Whole-body Model-Predictive Control applied to the HRP-2 Humanoid

J. Koenemann[1,2], A. Del Prete[2], Y. Tassa[3], E. Todorov[4], O. Stasse[2], M. Bennewitz[1] and N. Mansard[2]

***Abstract*— Controlling the robot with a permanently-updated optimal trajectory, also known as model predictive control, is the Holy Grail of whole-body motion generation. Before obtaining it, several challenges should be faced: computation cost, non-linear local minima, algorithm stability, etc. In this paper, we address the problem of applying the updated optimal control in real-time on the physical robot. In particular, we focus on the problems raised by the delays due to computation and by the differences between the real robot and the simulated model. Based on the optimal-control solver MuJoCo, we implemented a complete model-predictive controller and we applied it in real-time on the physical HRP-2 robot. It is the first time that such a whole-body model predictive controller is applied in real-time on a complex dynamic robot. Aside from the technical contributions cited above, the main contribution of this paper is to report the experimental results of this *première* implementation.**

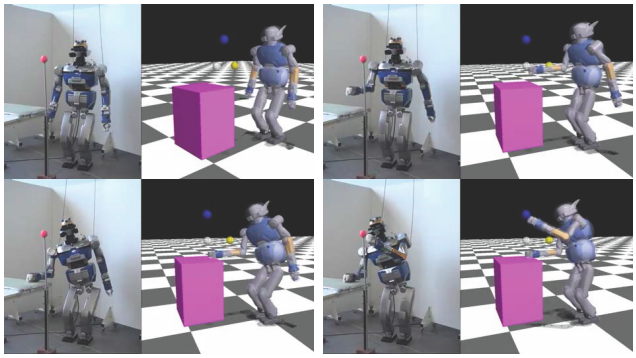


Fig. 1: Whole-body multi-contact experiment on HRP-2. The robot reaches for a target while it uses a table as an additional support to keep balance.

I. Introduction

A. Motivations

This papers deals with the problem of generating whole body motion with contacts in real time for humanoid robots. This generalized form of locomotion and manipulation would allow such robots to manipulate heavy objects, climb on uneven terrain, use the environment to stay balanced while in multiple contacts (as in Fig. 1) with one single controller. To achieve this the robot's dynamical model, forces, motor torques, self-collision, multi-contact dynamical balance have to be considered all together on a time horizon. This makes the problem very hard compared to classical instantaneous inverse kinematics. MuJoCo [1], [2] realizes this by using model predictive control and a specific contact model. So far it was mostly tested in simulation [3]. This paper describes a first evaluation towards a full deployment of this approach on a real humanoid robot, namely HRP-2.

B. State of the art

Whole-body motion is typically generated by inverse kinematics (IK) [4]. By keeping the model simple, IK is computationally efficient, but it cannot express the whole variety of constraints and tasks that we would typically expect from dynamic robots such as humanoids. For instance, the balance of a humanoid robot is an objective that is difficult to express with IK only, except by limiting the range of possible movements to only quasi-static ones. To extend the range of expressiveness of motion-generation techniques, the robotics community is switching to inverse dynamics, i.e. computing the instantaneous joint torques leading to the satisfaction of some operational constraints [5], referred to as operational-space inverse dynamics (OSID) in the following. Once more, the computation cost of the method is kept low by considering only the instantaneous linearization of the system at the current time instant. This more-complex OSID model provides more expressiveness than IK: for example, it is possible to consider instantaneous balance criteria like the ZMP on flat terrain [6] or the positivity of the contact forces in case of multiple contacts [7]. However, the instantaneous linearization once more limits the range of expressiveness. Continuing with the same example, balance control needs to consider the future evolution of the system, because the best definition of balance might be the capability of not falling in the future [8]. Basically, optimal control can be seen as an extension of OSID that considers the future evolution of the system [9], rather than its instantaneous linearization. The main difference is that the problem becomes nonlinear (and in general nonconvex) and thus much harder to solve. In particular, it is not longer possible to guarantee that the solver outputs the global minimum of the problem (not even that any global minimum exists [10]): we have to accept to work with a local minimum. The properties of this local optimum highly depends on the kind of nonlinear heuristic that was used to search it. A first class of heuristic accepts a time-consuming search among unfeasible solutions to finally output a solution that might be far from the provided initial guess [11], [12]. In that case, the optimal-control solver

[1]Jonas Koenemann and Maren Bennewitz are with University of Freiburg jonas.koenemann@yahoo.de, maren@informatik.uni-freiburg.de

[2]Andrea Del Prete, Olivier Stasse and Nicolas Mansard are with LAAS/CNRS, Toulouse adelpret@laas.fr, ostasse@laas.fr, nmansard@laas.fr

[3]Yuval Tassa is with Google UK, London yuval.tassa@gmail.com

[4]Emo Todorov is with the University of Washington todorov@cs.washington.edu

rather corresponds to a path planner, providing complex solutions, but taking minutes or hours of computation. On the other hand, it is possible to always keep a feasible solution during the search; in that case, the search algorithm is real-time (we can interrupt it at any time and get a feasible trajectory) and usually much faster. The trade-off is that the optimum is often of lower quality. In that case, the optimal-control solver rather corresponds to a controller, which is able to track a planned trajectory, but has only limited exploration capabilities [2].

C. Approach and contributions

We focus here on a specific optimal-control solver of this second class, named differential dynamic programming (DDP) [13]. DDP is a method to exploit the sparsity of the optimal-control problem without explicit constraints. It is implemented in the software MuJoCo [1]. Using multicore parallelization and a smooth dynamics approximation [14], MuJoCo can compute a 500ms trajectory for a 25-DOF humanoid robot in 50ms on a standard desktop computer. Based on this efficiency, it is possible to update in real-time the robot trajectory and control it from the output of the nonlinear optimal-control solver. This is referred to as model predictive control (MPC) [15]. As described in [16], MPC proposes some additional challenges with respect to standard optimal control. In particular, MPC has to be able to handle the noise due to the robot's sensors and the delay due to the computation time.

The contribution of this paper is to report the first application of whole-body nonlinear model predictive control on a humanoid robot (i.e. HRP-2). It is the first time that MPC is applied to the whole body of a humanoid robot, controlling the balance, handling different constraints (e.g. collisions, joint limits) and performing the specified reaching task at the same time.

In the next Section II, the MPC scheme and the DDP solver are introduced. In particular we describe choices and technical contributions that we developed to apply DDP as an MPC on a physical robot. In Section III, the connection between torque-based MPC model and physical position-driven actuators is described. The experimental setup is detailed in Section IV and the results are described and commented in Section V. Section VI draws the conclusions.

II. Model predictive control

A. Challenges in model predictive control

The optimal control problem (OCP) consists in finding the control and state trajectories optimizing a cost functional, typically composed of an integral term and a terminal term. OCP outputs an optimal state trajectory X^* that can be followed in open-loop from the starting point to the goal. Of course, open-loop tracking raises significant issues as soon as the predicted model does not perfectly match the physical system. The general idea of MPC is to update the OCP at every iteration of the control loop and only apply on the physical system the controls corresponding to the very first part of the trajectory [17]. MPC is a key methodology to properly control complex robots such as humanoids. It would for example unlock behaviors such as multi-contact locomotion.

The OCP is a planning problem, with its own difficulties of existence and approximation of the solution. Besides these problems, MPC raises additional specific challenges. First, in general we can not neglect computation times, and more generally neglect the delays between the sensor measurements and the computation of the associated control [18], [19]. We will see later that the empirical delay obtained on HRP-2 corresponds to 10 measurement-and-control cycles of the real robot. The use of the sensor measurements should be carefully considered with respect to this asynchronous delay. Several measurements can be merged to predict the robot state before each new computation. The state prediction must take into account the current set of optimal control candidates [20]. Additionally, each measurement can also be immediately used at the next control cycle, to servo the robot on the lastly-computed optimal trajectory. The DDP is particularly interesting for this last issue, because it outputs the optimal Riccati gains in addition to the optimal trajectory [21], providing an locally-optimal policy to feedback on new sensor measurements while waiting for the DDP solver to output a new trajectory candidate.

The control literature already described these three challenges and proposed some solutions, but the corresponding questions are still open. The main contribution of this paper is to experimentally exhibit these problems for the first time on a humanoid robot. In this section, we first recall the main facts about DDP that are necessary to understand the rest of the paper. We then describe with more details the three relevant problems that we noticed when applying MPC to the real robot, and quickly describe the preliminary solutions that we have applied in the experiments.

B. Differential dynamic programming

We consider in the following a direct resolution of OCP, i.e. iteratively approximating a local optimum to the OCP by a sampled (discrete-time) trajectory of state and control. This direct method boils down to the resolution of a large nonlinear optimization problem, using typically a Newton or quasi-Newton method. One of the important feature to take into account in the implementation of such a direct OCP solver is the intrinsic sparsity arising from the temporal structure of the problem. DDP [13] is an efficient way to handle this sparsity while keeping the implementation simple, in the particular case where the OCP is unconstrained. In that case, it is nearly equivalent to a Newton descent [22]. We considered a quasi-DDP descent named iterative LQR (iLQR), arising when neglecting the second-order derivatives of both the dynamics function and the cost residual [21]. In essence, this is very similar to the Gauss-Newton optimization descent. iLQR is particularly appropriate for control, since the second-order approximation increases the computational efficiency (with the drawback, not very important for control, that the exploration capabilities are lowered by the lack of second-order terms). iLQR

has largely been described in previous papers [21], [2], [3]. We recall the major technical facts in the Appendix. Mainly, the reader should be aware that DDP: i) is a Newton-like optimization scheme using only first-order derivatives, ii) takes advantage of the sparsity of the derivatives to keep a reasonable computation cost, and iii) is able to consider box constraints on the control variables. The solver outputs the optimal state trajectory X^* and the corresponding control U^*, but also the gains K of the optimal LQR to regulate X. We call $\Pi = (U, K)$ the output policy.

C. Cost overview and real-time property

The cost of one iteration of the DDP solver is in $\mathcal{O}(m^3 N)$, with m being the dimension of state and control, and N being the number of preview steps. In the experiments, this typically represents 10ms of computation (see Section VI for details). The solver additionally needs to compute the derivatives of both the dynamics and the cost function. For the humanoid robot in contact with the environment, the dynamics are computed by a contact solver and have no closed-form; the derivatives can only be approximated by finite differences. Thanks to the parallel computation on several cores, this typically takes 40ms in the experiments. These two parts are real-time: the computation time is bounded and typically has low variance. On the other hand, the solver also performs a line search (see Appendix), for which we do not know in advance the number of iterations. Each iteration is in $\mathcal{O}(N)$ and typically takes around 1ms in the experiments. The intermediate solutions of the line search are not valid because they cannot be applied on the robot: this final part of the algorithm is therefore not real-time.

In summary, despite the parallelization, the computation time of one iLQR iteration (around 50ms in the experiments) is not negligible with respect to the robot control rate (typically around 5ms) and it is not real-time. However, it is fast enough to be recomputed online (driving the motion of the robot) in a second thread running independently of the real-time control thread.

D. Real-time and fast-time loops

The robot control loop is typically real-time (the motors inputs **must** be updated at a regular and fast rate). On the opposite, the optimal-control computation is typically slower and cannot be guaranteed to converge in real-time, i.e. within a fixed time. The optimal-control solver must then be embedded in a second thread, which is independent of the real-time control thread. We say that this second thread is fast-time, i.e. it cycles as fast as possible, with some non-constant rate variance. It is not necessary that this second thread runs on the same computer as the control thread (in our case, we used a laptop-style CPU onboard the robot to host the control thread, and a desktop-style CPU offboard for the optimal-control solver).

The sensor measurements are received by the real-time control thread, and transferred to the optimal-control thread. Both the trajectory optimization and the network communication introduce a delay τ, named the *policy lag*. The lag has a varying length, depending on hardware and algorithm behavior.

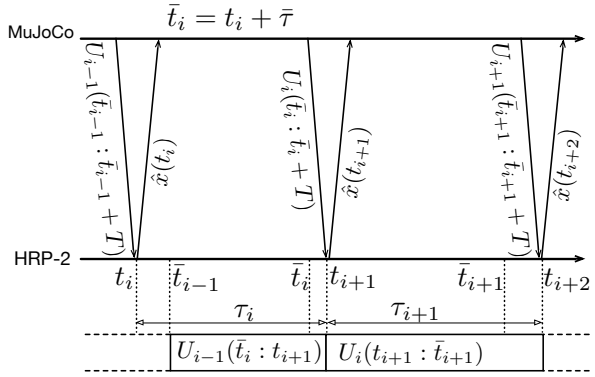


Fig. 2: Example of communication between the real-time control thread ("HRP-2") and the DDP thread ("MuJoCo"). At time t_i, the updated policy Π_{i-1} is received before expected ($\tau_{i-1} < \bar{\tau}$): the application of the new policy is delayed. At time t_{i+1} it is the opposite ($\tau_i > \bar{\tau}$): after discarding the first outdated samples, the updated policy is immediately applied. Again, at time t_{i+2}, the lag is overestimated, so the application of the updated policy is delayed.

E. State prediction

Each cycle of the optimal-control thread is denoted by the index i. It corresponds to a measurement of the robot $\hat{x}_i$ captured at time t_i, and to an output policy Π_i. The policy is received by the control thread after the lag τ_i. Since $t_i + \tau_i$ corresponds to the time of the next-cycle measurement, we denote it by t_{i+1}. The policy Π_i can be applied at best from t_{i+1}. It is therefore not necessary to try to optimize the controls Π_i during the policy lag (i.e. from t_i to t_{i+1}) since the robot will not be able to apply them. By using a proper estimation $\bar{\tau}$ of τ_i, we therefore predict the state at $t_i + \bar{\tau}$ by integrating the previous policy Π_{i-1} starting from the measurement $\hat{x}_i$.

When $\bar{\tau}$ is bigger than the actual τ_i, the policy switch is delayed (when the policy arrives at time t_{i+1}, the corresponding control has not been optimized and we have to wait until $t_i + \bar{\tau}$ to have an updated control). When $\bar{\tau}$ is smaller than the actual τ_i, the policy switch immediately applies, but the first controls are not used since they are outdated. A schematic view of the policy lag and state prediction is given in Fig. 2.

F. Tracking the state

The DDP outputs redundant information: the control policy (3) (composed of the feedforward term k and the feedback matrix K) and the corresponding state trajectory x^*. The control to send to the motor should then be:

$$u = -k - K(x - x^*)$$

, where $x - x^*$ is the deviation from the planned trajectory. The feedback gain K will then "distribute" the correction,

depending on the effect of the deviation on the cost. Alternatively, it is possible to neglect the feedback and only apply the feedforward $u = -k$. Or only the reference state x^* can be used as input to a separate controller. This last approximation has the advantage to lower the transmission bandwidth (K and k are not transferred). It is also interesting when the robot has a very good tracking system, or when the low-level control loop is difficult to model (and therefore to integrate in the dynamics of the MPC). This last point is discussed in Section IV.

We experimentally tested several ways of tracking the optimum computed by the MPC and we report the conclusions in the experimental Section VI.

G. MPC overview

Finally, the two main loops on the real-time control thread and on the optimal-control threads are summarized respectively in Alg. 1 and 2. In addition to these two loops, a "measurement" server runs on the optimal-control thread to receive the measurements sent from the control thread. The measurements are buffered when they arrive but most of them are simply discarded: only most recent measurement before starting a new optimal-control cycle is used. And a "policy" server runs on the real-time control thread to receive the optimum computed by the DDP. At the end of the transition interval, the second-last policy is discarded.

for $i = 0$ *till end, i++* **do**
  get state x_{t_i};
  predict state $x_{t_i+\bar{\tau}}$ by forward integration;
  initialize state $x := x_{t_i+\bar{\tau}}$;
  initialize policy $\Pi(t_i + \bar{\tau})$;
  while $i < maxiter$ *and* $improvement > \epsilon$ **do**
    $improvement, \Pi(t_i+\bar{\tau}) := DDP(x, \Pi(t_i+\bar{\tau}))$;
  end
  send control trajectory $\Pi(t_i + \bar{\tau})$;
end

Algorithm 1: Trajectory optimization loop

while *not finished* **do**
  send state x_{t_i};
  select latest trajectory Π valid for time t_i;
  apply control for current time $u = U(t_i)$;
end

Algorithm 2: Control loop

III. Actuator model

Typically the forward dynamic model of a mechanical system takes forces as inputs. This implies that we should be able to control the joint torques on our humanoid robot. However, HRP-2 has high-ratio gear boxes, which introduce large friction (especially static friction), making torque control challenging [23]. Implementing torque control on this kind of robots requires a torque feedback, which is currently not available on HRP-2. That is why HRP-2 is equipped with a high-gain position control, which allows for accurate tracking of the desired joint trajectories, but makes the robot stiff.

To account for this mismatch, we modeled the position controller in the forward dynamics of the system. On each motor a fast control loop ensures the tracking of the desired motor current i^d, which is computed as:

$$i^d = K_p(u - q) - K_d\dot{q},$$

where q is the joint position and K_p, K_d are the proportional and derivative gains, respectively. The control input u is then the desired joint position commanded to the low-level controller. The motor current i is directly proportional to the motor torque μ_m, which — neglecting the gear-box elasticity — differs from the joint torque μ (mainly) because of the friction torque μ_f:

$$\mu = \underbrace{K_m i}_{\mu_m} - \underbrace{K_v\dot{q} - \mu_s}_{\mu_f},$$

where we split the friction torque into two parts: viscous friction $K_v\dot{q}$ and static friction μ_s. Under the assumption of perfect current tracking we have then:

$$\mu = \underbrace{K_m K_p}_{K_1}(u - q) - \underbrace{(K_m K_d + K_v)}_{K_2}\dot{q} - \mu_s$$

To add the actuator dynamics inside the robot model we would need to identify K_1, K_2 and μ_s. Unfortunately modeling static friction is known to be a challenging problem [24] because it depends on position, temperature and external forces. However, due to the high gains, the position control compensates for most of the static friction, justifying our choice to neglect it. The identification is performed by exciting the system with sinusoidal trajectories on each joint, while collecting the desired positions u and real positions q_{meas}. Then a nonlinear least-squares problem is solved to identify the gains K_1, K_2 by minimizing the squared difference between the measured trajectory $x_{meas}^\top = \begin{bmatrix} q_{meas}^\top & \dot{q}_{meas}^\top \end{bmatrix}$ and the one obtained by simulating the system:

$$\min_{K_1,K_2} ||x - x_{meas}||^2 \qquad \text{s.t.} \qquad x_{i+1} = f(x_i, u_i, K_1, K_2)$$

For the simulation of the system we used the MuJoCo simulator, whereas for the optimization we used the Matlab function *lsqnonlin*.

The choice of modeling the position controller and the actuator inside the dynamics of our system has a number of pros and cons. On the bright side, MuJoCo is inverting the actuator-control model, simplifying the work of the onboard controller. Moreover, by bounding the control inputs we can ensure the respect of the joint limits. Furthermore, we could use a model-based reinforcement learning approach to identify on the fly the parameters of our model, while generating the optimal control policy. On the other side, in this case the actuator-control dynamics is rather simple and we could also invert it outside MuJoCo. Also, bounding the control inputs inside the joint limits could prevent the robot

from applying the desired forces on the environment. Finally, the full range of consequences of this choice are still unclear to us and we believe they need further investigation.

IV. Experimental Setup

A. Hardware

We carried out all the tests on the humanoid robot HRP-2. In the first tests we used only the upper body of the robot (10 DOFs, 4 in each arm and 2 in the torso). In the last tests we used instead its whole body (27 DOFs), thus dealing with an underactuated system. Onboard the robot we run a real-time linux on a 2.93 GHz CPU where the real-time position controller loops every 5ms, using only encoder feedback. The local stability of the robot is enforced by the proprietary stabilizer, which exploits the IMU and force/torque-sensor feedback. The machine used for the MPC computation is offboard the robot and it runs a non-real-time Windows on a 12-core 4 GHz CPU. The two machines are connected through a standard Wi-Fi (round-trip time below 1ms).

B. Software

In all our tests we used MuJoCo [1] for the online trajectory optimization. Inside MuJoCo a server thread waits for receiving the state from the robot, while a client thread sends the computed trajectories to the robot. Onboard the robot, a non-real-time server thread waits for the updated control sequence, takes care of the interpolation (MuJoCo's trajectories have a 20ms time step, while the control period is 5ms) and it updates the buffer containing the future control sequence. At the same time, the real-time loop takes care of the PD joint-position control, which applies the last control received from MuJoCo.

C. Overview of the Experiments

In the first test the robot had to use its upper body to reach a moving target. The target (i.e. a yellow ball) existed in simulation only and it was moved by the user through a 3D mouse. Using only the upper body we could neglect the balancing problem, and focus on validating the position control and the state prediction. Then in the second test the robot used its whole body to reach the same moving target while balancing (see Fig. 6). In the third test we ask the robot to make contact on a table with its right hand, and then to reach for a ball in front of it with the left hand (see Fig. 1).

V. Results

We tested the presented control architecture in different scenarios, both in simulation and on the real robot. The simulation results led us to modify the MPC scheme to overcome two issues: discontinuities in the control sequence and divergence of the DDP algorithm. Only the modified MPC scheme was then tested on the real robot.

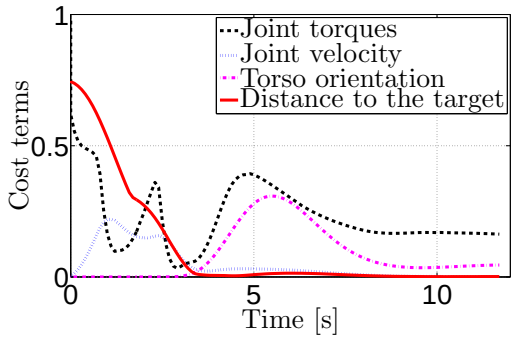


Fig. 3: Cost terms during upper-body reaching.

A. Position-Driven Model — Upper-Body Reaching

This test validates the high-gain position control that we inserted inside the dynamical model of the system. Fig. 3 shows the values (scaled to properly fit the plot) of the cost terms along time, which are: i) distance between hand and target, ii) z axis of torso (to keep it vertical), iii) joint velocities, iv) joint torques. Additionally, the whole-body dynamics and the contact constraints are enforced by the optimal-control solver, as well as the collision and joint-limit contraints.

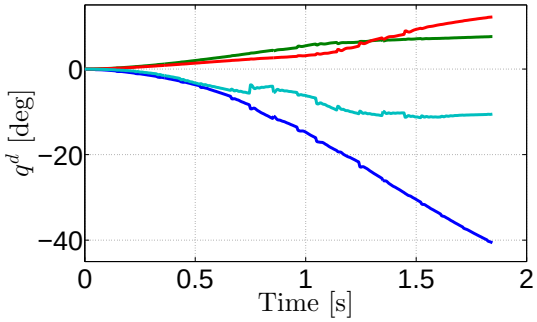
The experiment on the real robot (synchronized with the movement planned by the solver MuJoCo) is presented in the accompanying video

1) State Update and State Prediction: The first problem that we faced have been the discontinuities in the control sequence introduced by the *state update*. We observed a discontinuity every time the controller switched between two subsequent control sequences received from MuJoCo. Fig. 4 shows that these discontinuities are strongly attenuated when not updating the state. This led us not to use any *state update* in the experiments on the robot. To attenuate the discontinuities even more we performed a *state prediction* (see Section II-E). In other words, each optimization starts from the state predicted by the integration of the dynamics (considering the *policy lag*) rather than from the current state. Fig. 5 shows that the *state prediction* that we implemented actually helps reducing the discontinuities of the control sequences. Of course this approach would not work in case of strong external disturbances, but it allowed us to eliminate the above-mentioned discontinuities and focus on other crucial aspects.

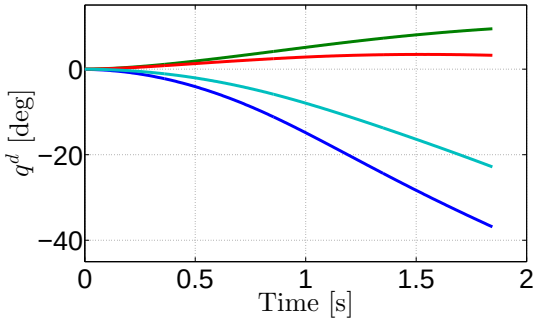
B. Torque-Driven Model — Whole-Body Reaching

The task was expressed through a cost function that trades off the following (conflicting) objectives: i) reach the target with the right hand, ii) keep the feet at their initial position, iii) keep the capture point at the center of support polygon, iv) keep the orientation of torso/waist upright, v) penalize joint velocities, iv) penalize joint torques. Like previously, the dynamics, contact and collision constraints are also enforced by the solver.

Fig. 7 shows the values of some cost terms, while Fig. 6 shows some screenshots of both the MuJoCo simulation and

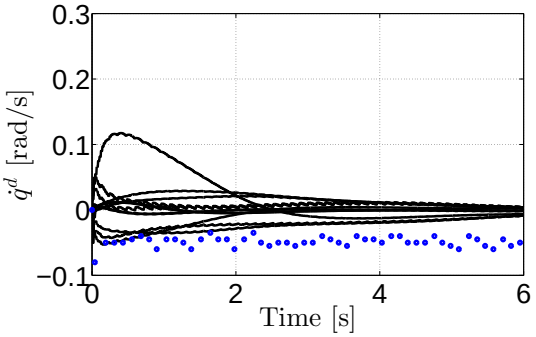


(a) Control sequence with state update.

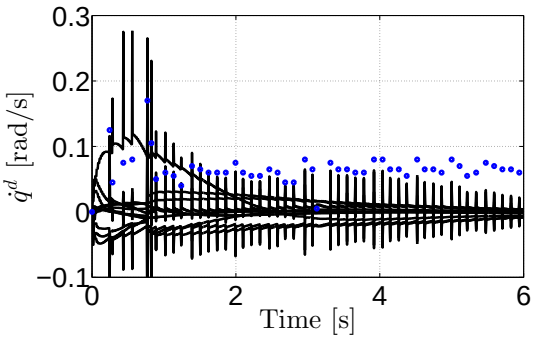


(b) Control sequence without state update.

Fig. 4: Control sequences for the 4 joints of the left arm during a reaching motion. The discontinuities are much larger when updating the state.



(a) With state prediction, expected policy lag $\bar{\tau} = 0.12$ s.



(b) Without state prediction, expected policy lag $\bar{\tau} = 0$ s.

Fig. 5: Numerical derivative of the control sequences for the 12 leg joints during a reaching motion. Derivatives are computed by finite differences. Blue circles represent the difference between the policy lag and the expected policy lag. When this is greater than zero a discontinuity is expected.

the real robot. The whole sequence can be watched in the accompanying video.

1) Actuator Model: In this test we could not use the position-driven model in MuJoCo because it led to divergence of the DDP. We believe that this did not happen in the first test because of the lack of contacts with the ground. To overcome this issue we used a standard torque-driven dynamic model; then we sent the state trajectories as references for the joint-position control.

2) Self-Collision Avoidance: Fig. 8 proves the effectiveness of the self-collision avoidance implemented in MuJoCo. Even if we moved the target inside the waist of the robot we see no penetration between the capsules used to approximate the robot's links.

C. Torque-Driven Model — Whole-Body Multi-Contact

Finally, we performed a reaching motion while controlling the robot balance with the other hand. This experiment confirms the capabilities of the optimal-control approach, that is able to unify many different operation modes of the robot in a single controller. For this test the cost function of the previous test has been marginally modified to bring the robot in contact. We removed the term penalizing inclinations of the torso/waist and we decreased the weight of the capture-point term. Moreover, we added a term to reach a target with the left hand. Contrary to the previous tests, here we used a sequence of four different cost functions to guide the motion of the robot over time. The difference between the

Fig. 6: Whole-body experiment on the real robot. The robot tracks a ball with its right hand while it keeps balance. The whole sequence can be seen in the accompanying video.

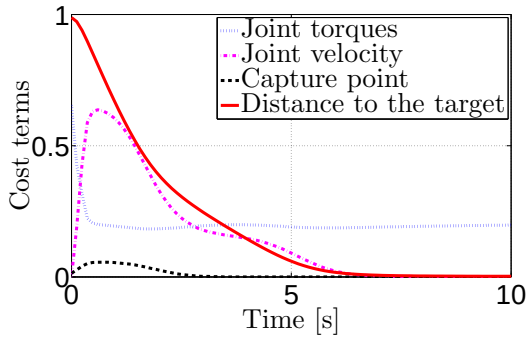


Fig. 7: Cost terms during whole-body reaching.

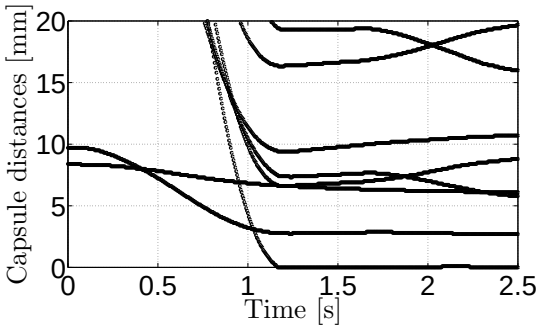


Fig. 8: Distances between the closest capsules. Negative distances would imply penetration.

cost functions was only the target positions of the right/left hands (see position of the colored balls in Fig. 1). To get the right hand in contact with the table we moved the associated target below the table. The whole sequence can be seen in the accompanying video.

VI. Conclusions

This paper described the first application of real-time whole-body MPC on a full humanoid robot. We carried out all the experiments on the HRP-2 robot using MuJoCo for the online trajectory optimization. Besides the experimental results, the contributions of this work are the modifications of the MPC scheme that we implemented to address two practical issues: the discontinuities in the control sequences and the actuation differences between model and real robot. These issues appear only when working with a real complex mechanical system, which is why they have been overlooked by previous simulation studies.

In the near future we will investigate how to reintroduce the *state update* inside the MPC scheme while maintaining smooth control sequences. Also we are in the process of implementing joint-torque control on HRP-2, which would allow us to use a torque-driven model for the trajectory optimization. Our goal is to gradually increase the complexity of the generated motion: our next step is to have the robot making dynamic motion while making and breaking contacts with the environment, e.g. walking with additional support of the hands.

Appendix
Differential Dynamic Programming

A. Problem definition

Consider a discrete dynamical system with state x and control u:

$$x_{i+1} = f(x_i, u_i) \tag{1}$$

The optimal control problem consists of minimizing a user-defined cost $J(x_0, U)$ over a certain time horizon N:

$$U^* = \underset{U}{\operatorname{argmin}} \underbrace{\sum_{i=0}^{N-1} l(x_i, u_i) + l_f(x_N)}_{J(x_0,U)}$$

Since DDP is an implicit method, it represents the trajectory through the control sequence $U = \{u_0, \dots, u_{N-1}\}$, while the states are computed integrating (1). Let us define the *cost-to-go* as:

$$J_j(x_j, U_j) = \sum_{i=j}^{N-1} l(x_i, u_i) + l_f(x_N),$$

where $U_j = \{u_j, \dots, u_{N-1}\}$. The *Value* at time i is the optimal cost-to-go starting at x_i:

$$V_i(x_i) = \min_{U_i} J_i(x_i, U_i) = \min_{u_i}[l(x_i, u_i) + V_{i+1}(f(x_i, u_i))]$$

B. Dynamic Programming

The Dynamic-Programming principle simplifies this minimization over the entire control sequence U to a cascade of minimizations over the single controls u_i. Starting by setting $V_N(x_N) = l_f(x_N)$, we can optimize backwards in time until we find the entire optimal control sequence U^*.

C. Differential resolution

Solving this cascade is not possible in the general case, due to the well-documented Bellman's curse of dimensionality. To solve each nonlinear optimization, the cost variation is therefore approximated with a quadratic function:

$$\begin{aligned} Q(\delta x, \delta u) =& l(x,u) - l(x+\delta x, u+\delta u) + \\ & V'(f(x,u)) - V'(f(x+\delta x, u+\delta u)) \\ \approx & \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}^\top \begin{bmatrix} 0 & Q_x^\top & Q_u^\top \\ Q_x & Q_{xx} & Q_{xu} \\ Q_u & Q_{ux} & Q_{uu} \end{bmatrix} \begin{bmatrix} 1 \\ \delta x \\ \delta u \end{bmatrix}, \end{aligned} \tag{2}$$

where the coefficients (i.e. first and second derivatives of Q with respect to x and u) are simply obtained as functions of the derivatives of the cost $l(x,u)$, the Value $V(x)$ and the dynamics $f(x,u)$ (see [2] for the exact expressions). While DDP uses the complete expressions of these coefficients, iLQG neglects the second derivatives of the dynamics in order to make the computation faster. In both cases, the solution of the quadratic approximation is then:

$$\delta u^* = \underset{\delta u}{\operatorname{argmin}}\, Q(\delta x, \delta u) = -\underbrace{Q_{uu}^{-1} Q_u}_{k} - \underbrace{Q_{uu}^{-1} Q_{ux}}_{K} \delta x \tag{3}$$

Substituting δu^* in (2) we get a quadratic model of V, which we need to propagate backwards in time.

D. Line search

After computing δu^* for the whole horizon we update the control policy $\hat{U}$ and the corresponding state trajectory $\hat{X}$ starting from $\hat{x}_0 = x_0$:

$$\hat{u}_i = u_i - \alpha k_i - K_i(\hat{x}_i - x_i)$$
$$\hat{x}_{i+1} = f(\hat{x}_i, \hat{u}_i),$$

where α is a backtracking search parameter, initially set to 1 and then iteratively reduced. The complete description of the nonlinear heuristic to adjust α is based on the Levenberg-Marquardt algorithm and it is described in [2].

REFERENCES

[1] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2012.

[2] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2012, pp. 4906–4913.

[3] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *Robotics and Automation (ICRA), IEEE International Conference on*, 2013.

[4] P. Baerlocher and R. Boulic, "Task-priority formulations for the kinematic control of highly redundant articulated structures," *Intelligent Robots and Systems*, no. 2, 1998.

[5] J. Park, "Control strategies for robots in contact," Ph.D. dissertation, Stanford, 2006.

[6] M. Vukobratović and B. Borovac, "Zero-moment pointthirty five years of its life," *International Journal of Humanoid Robotics*, vol. 1, no. 1, pp. 157–173, 2004.

[7] L. Saab, O. E. Ramos, N. Mansard, P. Soueres, and J.-y. Fourquet, "Dynamic Whole-Body Motion Generation under Rigid Contacts and other Unilateral Constraints," *IEEE Transactions on Robotics (to appear)*, pp. 1–17, 2013.

[8] P. Wieber, "Viability and predictive control for safe locomotion," *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2008.

[9] P. Geoffroy, N. Mansard, and M. Raison, "From Inverse Kinematics to Optimal Control," in *ARC, Advances in Robot Kinematics.*, 2014, pp. 409–418.

[10] J. Laumond, N. Mansard, and J. Lasserre, "Optimality in robot motion: optimal versus optimized motion," *Communications of the ACM*, 2014.

[11] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Transactions on Graphics*, vol. 31, no. 4, pp. 1–8, July 2012.

[12] K. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, 2009.

[13] D. Jacobson and D. Mayne, *Differential dynamic programming*. Elsevier, 1970.

[14] E. Todorov, "Implicit nonlinear complementarity: A new approach to contact dynamics," in *2010 IEEE International Conference on Robotics and Automation*, no. 5. Ieee, May 2010, pp. 2322–2329.

[15] M. Alamir, *Stabilization of nonlinear systems using receding-horizon control schemes: a parametrized approach for fast systems*. Springer (LNCIS), 2006.

[16] ——, "Fast NMPC: A reality-steered paradigm: Key properties of fast NMPC algorithms," *European Control Conference (ECC)*, June 2014.

[17] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[18] V. Zavala and L. Biegler, "The advanced-step nmpc controller: Optimality, stability and robustness," *Automatica*, vol. 45, no. 1, pp. 86–93, 2009.

[19] M. Alamir, "Monitoring control updating period in fast gradient based nmpc," in *Control Conference (ECC), 2013 European*. IEEE, 2013, pp. 3621–3626.

[20] ——, "Fast nmpc: Some good news and some facts to keep in mind," in *European Control Conference (ECC)*, Strasbourg, France, June 2014, [plenary talk].

[21] W. Li and E. Todorov, "Iterative linear-quadratic regulator design for nonlinear biological movement systems," in *ICINCO (1)*, 2004, pp. 222–229.

[22] D. M. Murray and S. J. Yakowitz, "Differential dynamic programming and Newton's method for discrete optimal control problems," *Journal of Optimization Theory and Applications*, vol. 43, no. 3, pp. 395–414, July 1984.

[23] S. Traversaro, A. Del Prete, R. Muradore, L. Natale, and F. Nori, "Inertial Parameter Identification Including Friction and Motor Dynamics," in *Humanoid Robots, 13th IEEE-RAS International Conference on*, Atlanta, Georgia, 2013, pp. 1–6.

[24] H. Olsson and K. Å ström, "Friction models and friction compensation," *European journal of control*, vol. 4, no. 3, pp. 176–195, 1998.