Adaptive Real-time Nonlinear Model Predictive Motion Control

Michael Neunert, Farbod Farshidian and Jonas Buchli

Abstract— **In this paper we present a framework for real-time, full state feedback, nonlinear model predictive motion control of autonomous robots. The proposed approach uses an iterative optimization algorithm, namely iterative Linear Quadratic Gaussian (iLQG) to solve the underlying nonlinear optimal control problem, simultaneously deriving feedforward and feedback terms. The resulting motion controller is updated online by continuously rerunning the solver in a model predictive control (MPC) setting. An additional optimization loop around the optimal control algorithm allows for a real-time, situation-dependent adaptation of the solver's parameters. This adds the possibility to influence the high level behavior of the system such as adapting the controllers time horizon or cost function. The performance of the proposed approach is validated in simulation and on the ball balancing robot Rezero. Therefore, this work presents one of very few implementations of full-state feedback, nonlinear, model-predictive control for motion control on a real robot. Results show that the framework is able to produce an optimized behavior of the system that is robust to large disturbances. The efficient implementation allows us to run the framework at high frame rate in real-time on standard computer hardware.**

I. INTRODUCTION

In robotics control the goal is to drive a system to a desired state via an optimized trajectory. In classical approaches this task is decomposed into two steps. The first step is finding a feasible, constraint compliant and near optimal trajectory. In the second step, a tracking controller is derived which ensures that the system follows this trajectory in presence of model inaccuracies or disturbances. While this separation simplifies the problem, both elements can have conflicting goals and the combination of both is often suboptimal. Hence, algorithms have been developed that simultaneously design feedforward and feedback control laws that implicitly encode an optimized trajectory and a matching control strategy. We refer to this combined approach as motion control.

As this combined approach uses a single optimization process, it can lead to improved performance, e.g. with respect to smoothness or tracking behavior of the controller. However, stability can usually still only be guaranteed in the vicinity of the predicted trajectory. In case of large disturbances the stability boundaries might be exceeded. But even if the boundaries are not exceeded, disturbances might lead to decreased performance. One approach to handle disturbances is to continuously solve an optimal control problem based on the current state of the system. This approach is known as model predictive control (MPC).

In our approach we apply full-state feedback nonlinear MPC. To solve the optimal control problem of MPC we use iterative Linear Quadratic Gaussian (iLQG) [1]. While it is a local optimization approach, it can be efficiently initialized to speed up convergence. Furthermore, as it only requires first order derivatives of the system dynamics, it can be implemented efficiently. However, other optimizers could be used here as well [2].

One contribution of our work is an additional adaptation loop around the MPC algorithm that can influence the high-level behaviour of the system. In this adaptation layer, the main elements of the MPC controller like the cost function, the time horizon, the initial control guess or the underlying system dynamics can be changed online. This allows for changes in the behavior that are difficult or even impossible to encode statically in a cost function such as context knowledge or high-level sensor feedback (e.g. visual perception). This way, we can transform the finite time horizon optimal control into an adaptive time horizon optimization.

Furthermore, we validate the performance of our approach on the ball balancing robot, Rezero. In contrast to other implementations e.g. [3] or [4], we directly design the control input to the system without an additional tracking loop, i.e. we implement full-state feedback, nonlinear MPC. Thus, this work is one of the very few demonstrations of full-state feedback, nonlinear MPC on hardware requiring high update rates (as is the case with autonomous robots).

While we demonstrate our approach on one specific robot the implementation of the described framework does not make any assumptions on the type of robot or its task. Therefore, it can be applied to various systems. As the framework is general, the user still simply has to provide a general robot description, a cost-function (including its derivatives with respect to state and control), and if desired a task-specific adaptation rule.

II. RELATED WORK

MPC has been used for decades in the control community mostly on systems with slow dynamics such as chemical plants [5]. As MPC is known to be computationally expensive, its application in robotics has been rather limited. However, the increase of computational power in recent times enables the usage in this field. MPC applications in robotics cover a broad spectrum of topics such as manipulators [4] aerial [2], [6] and ground vehicles [7], [8], [9] as well as legged robots [10], [11] naming only a few examples.

Solving MPC problems in real-time can be hard and computationally expensive. This is especially true in high dimensional state space or at relatively high update rates. Hence, in the previously mentioned examples different techniques are used to simplify solving the MPC problem.

Michael Neunert, Farbod Farshidian and Jonas Buchli are with the Agile & Dexterous Robotics Lab at the Institute of Robotics and Intelligent Systems, ETH Zurich, Switzerland. {neunertm, farbodf, buchlij}@ethz.ch

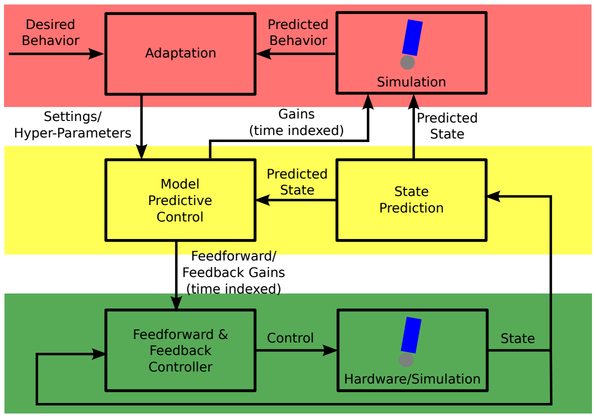

Fig. 1. Structure of our adaptive, real-time motion control framework. A standard full-state feedback controller (green) is complemented with an optimization-based, model predictive controller (yellow). Through an additional adaptation layer (red) higher-level behaviors can be specified.

In [2], [6], [11], [9] and [4] a linear model of the dynamics is assumed. This allows for solving the MPC at very high rates. However, these approaches might have decreased performance on strongly nonlinear systems. Another common simplification is to design a trajectory and add a tracking controller. This idea is used in e.g. [4] and [3]. While a low-level tracking controller can perform well, it is usually not taken into account by the MPC algorithm which might lead to decreased performance or constraint violations.

The approach used in this paper is similar to the one used in [10] and [12]. Both publications illustrate the capabilities of the underlying approach. However, the examples presented in [10] and [12] are in simulation only. As simulations do not perfectly replicate real physics (e.g. unmodelled motor dynamics) and as the latter are sometimes even excessively violated (e.g. no self-collision checking in [10]) it is not clear how these approaches perform on real systems.

III. Real-Time Motion Control Framework

A. Motion Control Problem

In this work, we are trying to solve a closed-loop motion control problem. We assume that the underlying system can be described as a nonlinear, control-affine system

$$\dot{x} = f(x) + g(x)(u + \epsilon) \tag{1}$$

where $f(x)$ and $g(x)$ denote the transition vector and the control gain matrix respectively. We assume that the control input u is perturbed by zero-mean Gaussian noise ϵ. Both $f(x)$ and $g(x)$ are assumed to be continuously differentiable with respect to state x and to control input u.

The goal is to find a linear, time-varying feedback and feedforward controller of the form

$$u(x,t) = \mathcal{K}(t)^T x'(t) \tag{2}$$

where $x'(t) = [1 \quad x(t)]^T$ denotes the augmented state vector such that the control gain matrix $\mathcal{K}(t)$ can contain both feedforward and feedback gains.

B. Description of the Motion Control Framework

In order to solve these motion control problems, we have implemented an adaptive, real-time motion control framework (see Figure 1). In this framework an optimization-based, model predictive controller (yellow background) is used to modify the gains of a standard feedback controller (green background). An additional adaptation layer (red background) allows to specify and change the desired behavior of the system by modifying settings or hyper-parameters (e.g. the time horizon) used in the MPC controller at runtime.

Usually, MPC designs a feedforward control input. Feedback control is then provided by closing the MPC loop. Thus, for highly dynamic systems this loop has to be run at a high frequency. In addition to a feedforward input, our MPC step also designs feedback gains. An inner, linear state feedback loop is then implemented using these gains. As the linear feedback control law allows for computing the control input very efficiently, the inner feedback loop can be run at very high rates. Therefore, the MPC loop is not required to be run at high update rates to ensure good performance. An additional outer adaptation loop can be used for altering parameters of the MPC solver, e.g. the time horizon or the cost function, to describe different tasks. This adaptation can be run at the same or at a lower rate than the MPC loop.

This hierarchical approach allows for rejecting disturbances and for adjusting to changes in the environment according to their time scales. Low amplitude, high frequency disturbances as well as system noise are handled by the feedback controller while larger amplitude, lower frequency disturbances are compensated for by the MPC loop. Slowly varying parameters, e.g. goal states provided from user input, can be adjusted to in the adaptation layer by modifying the hyper-parameters or cost function of the MPC loop.

C. State Prediction

Complex computations during the MPC or adaptation step can lead to an increased delay between the time of the state measurement and the time at which the new controller is computed and available to the feedback loop. By the time the control gains get applied, the state might have diverged from the measured state that the controller was initially designed for, leading to a decrease in performance.

As a countermeasure, we are predicting the state at the time of the estimated end of the controller gain calculation. This predicted state is then used as the initial state for the controller design step. The accuracy of this prediction is highly dependent on the model accuracy and the solver. However, when a good model is available and delays are short, an accurate prediction can be obtained. The time horizon for the state-prediction can be fixed, measured (online or offline) or inferred from a combination of both methods.

In our experiments, a good model of Rezero is available and its efficient implementation allows us to run the simulation in real-time at more than 1 kHz using a fourth-order Runge-Kutta integrator. As an estimate for the time horizon of the prediction step, we assume a fixed delay for communication delays based on the theoretical transmission speed of the involved interfaces. Additionally, we measure the delay introduced by the controller computation online and assume it being constant over two subsequent iterations. Thus, the prediction can adapt to medium-fast varying execution times, which can result from changes in the adaptation layer (e.g. changes in settings, hyper-parameters or the cost function can lead to different runtimes).

D. Adaptation

Through an efficient implementation of iLQG as well as the involved system dynamics and derivatives, we are able to run MPC at a very high rate (see Section V). Thus we can add an adaptation layer above the model predictive controller. This adaptation layer can be used to specify higher level behaviors that cannot be represented in the cost function.

In the current implementation, we are using this adaptation layer to vary the time horizon of the finite horizon optimal control problem to improve the control performance. Furthermore, we modify the cost function by altering the final costs of for the optimization (see Section V for more details).

While not used yet, the existing implementation would allows us to optimize additional settings and hyper-parameters of the optimal control problem such as changing parameters of the system dynamics (e.g. when running a parameter estimation in parallel) or modifying the initial guess for the controller (e.g. using pre-optimized motions). As a forward simulation is available in the adaptation layer, we can directly assess the control performance for the chosen parameters.

E. iLQG Algorithm

iLQG [1] is an iterative nonlinear optimal control algorithm. This method optimizes a time-indexed feedforward plus state feedback control law. The main idea behind this algorithm is to iteratively design a linear quadratic Gaussian regulator (LQG) for the linearized system over the latest estimation of the optimal trajectory. In many cases the costs in the iLQG algorithm converges rapidly to a local minimum which makes this algorithm a suitable choice for online optimization frameworks. Furthermore, the cost function in the iLQG algorithm has a general form which makes the design simpler and more flexible. This function is defined as

$$J = E\left[h\left(x(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(x(t), u(t)\right)\right] \tag{3}$$

where $l(x,u)$ and $h(x)$ are intermediate and terminal costs respectively and x and u are state and control input vectors

As iLQG uses the second order approximation of the cost function, it is considered a local optimization algorithm. Being a local optimizer, this algorithm requires to be initialized from a stable condition. Therefore, in this work we use a previously derived linear quadratic regulator (LQR) [13] as an initial control guess.

F. MPC Framework

MPC is based on iterative, finite-time-horizon optimization of a task cost function under the system dynamics constraints. The iterative scheme of this algorithm allows to replan the control law with respect to the last observed state.

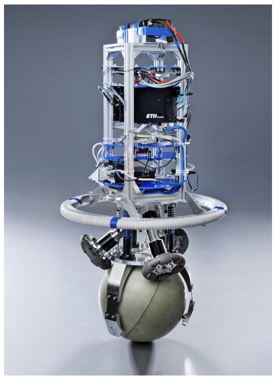

Fig. 2. Photograph of the ball balancing robot Rezero which is used during our experiments for validating the performance of the proposed approach. Due to its static instability and non-minimal phase behavior, it poses interesting challenges in terms of dynamics to the control approach. *Photo: Gerhard Born, Ringier AG*

Therefore, instead of solving the optimal control problem in the whole state space, this method locally designs the control law around the estimated trajectory initialized with the current state. Then by closing the loop over this local optimizer, it approximates the global controller.

In this work, we use the iLQG algorithm as a local optimizer. There are two options for initializing iLQG in an MPC loop. The first method is to use warm starting where iLQG is initialized with the controller calculated during the previous iteration of MPC. The second approach is to initialize iLGQ with a fixed initial guess. As we vary the time horizon of iLQG, the number of time-indexed control gains can be different between subsequent MPC iterations. Therefore, there is no clear way to warm start iLQG using the previous controller. However, iLQG still converges within few iterations when initialized with time-constant LQR gains.

G. Implementation

To allow fast execution suitable for real-time control, the entire framework is implemented in C++. The dynamics and derivatives for Rezero are generated using MATLAB's symbolic and code generation toolboxes. However, the motion control framework can also use forward dynamics generated by the Robotics Code Generator [14] as dynamics and numerical derivatives. Thus, the user only has to specify a text file based robot description, a cost function and its derivatives with respect to state and control to run the framework on most rigid body systems. In the future, we plan to add auto differentiation to replace the numerical derivatives and avoid any manual derivation of the cost function.

The same dynamics are used in several places of the toolbox, namely the state prediction, the adaptation (if required), the MPC (optimal control) algorithm and, if not run on real hardware, the simulation. However, simulation parameters like step length or solver type can be different for each use case. For solving the differential equation of the dynamics, we use ODEint [15] which provides a wide range of fixed and variable step solvers.

For iLQG and MPC we have customized implementation which are proven to work at reasonable speed (see Section V), even though they are not yet optimized for speed and hence still single core implementations.

IV. Hardware

For hardware tests, the ball balancing robot ("ballbot") Rezero (shown in Figure 2) is used. Even though the state space is not as large as e.g. legged robots, Rezero still has very interesting dynamics that make it a very suitable platform for the validation of the framework. First of all, the robot is statically unstable. This means it has to be stabilized at all times and also iterative control algorithms have to be initialized with a stable controller. Furthermore, it is a non-minimum phase system which does not allow to use simple motion planning algorithms. Additionally, the robot is fully torque controlled. This allows us to directly optimize over control torques rather than designing a trajectory. Last but not least, Rezero is a highly dynamic robot which requires fast real-time control loops.

A. *Robot Model*

For the simulation and the derivation of controllers a full nonlinear 3D system model of Rezero has been analytically derived [13]. This model describes the robot as two rigid bodies: the ball and the upper body. Furthermore, it assumes that no slip and no friction occurs. Furthermore, the actuation dynamics are neglected. The robot's state is defined as

$$x = \begin{bmatrix}\theta_x & \dot{\theta}_x & \theta_y & \dot{\theta}_y & \theta_z & \dot{\theta}_z & \varphi_x & \dot{\varphi}_x & \varphi_y & \dot{\varphi}_y\end{bmatrix}^T$$

which consists of the body angles with respect to gravity (θ_x, θ_y, θ_z) and its derivatives ($\dot{\theta}_x$, $\dot{\theta}_y$, $\dot{\theta}_z$) which represent the angular velocities. Furthermore, the state includes the rotational angles of the ball (φ_x, φ_y) as well as their derivatives ($\dot{\varphi}_x$, $\dot{\varphi}_y$). These states represent the ball's position with respect to a reference position (e.g. start position) and the ball's velocity respectively. The control output is represented by u which consists of the three motor torques (τ_1, τ_2, τ_3). Through onboard sensors (an inertial measurement unit and motor encoders) full-state feedback is provided. For the full description of the model see [13].

B. *Computational Hardware*

The low-level feedback loop (green box in Figure 1) is implemented on an ARM Cortex M4 micro-controller and runs in hard-real-time at 200 Hz. The motion planning framework (yellow box in Figure 1) is implemented in soft-real-time on regular PC hardware using an Intel Core i7 processor. Control gains and measured states are exchanged via serial communication between both computers.

V. Experiments

To verify the performance of the proposed approach, we run experiments on the physical hardware of Rezero. Due to the design of feedback controllers iLQG is able to handle modelling errors on Rezero quite well [16]. However, large disturbances still pose a problem since the control gains (both feedforward and feedback) are not recomputed. To assess to what extend MPC can improve the situation, we are performing tests with go-to tasks where the robot is supposed to reach a desired state in a cost-efficient way. In a first test, the goal state is kept constant while in a second test it is varied through user inputs. During both tests, the robot will be disturbed significantly by pushing/dragging it far away from its desired state.

A video summarizing the results can be found at `http://youtu.be/WzKu_IiX2xw`.

A. *Cost Function*

During all experiments, the following cost function is used during the iLQG optimization step

$$l(x,u) = x^T Q x + u^T R u$$
$$h(x) = 10(x - x_d)^T H (x - x_d)$$

where Q, R, H and x_d are defined as:

$$Q = diag(0.05, 0.1, 0.05, 0.1, 10, 0, 0.0025, 0, 0.0025)$$
$$R = diag(1.75, 1.75, 1.75)$$
$$H = diag(1, 2, 1, 2, 1, 2, 10r^2, 4r^2, 10r^2, 4r^2)$$
$$x_d = \begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0 & 3/r & 0 & 1/r & 0\end{bmatrix}^T$$

where r denotes Rezero's ball radius. In the final costs, a desired state x_d can be encoded. The intermediate cost limits the control effort and changes in orientation (yaw angle and rate). This cost function is very similar to the on used in our previous work [16] but in this work we additionally penalize the tilt angles, tilt angle velocities and ball velocities in x and y direction to produce less aggressive maneuvers.

B. *Adaptation*

During our experiments we push the robot far away from the goal state. When the robot gets pushed away far from its goal state, a constant finite horizon during optimization would lead to a more aggressive as the robot tries to reach the goal in time. However, with only one contact point to the ground, Rezero's tilt angle (directly proportional to the required acceleration for dynamic stability) is limited. Hence, an aggressive controller can make the system unstable. This is actually the case for Rezero's original LQR controller which becomes unstable for far setpoints (see video attachment). To avoid this issue, one could always choose a large time horizon. However, a large time horizon is computationally more expensive adding additional delay. Furthermore, as the goal state is only enforced through final costs, the robot will try to minimize control effort (the immediate costs) in the beginning of the trajectory. In a task where Rezero has to simply keep its current position, this would lead to larger drift in the early part of the execution.

To avoid this behavior we set the time horizon proportionally to the geometric distance of Rezero to the goal position. This can be seen as an approximation to setting an average velocity as we are fixing the distance to time ratio. To avoid high gains when Rezero is already at the desired position, we clamp the time horizon at a minimum value of 0.5 s. Through this approach we can avoid changing the cost function to achieve lower gains (even though the adaptation would allow to do so). Especially, in this case tuning the more intuitive time horizon seems easier than tuning the artificial cost function.

For the second test, we want to vary the goal state. This is a common task for autonomous robots in dynamic environments. To achieve this, the adaptation layer modifies the final cost that encodes the desired state.

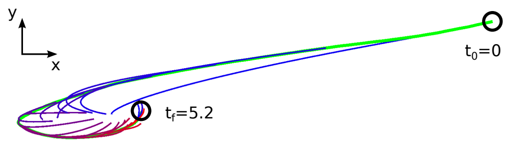

Fig. 3. Overhead plot of continuously recomputed, predicted MPC paths (blue to red gradient) and the executed path (green) of a go-to task on Rezero. The time horizon is adapted online based on the goal distance.

C. Settings

For our hardware tests and the benchmark of our algorithm, we use the settings described in this paragraph. Usually iLQG converges after a maximum of 3 iterations. To have a small safety margin, we run iLQG for 4 iterations. During each iteration, iLQG computes 50 feedforward and feedback control gains per simulated second. This means that each set of control gains is applied for 0.02 s. For the integration, we have chosen a fixed-step fourth order Runge-Kutta implementation as it provides good accuracy at fixed computational costs.

D. Experiment 1: Fixed Go-To Task under Disturbance

In the first experiment, Rezero is given the task to stay at its initial state. This desired state is encoded in the final costs of the cost function. To allow for an accurate tracking of the desired state but still ensure a graceful approach even for large disturbances, we use our adaptation scheme to vary the time horizon of the finite time horizon controller. During the test we perturb the robot manually to various distances to the initial state. These disturbances would exceed the stability margin of a most static feedback controllers (see comparison with LQR in the video attachment) unless they are low-gain and hence show bad tracking performance.

As can be seen in the video attachment, even when perturbed significantly, Rezero is able to gracefully return to its initial position. Due to the MPC structure, Rezero is able to quickly adapt to disturbances during runtime, such that it can be disturbed any time.

In Figure 3 an overhead position plot of one return to the initial position after a perturbation is illustrated. The black circles indicate the starting point (after the perturbation has finished) and the final position. The optimized, predicted trajectories are indicated using a gradient that starts with blue (beginning of the trajectory) and ends in red (end of trajectory). These trajectories are obtained by forward simulating a noiseless model of the system dynamics. In green, the actually executed trajectory is shown. As seen in the plot, the MPC controller gradually adjusts the plan to a circular "swing-in" motion. This behavior seems to be favourable in terms of the cost function as it allows a more graceful approach. Due to the limited time horizon, the initial optimization does not converge this solution but prefers a straight goal approach. Ultimately, the controller converges to a circular balancing trajectory. This trajectory can also be observed in tests with the LQR controller which suggests that this behavior emerges from the system dynamics.

During the approach to the goal state, the time horizon is varied according to the presented adaptation scheme. This

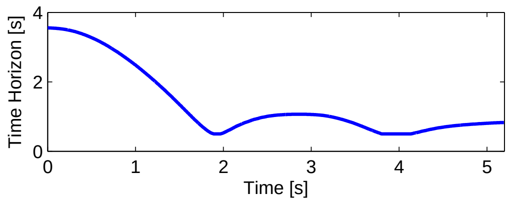

Fig. 4. Scaling of the time horizon for the finite time optimal control algorithm (iLQG). While approaching the goal, it is reduced (limited to a lower boundary of 0.5 s) to ensure a good position tracking.

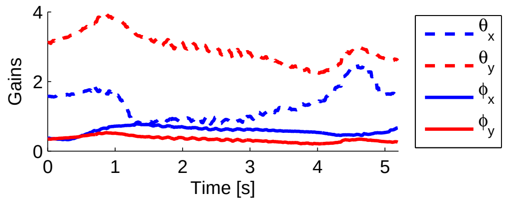

Fig. 5. Evolution of tilt angle and ball rotation angle feedback gains during a go-to task. The slowly varying gains validate the smooth behavior that the MPC structure generates.

scheme increases the time horizon for larger distances and clamps it at a lower boundary. In Figure 4 the time horizon is plotted over time for the same sequence previously used in Figure 3. As simple heuristics for changing the time horizon and the cost functions are used in the adaptation loop, we are varying these parameters at the same rate as running the MPC algorithm (which is at maximum 200 Hz).

Since the feedback control loop on Rezero runs at 200 Hz, the MPC rate is limited to 200 Hz. Our benchmarks (see Section V-F) show, that the MPC loop cannot keep this rate for large prediction time horizons. However, our approach produces gains that are smooth over time. This is illustrated in Figure 5 where we show the gain variation for the feedback gains of the robot's tilt angles as well as its ball angles during execution. Due to this smooth variation, the MPC loop can run at much lower rates than the feedback controller. Note that Figures 4 and Figure 5 are based on the same dataset. This shows that the gains are well behaved also during a smooth variation of the time horizon.

E. Experiment 2: Varying Go-To Task under Disturbance

In the previous experiment we have shown the behavior of our motion control framework under disturbances but for a constant goal point. To show that our approach is also able to handle faster varying changes of the goal state and therefore also the cost function, we conduct a test where the goal state is set by an operator on a joystick. The commands given consist of different kind of variations of the goal point including ramp and step inputs.

The resulting behavior can best be observed in the video attachment. The robot is still able to handle disturbances robustly, even under dynamic changes of the cost function.

F. Implementation Benchmark

To verify the speed of the implementation, we conduct a speed test. For this test, we use the parameters described in Subsection V-C. We fix the time horizon to 3 seconds during the test. Furthermore, we use artificial initial states for iLQG. Rezero's dynamics and the corresponding derivatives are implemented based on the analytical model.

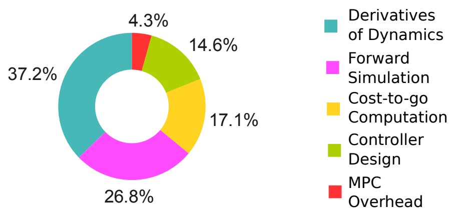

Fig. 6. Runtime fractions of the MPC framework. The largest fractions are forward dynamics and derivatives ($>$ 60%) and the iLQG algorithm itself ($\sim$30%) while the overhead of MPC is low ($\sim$4%).

The benchmark results show that for the given settings MPC can be run at a loop rate of around 75 Hz. As it can be assumed from the implementation, the runtime scales approximately linearly in time horizon, iLQG frequency and iterations. Assuming all other parameters constant, tests verify that a time horizon of 5 seconds leads to an achievable MPC rate of 45 Hz, while at a time horizon of 0.5 seconds we can run the loop well above 300 Hz. While not required for our application, one could change almost any parameter, e.g. iLQG frequency, integrator type or number of iterations to speed up the implementation. Also, our implementation is not yet speed optimized and still runs on a single core.

With the help of a profiler, we have also investigated how much time the implementation requires for the individual steps. Our results, illustrated in Figure 6 show that most time is spent for the calculation of the forward simulations and derivatives used within the iLQG algorithm. The computation of the cost and the control design update step which are also part of the iLQG algorithm account for about 30% of the runtime. With 4% the overhead of MPC, the state prediction and data handling remains low. These results underline the importance of a fast implementation for dynamics and derivatives when using iLQG. While we can auto-generate optimized code based on an analytical model using the Robotics Code Generator [14], we will investigate on automatic differentiation for the latter.

VI. Conclusion

This work presents an online implementation of full state feedback, nonlinear MPC on an autonomous robot. In contrast to most of the MPC implementations on real hardware that use a simplified or linear model of the system, we have focused on a real-time implementation that leverages the full knowledge of the nonlinear system dynamics. This allows us to run MPC on the full state space, optimizing the feedback and feedforward gains. Therefore, we directly optimize over the control law rather than seperately designing a trajectory and a tracking controller.

Furthermore, by introducing an adaptive scheme for the MPC parameters, we can tackle the limitations resulting from the finite-time horizon control problem. This adaptation helps to maintain a short optimization horizon in absence of large disturbances but still allows us to increase the time horizon when needed. Additionally, we use the adaptation layer to modify the cost function which allows us to interactively change the robot's task at runtime.

As experiments on the hardware show, the implemented MPC structure results in a good performance. Even in the presence of large, unmodelled disturbances and a varying task (cost function), the controller remains stable and adjusts its behavior in real-time. The combined design of feedback and feedforward gains leads to a compliant controller that can gently give in but also accurately tracks a goal position.

VII. Future Work

The presented approach has shown good performance on Rezero subject to challenging dynamics. In the future, we will assess the framework's performance on more complex systems like legged systems with increased nonlinear dynamics. The existing implementation already leaves enough headroom for the use on such systems and can be further improved if required. Furthermore, we plan to investigate how well our approach can deal with hybrid dynamics.

Acknowledgement

We gratefully acknowledge Péter Fankhauser and the Autonomous Systems Lab for their support with Rezero.

This research has been funded partially through a Swiss National Science Foundation Professorship award to Jonas Buchli.

References

[1] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *IEEE American Control Conference*, 2005.
[2] P. Bouffard, A. Aswani, and C. Tomlin, "Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results," in *IEEE International Conference on Robotics and Automation*, 2012.
[3] T. Erez, S. Kolev, and E. Todorov, "Receding-horizon online optimization for dexterous object manipulation," preprint available onliine.
[4] R. Ginhoux, J. Gangloff, M. de Mathelin, L. Soler, M. Sanchez, and J. Marescaux, "Beating heart tracking in robotic surgery using 500 hz visual servoing, model predictive control and an adaptive observer," in *IEEE International Conference on Robotics and Automation*, 2004.
[5] M. Morari and J. H Lee, "Model predictive control: past, present and future," *Computers and Chemical Engineering*, vol. 23, no. 4–5, 1999.
[6] K. Alexis, C. Papachristos, G. Nikolakopoulos, and A. Tzes, "Model predictive quadrotor indoor position control," in *Mediterranean Conference on Control Automation*, 2011.
[7] N. Keivan and G. Sibley, "Realtime simulation-in-the-loop control for agile ground vehicles," in *Towards Autonomous Robotic Systems*, ser. Lecture Notes in Computer Science. Springer, 2014.
[8] C. Ostafew, A. Schoellig, and T. Barfoot, "Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments," in *IEEE International Conference on Robotics and Automation*, 2014.
[9] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, 2014.
[10] Y. Tassa, T. Erez, and E. Todorov, "Synthesis and stabilization of complex behaviors through online trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.
[11] H. Diedam, D. Dimitrov, P. B. Wieber, K. Mombaur, and M. Diehl, "Online walking gait generation with adaptive foot positioning through linear model predictive control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008.
[12] T. Erez, Y. Tassa, and E. Todorov, "Infinite-horizon model predictive control for periodic tasks with contacts," *Robotics: Science and Systems VII*, 2012.
[13] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," *Bachelor thesis, ETH Zurich*, 2010.
[14] M. Frigerio, J. Buchli, and D. Caldwell, "Code generation of algebraic quantities for robot controllers," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.
[15] K. Ahnert and M. Mulansky, "Odeint – solving ordinary differential equations in c++," in *Proceedings of the AIP Conference*, 2011.
[16] F. Farshidian, N. Neunert, and J. Buchli, "Learning of closed-loop motion control," 2014, in print: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).