Real-Time Motion Planning of Legged Robots: A Model Predictive Control Approach

Farbod Farshidian, Edo Jelavić, Asutosh Satapathy, Markus Giftthaler, Jonas Buchli

Abstract— We introduce a real-time, constrained, nonlinear Model Predictive Control for the motion planning of legged robots. The proposed approach uses a constrained optimal control algorithm known as SLQ. We improve the efficiency of this algorithm by introducing a multi-processing scheme for estimating value function in its backward pass. This pass has been often calculated as a single process. This parallel SLQ algorithm can optimize longer time horizons without proportional increase in its computation time. Thus, our MPC algorithm can generate optimized trajectories for the next few phases of the motion within only a few milliseconds. This outperforms the state of the art by at least one order of magnitude. The performance of the approach is validated on a quadruped robot for generating dynamic gaits such as trotting.

I. INTRODUCTION

One of the essential requirements for robust planning in real world applications is the capability of finding solutions in real-time to adjust the plan with the current state measurements. Many of today's online approaches have achieved this efficiency through task decomposition and model reduction approaches. The main idea behind these approaches is to decompose the locomotion problem into a set of simpler tasks which effectively reduces the number of control coordinates in each subtask. This simplification is the key to make the computation of the motion planner faster [1], [2] and facilitates finding solutions in real-time. Thus, these approaches are often used in most of the practical implementations of the Model Predictive Control (MPC) in legged robots [3–7]. However, the simplification generally comes at the cost of limiting the maneuverability. This, in turn, can reduce the reachable set of solutions and renders the task synergy synthesis approaches overly conservative.

In contrast to the task decomposition approach, single task formulation offers the potential to treat the whole aspects of planning as a single problem without sacrificing performance. Due to the high complexity of legged systems, hand-designing the plan is often impractical. Therefore, an optimization method is often used to plan the robot's motion based on an user-defined performance index. This optimization problem is often formulated as an optimal control problem which provides the theoretical basis for designing a control policy. In general, there is no closed form solution for the optimal control problem with nonlinear system dynamics and cost function. Thus, the whole-body

*All authors are with the Agile & Dexterous Robotics Lab, ETH Zürich, Switzerland, email: {farbodf, jelavice, sasutosh, mgiftthaler, buchlij}@ethz.ch

optimization often renders a computationally expensive numerical problem which can not be solved in real-time.

While many of the applied optimization methods for motion planning of the legged robots do not scale favorably, the optimization methods based on using a Gauss-Newton Hessian approximation and a Riccati backward sweep demonstrate a great potential to be run in real-time on the high dimension problems. The notable examples of such approaches are DDP-based methods such as iLQR/G [8], and SLQ (Sequential Linear Quadratic) [9]. Application of these algorithms in an MPC settings have been shown previously for a humanoid robot [10]. In this work, we will show an implementation of a constrained SLQ algorithm in an MPC setting for motion planning of a quadrupedal robot. Unlike [10], we use a relatively longer time horizon due to our new approach which allows us to distribute the most expensive part of the computation in parallel.

Contributions

In this contribution, we introduce a constrained, nonlinear MPC approach for the motion planning of legged robots which its performance exceeding the current state of the art in robotics applications by at least one order of magnitude. Our MPC algorithm continuously re-optimizes the state and control input trajectories for the next few phases of the motion within only a few milliseconds. In order to achieve such a performance, we propose a variant of the SLQ algorithm which uses a multi-processing scheme for estimating value function in its backward pass.

We demonstrate the performance of this algorithm for planning highly dynamic gaits such as trotting in an MPC fashion. The robustness and real-time planning capabilities of the approach is verified by inserting significant disturbances during execution. To the best of our knowledge, this work is the first to demonstrate a whole-body nonlinear MPC for periodic gait generation of the legged systems. Last but not least, our solver is available as an open-source software [11].

II. MODEL PREDICTIVE CONTROL APPROACH

Our nonlinear MPC approach uses an efficient SLQ algorithm as its solver which can solve optimal control problems with nonlinear dynamics, cost, and equality constraints [12]. The performance of this approach for generating various gaits and obstacle avoidance has been shown previously in [12]. In this paper, we show how this algorithm can be used in an MPC loop. To this end, we first briefly introduce our optimization algorithm. Then, we propose a new approach which allows us to break the most computationally intensive

part of the algorithm into several smaller calculations which can be carried out simultaneously. Finally, we discuss our MPC approach.

A. An Overview of the SLQ Algorithm

The continuous constrained SLQ algorithm is based on dynamic programming, which designs both a feedforward plan and a feedback controller through a quadratic approximation of the value function. Our continuous-time SLQ algorithm can handle state-input equality constraints through a Lagrangian method and state-only constraints through a penalty method. The complexity of the algorithm scales linearly with the time horizon of the optimization [12].

In general, planning algorithms based on Nonlinear Programming (NLP) require first to transcribe the infinite-dimensional, continuous problem to a finite-dimensional NLP. This discretization is often carried out using heuristics, which can result in numerically poor or practically infeasible solutions. In contrast, the continuous-time SLQ uses variable step-size ODE (Ordinary Differential Equation) solvers in its forward and backward passes. Given the desired accuracy, it can automatically discretize the problem using the error control mechanism of the variable step-size ODE solvers. Furthermore, using such an adaptive scheme – in average – decreases the number of discretized points in comparison to the discrete-time SLQ algorithm. This, in turn, can improve the run-time of an iteration since the number of calculations significantly decreases for the expensive operations such as linearization of the dynamics and the constraints.

In this paper, we use the SLQ algorithm for solving optimal control problem for switched systems. Switched systems are a subclass of a more general family known as hybrid systems. A hybrid system consists of a finite number of dynamical subsystems subjected to discrete events. These events are triggered either by an external input, or through intersection of the state trajectory to certain manifolds known as the switching surfaces. Upon triggering an event, a transition to a different subsystem takes place which can be followed by a sudden jump in the state vector. As more redistricted hybrid models, switched systems are characterized by continuous transitions of state trajectory during the switching moments. Here, we have further restricted our switched system model by assuming that the transitions are triggered by predefined switching times, in between predefined sequence of subsystems. For more general treatment of this problem refer to [12], [13].

We formulate the constrained optimal control problem for switched systems in the finite time interval $[t_0, t_I]$ as

$$
\begin{aligned}
\underset{\mathbf{u}(\cdot)}{\text{minimize}} \quad & \sum_{i=0}^{I-1} \Phi_i(\mathbf{x}(t_{i+1})) + \int_{t_i}^{t_{i+1}} L_i(\mathbf{x}, \mathbf{u}, t) dt \\
\text{subject to} \quad & \dot{\mathbf{x}} = \mathbf{f}_i(\mathbf{x}, \mathbf{u}), \quad \mathbf{x}(s_0) = \mathbf{x}_0, \quad \mathbf{x}(t_i^-) = \mathbf{x}(t_i^+) \\
& \mathbf{g}_{1i}(\mathbf{x}, \mathbf{u}, t) = 0, \quad \mathbf{g}_{2i}(\mathbf{x}, t) = 0,
\end{aligned}
\tag{1}
$$

where t_1 to t_{I-1} are the switching times and I is the number of subsystems. For each mode i, the nonlinear cost function consists of a terminal cost, $\Phi_i(\cdot)$, and an intermediate cost,

$L_i(\cdot)$. Here, $\mathbf{f}_i(\cdot)$, $\mathbf{g}_{1i}(\cdot)$, and $\mathbf{g}_{2i}(\cdot)$ are respectively the system dynamics, the state-input constraints, and the state-only constraints in mode i.

The SLQ algorithm iteratively solves the extremal problem around the latest estimation of the optimal trajectories and improves the optimal control policy based on the solution of this local extremal problem. The local extremal problems are defined by the linearized system dynamics and constraints and the quadratic approximation of the cost function. The first step of each iteration is a forward integration of the system dynamics using the last approximation of the optimal controller. Next, a quadratic approximation of the cost function is calculated over the nominal state and input trajectories obtained from the forward integration.

$$
\begin{aligned}
\widetilde{J} &= \sum_{i=0}^{I-1} \widetilde{\Phi}_i(\delta\mathbf{x}(t_{i+1})) + \int_{t_i}^{t_{i+1}} \widetilde{L}_i(\delta\mathbf{x}, \delta\mathbf{u}, t) dt \\
\widetilde{\Phi}_i(\delta\mathbf{x}) &= q_{f,i} + \mathbf{q}_{f,i}^\top \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{x}^\top \mathbf{Q}_{f,i} \delta\mathbf{x} \\
\widetilde{L}_i(\mathbf{x}, \mathbf{u}, t) &= q_i(t) + \mathbf{q}_i(t)^\top \delta\mathbf{x} + \mathbf{r}_i(t)^\top \delta\mathbf{u} + \delta\mathbf{x}^\top \mathbf{P}_i(t) \delta\mathbf{u} \\
&\quad + \frac{1}{2} \delta\mathbf{x}^\top \mathbf{Q}_i(t) \delta\mathbf{x} + \frac{1}{2} \delta\mathbf{u}^\top \mathbf{R}_i(t) \delta\mathbf{u}
\end{aligned}
\tag{2}
$$

where $q_i(t)$, $\mathbf{q}_i(t)$, $\mathbf{r}_i(t)$, $\mathbf{P}_i(t)$, $\mathbf{Q}_i(t)$, and $\mathbf{R}_i(t)$ are the coefficients of the Taylor expansion of the ith cost function in Equation (1) around the nominal trajectories. $\delta\mathbf{x}$ and $\delta\mathbf{u}$ are the deviations of state and input from the nominal trajectories. Constrained SLQ also uses linear approximations of the system dynamics and constraints in Equation (1) around the nominal trajectories.

$$
\begin{aligned}
\delta\dot{\mathbf{x}} &= \mathbf{A}_i(t)\delta\mathbf{x} + \mathbf{B}_i(t)\delta\mathbf{u} \\
\mathbf{C}_i(t)\delta\mathbf{x} &+ \mathbf{D}_i(t)\delta\mathbf{u} + \mathbf{e}_i(t) = \mathbf{0} \\
\mathbf{F}_i(t)\delta\mathbf{x} &+ \mathbf{h}_i(t) = \mathbf{0}
\end{aligned}
\tag{3}
$$

Based on this LQ approximation, a generalized, constrained LQR algorithm is used to find an update to the feedback-feedforward control policy. For a more detailed discussion of the algorithm's derivation, refer to [12].

B. FastSLQ Algorithm

Each iteration of the SLQ algorithm consists of three main steps namely forward integration of system dynamics (i.e. forward pass), constructing the LQ approximation of the nonlinear problem, and solving Riccati-like equations (i.e. backward pass). Computing the LQ approximation in parallel has been proven to be essential in high dimensional problems. However, due to the sequential nature of integration, the forward and backward passes have been often implemented as single processes. In this section, we propose a variant of the SLQ algorithm which uses a parallel-processing scheme for calculating the backward pass. In practice, the backward pass is the most expensive part of the computations. Thus, parallel computation of the backward pass can significantly improve the speed of the algorithm. To this end, we refer to this algorithm as FastSLQ.

In FastSLQ, we need first to divide the optimization time horizon into several disjoint intervals. A natural partitioning

578

Algorithm 1 FASTSLQ Algorithm

1: **Given:**
2:   Initial stable control policy, $\{\mathbf{u}_i(\mathbf{x},t)\}_{i=0}^{I-1} = \{\mathbf{u}_{ff,i}(t) + \mathbf{K}_i(t)\mathbf{x}\}_{i=0}^{I-1}$
3:   Initial value function, $\{V_i(\mathbf{x},t), V_{e,i}(\mathbf{x},t)\}_{i=0}^{I-1}$
4:   Heuristic function for approximating infinite time problem $V_I(\mathbf{x},t) = V^{lqr}(\mathbf{x})$
5: **repeat**
6:   **Forward integrate the system dynamics** using adaptive step-size integrator.
7:     $\tau: \overline{\mathbf{x}}(t_0), \overline{\mathbf{u}}(t_0), \overline{\mathbf{x}}(t_1), \overline{\mathbf{u}}(t_1)\ldots\overline{\mathbf{x}}(t_{N-1}), \overline{\mathbf{u}}(t_{N-1}), \overline{\mathbf{x}}(t_N = t_I)$
8:   **for** $i = I-1$ to 0 **in parallel do**
9:     **Quadratize cost function** along the trajectory τ
10:     **Linearize the system dynamics and constraints** along the trajectory τ
11:     Compute the constrained LQR problem coefficients
12:       $\mathbf{D}_i^{\dagger} = \mathbf{R}_i^{-1}\mathbf{D}_i^{\top}(\mathbf{D}_i\mathbf{R}_i^{-1}\mathbf{D}_i^{\top})^{-1}, \quad \widetilde{\mathbf{A}}_i = \mathbf{A}_i - \mathbf{B}_i\mathbf{D}_i^{\dagger}\mathbf{C}_i$
13:       $\widetilde{\mathbf{C}}_i = \mathbf{D}_i^{\dagger}\mathbf{C}_i, \quad \widetilde{\mathbf{D}}_i = \mathbf{D}_i^{\dagger}\mathbf{D}_i, \quad \widetilde{\mathbf{e}}_i = \mathbf{D}_i^{\dagger}\mathbf{e}_i$
14:       $\widetilde{\mathbf{Q}}_i = \mathbf{Q}_i + \widetilde{\mathbf{C}}_i^{\top}\mathbf{R}_i\widetilde{\mathbf{C}}_i - \mathbf{P}_i\widetilde{\mathbf{C}}_i - (\mathbf{P}_i\widetilde{\mathbf{C}}_i)^{\top} + \rho\mathbf{F}_i^{\top}\mathbf{F}_i$
15:       $\widetilde{\mathbf{q}}_i = \mathbf{q}_i - \widetilde{\mathbf{C}}_i^{\top}\mathbf{r}_i + \rho\mathbf{F}_i^{\top}\mathbf{h}_i, \quad \widetilde{\mathbf{R}}_i = (\mathbf{I} - \widetilde{\mathbf{D}}_i)^{\top}\mathbf{R}_i(\mathbf{I} - \widetilde{\mathbf{D}}_i)$
16:       $\mathbf{L}_i = \mathbf{R}_i^{-1}(\mathbf{P}_i^{\top} + \mathbf{B}_i^{\top}\mathbf{S}_i)$
17:       $\widetilde{\mathbf{l}}_i = \mathbf{R}_i^{-1}(\mathbf{r}_i + \mathbf{B}_i^{\top}\mathbf{s}_i), \quad \widetilde{\mathbf{l}}_{e,i} = \mathbf{R}_i^{-1}\mathbf{B}_i^{\top}\mathbf{s}_{e,i}$
18:     Calculate final value for Riccati-like equations
19:       $\mathbf{S}_i(t_{i+1}) = \mathbf{Q}_{f,i} + \frac{\partial^2}{\partial\mathbf{x}^2}V_{i+1}(\overline{\mathbf{x}}(t_{i+1}), t_{i+1})$
20:       $\mathbf{s}_i(t_{i+1}) = \mathbf{q}_{f,i} + \frac{\partial}{\partial\mathbf{x}}V_{i+1}(\overline{\mathbf{x}}(t_{i+1}), t_{i+1}), \mathbf{s}_{e,i}(t_{i+1}) = \frac{\partial}{\partial\mathbf{x}}V_{e,i+1}(\overline{\mathbf{x}}(t_{i+1}), t_{i+1})$
21:       $s_i(t_{i+1}) = q_{f,i} + V_{i+1}(\overline{\mathbf{x}}(t_{i+1}), t_{i+1}) + V_{e,i+1}(\overline{\mathbf{x}}(t_{i+1}), t_{i+1})$
22:     **Solve the final-value Riccati-like equations** in interval $[t_i, t_{i+1}]$
23:       $-\dot{\mathbf{S}}_i = \widetilde{\mathbf{A}}_i^{\top}\mathbf{S}_i + \mathbf{S}_i^{\top}\widetilde{\mathbf{A}}_i - \widetilde{\mathbf{L}}_i^{\top}\widetilde{\mathbf{R}}_i\widetilde{\mathbf{L}}_i + \widetilde{\mathbf{Q}}_i$
24:       $-\dot{\mathbf{s}}_i = \widetilde{\mathbf{A}}_i^{\top}\mathbf{s}_i - \widetilde{\mathbf{L}}_i^{\top}\widetilde{\mathbf{R}}_i\widetilde{\mathbf{l}}_i + \widetilde{\mathbf{q}}_i$
25:       $-\dot{\mathbf{s}}_{e_i} = \widetilde{\mathbf{A}}_i^{\top}\mathbf{s}_{e_i} - \widetilde{\mathbf{L}}_i^{\top}\widetilde{\mathbf{R}}_i\widetilde{\mathbf{l}}_{e_i} + (\widetilde{\mathbf{C}}_i - \widetilde{\mathbf{L}}_i)^{\top}\mathbf{R}_i\widetilde{\mathbf{e}}_i$
26:       $-\dot{s}_i = q_i - \widetilde{\mathbf{l}}_i^{\top}\widetilde{\mathbf{R}}_i\widetilde{\mathbf{l}}_i$
27:     **Compute value function update**
28:       $V_i(\mathbf{x},t) = s_i(t) + (\mathbf{x} - \overline{\mathbf{x}}(t))^{\top}\mathbf{s}_i(t) + \frac{1}{2}(\mathbf{x} - \overline{\mathbf{x}}(t))^{\top}\mathbf{S}_i(t)(\mathbf{x} - \overline{\mathbf{x}}(t))$
29:       $V_{e,i}(\mathbf{x},t) = (\mathbf{x} - \overline{\mathbf{x}}(t))^{\top}\mathbf{s}_{e,i}(t)$
30:     **Compute controller update**
31:       $\mathbf{L}_i = -(\mathbf{I} - \widetilde{\mathbf{D}}_i)\widetilde{\mathbf{L}}_i - \widetilde{\mathbf{C}}_i, \quad \mathbf{l}_i = -(\mathbf{I} - \widetilde{\mathbf{D}}_i)\widetilde{\mathbf{l}}_i, \quad \mathbf{l}_{e,i} = -(\mathbf{I} - \widetilde{\mathbf{D}}_i)\widetilde{\mathbf{l}}_{e,i} - \widetilde{\mathbf{e}}_i$
32:       $\mathbf{u}_{ff,i} = \overline{\mathbf{u}} + \alpha\mathbf{l}_i + \mathbf{l}_{e,i} - \mathbf{L}_i\overline{\mathbf{x}}, \quad \mathbf{u}_i(\mathbf{x},t) = \mathbf{u}_{ff,i}(t) + \mathbf{L}_i(t)\mathbf{x}$
33:   **end for**
34:   **line search scheme**: optimize the learning rate, α.
35: **until** convergence or maximum number of iterations
36: **return** Optimized control policy and value function, $\{\mathbf{u}_i(\mathbf{x},t), V_i(\mathbf{x},t), V_{e,i}(\mathbf{x},t)\}$

scheme for our switched system formulation is based on the switching moments but in general, any other partitioning approach can be considered. In order to solve the Riccati-like equations in each of these partitions, we should estimate the final value of the equations in each partition.

A naive approach to compute the backward passes of these partitions in parallel is the following. In each iteration, all processes – in parallel – integrate the Riccati-like equations of each partition backward in time. The final-value of these equations can be calculated based on the solution of the following partition in the previous iteration. This is in contrast to the sequential approach which waits until the computation of the following partition to complete. However, since the nominal trajectories of the previous iteration are different from the current ones, the Riccati-like equations in the subsequent iterations are different.

To tackle this issue, we should first notice that SLQ uses the Bellman equation of optimality to locally estimate the value function. This local estimation is based on a quadratic approximation of the value function around the nominal trajectories. The coefficients of this quadratic model are calculated through the Riccati-like equations in the backward pass. FASTSLQ leverages this approximation in order to correct the final values obtained from the previous iteration. To do so, it employs the value function approximation of the following partition from the previous iteration to estimate its value and its first order derivative at the partition's final time and the corresponding nominal state. Then, it uses the following equations to improve the estimation of the final

values of the Riccati-like equations.

$$\mathbf{S}_i^k(t_{i+1}) = \mathbf{Q}_{f,i}^k + \frac{\partial^2}{\partial\mathbf{x}^2}V_{i+1}^{k-1}(\overline{\mathbf{x}}^k(t_{i+1}), t_{i+1})$$

$$\mathbf{s}_i^k(t_{i+1}) = \mathbf{q}_{f,i}^k + \frac{\partial}{\partial\mathbf{x}}V_{i+1}^{k-1}(\overline{\mathbf{x}}^k(t_{i+1}), t_{i+1})$$

$$\mathbf{s}_{e,i}^k(t_{i+1}) = \frac{\partial}{\partial\mathbf{x}}V_{e,i+1}^{k-1}(\overline{\mathbf{x}}^k(t_{i+1}), t_{i+1})$$

$$s_i^k(t_{i+1}) = q_{f,i}^k + V_{i+1}^{k-1}(\overline{\mathbf{x}}^k(t_{i+1}), t_{i+1}) + V_{e,i+1}^{k-1}(\overline{\mathbf{x}}^k(t_{i+1}), t_{i+1})$$

where we have defined

$$\delta\mathbf{x}^k(\mathbf{x},t) = \mathbf{x} - \overline{\mathbf{x}}^k(t)$$

$$V_{e,i}^k(\mathbf{x},t) = \delta\mathbf{x}^k(\mathbf{x},t)^{\top}\mathbf{s}_{e,i}^k(t)$$

$$V_i^k(\mathbf{x},t) = s_i^k(t) + \delta\mathbf{x}^k(\mathbf{x},t)^{\top}\mathbf{s}_i^k(t) + \frac{1}{2}\delta\mathbf{x}^k(\mathbf{x},t)^{\top}\mathbf{S}_i^k(t)\delta\mathbf{x}^k(\mathbf{x},t),$$

the subscript i and superscript k refer to the subsystem index and the iteration number respectively. t_is are the switching times (partitioning times), $V_i^k(\cdot)$ and $V_{e,i}^k(\cdot)$ are the value function approximation for the system in the null space and projected space of the constraints respectively. Finally, $\overline{\mathbf{x}}^k(\cdot)$ is the nominal state trajectory in iteration k.

A high level illustration of the backward pass of FASTSLQ is shown in Fig. 1. Note that, since FASTSLQ provides a quadratic approximation of the value function, the correction is only effective up to the first order terms (i.e. $\mathbf{s}(\cdot), \mathbf{s}_e(\cdot), s(\cdot)$) and the second order term (i.e. $\mathbf{S}(\cdot)$) is directly used from the previous iteration.

A summary of FASTSLQ, is given in Algorithm 1. For a reliable implementation of FASTSLQ, extra care should be taken when selecting the learning rate in the line search scheme. The line search scheme in the original SLQ algorithm favors the largest learning step which has a lower cost than the nominal cost. In order to ensure that the changes of the nominal trajectories in successive iterations are small enough that the quadratic approximation of the value function is valid, an extra criterion should be included in the line search scheme of the FASTSLQ algorithm. To this end, the following criterion is appended to the line search scheme's conditions. The new cost associated with the updated control policy should be close enough to the expected cost which is estimated through the approximate value function. This line search scheme is in particular necessary in the initial iterations when the solution is far from the optimal one. [1]

The backward pass of the SLQ-type algorithms can be considered as a bootstrapping approach for estimating the value function. In the original SLQ algorithm, the state sweeps are carried out in an especial order – from the final state towards the initial state. Due to the specific structure of the problem, this process converges in a single sweep. In contrast, FASTSLQ uses a different sweeping order which allows to implement the process in parallel. At a very high level, this is similar to how the value function is estimated in asynchronous dynamic programing approach where the state sweep can be carried out in an arbitrary order as long as all states are visited.

---

[1]Note that this is not a major issue in a real-time iteration MPC since the optimal solution of the subsequent problems are in the close neighborhood.
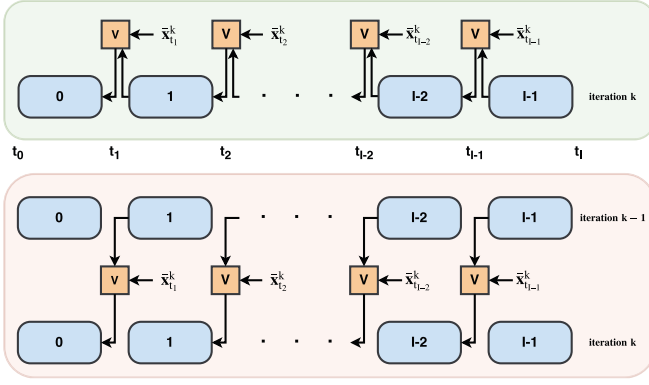
Fig. 1. Comparison of the backward passes of SLQ and FASTSLQ for estimating the value function based on solving Riccati-like equations. The top graph (green box) illustrates the sequential approach in which the Riccati-like equations are solved backward in time. The bottom graph (red box) illustrates the parallel computation approach. In this approach, instead of waiting for the solution of the neighboring partition, the approximation of the value function from the previous iteration is used. However, in order to account for the changes of the nominal trajectories in two iterations, the values function and its derivatives are re-evaluated at the current nominal trajectories.

C. Dealing with State-Input Inequality Constraints

In order to deal with state-input inequity constraints, we employ an approach similar to [14]. While this method does not necessarily result in an optimal solution, it ensures that the planned trajectories are constraint-satisfactory. In this method, if at any time during the algorithm's forward pass, an inequity constraint is violated, the control vector is projected on the plane defined by the linearized inequality constraint.

D. FASTSLQ-MPC with Real-Time Iteration Scheme

The real-time scheme is proposed for the scenarios such as MPC where we need to solve a sequence of optimization problems, but we do not have the time to iterate each problem to convergence. In the real-time iteration MPC, the optimal control solver never attempts to iterate to convergence, but instead only takes one iteration towards the solution of the most current MPC problem (triggered with new measurements), before proceeding to the next one [15].

In the absence of disturbances and if the terminal time of the optimization is not receded, this scheme subsequently delivers approximations of the optimal policy that become better and better over iterations. However, in the presence of disturbances or receding horizon MPC scheme, it is crucial to the success of this method that the transition between subsequent problems be carefully designed.

In the presence of disturbances, the initial state of the subsequent optimization problem would deviate from the planned state. Thus, in a naive real-time iteration implementation, depending on the amplitude of the deviations, the convergence can be drastically affected. In order to reduce this effect, the current solution of the optimizer should be corrected based on the state deviations. In the FASTSLQ-MPC algorithm, such a correction can be achieved using its locally linear feedback policy. This policy generalizes the optimal open loop solution to a vicinity of the optimal trajectories while respecting constraints. Thus local adjustment

---

Algorithm 2 FASTSLQ-MPC Algorithm

1: **Given:**
2:     Initial stable control policy, $\{\mathbf{u}_i(\mathbf{x},t)\} = \{\mathbf{u}_{ff,i}(t) + \mathbf{K}_i(t)\mathbf{x}\}$.
3:     LQR Quadratic cost function, $\{\mathbf{Q}_i^{lqr}, \mathbf{R}_i^{lqr}\}$, for heuristic value function.
4: **repeat**
5:     **Get the current time and state**, $t_0, \mathbf{x}_0$.
6:     **if** $t_f - t < t_h$ **then**
7:         **Append a new subsystem**, $I - 1$, based on the gait pattern.
8:         **Adjust final time:** $t_f = t_I$
9:         **Calculate an LQR** for subsystem I using linearized system at $\mathbf{x}(t_{I-1})$.
10:        **Append control policy** with LQR controller: $\mathbf{u}_{I-1}(\mathbf{x},t) = K^{lqr}(\mathbf{x} - \mathbf{x}(t_I))$.
11:        **Set final cost** using LQR quadratic value function: $V_I(\mathbf{x},t) = V^{lqr}(\mathbf{x})$.
12:    **end if**
13:    **Adjust the previous controller**
14:        Rollout from $(t, \mathbf{x})$ using policy $\{\mathbf{u}_i(\mathbf{x},t)\}$ and get nominal trajectories
15:        $\tau : \bar{\mathbf{x}}(t_0), \bar{\mathbf{u}}(t_0), \bar{\mathbf{x}}(t_1), \bar{\mathbf{u}}(t_1) \dots \bar{\mathbf{x}}(t_{N-1}), \bar{\mathbf{u}}(t_{N-1}), \bar{\mathbf{x}}(t_N = t_I)$
16:        Adjust the previous controller feedforward component, $\{\mathbf{u}_{ff,i}(t)\}$.
17:        $\mathbf{u}_{ff,i}(t) = \bar{\mathbf{u}}(t) - \mathbf{K}_i(t)\bar{\mathbf{x}}(t)$
18:    **Perform single iteration of SLQ**
19:        Set adjusted policy, $\{\mathbf{u}_i(\mathbf{x},t)\}_{i=0}^{I-1} = \{\mathbf{u}_{ff,i}(t) + \mathbf{K}_i(t)\mathbf{x}\}_{i=0}^{I-1}$.
20:        Set augmented value function, $\{V_i(\mathbf{x},t), V_{e,i}(\mathbf{x},t)\}_{i=0}^{I}$.
21:        Perform single iteration of SLQ in time period $[t_0, t_f]$.
22:    **Get the feedback policy and value funtion**
23:        $\{\mathbf{u}_i(\mathbf{x},t)\}_{i=0}^{I-1}, \{V_i(\mathbf{x},t), V_{e,i}(\mathbf{x},t)\}_{i=0}^{I}$.
24: **until** finished

---

of the plan can be realized by using the optimal feedback policy of the SLQ instead of using optimized trajectories.

In addition to disturbance, we should also address the issue arises from the shifting of the optimization's terminal time in order to maintain the time horizon of optimization. This issue arises because of the finite-time optimization setup. To tackle it, we increment the terminal cost of the optimization with the value function of a fictitiously infinite-time LQR solution defined at the final state. Moreover, we have employed a varying time horizon scheme, in which the final time is set to a future switching time such that the time horizon includes exactly n complete switching modes. For example, for $n = 2$ at time t where subsystem i is active ($t_i \leq t < t_{i+1}$), the terminal time will be set to t_{i+3}. This means that if the average activation time of subsystems is Δt, the optimization time horizon varies in between $2\Delta t$ and $3\Delta t$.

In this way, the terminal time of the MPC optimization is always set to moments of mode switch. As long as the final states of the modes are controllable, they can be stabilized by fictitiously linear LQR controllers. Thus, these LQR problems have finite value functions. Defining such a stable LQR problem also allows using a quasi-infinite horizon MPC approach [16], [17]. The stability of this quasi-infinite horizon MPC can be guaranteed as long as the system is controllable at the terminal time. For the gait studied in this paper, this switching moments correspond to 4-leg stance mode in which the system is fully actuated and controllable. Our MPC approach is described in Algorithm 2.

III. EXPERIMENTAL SETUP

A. Platform Description

For evaluating our MPC approach, we use a hydraulically actuated quadruped robot known as HyQ [18]. HyQ is a fully torque controlled quadruped and it features three joints per leg. All joints are equipped with absolute and relative encoders. The joint torques are measured by load cells which are also used to estimate the contact forces.

In our setup, we use a dedicated computer to run the MPC control loop. This computer has an Intel Corei7 4790

580

Authorized licensed use limited to: Zhejiang University. Downloaded on February 02,2023 at 12:13:57 UTC from IEEE Xplore. Restrictions apply.

processor (8 M Cache, 3.60 GHz processor frequency). The MPC loop receives the current state of the robot from the midlevel control computer that executes the tracking controller described in subsection III-D. This tracking controller runs at 250 Hz. The midlevel controller then sends desired torque to a lowlevel torque controller. In return, it receives current state measurements and computes the base and ground state estimates. The lowlevel controller runs a torque tracking control loop for actuators at 1 KHz.

In order to estimate the base state, we use a Kalman filter approach introduced in [19]. For estimating the ground plane, we use a simple approach which approximates the ground plane by passing a plane at the stance feet of the robot (in two consequential phases).

B. Modeling Framework

In this paper, we choose to use a whole-body modeling approach known as the Center of Mass (CoM) dynamics plus full kinematics [20]. This model includes 12 states describing the CoM motion as well as robot's joint positions which describe the full kinematic of the robot. The control input of this model includes the contact forces at end-effectors and the joint velocities. Due to the impact forces at the touch-down moments of swing legs, the state trajectories are not continuous. Thus, this model is a hybrid model. However, here, we model the legged robot as a switched system.

This transition from a hybrid model to a switching model requires to assume that there is no state jump at the moments of phase transitions. Here, we reinforce such an assumption by designing swing leg trajectories with zero approaching velocity at the touch-downs. Assuming that the phase sequence of the motion is predefined, we can construct a switched model with a set of constraints on the velocities and the contact forces at end-effectors [12]. For HyQ, The CoM dynamics plus full kinematics model has 24 states and 24 inputs. It includes 12+NUMBEROFSWINGLEGS active state-input equality constraints at each phase of motion. The equation of motion for this model is as following

$$
\begin{cases}
\dot{\theta} = \mathbf{T}(\theta)\left(\omega - {}_B\mathbf{J}^{\omega}_{com}(\mathbf{q})\,\mathbf{u}\right) \\
\dot{\mathbf{p}} = \mathbf{R}(\theta)\,\mathbf{v} \\
\dot{\omega} = \mathbf{I}^{-1}(\mathbf{q})\left(\dot{\mathbf{I}}(\mathbf{q},\mathbf{u}) - \omega \times \mathbf{I}(\mathbf{q})\omega + \sum_{i=1}^{4}\mathbf{r}_{f_i}(\mathbf{q}) \times \lambda_{f_i}\right) \\
\dot{\mathbf{v}} = \mathbf{g}(\theta) + \frac{1}{m}\sum_{i=1}^{4}\lambda_{f_i} \\
\dot{\mathbf{q}} = \mathbf{u}
\end{cases}
$$

$$
\begin{cases}
\mathbf{u}_{f_i}(\mathbf{q},\mathbf{u}) = \mathbf{0} & \text{if } i \text{ is a stance leg} \\
\mathbf{u}_{f_i}(\mathbf{q},\mathbf{u})\cdot \hat{n} = c(t), \quad \lambda_{f_i} = \mathbf{0} & \text{if } i \text{ is a swing leg}
\end{cases}
$$

where $\mathbf{T}$ is a matrix which transforms the angular velocities in the base frame to the Euler angles derivatives in the global frame. $\mathbf{R}$ is the rotation matrix of the base with respect the global frame, $\mathbf{g}$ is the gravitational acceleration in the body frame, $\mathbf{I}$ and m are respectively the total moment of inertia about the CoM and the total mass. ${}_B\mathbf{J}^{\omega}_{com}$ is the Jacobian matrix of CoM rotation with respect to robot's base frame. $\mathbf{r}_{f_i}$, $\mathbf{u}_{f_i}$, λ_{f_i} are respectively the position, velocity and contact force vector of foot $i \in \{1,2,3,4\}$.

We use a predefined swing leg trajectory, $c(t)$, in the orthogonal direction of the contact surface ($\hat{n}$) which ensures
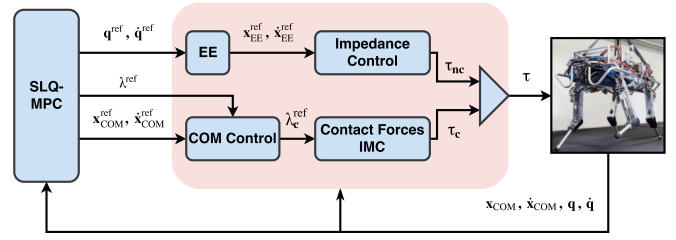

Fig. 2. Overview of the motion control and planning structure. The planner is illustrated at far left and the motion controller (red block) in middle. The EE block transforms the joint's coordinate to the end-effector's Cartesian coordinate which is controlled by an impedance controller (inverse-dynamics + PD). The CoM controller stabilizes the CoM by providing correction to the CoM acceleration which is then mapped to an equivalent correction to λ_{ref} of the MPC. Finally, the IMC tracks the corrected end-effector's forces.

that the touch-down takes place according to the predefined switching times. Furthermore, it ensures that the velocity of the swing foot before the contact is zero. The unilateral and the friction constraints on contact forces are enforced by the method introduced in subsection II-C.

C. Model Comparison

The computational complexity of SLQ scales cubically with respect to the sum of state's and input's dimensions, $O\left((n_x + n_u)^3\right)$. In our switched model formulation for HyQ, this sum is of dimension 48. A comparable model of HyQ which results in a same computational complicity is the rigid body modeling approach with a soft contact model [21]. The state space in this model is of dimension 36 which consists of the base pose and twist as well as joint angles and velocities. The control inputs only includes the joint torques with dimension 12. The contact forces are calculated using a soft model consisting of nonlinear springs and dampers.

Each of these models has their advantages and disadvantages. The switched model uses hard constraints to satisfy contact unilateral constraints. However, the full model with soft contacts uses a penalty method in which constraints are not always fully satisfied. Thus, special care should be taken to implement them on hardware and in general achieving a good go-to task is hard. On the other hand, the optimized plan for the swing leg velocities in the full model with soft contact is always continuous but the velocities of the swing legs in the switched model are piecewise continuous. Therefore, in the switched model, we need to pre-filter the planned joint velocities before applying them on hardware.

D. Motion Control Structure

In our switched model approach the contact forces at end-effectors are part of the control authority. However, the actual commands to the robot are the joint torques. There are two ways to realize this contact forces. Either, we should map them back to an equivalent joint torques or we should directly control the contact forces. Here, we have chosen the direct approach. To this end, we use a robust motion control approach introduced in [22] which relies on the robust tracking of contact forces in face of rigid body model mismatch, actuator dynamics, delays, contact surface stiffness, and unobserved ground profiles.
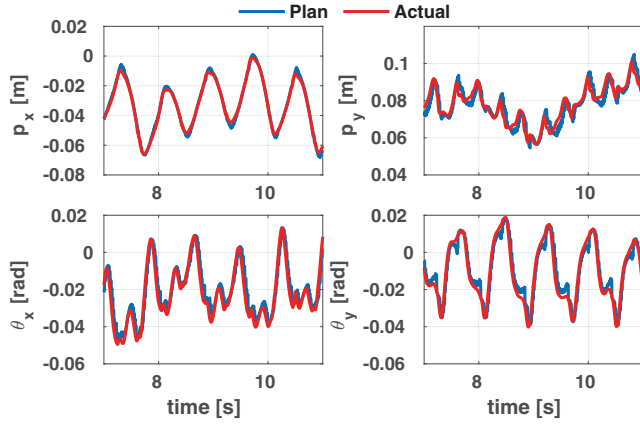
Fig. 3. CoM pose plots for HyQ trotting in-place. The blue lines are the planned references and the red lines are the actual states. The robot nicely maintains its pose and the position and orientation angles are varying in a small range. The periodic pattern of motion is clearly visible in these plots.
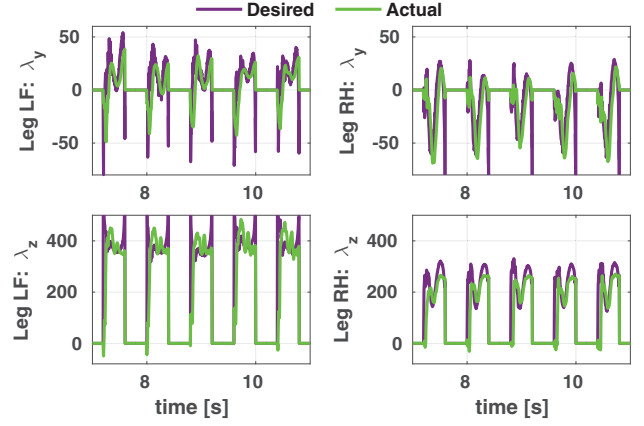

Fig. 4. Contact forces for the trotting in-place task. The planned forces are in purple and the estimated forces are in green. The contact forces are relatively smooth during support phases, which facilitates hardware experiments. The contact forces also nicely reflect the stepping pattern.

Fig. 2 demonstrates an overview of the motion controller introduced in [22]. In order to manipulate the contact force directly, this structure uses an especial system decomposition which allows to control the swing leg trajectories and contact forces independently. This motion control structure consists of three main components: contact force controller, swing leg controller, and CoM controller. The contact force controller uses a robust Internal Model Controller (IMC) to track the planned contact forces. The swing leg controller uses an inverse dynamics scheme plus a PD controller to track the desired end-effector motion in the Cartesian space. Finally, the CoM controller is responsible for tracking desired, feasible motions of CoM.

In theory, if the MPC loop runs fast enough (e.g. 250 Hz), its re-planing scheme should be able to stabilize the CoM. However, in practice our MPC loop run at 60 Hz which is slower than it can be used as a stabilizing controller. In this case a CoM tracking controller is employed in order to deal with the discrepancies between the model and hardware. Our CoM controller uses a PD feedback controller on the planned trajectories which provides correction to the CoM acceleration which is later mapped to an equivalent correction to the MPC planned contact forces.

IV. RESULTS

In this section, we show how FASTSLQ-MPC can be applied for planning of the periodic gait patterns on real hardware. We also demonstrate the robustness and re-planning capabilities of the approach by adding significant disturbances during execution. Finally, we benchmark the run-time speed of our FASTSLQ implementation.

A. Motion Planning for HyQ through FASTSLQ-MPC

In order to assess the performance of our MPC algorithm, we have performed a series of experiments on HyQ. All of the experiments presented in this section are implemented on hardware as well as two different physics engines namely SL and Gazebo. The performance of the robot was consistent across different simulators and was comparable to the hardware results. Due to the limited space, here, we only focus on

the hardware results. However, in the attached video[2], more results are presented in both SL and Gazebo simulators.

Three different experiments are presented in this section: "trotting in-place task", "go-to task", and "disturbance rejection task". Here, we mainly focus on the trotting gait. We also assume that the duration of each phase of motion is 400 [ms] regardless of the given task. We use similar cost functions for all of the experiments. The cost function in each phase of motion (each subsystem) has a simple quadratic form.

$$J_i = \frac{1}{2} \left(\mathbf{x}(t_{i+1}) - \mathbf{x}_d(t_{i+1}) \right)^\top \mathbf{Q}_{f,i} \left(\mathbf{x}(t_{i+1}) - \mathbf{x}_d(t_{i+1}) \right)$$
$$+ \int_{t_i}^{t_{i+1}} \frac{1}{2} \mathbf{x}(t)^\top \mathbf{Q}_i \mathbf{x}(t) + \frac{1}{2} \mathbf{u}(t)^\top \mathbf{R}_i \mathbf{u}(t) \, dt$$

where $\mathbf{Q}_{f,i}$, $\mathbf{Q}_i$, and $\mathbf{R}_i$ are constant, diagonal matrices of appropriate dimensions. $\mathbf{x}_d(t_{i+1})$ defines the desired goal state which can be used by user to command the robot to move around. This variable is constant for the in-place trotting task and it is set to the commanded goal state for the go-to task. For all of the results presented in this section, we use 2 phases ahead setting for the MPC time horizon. In this case, the time horizon varies from 0.8 [s] to 1.2 [s] (refer to the discussion in II-D). Using the four independent threads of our processor, the MPC loop runs at about 60 Hz. In the following, we discuss our results in details.

a) Trotting in-place task: In this task, HyQ trots in-place and tries to maintain its pose. Fig 3 shows the xy position of CoM as well as the roll and the pitch angles of the base. The blue lines are the planned references and the red lines are the actual states. As you see, the references are nicely tracked and the position and orientation angles are varying in a small range. The periodic pattern of motion is clearly visible in these plots.

Fig 4 shows the yz-direction contact forces at the same time period for two opposing legs namely left-front, LF, leg and right-hind, RH, leg. The planned forces are in purple and the estimated forces are in green. The contact forces nicely visualize the stepping sequence where the discontinuities in the contact forces concur with a moment of touch-down or
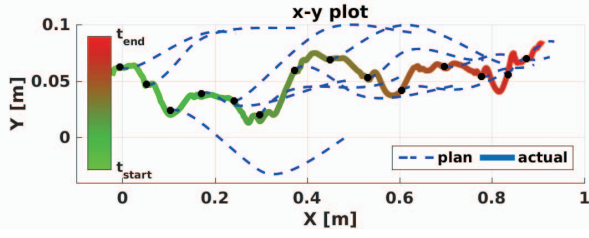
---

[2]https://youtu.be/EYGVmcd9uds

Fig. 5. Overhead plot of continuously recomputed MPC paths for CoM (blue dashed line) and the executed path (green to red gradient) of a go-to task on HyQ. The time horizon is adapted online based on the goal distance. The 1 [m] trotting forward command is received at t_{start} and after s single update of the MPC optimizer (under 20 [ms]) the plan for the next two steps are calculated and send it to robot. The computed plans are down-sampled and the plans are truncated to half of their total length.

lift-off. Due to the imbalance of the robot's weight, the force profiles of the two legs have different patterns.

b) Go-to task: In this task, after the robot starts trotting, we command it to move 1 [m] ahead (*x* direction). The time horizon in which the task should be completed is calculated based on a heuristic. For a motion with a predefined stepping time, the problem of estimating time horizon is equivalent to defining the number of steps for which HyQ requires to reach to the goal position. To this end, we assume a virtual average stride length and based on the goal position displacement, we calculate the number of steps. In this experiment, we choose average stride length of 35 [cm]. We have tried different average stride lengths on simulation and we ultimately choose 35 [cm]. However in general with the higher values, we can observe more dynamic motions.

Fig. 5 shows the overhead view of this motion in *xy* plane using a gradient color scheme which reflects the time evolution of the motion. A subsampled set of computed plans is also demonstrated with dashed blue lines. Here, we have only plotted the first half of the plans. This graph demonstrates that the realized CoM position and the plans are smooth and the MPC plans try to guide the robot toward the final goal position.

c) Disturbance rejection task: In this task, we command HyQ to trot in-place. For evaluating the disturbance rejection capability of our MPC, We pull/push the robot sidewise, i.e. *y* direction (refer to video). Fig. 6 shows the time around one of these unknown disturbances (the gray area), where the disturbance force results is a sudden increase of velocities (in particular the *y* direction). This plot also shows the *xy* contact forces as well as the foot *xy* motion in the global frame for a specific leg (left-hind). The robot uses two different strategies to reject the unknown external force resulted from the pulling. During the stance phase, it manipulates the contact forces in a way to resist the disturbance force while respecting the friction cone. Then in the swing phase, it reacts to the external force through a side-stepping motion. Fig. 7 shows the snapshots of this experiment. In the video of this task, you can see that the robot maintains its balance against relatively strong pulls. The video also demonstrates the performance of the robot in response to the disturbances in the *x* direction.
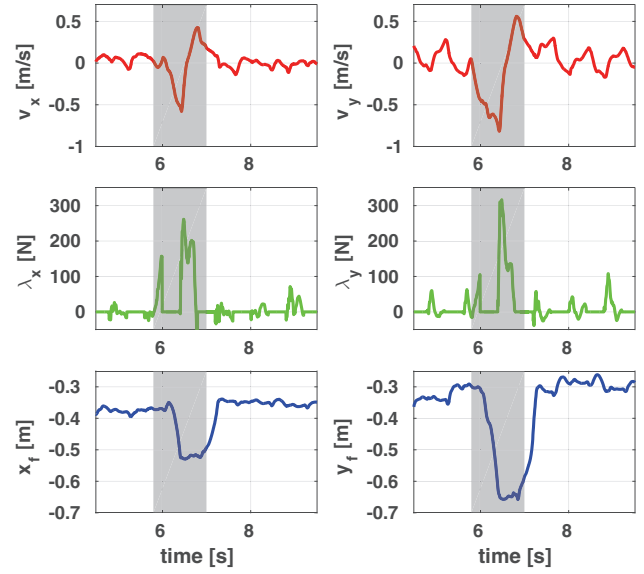

Fig. 6. CoM velocity and the left-hind leg's contact force and position in the *x* and *y* directions. The gray area shows the trajectories at the time around a disturbance, where the disturbance force results is a sudden increase in the CoM velocity. The MPC plan for the contact forces and the foothold position nicely tries to stabilize the CoM.

TABLE I

COMPARISON BETWEEN THE FREQUENCY OF THE MPC ALGORITHM FOR DIFFERENT NUMBER OS SUBSYSTEMS (NUMBER OF PARTITIONS) AND DIFFERENT NUMBER OF THREADS. THE VALUE ARE IN HZ.

Min Num. Subsystems	Num. Threads		
	1	2	4
2	31.9 ± 0.7	40.4 ± 0.3	61.7 ± 1.6
4	17.5 ± 0.2	25.2 ± 0.3	32.3 ± 1.4

B. FASTSLQ *Benchmarking*

Table I shows the frequency of the MPC loop for a various number of threads and number of subsystems (number of phases ahead) for planning. By looking at each column of this table, we notice that as the number of subsystems doubles the frequency reduces almost to half. This is due to the linear computational complexity of SLQ with respect to the optimization time horizon. As discussed in subsection II-B, an important characteristic of FASTSLQ is that it scales more favorably with respect to the time horizon. By comparing the values with the same color in Table I this characteristic becomes evident. We can see that while the time horizon is doubled the frequency does not reduce to half since the FASTSLQ algorithm nicely benefits from the extra computational power (extra threads).

To better understand this, Fig. 8 demonstrates the average CPU time required for the three main operations of SLQ and FASTSLQ. As you see on the left, the computational bottle-neck of SLQ is its backward pass which is almost 65% of the total computation. FASTSLQ, in contrast, leverages fully from the extra processing threads and drastically reduces the computation load of the backward pass (Fig. 8 right graph).

V. CONCLUSION

In this paper, we have introduced a real-time, constrained, nonlinear MPC approach for the motion planning of legged
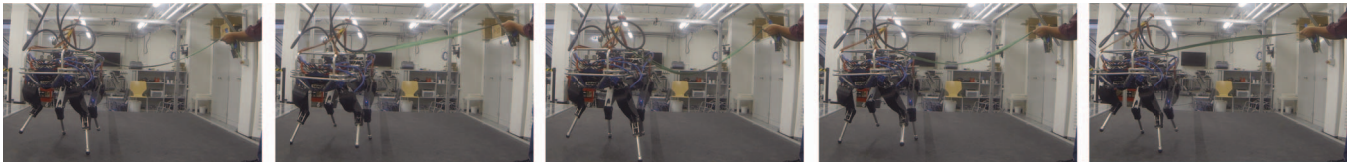
Fig. 7. Disturbance rejection task. We insert a lateral disturbance by strongly pulling the robot from its right during trotting (second snapshot). In order to keep its balance, HyQ demonstrates a side-stepping motion in the third snapshot. Then, it eventually moves back to its initial position. The side-stepping in this experiment is about 40 cm.
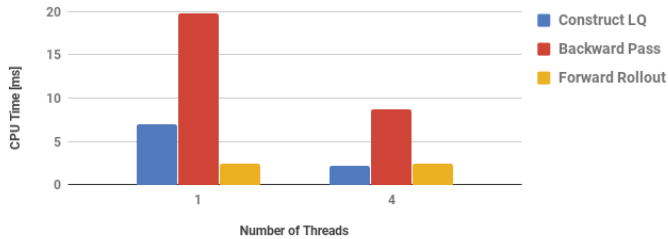

Fig. 8. Comparison of the required CPU time for three main operations of SLQ with 1 thread and FASTSLQ with 4 threads. The values are calculated by averaging over 1500 iterations of the algorithms in the MPC loop.

robots. The proposed approach uses a constrained SLQ algorithm in order to solve the MPC optimization problems. Moreover, we have introduced the FASTSLQ algorithm which allows us to calculate the backward pass of SLQ algorithm in parallel. This drastically reduces the computational complexity of the backward pass which in turn improves the MPC loop frequency.

The FASTSLQ-MPC algorithm introduced in this paper can generate optimized trajectories for the next few phases of the motion within only a few milliseconds. This work shows the first application of whole-body MPC on legged robots for generating periodic gait patterns. We demonstrate that FASTSLQ-MPC can be run at rates that exceed the state of the art by an order of magnitude. For example, in the case of 2 subsystems, the MPC loop can run at about 60 Hz. The performance of our FASTSLQ-MPC motion planner has been tested on both hardware and simulation for generating trotting gait. The capability of the planner for tracking user-defined goal as well as disturbance rejection is nicely shown on hardware and simulation.

REFERENCES

[1] A. Takanishi, H.-o. Lim, M. Tsuda, and I. Kato, "Realization of dynamic biped walking stabilized by trunk motion on a sagittally uneven surface," in *Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on.* IEEE, 1990, pp. 323–330.

[2] T. Buschmann, S. Lohmeier, M. Bachmayer, H. Ulbrich, and F. Pfeiffer, "A collocation method for real-time walking pattern generation," in *2007 7th IEEE-RAS International Conference on Humanoid Robots.* IEEE, 2007, pp. 1–6.

[3] P. Wieber, "Trajectory free linear model predictive control for stable walking in the presence of strong perturbations," in *IEEE-RAS 6th International Conference on Humanoid Robots*, 2006, pp. 137–142.

[4] J. Urata, K. Nshiwaki, Y. Nakanishi, K. Okada, S. Kagami, and M. Inaba, "Online decision of foot placement using singular lq preview regulation," in *IEEE-RAS International Conference on Humanoid Robots*, 2011, pp. 13–18.

[5] R. Wittmann, A. C. Hildebrandt, D. Wahrmann, F. Sygulla, D. Rixen, and T. Buschmann, "Model-based predictive bipedal walking stabilization," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, 2016, pp. 718–724.

[6] S. Feng, X. Xinjilefu, C. G. Atkeson, and J. Kim, "Robust dynamic walking using online foot step optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[7] M. Naveau, M. Kudruss, O. Stasse, C. Kirches, K. Mombaur, and P. Soures, "A reactive walking pattern generator based on nonlinear model predictive control," *IEEE Robotics and Automation Letters*, pp. 10–17, 2017.

[8] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference.*, 2005.

[9] A. Sideris and J. E. Bobrow, "An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems," in *American Control Conference, 2005. Proceedings of the 2005*, 2005.

[10] J. Koenemann, A. D. Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body model-predictive control applied to the hrp-2 humanoid," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 3346–3351.

[11] "OCS2: An open source library for Optimal Control of Switched Systems," https://adrlab.bitbucket.io/ocs2, 2017, [Online; accessed 30-September-2017].

[12] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[13] F. Farshidian, M. Kamgarpour, D. Pardo, and J. Buchli, "Sequential linear quadratic optimal control for nonlinear switched systems," in *International Federation of Automatic Control (IFAC)*, 2017.

[14] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1168–1175.

[15] 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.

[16] H. Chen and F. Allgöwer, "A quasi-infinite horizon nonlinear model predictive control scheme with guaranteed stability." *Automatica*, vol. 34, no. 10, pp. 1205–1217, 1998.

[17] M. Morari and J. H. Lee, "Model predictive control: past, present and future," *Computers & Chemical Engineering*, pp. 667–682, 1999.

[18] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella, and D. G. Caldwell, "Design of hyq–a hydraulically and electrically actuated quadruped robot," *Proceedings of the Institution of Mechanical Engineers, Journal of Systems and Control Engineering*, 2011.

[19] M. Bloesch, M. Hutter, M. A. Hoepflinger, S. Leutenegger, C. Gehring, C. D. Remy, and R. Siegwart, "State estimation for legged robots-consistent fusion of leg kinematics and imu," *Robotics*.

[20] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *IEEE-RAS International Conference on Humanoid Robots*, 2014.

[21] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Robotics and Automation Letters*, 2017.

[22] F. Farshidian, E. Jelavić, A. W. Winkler, and J. Buchli, "Robust whole-body motion control of legged robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.