surface was wrong, or the trajectory relied excessively on frictional contact. Regardless, the non-ensembled trajectories either need more accurate models to plan through, or can only be used for simpler tasks.

Trajectories optimized with Ensemble-CIO had a much higher chance of completing the task, and demonstrated more cautious behavior: the feet were lifted off the ground higher to clear ensembled contact points, and more stable trajectories were chosen overall.

Feedback had some interesting effects. When using feedback, the robot would frequently experience some form of

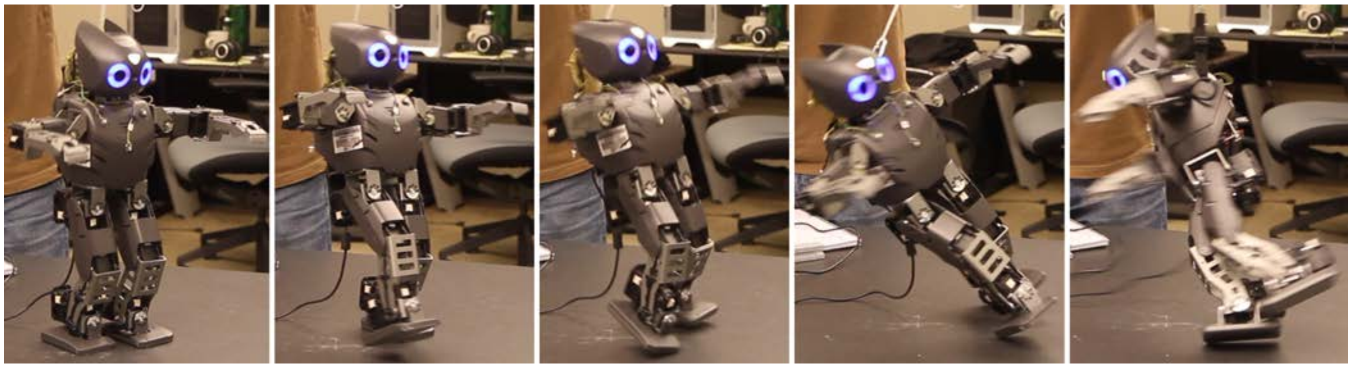


Fig. 4. Example of a failed trial.

overshoot when correcting for errors. This would obviously be problematic for trying to remain stable, and shows that feedback with non-ensembled trajectories in fact prevented it from successfully completing. The reason, we suspect, is that as both the trajectory and feedback were calculated from an incorrect model, errors built up more quickly, and could not be correctly compensated for.

Feedback with the ensembled trajectories did show improvement, however. Trajectories would frequently reach 'tipping points' where a successful transfer of mass would need to take place to continue the trajectory – otherwise, the robot would fall. Feedback would help in these instances to maintain the correct velocity within the joints to move correctly past the tipping point.

## VIII. Future Work

While this work demonstrates the ability of our ensemble method to make optimized dynamic trajectories feasible for use on robotic hardware, there are more avenues to explore. First is the types of ensembling and how they pertain to different actions. Robots that rely on contact with their environment might need different parameters to be ensembled than robots that do not.

Secondly, the feedback policy's range of use seems to be narrow in this experiment. Strong perturbations from the trajectory cannot be recovered. How strong remains to be seen and measured. Furthermore, we wish to explore how increasing the state space of the feedback can improve the feedback policy or increase robustness. Instead of just joint positions and velocities, we could include the robot's root orientation and position, and associated velocities. We hypothesize that this would increase the degree of success of a given action – walking forward a specific distance would be more accurate to that specific distance as we would have a control gain on the error in the distance. Some sensors we consider using are the Darwin's internal IMU or foot force sensors, or an external Phasespace position and orientation tracking system.

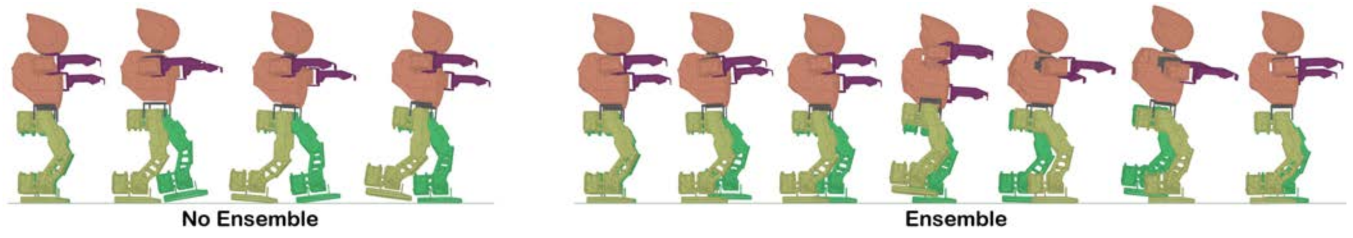
Alternatively, the simple feedback policy could be replaced with a neural network or another type of controller optimized via a policy gradient method. This could result in very different policies for different models, but all derived from the same parameters. Neural network policies (which we are currently developing in another project) could be used to increase robustness by expanding the stability region. This means, of course, measuring the window of robustness. This could be done in simulation, or by injecting noise into the sensor measurements or control outputs of the robot and again measuring the success of the action.

## IX. Acknowledgements

This work was funded by US National Science Foundation. Thanks to Yifan Hou for helpful discussions and assistance with the experiments.

## References

[1] K. Byl and R. Tedrake, "Metastable walking machines," *International Journal Robotics Research*, 2009.
[2] S. Collins, A. Ruina, R. Tedrake, and M. Wisse, "Efficient bipedal robots based on passive dynamic walkers," *Science*, 2005.
[3] 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.
[4] S. Levine and V. Koltun, "Variational policy search via trajectory optimization," *NIPS*, 2013.
[5] 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
[6] I. Mordatch, J. M. Wang, E. Todorov, and V. Koltun, "Animating human lower limbs using contact-invariant optimization," *ACM Trans. Graph.*, vol. 32, no. 6, pp. 203:1–203:8, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2508363.2508365
[7] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *International Journal of Robotics Research*, 2014.
[8] B. Stephens and C. Atkeson, "Push recovery by stepping for humanoid robots with force controlled joints," *Humanoids*, 2010.
[9] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," *IROS*, 2012.
[10] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: a physics engine for model-based control," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.
[11] M. Vukobratovic and B. Borovac, "Zero-moment point thirty-five years of its life," *International Journal of Human Robotics*, 2004.
[12] J. M. Wang, D. J. Fleet, and A. Hertzmann, "Optimizing walking controllers for uncertain inputs and environments," in *ACM SIGGRAPH 2010 Papers*, ser. SIGGRAPH '10. New York, NY, USA: ACM, 2010, pp. 73:1–73:8. [Online]. Available: http://doi.acm.org/10.1145/1833349.1778810



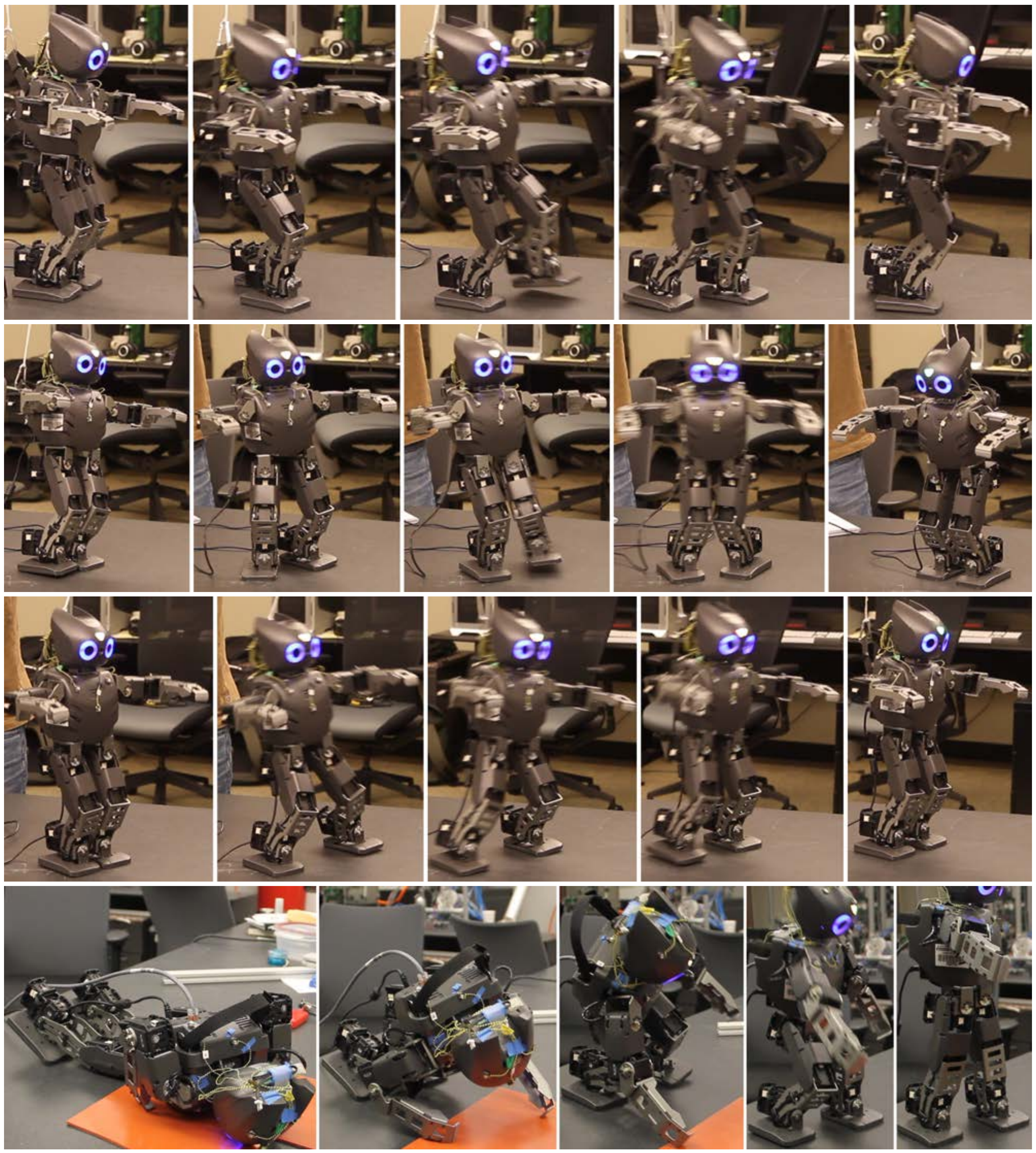


Fig. 5. Top: comparison of no-enselmbe and ensemble optimization in simulation. Bottom: illustration of the behavior in each task. We have also added the getting-up task which was not included in the main experiment.