2024 European Control Conference (ECC)
June 25-28, 2024. Stockholm, Sweden

Efficient Zero-Order Robust Optimization for Real-Time Model Predictive Control with `acados`

Jonathan Frey[1,2], Yunfan Gao[1,3], Florian Messerer[1], Amon Lahr[4], Melanie Zeilinger[4], Moritz Diehl[1,2]

***Abstract*— Robust and stochastic optimal control problem (OCP) formulations allow a systematic treatment of uncertainty, but are typically associated with a high computational cost. The recently proposed zero-order robust optimization (zoRO) algorithm mitigates the computational cost of uncertainty-aware MPC by propagating the uncertainties separately from the nominal dynamics. This paper details the combination of zoRO with the real-time iteration (RTI) scheme and presents an efficient open-source implementation in `acados`, utilizing `BLASFEO` for the linear algebra operations. In addition to the scaling advantages posed by the zoRO algorithm, the efficient implementation drastically reduces the computational overhead, and, combined with an RTI scheme, enables the use of tube-based MPC for a wider range of applications. The flexibility, usability and effectiveness of the proposed implementation is demonstrated on two examples. On the practical example of a differential drive robot, the proposed implementation results in a tenfold reduction of computation time with respect to the previously available zoRO implementation.**

I. Introduction

A dedicated, explicit treatment of uncertainties allows practical model predictive control (MPC) applications to avoid heuristic and overly conservative safety margins and to harness unused optimality potential. A wide variety of uncertainty-aware OCP formulations exists, such as min-max MPC, scenario-tree MPC and tube-based MPC with tubes of various shapes [1], [2], [3], [4]. This paper focuses on problem formulations where the uncertainties are represented by ellipsoidal tubes in the robust case, and independent distributed noise in the stochastic case. In the case of linear dynamics and constraints, constraint tightenings can be precomputed and such problems can be solved at no additional computational cost compared to their nominal correspondents. In the case of nonlinear dynamics, it is sometimes possible to design constraint tightenings offline by scaling the uncertainty set [5]. However, the state-independent overapproximation of the uncertainty may lead to significant conservatism of the controller, especially when the shape of the actual uncertainty set strongly varies along the prediction horizon.

[1]Department of Microsystems Engineering (IMTEK), University Freiburg, 79110 Freiburg, Germany {name.surname}@imtek.uni-freiburg.de
[2]Department of Mathematics, University Freiburg, Germany
[3]Robert Bosch GmbH, Corporate Research, Stuttgart, Germany yunfan.gao@de.bosch.com
[4]Institute for Dynamic Systems and Control, ETH Zurich, 8092 Zurich, Switzerland

This research was supported by DFG via Research Unit FOR 2401, project 424107692 on Robust MPC and 525018088, by BMWK via 03EI4057A and 03EN3054B, and by the EU via ELO-X 953348.

In this work, we thus consider constraint tightenings based on an approximate uncertainty propagation performed online. Treating the nonlinear form of such problems conventionally with standard OCP solvers requires a state augmentation that is quadratic in the original state dimension. Since the computational cost of most OCP solvers is cubic in the state dimension, this complicates the exact treatment of tube-based OCPs in real-time MPC applications with an overall computational burden growing with the sixth power of the state dimension. As a remedy, the recently proposed zero-order robust optimization (zoRO) algorithm allows one to treat these OCPs at a computational complexity of the order corresponding to the one of a nominal OCP [6], [7], while returning a suboptimal, yet feasible, point at convergence [6]. This algorithmic novelty has brought the application of tube-based MPC with online uncertainty propagation on real systems within reach, as zoRO has been applied successfully on the experimental setup of a differential drive robot [8]. Furthermore, the zoRO algorithm has been extended in [9] to incorporate learning-based model uncertainty, e.g., Gaussian process-based MPC [10] or Bayesian last-layer networks [11]. In addition, the problem formulation allows one to incorporate precomputed linear feedback laws.

Existing zoRO implementations have been written in Python and are limited to their specific use-cases [6], [8], [9]. In contrast to this, we present an efficient, flexible, open-source `C` implementation of zoRO, which can be conveniently used from Python. The high-performance implementation utilizes intermediate results from the `acados` SQP solver and performs the uncertainty propagation as well as backoff computation using `BLASFEO` for the linear algebra operations [12], leading to significant computational speedups compared to the previous implementations.

In the following, we give a brief introduction into the OCP formulation (Section II) and the zoRO algorithm (Section III), before presenting the main features of the high-performance implementation (Section IV). Finally, Section V presents numerical experiments and Section VI concludes the paper.

Notation. This paper uses the following Notation The identity matrix of dimension n is written as $\mathbb{1}_n$. For a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$, the gradient symbol denotes the transposed of the Jacobian $\nabla f = \frac{\partial f}{\partial x}^\top$. The concatenation of vectors $x \in \mathbb{R}^n, y \in \mathbb{R}^m$ is denoted as $(x, y) = [x^\top, y^\top]^\top$.

II. Robust & Stochastic Optimal Control

Let us regard optimal control problems (OCPs) of the form

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}\\ P_0,\dots,P_N}} \quad \sum_{k=0}^{N-1} l(u_k, x_k) + M(x_N) \tag{1a}$$

$$\text{s.t.} \quad x_0 = \bar{x}_0, \tag{1b}$$

$$P_0 = \bar{P}_0, \tag{1c}$$

$$x_{k+1} = \psi_k(x_k, u_k, 0), \tag{1d}$$

$$P_{k+1} = \Phi_k(x_k, u_k, P_k), \tag{1e}$$

$$0 \geq h_k(x_k, u_k) + \beta_k(x_k, u_k, P_k), \tag{1f}$$

$$0 \geq h_N(x_N) + \beta_N(x_N, P_N), \tag{1g}$$

$$\text{with} \quad k = 0, \dots, N-1$$

where $x_k \in \mathbb{R}^{n_x}$ are the state variables for $k = 0, \dots, N$, and $u_k \in \mathbb{R}^{n_u}$ are the controls for $k = 0, \dots, N-1$. The nonlinear, discrete-time system dynamics are given by $\psi_k : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_w} \to \mathbb{R}^{n_x}$ and additionally depend on an uncertain variable $w_k \in \mathbb{R}^{n_w}$. The resulting uncertain trajectory is represented by the nominal trajectory $(x_0, u_0, \dots, u_{N-1}, x_N)$ and the matrices $P_0, \dots, P_N \in \mathbb{R}^{n_x \times n_x}$. A simulation with zero noise characterizes the nominal trajectory (1d); equation (1e) describes the uncertainty dynamics. Here, the positive (semi)definite matrices P_k represent uncertainty in state space and have the meaning of covariance matrices or ellipsoidal shape matrices.

This formulation covers uncertainty-aware NMPC for both a stochastic and robust setting. In the stochastic setting, each w_k follows an independent normal distribution with zero mean and covariance W_k, e.g. $w_k \sim \mathcal{N}(0, W_k)$. Here, x_k and P_k parameterize mean and variance of a stochastic state χ_k, usually approximated as $\chi_k \sim \mathcal{N}(x_k, P_k)$, which approximates the exact state distribution over the horizon for fixed controls.

In the robust setting, the assumption is that the noise variables are contained within an ellipsoidal set

$$(w_0, \dots, w_{N-1}) \in \mathcal{E}\left(0, \operatorname{blkdiag}(W_0, \dots, W_{N-1})\right). \tag{2}$$

Here, $\mathcal{E}(q, Q) := \{\, x \in \mathbb{R}^{n_q} \mid (x-q)^\top Q^{-1}(x-q) \leq 1 \,\}$ denotes an ellipsoid with center $q \in \mathbb{R}^{n_q}$ and shape described by a positive definite $Q \in \mathbb{R}^{n_q \times n_q}$. This results in an ellipsoidal tube such that $\chi_k \in \mathcal{E}(x_k, P_k)$.

The nominal initial state is $\bar{x}_0$ and has uncertainty described by $\bar{P}_0$, e.g. the mean and covariance of the current state estimate in the stochastic setting. The inequality constraints (1f) and (1g) consist of the nominal term $h_k(\cdot)$ and an additional backoff $\beta_k(\cdot)$ to account for the uncertainty, representing chance constraints in the stochastic setting and worst-case constraints in the robust setting [13].

The uncertainty propagation can be written as

$$\begin{aligned}&\Phi_k(P_k, x_k, u_k) \\ &\quad = (A_k + B_k K_k) P_k (A_k + B_k K_k)^\top + G_k W_k G_k^\top,\end{aligned} \tag{3}$$

where $A_k := \frac{\partial \psi_k}{\partial x}(x_k, u_k, 0)$, and $B_k := \frac{\partial \psi_k}{\partial u}(x_k, u_k, 0)$. The matrices $G_k = \frac{\partial \psi_k}{\partial w}(x_k, u_k, 0)$ are constant in case of additive noise. Additionally, it is possible to include precomputed linear feedback gains $K_k \in \mathbb{R}^{n_u \times n_x}$ to reduce conservatism [1].

The backoff term for a constraint component $h_{k,i}(\cdot)$ of $h_k \colon \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_{h,k}}$ for $i \in \{1, \dots, n_{h,k}\}$ is given by

$$\begin{aligned}&\beta_{k,i}(x_k, u_k, P_k) := \\ &\quad \gamma \sqrt{\nabla h_{k,i}(x_k, u_k)^\top \begin{bmatrix} \mathbb{1}_{n_x} \\ K \end{bmatrix} P_k \begin{bmatrix} \mathbb{1}_{n_x} \\ K \end{bmatrix}^\top \nabla h_{k,i}(x_k, u_k)}.\end{aligned} \tag{4}$$

In the robust setting, the backoff factor γ equals 1, while in the stochastic setting, γ is the number of standard deviations that the nominal trajectory should maintain a distance from the nominal bounds. Depending on the interpretation of x_k, P_k as a normal distribution or the first two moments of a general distribution, the choice of γ corresponds to different probabilities of constraint satisfaction, which can be derived via Chebyshev's inequality or the inverse normal cumulative density function, c.f. [14].

III. Zero-Order Robust Optimization (zoRO)

The key idea of zoRO is to eliminate the uncertainty matrices from the OCP and solve reduced subproblems in the nominal variables only,

$$\min_{\substack{x_0,\dots,x_N,\\ u_0,\dots,u_{N-1}}} \quad \sum_{k=0}^{N-1} l(u_k, x_k) + M(x_N) \tag{5a}$$

$$\text{s.t.} \quad x_0 = \bar{x}_0, \tag{5b}$$

$$x_{k+1} = \psi_k(x_k, u_k, 0), \tag{5c}$$

$$0 \geq h_k(x_k, u_k) + \hat{\beta}_k, \tag{5d}$$

$$0 \geq h_N(x_N) + \hat{\beta}_N, \tag{5e}$$

$$\text{with} \quad k = 0, \dots, N-1,$$

where the values $\hat{\beta}_k$ approximate the backoff terms $\beta_k(x_k, u_k, P_k)$. For a detailed derivation, we refer the interested reader to [6], [7].

The zoRO algorithm alternates the following two steps:

1) approximate the backoff terms using the current guess of the nominal trajectory, by performing the uncertainty propagation (3) and the backoff computation (4)
2) solve subproblem (5) approximately

Note that the accuracy up to which the subproblems are solved is an implementation choice and depends on the solver used to tackle the subproblems (5). For example, the subproblems might be solved to convergence using an interior point method, or with a limited number of SQP iterations, c.f. Section IV-B.

A. Convergence properties

Comparing the solution of original problem (1) with a Newton-type optimization method and the alternation performed by zoRO, it can be seen that, while in the original problem the uncertainty matrices correspond to optimization variables, in the reduced subproblem they are inserted as fixed parameters. Thus, it is possible to interpret the zoRO algorithm as an inexact Newton-type method, which employs

a tailored Jacobian approximation at each iteration, namely, by neglecting the sensitivities of the uncertainty matrices with respect to the state and input variables [6], [7].

By standard arguments for Newton-type optimization, see e.g. [15], the zoRO algorithm hence returns a suboptimal, yet feasible, solution of the original problem.

Moreover, due to the structure of the tailored Jacobian approximation, the approximation error scales with the magnitude of the uncertainty, leading to favorable convergence properties for sufficiently small level of uncertainty under the assumption of strong regularity, shown in [6]. First, the incurred suboptimality approaches zero, i.e., the zoRO algorithm recovers the optimal solution to (1) in the limit for vanishing uncertainties. Second, for sufficiently small uncertainties, zoRO converges linearly provided that the unmodified SQP algorithm converges. While the results in [6] hold for the case of constant additive uncertainties W, the second convergence result has been extended to the case of state- and input-dependent uncertainties $W(y)$ in [9].

Note that it is possible to compensate for the suboptimality that the tailored Jacobian approximation incurs by means of an adjoint correction [15], [16], as done in [7], at the additional expense of computing the neglected sensitivities in the direction of the corresponding Lagrange multipliers with the backwards-mode of automatic differentiation. In this paper, we do not address this variant due to the higher computational cost and the focus on real-time feasibility.

B. Remark on Nonlinear Constraints

Previous works have used the term zoRO for two slightly different variants of the algorithm. In [8], the NLP with fixed backoff terms (5) is solved with an SQP algorithm. On the other hand, [6], [7], [9] directly take the SQP perspective and suggest solving quadratic subproblems with the constraints

$$\begin{aligned} 0 \geq\ & h_{k,i}(\bar{y}_k) + \beta_{k,i}(\bar{y}_k, \bar{P}_k) + \nabla_y h_{k,i}(\bar{y}_k)\Delta y_k \\ & + \nabla_y \beta_{k,i}(\bar{y}_k, \bar{P}_k)^\top \Delta y_k + \nabla_P \beta_{k,i}(\bar{y}_k, \bar{P}_k)^\top \Delta P_k \end{aligned} \tag{6}$$

see [6, Eq. (14)], where $\bar{y}_k = (\bar{x}_k, \bar{u}_k)$ and $\bar{P}_k$ is the current linearization point.

Thus, compared with [6], [7], [9], performing a single SQP iteration on (5) corresponds to additionally neglecting the derivative of the backoff with respect to the nominal variables $\nabla_y \beta_{k,i}$. Further, in (5), the backoff $\beta_{k,i}$ is evaluated at $\bar{P}_k^+$, instead of linearizing $h_{k,i}$ with respect to P_k, as in (6).

In order to compute the backoff taking its partial derivatives into account as in (6), on the one hand, the term $\nabla_P \beta_{k,i} \Delta P_k$ has to be precomputed and added to $\beta_{k,i}$. Thereby, the difference ΔP_k is computed as $\Delta P_k = \bar{P}_k^+ - \bar{P}_k$, where $\bar{P}_k^+$ denotes the uncertainty matrix after the performing the uncertainty propagation at $\bar{y}_k$. On the other hand, for the term $\nabla_y \beta_{k,i}(\bar{y}_k, \bar{P}_k)^\top \Delta y_k$ the derivative of the backoff terms $\beta_{k,i}$ with respect to the nominal variables x_k, u_k are needed. Those are

$$\begin{aligned} \nabla_y \beta_{k,i} = \frac{\gamma^2}{\beta_{k,i}} \cdot & \\ \left(\nabla_y h_{k,i}(\cdot,\cdot)^\top \begin{bmatrix} \mathbb{1}_{n_x} \\ K \end{bmatrix} P_k \begin{bmatrix} \mathbb{1}_{n_x} \\ K \end{bmatrix}^\top \nabla_{yy} h_{k,i}(\cdot,\cdot) \right)^\top, & \end{aligned} \tag{7}$$

which vanish for linear constraints $h_{k,i}$. Neglecting those terms results again in an inexact Jacobian.

We note that the convergence results of [6] for the zoRO algorithm still hold under the same assumptions when neglecting these additional derivatives. This can be seen by the fact that the additionally neglected parts of the Jacobian also scale with the magnitude of the uncertainty. Thus, they similarly recover the unmodified Jacobian in the limit for vanishing uncertainties and, ultimately, the convergence properties outlined in Section III-A.

For computational efficiency and ease of implementation, the remainder of this work considers the zoRO version that neglects the terms in (7).

IV. `acados` zoRO SQP implementation

In this section, we detail the ingredients for an efficient zoRO implementation with SQP and RTI in `acados` [17].

A. Real-time iterations

The real-time iteration (RTI) scheme [18] allows one to split an SQP iteration into a preparation and a feedback phase. The preparation phase is carried out based on the initial guess and evaluates the nonlinear functions and its (approximate) derivatives to form the (approximate) Hessian of the Lagrangian. The Hessian can then be used to prepare the linear algebra operations for the feedback phase. The feedback phase takes the best available estimate of the current state value $\bar{x}_0$, evaluates the remaining parts of the Lagrange gradient, and solves the QP subproblem.

We note that typically, only the component corresponding to the constraint $\bar{x}_0 = x_0$ is evaluated in this phase [18]. However, to allow updating all constraint bounds, such as the backoffs in the zoRO algorithm, the `acados` RTI implementation was adapted as follows: It makes all constraint evaluations available in the preparation phase, and just subtract its bounds in the feedback phase, to complete the computation of the Lagrange gradient. The only additional computational cost for this is performing additions corresponding to the number of constraints in the feedback instead of the preparation phase.

B. zoRO with SQP & RTI

When using an SQP solver, as is the focus in this paper, a common choice is to update the backoffs after each SQP iteration. For an efficient implementation of zoRO-SQP, the current linearization of the dynamics and constraints, namely A_k, B_k and $\nabla h_{k,i}(x_k, u_k)$ can be taken directly from the memory of the solver. The backoffs can then be inserted into the subproblem by adapting the bounds of the constraints.

More specifically, when using the RTI variant, one can compute the backoff terms as part of the preparation phase,

i.e. first perform the preparation step of (5) then update the backoff terms. Once the new state estimate is available, use $\bar{x}_0$ to perform the feedback phase on the QP approximating the subproblem (5) and deploy the solution. Note that, when employing the RTI variant of zoRO, the algorithm is equivalent to performing RTI and using the previous MPC solution to propagate the uncertainties and compute the backoffs. Using the previous MPC solution for backoff computation with reduced computation time has also been performed in [10].

C. Remark on non-tightened constraints

In many practical OCP formulations, it is not desired to tighten all the constraints. Thus, we define the index sets $\mathcal{I}_{k,\text{tight}} \subseteq \{1, \dots, n_{h,k}\}$, which denote the subset of constraints that are tightened in (1), i.e., for all other constraints $h_{k,i}$ with $i \notin \mathcal{I}_{k,\text{tight}}$, the corresponding backoff is always zero.

Constraints that are not supposed to be tightened are for example bounds on slack variables. Additionally, two inequalities can be used to encode an equality constraint, such as for the initial state constraint. Furthermore, inequalities in `acados` are always formulated as two-sided, and if one of the bounds is a place holder, those do not need to be tightened.

D. `acados` template interface

An important aspect of the presented implementation is to specify information on the zoRO algorithm compactly in a predefined format, transfer it to `C` code and couple it with the `acados` solver. This is achieved by utilizing the `acados` template interface workflow, which is briefly described in the following.

The `acados` high-level interfaces to Python and Matlab allow one to conveniently and compactly formulate an OCP and specify solver options. The nonlinear problem functions can be formulated using `CasADi` [19] symbolics which are generated as `C` code with the required derivatives using automatic differentiation. The whole problem description is written to a `json`-file which is then used to render different templates. Most importantly, a `C` file is generated which uses the `acados` shared library to create a solver specific to the problem formulation and loads the nonlinear functions.

E. Generic Custom Updates

In order to allow updating numerical data in the solver efficiently, i.e., without any calling functions outside the `C` stack, we added the option of using a custom `C` update function in between solver SQP solver calls. This custom update function

- can access the solver;
- can allocate its own memory and store a pointer to it in the solver `capsule`, implemented by calling an `_init` function at the end of the creation process of the `acados` OCP solver and a `terminate` function which is responsible for freeing it;
- is compiled together with the problem specific OCP solver;
- can be conveniently used in the prototyping phase, as it is fully integrated in the Python interface and can simply be called using: `AcadosOcpSolver.custom_update()`.

Moreover, the user is allowed to write information specific to their problem into the `json` file and use that to generate the custom update function from a custom template. The `C` template implements the zoRO update and allows the user to specify various options, as detailed in the next section.

We note that this feature can be useful to efficiently implement a variety of other advanced MPC schemes, other than zoRO, which require one to update parameters, constraint bounds, or other numerical data in the OCP solver in between SQP iterations. Some examples include

- the advanced-step RTI variants [20],
- an efficient algorithm for robust MPC with optimal linear feedback [21],
- custom backoff computations based on, e.g., poly- or zonotopic tubes,
- state-dependent modelling, e.g. via splines, without explicit incorporation (and therefore differentiation) in the nonlinear functions of the SQP scheme.

F. Template-based zoRO implementation

The uncertainty propagation and the constraint tightening of the zoRO algorithm is implemented in a template-based `C` function using `BLASFEO` [12] for all the linear algebra operations. The constraint and model linearizations, i.e., $\nabla h_k, A_k, B_k$, are obtained from the `acados` solver, using the `C` interface, while linear constraints and simple bounds on states and controls are handled efficiently.

We added the Python class `ZoroDescription` to conveniently specify the zoRO specific information. Specifically, this contains:

- the initial uncertainty matrix $\bar{P}_0$,
- a constant feedback $K = K_k$, for $k = 0, \dots, N-1$,
- a constant covariance $W = W_k$ for $k = 0, \dots, N-1$,
- a constant sensitivity function $G(x, u) = \frac{\partial \psi}{\partial w}(x, u)$,
- the index sets of constraints to be tightened: $\mathcal{I}_{k,\text{tight}}$, which are assumed to be constant for the intermediate shooting nodes $k = 1, \dots, N-1$, but might be different for the initial $k = 0$ and terminal $k = N$ shooting node,
- the backoff factor γ.

The matrices W_k are often constant, which corresponds to an invariant noise distribution. It is possible to update the W_k matrices outside, based on the current nominal iterates to realize a state-, control-, or time-dependent noise distribution, as e.g. done in [9]. Moreover, the initial uncertainty matrix $\bar{P}_0$ can be conveniently changed during runtime. The implementation could be easily modified to allow for different matrices K_k, W_k at each shooting node and respective updates during runtime.

V. Numerical Experiments

This Section presents numerical results comparing the proposed zoRO implementation, labeled *zoRO-24*, with previously existing implementations that were tailored to the specific examples in [6], [8] and are labeled *zoRO-21*. The experiments have been performed on a Laptop with an Intel i5-8365U CPU, 16 GB of RAM running Ubuntu 22.04. All code to reproduce the results is open-source. All experiments have been conducted with `acados` v.0.2.5.

A. Chain benchmark

The code of the chain benchmark used in [6] has been adapted to contain the new zoRO implementation zoRO-24[1]. Additionally, the benchmark includes a *nominal* controller, i.e., without uncertainties and backoffs, a *naive* robust version, where the uncertainty is implemented by augmenting the state only exploiting symmetry of the uncertainty matrices, and the zoRO implementation *zoRO-21* proposed in [6], which propagates the uncertainties and computes the backoffs in Python.

Figure 1 shows a timing comparison of these variants when varying the number of masses. One can observe that the computation times of both zoRO implementations scale with the same order of complexity. The zoRO-24 implementation is able to bring the computational cost of zoRO much closer to the one of the nominal OCP. For higher state dimensions, the computation time is dominated by other computations, such as the QP solution.

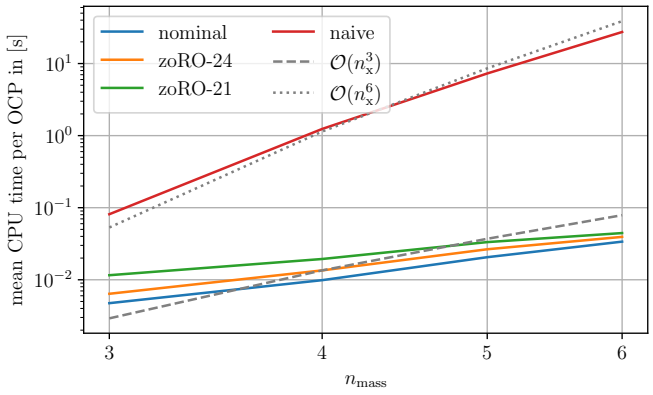

Fig. 1: Mean computation time for solving one OCP for different number of masses n_{mass}.

B. Differential drive robot

In order to illustrate the effectiveness of our implementation on a practically relevant problem, we regard the system of a differential drive robot, which has been considered in [8] and controlled with a zoRO implementation which uses Python for the uncertainty propagation and backoff computation, labeled zoRO-21. In this system, the state vector is given by $x = (p_x, p_y, \theta, v, \omega)^\top$, where p_x, p_y parametrize the 2D-position of the center of the robot, θ is the heading angle, v the forward velocity and ω the angular velocity. The controls are the forward acceleration a and angular acceleration α. The ODE is given by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = (v\cos(\theta), v\sin(\theta), \omega, a, \alpha)^\top. \tag{8}$$

In this OCP, the cost penalizes deviations from a given reference trajectory by linear least-squares. A multiple shooting discretization with 20 shooting intervals and a horizon of 2.0s is used. To arrive at discrete dynamics, an implicit Runge-Kutta integrator of order four with Gauss-Legendre Butcher tableau is chosen, which performs three Newton iterations over which the Jacobian is reused. The constraints consist of bounds on the controls and the states v, ω, and collision avoidance constraints of the form:

$$\left\|(p_x, p_y)^\top - (q_{x,i}, q_{y,i})^\top\right\|_2 \geq r + r_i^{\text{obs}}, \tag{9}$$

where r denotes the radius of the robot, r_i^{obs} the radius of the obstacle i at position $(q_{x,i}, q_{y,i})$ for $i = 1, \ldots, n_{\text{obs}}$.

The index set of constraints to be tightened $\mathcal{I}_{k,\text{tight}}$ is set to contain the upper bounds on v and ω, the lower bound on v and the collision avoidance constraints. The terminal constraint consists of very tight bounds on the velocities,

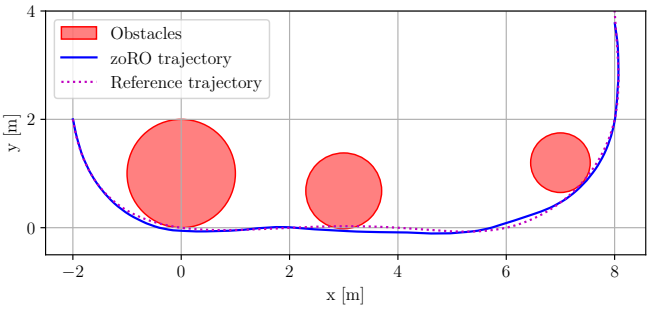

Fig. 2: Closed loop trajectory differential drive robot.

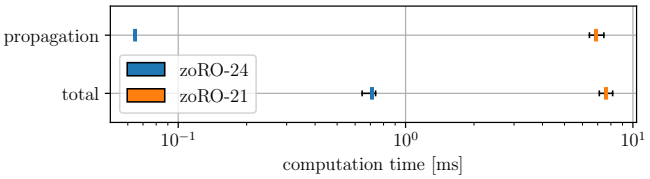

Fig. 3: Computation times of zoRO variants. The whiskers indicate the minimum and maximum value.

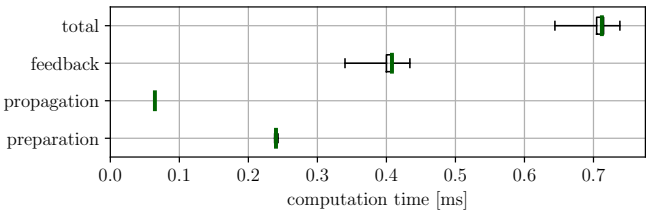

Fig. 4: Computation times for the proposed zoRO implementation zoRO-24 on the differential drive robot. The whiskers indicate the minimum and maximum value.

[1] `https://github.com/FreyJo/zoro-NMPC-2021`

corresponding to the nominal trajectory of the robot not moving at the end of the horizon. Those constraints are not tightened to avoid infeasibility.

The robot is simulated in closed-loop by adding process noise to the output of an RK4 integrator at every simulation instance. The process noise is sampled from a multivariate normal distribution with zero mean and covariance $W = \text{diag}\left(2 \cdot 10^{-6}, 2 \cdot 10^{-6}, 4 \cdot 10^{-6}, 1.5 \cdot 10^{-3}, 7 \cdot 10^{-3}\right)$. The initial uncertainty matrix is $\bar{P}_0 = W$, and $\gamma = 3.0$. The QP subproblems are solved using full condensing and `DAQP` [22]. The code to reproduce the results discussed next is publicly available[2].

Figure 2 visualizes the closed-loop trajectory and a reference trajectory of the robot in a scenario with three obstacles. We observe that the robot is able to avoid the obstacles, even in the described scenario with process noise. Figure 3 visualizes the computation times of both zoRO implementations. All timings are obtained by running the exact same simulation 50 times with the same noise realization and taking the minimum of each execution to remove artifacts. It can be seen that the total computation time of zoRO-24 is roughly ten times lower compared to zoRO-21. The share of computation time for the propagation and backoff computation is $\approx 90\%$ for the zoRO-21 implementation and only $\approx 10\%$ for zoRO-24.

Figure 4 shows how the computation times of different algorithmic components of zoRO-24 on this example. The total computation time consists of the times corresponding to the `acados` *preparation* and *feedback* phases, and the uncertainty propagation and backoff computation, labeled as *propagation*. The preparation step contains the linearization operations of the OCP, such as nonlinear constraints and simulation of the dynamics with sensitivities. The computation time of the feedback phase is dominated by the *QP* solution. The propagation and backoff computation can be carried out either as part of the preparation or the feedback phase. The initial uncertainty matrix $\bar{P}_0$ is constant in this example. However, if it was obtained from a state estimator, it might be worth to use the new initial uncertainty of the initial state, since the corresponding computations can be carried out quickly relative to the QP solution.

VI. Conclusion & Outlook

This paper presented an efficient implementation of the zoRO algorithm for real-time application with an SQP-type solver in `acados` detailing its stochastic and robust interpretation. We believe that this work allows the realization of embedded, tube-based MPC in a wider range of real-world control tasks.

The effectiveness and flexibility of the proposed implementation has been demonstrated on two examples from previous studies. Future work includes the extension of the presented implementation to optionally perform adjoint corrections as proposed in [7] and to optimize over linear feedback matrices [21].

[2]`https://github.com/acados/acados/tree/v0.2.5/examples/acados_python/zoRO_example/diff_drive`

References

[1] D. Mayne, E. Kerrigan, E. J. van Wyk, and P. Falugi, "Tube-based robust nonlinear model predictive control," *International Journal of Robust and Nonlinear Control*, vol. 21, pp. 1341–1353, 2011.

[2] D. Kouzoupis, *Structure-exploiting numerical methods for tree-sparse optimal control problems*. PhD thesis, Univ. of Freiburg, 2019.

[3] D. Telen, M. Vallerio, L. Cabianca, B. Houska, J. Van Impe, and F. Logist, "Approximate robust optimization of nonlinear systems under parametric uncertainty and process noise," vol. 33, pp. 140–154.

[4] J. Gillis and M. Diehl, "A positive definiteness preserving discretization method for nonlinear Lyapunov differential equations," in *Proc. IEEE Conf. Decis. Control (CDC)*, 2013.

[5] J. Köhler, M. A. Müller, and F. Allgöwer, "A novel constraint tightening approach for nonlinear robust model predictive control," *Annual American Control Conference (ACC)*, 2018.

[6] A. Zanelli, J. Frey, F. Messerer, and M. Diehl, "Zero-order robust nonlinear model predictive control with ellipsoidal uncertainty sets," *Proc. the IFAC Conf. Nonlinear Model Predictive Control (NMPC)*, 2021.

[7] X. Feng, S. D. Cairano, and R. Quirynen, "Inexact Adjoint-based SQP Algorithm for Real-Time Stochastic nonlinear MPC," in *Proc. IFAC World Congr.*, 2020.

[8] Y. Gao, F. Messerer, J. Frey, N. van Duijkeren, and M. Diehl, "Collision-free motion planning for mobile robots by zero-order robust optimization-based mpc," in *Proc. Eur. Control Conf. (ECC)*, 2023.

[9] A. Lahr, A. Zanelli, A. Carron, and M. N. Zeilinger, "Zero-order optimization for Gaussian process-based model predictive control," *European Journal of Control*, p. 100862, 2023.

[10] L. Hewing, J. Kabzan, and M. N. Zeilinger, "Cautious model predicitive control using gaussian process regression," *IEEE Transaction on Control Systems Technology*, vol. 28, no. 6, pp. 2736–2743, 2020.

[11] F. Fiedler and S. Lucia, "Model predictive control with neural network system model and Bayesian last layer trust regions," in *2022 IEEE 17th International Conference on Control & Automation (ICCA)*, pp. 141–147.

[12] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "BLASFEO: Basic linear algebra subroutines for embedded optimization," *ACM Trans. Math. Softw.*, vol. 44, no. 4, pp. 42:1–42:30, 2018.

[13] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski, *Robust optimization*. Princeton Univ. Press, 2009.

[14] T. A. N. Heirung, J. A. Paulson, J. O'Leary, and A. Mesbah, "Stochastic model predicitive control – how does it work?," *Comp. Chem. Eng.*, vol. 114, pp. 158–170, 2018.

[15] H. G. Bock, M. Diehl, E. A. Kostina, and J. P. Schlöder, "Constrained optimal feedback control of systems governed by large differential algebraic equations," in *Real-Time and Online PDE-Constrained Optimization*, pp. 3–22, SIAM, 2007.

[16] L. Wirsching, H. G. Bock, and M. Diehl, "Fast NMPC of a chain of masses connected by springs," in *Proc. the IEEE International Conf. Control Applications, Munich*, pp. 591–596, 2006.

[17] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Math. Program. Comput.*, Oct 2021.

[18] M. Diehl, H. G. Bock, and J. P. Schlöder, "Real-time iterations for nonlinear optimal feedback control," in *Proc. IEEE Conf. Decis. Control and Eur. Control Conf. (CDC-ECC)*, pp. 5871–5876, 2005.

[19] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – a software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.

[20] A. Nurkanović, A. Zanelli, S. Albrecht, and M. Diehl, "The Advanced Step Real Time Iteration for NMPC," in *Proc. IEEE Conf. Decis. Control (CDC)*, 2019.

[21] F. Messerer and M. Diehl, "An efficient algorithm for tube-based robust nonlinear optimal control with optimal linear feedback," in *Proc. IEEE Conf. Decis. Control (CDC)*, 2021.

[22] D. Arnstrom, A. Bemporad, and D. Axehill, "A dual active-set solver for embedded quadratic programming using recursive LDL^T updates," *IEEE Trans. Automatic Control*, 2022.