Preprint February 16, 2016 – To appear in Proceedings of the 2016 IEEE/RSJ International Conference on Robotics and Automation (ICRA 2016) - (c) IEEE

Fast Nonlinear Model Predictive Control for Unified Trajectory Optimization and Tracking

Michael Neunert[1], Cédric de Crousaz[1], Fadri Furrer[2], Mina Kamel[2], Farbod Farshidian[1], Roland Siegwart[2], Jonas Buchli[1]

***Abstract*— This paper presents a framework for real-time, full-state feedback, unconstrained, nonlinear model predictive control that combines trajectory optimization and tracking control in a single, unified approach. The proposed method uses an iterative optimal control algorithm, namely Sequential Linear Quadratic (SLQ), in a Model Predictive Control (MPC) setting to solve the underlying nonlinear control problem and simultaneously derive the optimal feedforward and feedback terms. Our customized solver can generate trajectories of multiple seconds within only a few milliseconds. The performance of the approach is validated on two different hardware platforms, an AscTec Firefly hexacopter and the ball balancing robot Rezero. In contrast to similar approaches, we perform experiments that require leveraging the full system dynamics.**

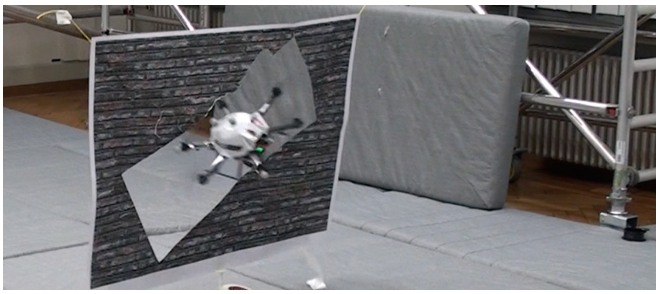


Fig. 1. Flight through a tilted window with an Asctec Firefly hexacopter, with a simultanously optimized trajectory and gains from the proposed SLQ-MPC.

I. Introduction

While recent developments in robotics progress with increasing speed, we still do not see many real world applications. One major challenge for such applications are dynamic environments. Unpredicted changes in its surroundings may require the robot to react quickly and replan its motion within a short time. Despite such sudden changes, the motions should be smooth and adjusted according to the application or current task. Many motion control problems in mobile robotics are solved by addressing trajectory planning and tracking separately. This classical approach consists of a trajectory generation block that is responsible of generating a feasible trajectory and a feedback controller policy that tries to track the generated trajectory. Since the trajectory planner is required to solve a complex planning problem, respecting the system dynamics and constraints, it cannot react very dynamically to changes. The controller on the other hand is often tracking a fixed trajectory with no authority to adjust it. Furthermore, the planner requires knowledge about the controller to precisely predict the system's behavior.

In this work, we are proposing to address this issue with an integrated approach that solves the motion planning and tracking problem by generating optimal feedforward and feedback control gains by solving a single optimal control problem. The unconstrained optimal control problem is solved using a Sequential Linear Quadratic (SLQ) solver [1], which is a very efficient iterative, nonlinear, and locally optimal approach. This kind of approach can also be favorable if the state estimation is likely to be discontinuous and thus requires fast replanning. Such scenarios frequently occur during loop closures in Simultaneous Localization and Mapping (SLAM) applications or after regaining measurements after sensor outages, as it is often the case with Global Positioning System (GPS) measurements in urban environments. Being able to rapidly generate new intermediate trajectories instead of letting the controller act on large state deviations prevents aggressive maneuvers. Therefore, we propose to use SLQ in a Model Predictive Control (MPC) setting by repeatedly solving the optimal control problem at run time.

[1]Agile & Dexterous Robotics Lab, ETH Zurich {neunertm, cedricd, farbodf, buchlij}@ethz.ch

[2]Autonomous Systems Lab, ETH Zurich {fadri.furrer, mina.kamel}@mavt.ethz.ch, rsiegwart@ethz.ch

A. Related Work

Recently, MPC approaches have become more popular in robotics [2]–[6], thanks to increasing computation capabilities and novel efficient algorithms. In [7], [8] a framework for fast Nonlinear Model Predictive Control (NMPC) is presented and used in [9] for fast attitude control of a Unmanned Aerial Vehicle (UAV), while in [10] a code generator for embedded implementation of a linear MPC based on an interior-point solver is shown. Most of these methods are solving a constrained MPC problem, which is computationally complex and thus require a trade-off between time horizon and policy lag.

To overcome this problem, we are solving an unconstrained MPC problem using SLQ. One well-known variant of SLQ called iterative Linear Quadratic Gaussian (iLQG) is presented in [11] and its capabilities are demonstrated in various simulations. Following research [12], [13] has shown the feasibility of such approaches in simulation. In recent work by others [14], and the authors [15], SLQ was successfully demonstrated on hardware. However, in these projects the control is not recomputed during run time, which limits these approaches in terms of disturbance rejection, reaction to unperceived situations and imperfect state esti-

mation. These limitations can be mitigated by resolving the optimal control problem in an MPC manner. First implementations of such an approach were demonstrated in [16], [17]. However, in this work time horizons are short and additional tracking controllers are used instead of directly applying the feedforward and feedback gains to the actuation system. We believe that by allowing SLQ-MPC to directly act on the actuation system, performance can be significantly improved. To the best of our knowledge, the presented work is the first application of SLQ-based MPC applied to highly dynamic, nonlinear mobile robots, directly controlling their actuation systems. In experiments, we demonstrate the capabilities of the approach in tasks that leverage the full system dynamics. Thus, the resulting behaviors cannot be easily reproduced by simple controller and planner schemes.

With our efficient implementation of the SLQ solver, we are able to solve the MPC problem for time horizons of multiple seconds in only a few milliseconds. This performance is unmatched by many other MPC algorithms. Yet, the proposed framework does not only provide a feedforward trajectory but also computes optimal feedback gains which can be used by a higher rate closed loop controller. Therefore, the proposed approach can effectively substitute a tracking controller while simultaneously providing the functionality as a local planner. By initializing the optimal control solver with the infinite horizon Linear Quadratic Regulator (LQR) solution, feedback gains converge to steady-state gains towards the end of the trajectory, which can stabilize the system even if the solver fails to compute an updated trajectory and updated gains in time.

II. Finite Horizon Optimal Control Framework

A. Problem Statement

In this work, we are solving an unconstrained, nonlinear optimal control problem in a nonlinear model predictive control manner. We assume a general nonlinear system

$$\dot{\boldsymbol{x}}(t) = f(\boldsymbol{x}(t), \boldsymbol{u}(t)) \tag{1}$$

where $x(t)$ and $u(t)$ denote the system's state and control input, respectively. $f(x(t), u(t))$ denotes the transition vector. $f(x(t), u(t))$ is assumed to be differentiable with respect to the state $x(t)$ and to the control input $u(t)$. Furthermore, we assume that all states are observable. Our goal is to find a linear, time-varying feedback and feedforward controller of the form

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{u}_{ff}(t) + \mathcal{K}(t)\boldsymbol{x}(t) \tag{2}$$

with the control gain matrix $\mathcal{K}(t)$ and feedforward control action $u_{ff}(t)$. The time-varying controller is found by iteratively solving a finite-horizon optimal control problem that minimizes a cost function

$$J = h\left(\boldsymbol{x}(t_f)\right) + \int_{t=0}^{t_f} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt \tag{3}$$

where $l(\boldsymbol{x}, \boldsymbol{u})$ and $h(\boldsymbol{x})$ are intermediate and terminal cost.

Algorithm 1 SLQ Algorithm

Given
- System dynamics: $\boldsymbol{x}(t+1) = \boldsymbol{f}\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
- Cost function $J = h\left(\boldsymbol{x}(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right)$
- Initial stable control law, $\mathbf{u}(\boldsymbol{x}, t)$

repeat
- t(0) simulate the system dynamics:
  $\tau : \boldsymbol{x}_n(0), \boldsymbol{u}_n(0), \boldsymbol{x}_n(1), \boldsymbol{u}_n(1), \ldots, \boldsymbol{x}_n(t_f-1), \boldsymbol{u}_n(t_f-1), \boldsymbol{x}_n(t_f)$
- Linearize the system dynamics along the trajectory τ:
  $\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
  $\{\boldsymbol{A}(t)\}_0^{t_f-1}$, $\boldsymbol{A}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t),\boldsymbol{u}(t)}$
  $\{\boldsymbol{B}(t)\}_0^{t_f-1}$, $\boldsymbol{B}(t) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t),\boldsymbol{u}(t)}$
- Quadratize cost function along the trajectory τ:
  $\tilde{J} \approx p(t) + \delta\boldsymbol{x}^T(t_f)\boldsymbol{p}(t_f) + \frac{1}{2}\delta\boldsymbol{x}^T(t_f)\boldsymbol{P}(t_f)\delta\boldsymbol{x}(t_f)$
  $+ \sum_{t=0}^{t_f-1} q(t) + \delta\boldsymbol{x}^T\boldsymbol{q}(t) + \delta\boldsymbol{u}^T\boldsymbol{r}(t)$
  $+ \frac{1}{2}\delta\boldsymbol{x}^T\boldsymbol{Q}(t)\delta\boldsymbol{x} + \frac{1}{2}\delta\boldsymbol{u}^T\boldsymbol{R}(t)\delta\boldsymbol{u}$
- Backwards solve the Riccati-like difference equations:
  $\boldsymbol{P}(t) = \boldsymbol{Q}(t) + \boldsymbol{A}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t) +$
  $\boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{K}(t) + \boldsymbol{K}^T(t)\boldsymbol{G} + \boldsymbol{G}^T\boldsymbol{K}(t)$
  $\boldsymbol{p}(t) = \boldsymbol{q} + \boldsymbol{A}^T(t)\boldsymbol{p}(t+1) + \boldsymbol{K}^T(t)\boldsymbol{H}\boldsymbol{l}(t) + \boldsymbol{l}^T(t)\boldsymbol{g} + \boldsymbol{G}^T\boldsymbol{l}(t)$
  $\boldsymbol{H} = \boldsymbol{R}(t) + \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{B}(t)$
  $\boldsymbol{G} = \boldsymbol{B}^T(t)\boldsymbol{P}(t+1)\boldsymbol{A}(t)$
  $\boldsymbol{g} = \boldsymbol{r}(t) + \boldsymbol{B}^T(t)\boldsymbol{p}(t+1)$
  $\boldsymbol{K}(t) = -\boldsymbol{H}^{-1}\boldsymbol{G}$ % feedback update
  $\boldsymbol{l}(t) = -\boldsymbol{H}^{-1}\boldsymbol{g}$ % feedforward increment

Line search
- Initialize $\alpha = 1$

repeat
- Update the control:
  $\mathbf{u}(\boldsymbol{x}, t) = \boldsymbol{u}_n(t) + \alpha\boldsymbol{l}(t) + \boldsymbol{K}(t)\left(\boldsymbol{x}(t) - \boldsymbol{x}_n(t)\right)$
- Forward simulate the system dynamics:
  $\tau : \boldsymbol{x}_n(0), \boldsymbol{u}_n(0), \boldsymbol{x}_n(1), \ldots, \boldsymbol{x}_n(t_f-1), \boldsymbol{u}_n(t_f-1), \boldsymbol{x}_n(t_f)$
- Compute new cost:
  $J = h\left(\boldsymbol{x}(t_f)\right) + \sum_{t=0}^{t_f-1} l\left(\boldsymbol{x}(t), \boldsymbol{u}(t)\right) dt$
- decrease α by a constant α_d:
  $\alpha = \alpha/\alpha_d$

until found lower cost or number of maximum line search steps reached

until maximum number of iterations or converged ($\boldsymbol{l}(t) < l_t$)

B. Overview of the Control Framework

Our control framework consists of two elements. The first element is the optimal control solver that repeatedly solves the finite horizon control problem, designing a feedforward trajectory and time-varying feedback gains. The second element is a closed loop controller which applies these gains. This second, inner control loop can run at a higher rate than the outer solver, which allows the solver to take slightly more time in designing new control trajectories, while the inner loop ensures a high control frequency. 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 are handled by the feedback controller while larger amplitude, lower frequency disturbances can be compensated for by the optimal control loop. In contrast to cascaded control loops, we use a single optimal control algorithm to design the feedforward trajectory and feedback gains. Therefore, feedback gains are not fixed but tuned to match the trajectory.

C. Optimal Control Solver

To solve the unconstrained nonlinear optimal control problem as stated in Eq. (1) and Eq. (3), we use a custom solver implementing SLQ control (see Algorithm 1). SLQ optimizes a time-varying feedforward and feedback controller with

respect to a given cost function by iteratively linearizing the system around a reference trajectory and computing a quadratic approximation of the cost function around the trajectory and then backwards solving the resulting time-varying, linear quadratic optimal control problem. SLQ needs to be provided with a model of the system dynamics as shown in Eq. (1). In case the system is not statically stable, a stabilizing controller needs to be provided. The possibly nonlinear model and the stabilizing controller are then used to generate a reference trajectory. For the following control update step around the trajectory, SLQ requires a linearized model. Such a model can be computed numerically or analytically. While numerical differentiation has been shown to work [14], we are using analytically derived linearizations [18], as they are more accurate and faster to compute.

1) Cost Function Structure: In order to apply our optimal control solver, a cost function in the form of Eq. (3) needs to be provided. Since in SLQ a quadratic approximation of the cost function is computed, convergence can be improved by already choosing a quadratic cost function. Additionally, quadratic costs are frequently used in control and thus researchers have developed an intuition in tuning these. The cost functions used in this work are of the following form

$$J = \bar{x}(t_f)^T H \bar{x}(t_f) + \int_{t=0}^{t_f} \bar{x}(t)^T Q \bar{x}(t) + \bar{u}(t)^T R \bar{u}(t) + W(x,t)\, dt \tag{4}$$

where $\bar{x}(t)$/$\bar{u}(t)$ represent deviations of state/input from a desired state/input. H, Q and R are the final cost, intermediate cost and input cost weighting matrices respectively. To be able to include multiple waypoints in our solver, we add an intermediate waypoint cost $W(x,t)$ defined as

$$W(x,t) = \sum_{n=0}^{N} \hat{x}(t)^T W_p \hat{x}(t) \sqrt{\frac{\rho_p}{2\pi}} \exp\left(-\frac{\rho_p}{2}(t-t_p)^2\right) \tag{5}$$

where N is the number of waypoints, $\hat{x}(t)$ is the deviation from current to waypoint state, W_p is the waypoint cost matrix, t_p the time at which the desired waypoint should be reached, and ρ_p the temporal spread of the waypoint. W_p and t_p can be chosen separately for each waypoint. Even though we include waypoints as soft constraints, results show that our formulation of immediate waypoints is able to enforce a certain state on a trajectory well (see Fig. 6). At the same time, such soft constraints allow the controller to slightly violate these constraint if not dynamically feasible. This relaxes the requirements for a high-level planner to generate perfectly feasible trajectories, which is a strict requirement when using hard constraints. Including waypoints in the cost function has another advantage: the waypoint weighting matrix W allows to configure the importance of individual states which makes tuning more intuitive. Since the weighting matrix W_p does not only influence the feedforward trajectory but the feedback gains as well, SLQ also designs optimal feedback gains that match this importance weighting.

Algorithm 2 SLQ-MPC Algorithm

Given
- System dynamics: $\boldsymbol{x}(t+1) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t))$
- Input cost weights $\boldsymbol{R}$ for SLQ/LQR
- State cost weights $\boldsymbol{Q}_{slq}$ for SLQ
$J = \bar{x}(t_f)^T H \bar{x}(t_f) + \sum_{t=0}^{t_f-1} \bar{x}(t)^T Q_{slq} \bar{x}(t) + \bar{u}(t)^T R \bar{u}(t) + W(x,t)$
- State cost weights for LQR Q_{lqr}

repeat

Adjust Setup
- Acquire current state $x(0)$
- Optional: Predict state after policy lag by forward simulating dynamics with previous controller
$x(0) = x_{t_{lag}}$
- Optional: Update the time horizon
- $t_f = distance(x(0), x(t_f))$

Initialize SLQ with Controller
- Linearize the system dynamics around current state $x(0)$:
$\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
$\boldsymbol{A}(0) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(0),\boldsymbol{u}(0)}$
$\boldsymbol{B}(0) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(0),\boldsymbol{u}(0)}$
- Compute LQR at linearized state
$\boldsymbol{K}(0) = lqr(\boldsymbol{Q}_{lqr}, \boldsymbol{R}, \boldsymbol{A}(0), \boldsymbol{B}(0))$
- Initialize SLQ with LQR around current state
$slq.init(\boldsymbol{K}(0), \boldsymbol{x}_0)$

Optional: Update Cost Function
- Update desired and final state $x_d(t)$, $x(t_f)$
- Update intermediate weights $W(x,t)$
- Linearize the system dynamics around final state $x(t_f)$:
$\delta\boldsymbol{x}(t+1) = \boldsymbol{A}(t)\delta\boldsymbol{x}(t) + \boldsymbol{B}(t)\delta\boldsymbol{u}(t)$
$\boldsymbol{A}(t_f) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}|_{\boldsymbol{x}(t_f),\boldsymbol{u}(t_f)}$
$\boldsymbol{B}(t_f) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}|_{\boldsymbol{x}(t_f),\boldsymbol{u}(t_f)}$
- Solve LQR Ricatti equation at final state
$\boldsymbol{P}(t_f) = ricatti(\boldsymbol{Q}_{lqr}, \boldsymbol{R}, \boldsymbol{A}(t_f), \boldsymbol{B}(t_f))$
- Set final cost of SLQ cost function to Ricatti solution $\boldsymbol{H} = \boldsymbol{P}(t_f)$

Solve using SLQ
- Run SLQ algorithm
- $[\boldsymbol{u}_{ff}(t), \boldsymbol{K}(t), \boldsymbol{x}(t)] = slq.solve(\boldsymbol{f}, \boldsymbol{Q}_{slq}, \boldsymbol{R}, \boldsymbol{H}, \boldsymbol{W}, t_f)$
- Send control to closed-loop controller
$control.apply(\boldsymbol{u}_{ff}(t), \boldsymbol{K}(t))$
- Optional: Record policy lag t_{lag}

until finished

D. Extending SLQ to MPC

In this work, we are using SLQ in an MPC scheme. Algorithm 2 summarizes the proposed approach. The main concept is to run SLQ in an infinite loop, continuously recomputing the control trajectory while adjusting its parameters. The MPC loop is triggered by a new state measurement and is run until convergence.

1) State Prediction: We define policy lag as the time between the initial state for which a control policy was designed and the time this policy gets executed. A long policy lag can have severe consequences on the control performance, especially for complex dynamic systems with a rapidly changing state. To compensate for a potential policy lag, one can estimate or measure the lag and predict the first state where the new policy will be executed. This state can then be used as the initial condition for the policy design.

2) Initialization: SLQ needs to be provided with an initial controller. The convergence speed of SLQ depends on this initial guess. When running an MPC at a high rate, subsequently designed control trajectories usually do not deviate significantly from each other and thus provide a good initialization for the next design step. However, it is not trivial to initialize SLQ with the previous controller if the time horizon is changing, especially when the time horizon increases. As an alternative, SLQ can be initialized

with any stable controller. Since for the presented problems SLQ converges in less than three iterations without warm starting, we initialize it with an LQR controller stabilizing the system around its initial state.

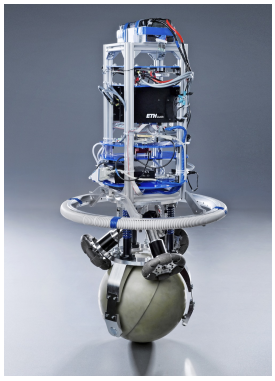
3) Waypoints: As shown in Eq. (5), our waypoints are localized in time. To ensure that a waypoint is reached at a specific time, we shift the waypoint time t_p in Eq. (5) by the time elapsed since the start of the execution of the plan.

4) Final Cost: Manually tuning the final cost H in Eq. (4) is difficult. Thus, we set H to the solution of the infinite horizon LQR Ricatti equation. The input weighting R is chosen to be equal for both SLQ and LQR. Thus, the final gains of each trajectory correspond to the LQR gains.

III. Test Platforms: Ballbot and Hexacopter

Two platforms were used to demonstrate the applicability of the presented framework. The hardware as well as the dynamic models for both systems are described below.

A. Ballbot



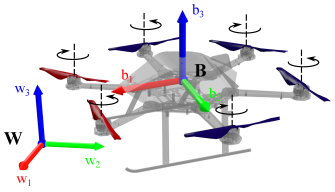


Fig. 2. Visualization of both hardware platforms used during the experiments. Both systems are highly dynamic, unstable systems and thus allow for evaluating the performance of the presented approach. *Photo Rezero: Gerhard Born, Ringier AG*

1) System Description: For our first hardware tests, we use the ball balancing robot ("ballbot") Rezero, shown in Fig. 2. Rezero has very interesting dynamics that render it a suitable platform for the validation of the framework. It is a statically unstable, non-minimum phase system which does not allow for using simple motion planning algorithms. Additionally, the robot is fully torque controlled, allowing for directly optimizing control torques. The controller executing the trajectory runs at 200 Hz on an ARM Cortex M4 while the MPC solver runs on an onboard Intel Core i7 computer.

2) Model Description: A full nonlinear 3D system model of Rezero has been derived analytically [18]. This model describes the robot as two rigid bodies: the ball and the upper body. Furthermore, it assumes that no slip and no rolling friction occurs. Additionally, the actuation dynamics are neglected. The robot's state is defined as $x = \begin{bmatrix}\theta_x & \dot{\theta}_x & \theta_y & \dot{\theta}_y & \theta_z & \dot{\theta}_z & \varphi_x & \dot{\varphi}_x & \varphi_y & \dot{\varphi}_y\end{bmatrix}^T$ containing 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. We denote the control output, the three motor torques (τ_1, τ_2, τ_3), by u. Through onboard sensors (an inertial measurement unit and motor encoders), full-state feedback is provided. For the full model description see [18].

B. Hexacopter

1) System Description: The second platform is a Firefly hexacopter by Ascending Technologies, as depicted in Fig. 2. A Firefly has a Flight Control Unit (FCU) based on an ARM Cortex-M4 micro controller. The optimal control algorithm is computed on an onboard computer with an Intel Core i7 processor, sending velocity commands to the FCU. The current pose estimate obtained from a Vicon tracking system is sent to the computer via a wireless network.

2) Model Description: We are using a widely used dynamic model for Micro Aerial Vehicles (MAVs) as e.g. described in [19]–[21]. We assume that the axis of each propeller is parallel to the z–axis of the body frame $\mathbb{B}$ as shown in Fig. 2. Each propeller i generates a thrust f_i along its axis that is proportional to the square of the propeller's rotational speed n_i. As the motor dynamics are considerably faster compared to those of the vehicle body, they are neglected. The generated thrust and moment from the i-th propeller is given by $f_i = k_n n_i^2$ and $M_i = (-1)^i k_m f_i$, where k_n and k_m are positive constants and M_i is the generated moment. Note that the sign of each moment M_i depends on the rotation direction of the i-th propeller.

Let $\boldsymbol{p}$ be the position and $\mathbf{v}$ the velocity of the center of mass expressed in the inertial frame $\mathbb{W}$, g the acceleration of gravity, T the total thrust generated by the propellers, m the total mass, ω the angular velocity of the vehicle expressed in the body-frame and $\mathbf{J}$ the inertia matrix of the vehicle with respect to the body frame. $\mathbf{R}$ is the rotation matrix from the body to the inertial frame. The system dynamics can be described by the following equations [20], [21]:

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{6a}$$

$$\dot{\mathbf{v}} = \mathbf{R}\begin{bmatrix}0 & 0 & T/m\end{bmatrix}^T + \begin{bmatrix}0 & 0 & -g\end{bmatrix}^T, \tag{6b}$$

$$\dot{\mathbf{R}} = \mathbf{R}\lfloor\boldsymbol{\omega}\times\rfloor, \tag{6c}$$

$$\mathbf{J}\dot{\boldsymbol{\omega}} = \boldsymbol{\omega}\times\mathbf{J}\boldsymbol{\omega} + \mathcal{A}\begin{bmatrix}f_1 & \dots & f_6\end{bmatrix}^T, \tag{6d}$$

where the operator $\lfloor\times\rfloor$ represents skew symmetric calculation, $\mathcal{A}$ is an appropriate matrix that maps the propellers thrust into moments around each axis of the body frame. For a hexacopter the definition of $\mathcal{A}$ can be found in [22].

IV. Evaluation and Experiments

In order to assess the performance of our algorithm, we performed different control tasks on both hardware platforms. Furthermore, we studied the runtime of our optimal control solver to quantify its performance.

A. MPC and Optimal Control Parameters

In our approach, we tried to keep the number of parameters low and made them intuitively tunable. Yet, parameter tuning cannot be avoided, especially given the generality of the

approach. Therefore, we briefly describe how to set these parameters and reason about our choices for them.

1) Cost Function Parameters: The final cost matrix H in Eq. (4) can be tuned to influence the systems approach towards the goal point and the final gains of the trajectory. In the ballbot examples hand-tuned values for H worked well. However, in the hexacopter experiments we are setting H to approximate the infinite time horizon optimal control problem, i.e. the LQR solution centered around the desired goal position. In general, if only diagonal weightings are considered the cost function has about 40 parameters to be tuned. However, considering a pre-tuned LQR controller providing H and R, we only have 24 open parameters. Given the symmetry in the system and not using unintuitive intermediate position gains in Q, we end up with less than 10 general parameters and less than 10 waypoint specific weightings. While the latter are possibly task dependent, we keep these the same throughout all experiments with the exception of an added roll term in Section IV-C. Since all parameters have an intuitive meaning and the solver is insensitive to them, tuning the costs is simpler than expected.

2) Time Horizon: Choosing the time horizon for the optimal control problem is a common issue. Ideally, one would set the horizon based on the minimum time required to reach the final state without exceeding given state and input constraints. In practice however, many high level planners already present the controller with a desired time horizon. In our experiments, we heuristically set the distance to the p-norm between current and goal position which works well.

3) Solver Settings: In most cases, our optimal control solver converges in one to three iterations. To have some safety margin, we set the maximum number of iterations to five. When updating the controller, the solver can apply a line search scheme. However, the solver did not make use of it in any of the experiments.

B. Results of Ballbot Experiments

First, we run a set of experiments on the physical hardware of Rezero. Due to the design of feedback controllers, SLQ is able to handle modelling errors on Rezero quite well [15]. However, large disturbances still pose a problem when 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.

1) Fixed Go-To Task under Disturbance: In the first experiment, Rezero is given the task to stay at its initial state which is encoded in the final term of the cost function. During the test we perturb the robot manually to various distances to the initial state. These disturbances would eventually exceed the stability margin of most static feedback controllers (see comparison with LQR in the video[1] attachment), unless they are low-gain and hence show bad tracking performance. In Fig. 3 an overhead position plot

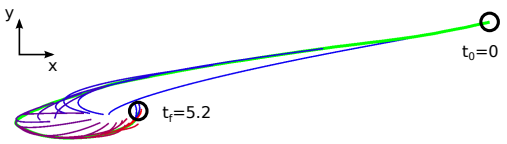


Fig. 3. Overhead plot of continuously recomputed, predicted MPC paths (blue to red gradient) and the executed path (green) of a go-to task on Rezero. The time horizon is adapted online based on the goal distance.

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 to this solution and prefers a straight goal approach, but ultimately, the controller converges to a circular balancing trajectory. This trajectory can also be observed in tests with the LQR, which suggests that this behavior emerges from the system dynamics.

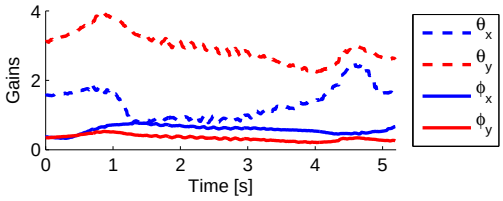


Fig. 4. Evolution of tilt angle and ball rotation angle feedback gains during a go-to task. The slowly varying gains validate the smooth behavior that the MPC structure generates.

2) Varying Go-To Task under Disturbance: 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 through a joystick. The commands given consist of different variations of the goal point including ramp and step inputs. The resulting behavior can best be observed in the video[1] attachment. The robot is able to handle disturbances robustly even under dynamic changes of the cost function. Since the ballbot is a service robot platform, the cost function weights have been tuned for compliant, non-aggressive behavior.

C. Results of Hexacopter Experiments

1) Two Window Task: On the hexacopter we evaluate the performance of our approach with a dynamic re-planning task. In this task we emulate a wall which has two windows. The left one is horizontal and has a size of 1.0 x 0.6 m, whereas the hexacopter's bounding box (with Vicon markers)

[1] https://youtu.be/Y7-1CBqs4x4

is roughly 0.64 x 0.2 m. The window on the right is tilted by 30° on the wall plane. This window is smaller and only spans 0.85 x 0.4 m which prohibits horizontal passes of the hexacopter. We emulate a simple hexacopter level planner that sends two desired intermediate waypoints around the window to be passed, one 0.15 m before and one 0.15 m behind. This way, the controller is prevented from cutting corners. These waypoints are 13 Degrees of Freedom (DoF) waypoints, i.e. amongst time and position, they also include orientation as well as linear and angular velocities. In real

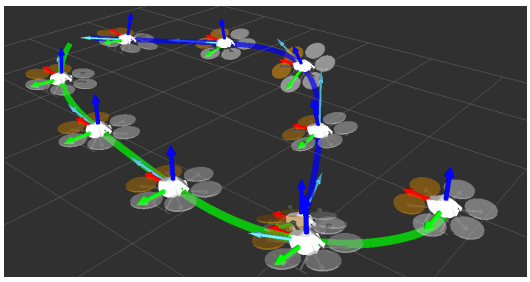


Fig. 5. At the bottom right the hexacopter is shown in its initial state. First, it plans a trajectory through a waypoint on the left, shown in green. During execution (shortly after the second hexacopter on the green trajectory), the intermediate waypoint is manually switched to the right, and a new trajectory in blue, is optimized through this waypoint.

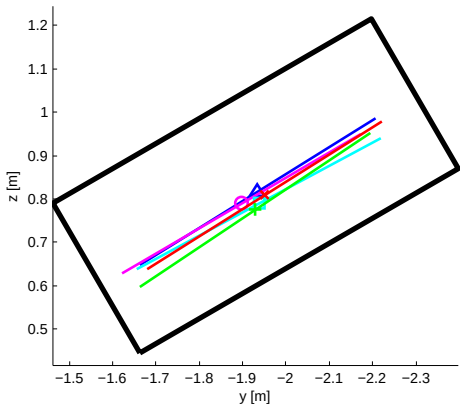


Fig. 6. Five y- and z-positions of the CoG with its rolling angles ϕ of a hexacopter flying through a 30°–tilted window. The length of the five lines shows the diameter of the hexacopter, which is 64 cm.

world applications, a MAV is subject to quick changes to the environment such as suddenly appearing obstacles, requiring quick changes of plans. We simulate such a scenario by switching between both window waypoint sets unpredictably. Figure 5 shows how our control approach solves such task changing requests. The hexacopter first plans a trajectory through the left window. After obtaining the replanning request, the controller lays a trajectory starting at the current state through the waypoints associated to the right, angled window. As benchmarks presented in Section IV-D show, this replanning is completed in less than 25 ms, which demonstrates the potential of the presented approach: The high-level planner is only required to plan waypoints instead of a fully dynamically compliant trajectory. The controller itself is given the freedom to plan an optimal, model-based, dynamically consistent trajectory. During flight the MPC controller is faced with different initial conditions. Thus all trajectories have to be optimized online. Passing the window precisely can be regarded as task critical. Therefore, we evaluate the precision of the window passing both in terms of state and time. Figure 6 shows the position and orientation of the hexacopter at the desired window waypoint time projected onto the wall plane. We measured the following y and z positions and roll angles ϕ:

$$\begin{aligned} \boldsymbol{y_{pos}} &= [-1.866, -1.903, -1.863, -1.848, -1.872]\, m \\ \boldsymbol{z_{pos}} &= [0.816, 0.789, 0.790, 0.809, 0.775]\, m \\ \boldsymbol{\phi} &= [-31.79, -30.40, -28.08, -32.23, -33.70]^\circ \end{aligned}$$

The center of the window is located at $x = 0.0\,\text{m}$ $y = -1.87\,\text{m}$, and $z = 0.83\,\text{m}$, with a roll angle of $-30°$. During the five window passages we measured deviations of the y-position of in the range of $\pm 3\,\text{cm}$, in the z-position we were slightly below the waypoint but never more 6 cm, and the roll-angle ϕ differed less than $\pm 4°$ over the five courses. The two intermediate waypoints were set to different y-values, namely $y = -2.13\,\text{m}$ as we encountered a static offset in preliminary test flights. The z and ϕ values were set to the measured values of the window's elevation and tilting angle. In our experiments we focused on precision, repeatability and consistency of the trajectory execution, even under different initial conditions, and not on absolute accuracy. Hence, we did not incorporate an integral part in the controller which would eliminate the aforementioned offset in y-position.

D. Optimal Control Solver Benchmark

To verify the speed of the implementation, we conducted a runtime speed test. For this test, we used the hexacopter, as the state and control space are larger than for the ballbot model. The dynamics and derivatives, on the other hand, are slightly more complex for the ballbot system model. Hence, we experienced comparable timings on both systems.

During the window passing experiments, we recorded the times needed to solve the MPC problem summarized in Algorithm 2, including both LQR computations for initialization. To reduce measurement noise, we grouped the timings over all approximately 4000 planned trajectories by their time horizon, using bins of 200 ms and plotted their average policy lag in Fig. 7. Since the complexity of Algorithm 1 and Algorithm 2 scales linear with time, the policy lag also scales linearly. These timings indicate that the policy lag for trajectories of up to 4 s are generally below 25 ms. The low policy lag demonstrates that state prediction and a better initial guess is not necessary for the presented applications. However, quick tests have shown that e.g. the SLQ frequency, integrator type, or number of iterations could be reduced to further decrease the policy lag. Additionally, in the experiments and benchmarks, we used our single core implementation of the algorithm. The multi core version also available would further speed up the performance by an order of 50-400 % depending on the CPU model. Thus we assume that SLQ-MPC is also applicable to more complex systems.

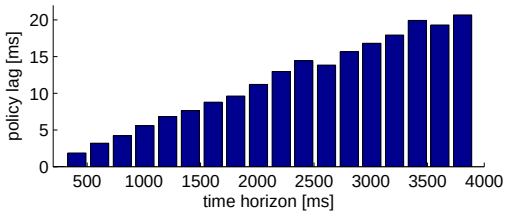


Fig. 7. Policy lag of approximately 4000 optimized trajectories over time horizons of up to 4 s, during flight experiments with a hexacopter.

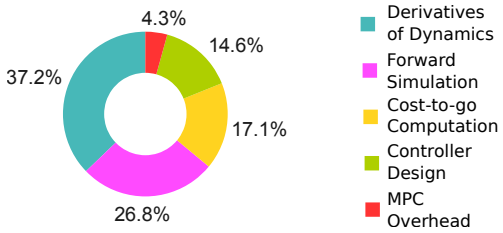


Fig. 8. Runtime fractions of the MPC framework. The largest fractions are forward dynamics and derivatives ($> 60\,\%$) and the iLQG algorithm itself ($\sim 30\,\%$) while the overhead of MPC is low ($\sim 4\,\%$).

With the help of a profiler, we investigated how much time the implementation requires for the individual steps. Our results, illustrated in Fig. 8, show that most time is spent for the calculation of the forward simulations and derivatives used within the SLQ algorithm. The computation of the cost and the control design update step, which are also part of the SLQ algorithm, account for about 30 % of the runtime. The overhead of the MPC, the state prediction, and data handling remains low with 4 %.

V. Conclusion

In this paper we applied SLQ-MPC to solve a nonlinear optimal control problem in a receding horizon fashion for simultaneous trajectory optimization and tracking control. The approach is applied to two different systems, the ballbot Rezero and an AscTec Firefly hexacopter. Hardware experiments demonstrate the capabilities of the approach to plan dynamic trajectories and suited feedback gains for very different tasks. On the ballbot, we are able to design a compliant behavior ideal for crowded environments. On the hexacopter, we demonstrate agile flight maneuvers with accurate tracking and high repeatability. These tasks underline the importance of directly applying the optimized gains to the system and the neccessity for long time horizons. The experiments also exhibit the dynamic replanning capabilities of the approach and display how the algorithm could be combined with a high level planner. Benchmark results show that the MPC problem can be solved in only few milliseconds, even for time horizons of several seconds, indicating that the approach can scale well to more complex systems. In the future we plan to integrate the controller with a high level planner. Also, we will add integral action or model parameter estimation to compensate for steady state errors.

Acknowledgements

This research has been funded through a Swiss National Science Foundation Professorship award to Jonas Buchli and the NCCR Robotics.

References

[1] A. Sideris and J. E. Bobrow, "An efficient sequential linear quadratic algorithm for solving nonlinear optimal," in *Control Problems, Proceeding of the 2005 IEEE Conference on Decision and Control*.

[2] 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.

[3] 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.

[4] K. Alexis, C. Papachristos, G. Nikolakopoulos, and A. Tzes, "Model predictive quadrotor indoor position control," in *Mediterranean Conference on Control Automation*, 2011.

[5] 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.

[6] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1: 43 scale rc cars," *Optimal Control Applications and Methods*, 2014.

[7] B. Houska, H. Ferreau, and M. Diehl, "ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, 2011.

[8] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating Microsecond Solvers for Nonlinear MPC: a Tutorial Using ACADO Integrators," *Optimal Control Applications and Methods*, 2014.

[9] M. Kamel, K. Alexis, M. Achtelik, and R. Siegwart, "Fast nonlinear model predictive control for multicopter attitude tracking on so(3)," in *IEEE Multi-Conference on Systems and Control*, 2015.

[10] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl, "Auto-generated Algorithms for Nonlinear Model Predicitive Control on Long and on Short Horizons," in *Conference on Decision and Control*, 2013.

[11] 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.

[12] 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.

[13] T. Erez, Y. Tassa, and E. Todorov, "Infinite-horizon model predictive control for periodic tasks with contacts," *Robotics: Science and Systems VII*, 2012.

[14] G. Garimella and M. Kobilarov, "Towards model-predictive control for aerial pick-and-place," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, May 2015, pp. 4692–4697.

[15] F. Farshidian, N. Neunert, and J. Buchli, "Learning of closed-loop motion control," in *International Conference on Intelligent Robots and Systems*, 2014.

[16] J. Koenemann, A. Del Prete, Y. Tassa, E. Todorov, O. Stasse, M. Bennewitz, and N. Mansard, "Whole-body Model-Predictive Control applied to the HRP-2 Humanoid," Mar. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01137021

[17] T. Erez, S. Kolev, and E. Todorov, "Receding-horizon online optimization for dexterous object manipulation," preprint available onliine.

[18] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," *Bachelor thesis, ETH Zurich*, 2010.

[19] P. Martin and E. Salaun, "The true role of accelerometer feedback in quadrotor control," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, May 2010, pp. 1623–1629.

[20] S. Bouabdallah and R. Siegwart, "Full control of a quadrotor," in *International Conference on Intelligent robots and systems*, 2007.

[21] T. Lee, M. Leoky, and N. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, Dec 2010, pp. 5420–5425.

[22] M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Inversion based direct position control and trajectory following for micro aerial vehicles," in *International Conference on Intelligent Robots and Systems*, 2013.