Optimizing at All Scales: Edge (Non)linear Model Predictive Control from MCUs to GPUs

Emre Adabag[†*], Xuyei Bu[†*], Khai Nguyen[‡*], Sam Schoedel[§*],
Anoushka Alavilli[¶], Miloni Atal[†], William Gerard[†], Elakhya Nedumaran[¶],
Zachary Manchester[§], and Brian Plancher[‖]
[†]Fu Foundation School of Applied Science, [‖]Barnard College, Columbia University
[‡]Mechanical Engineering, [§]Robotics Institute, [¶]Electrical & Computer Engineering, Carnegie Mellon University
[*]Equal Contribution, Authors Ordered Alphabetically

***Abstract*—Model predictive control (MPC) is a powerful tool for controlling highly dynamic robotic systems subject to complex constraints. However, MPC, and its underlying (nonlinear) optimization algorithms, are often too computationally demanding to meet real-time rates for robotic platforms, both large and small. These problems are exacerbated by the end of Dennard Scaling and Moore's law, which have led to a utilization wall that limits the performance a single CPU chip can deliver. As such, we now need to look to the field of software performance engineering to co-design our solvers for their target hardware architectures. As such, in our recent works, by leveraging a combination of parallelism, approximation, and structure exploitation, we have enabled and accelerated (nonlinear) trajectory optimization solvers for real-time performance on non-standard computational hardware, ranging from microcontrollers (MCUs) to graphical processing units (GPUs). This has led to real-time MPC onboard an MCU powered 27g quadrotor for dynamic obstacle avoidance, as well as simulated whole-body nonlinear MPC at kHz rates for a GPU powered manipulator for high speed trajectory tracking.**

I. Introduction

Model Predictive Control (MPC) has enabled reactive and dynamic online control for robots while respecting complex control and state constraints such as those encountered during dynamic obstacle avoidance and contact events [56, 10, 28, 21, 17, 50, 53]. However, despite MPC's many successes, its practical application is often hindered by computational limitations, which can necessitate algorithmic simplifications, especially for systems requiring high control rates for safe and effective operation [59, 42, 33, 34].

Compounding this issue, the end of Moore's Law and Dennard Scaling have led to a utilization wall that limits the performance a single CPU chip can deliver [55, 12]. As such, for computationally bounded algorithms, like MPC, computer scientists have had to look beyond the CPU to exploit large-scale parallelism available on alternative computing platforms such as GPUs. Several recent efforts have shown that significant computational benefits are possible by exploiting the natural parallelism in the computation of dynamics and cost functions on GPUs and FPGAs [4, 40, 44, 33, 45, 24, 35, 58]. However, multiple-shooting and consensus approaches to computing trajectory updates at each algorithmic iteration [15, 14, 20, 18, 29, 26] have only seen modest gains when implemented on alternative hardware platforms [42, 43].

At the same time, there has been an explosion of interest in tiny, low-cost robots that can operate in confined spaces, making them a promising solution for applications ranging from emergency search and rescue [30] to routine monitoring and maintenance of infrastructure and equipment [9, 11]. These robots are limited to low-power, resource-constrained microcontrollers (MCUs) for their computation [13, 41]. These microcontrollers feature orders of magnitude less RAM, flash memory, and processor speed compared to the CPUs and GPUs. Consequently, many examples in the literature of intelligent robot behaviors executed on these tiny platforms rely on off-board computers [2, 54, 22, 27, 57, 52, 8].

In this work we show that in order to overcome these challenges at all scales we need to look to the field of software performance engineering and leverage holistic algorithm-hardware co-design. Through this approach we can meet performance targets while respecting real-world computing constraints. We demonstrate the power of this approach through two projects. First we describe MPCGPU (https://github.com/a2r-lab/MPCGPU) [1, 7], a GPU-accelerated, NMPC solver that exploits the structured sparsity and the natural parallelism in direct trajectory optimization through a custom preconditioned conjugate gradient solver at its core. Our experiments show that MPCGPU increases the scalability and real-time performance of NMPC, solving larger problems, at faster rates. Next we describe TinyMPC (https://tinympc.org) [36, 48], an MCU-optimized implementation of convex MPC using the alternating direction method of multipliers (ADMM) algorithm. To the best of the authors' knowledge, TinyMPC is the first MPC solver tailored for execution on these MCUs that has been demonstrated onboard a highly dynamic, compute-limited robotic system, and can support second-order cone constraints.

II. Background

A. Direct Trajectory Optimization

In most MPC formulations, a trajectory optimization problem [5] is solved at each control step. These problems solve a (nonlinear) optimization problem to compute a robot's path through an environment as a series of states $X = \{x_0, \cdots, x_N\}$ and controls $U = \{u_0, \cdots, u_{N-1}\}$ for $x \in \mathbb{R}^n$ and $u \in \mathbb{R}^m$, model the robot as a discrete-time dynamical system,

$$x_{k+1} = f(x_k, u_k, h), \quad x_0 = x_s, \tag{1}$$

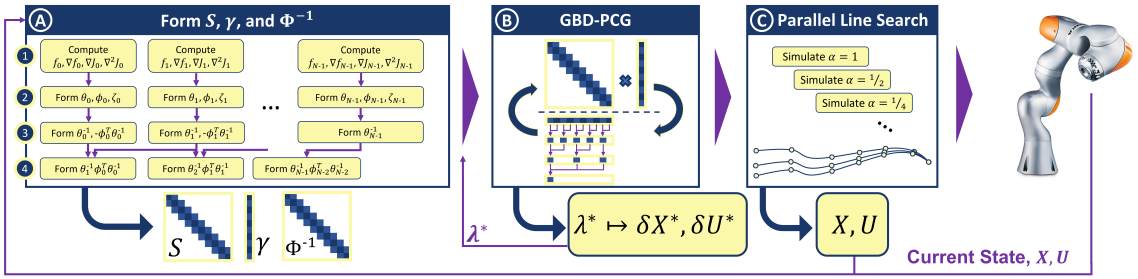


Fig. 1: A high level overview of MPCGPU which: 1) in parallel on the GPU computes S, γ, and Φ^{-1} and stores those values in an optimized dense format, 2) uses our GBD-PCG solver to compute λ^* and reconstructs $\delta X^*, \delta U^*$ through GPU-friendly matrix-vector multiplications and vector reductions, and 3) leverages a parallel line search to compute the final trajectory, X, U. This trajectory is then passed to the (simulated) robot and the current state of the (simulated) robot is measured and fed back into our solver which is run again, warm-started with our last solution.

with a timestep h, minimize an additive cost function,

$$J(X,U) = \ell_f(x_N) + \sum_{k=0}^{N-1} \ell(x_k, u_k), \tag{2}$$

and may also be subject to additional constraints (e.g., torque limits, obstacle avoidance constraints),

$$h(X,U) \leq 0. \tag{3}$$

In the nonlinear case, Taylor approximations of $f(\cdot)$, $h(\cdot)$, and $J(\cdot)$ are taken at each iteration of the algorithm, and regularization is applied, to produce a convex optimization problem. The solution to this convex problem is then often followed by a line search [37].

B. The Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) is a popular and efficient approach for solving convex optimization problems [6]. Given a generic problem of minimizing a convex function $f(x)$ according to a constraint $x \in \mathcal{C}$, we can form the equivalent problem (with the slack z),

$$\begin{aligned} \min_{x} \quad & f(x) + I_{\mathcal{C}}(z) \\ \text{subject to} \quad & x = z. \end{aligned} \tag{4}$$

The augmented Lagrangian of the transformed problem (4) is (with Lagrange multiplier λ and scalar penalty weight ρ):

$$\mathcal{L}_A(x,z,\lambda) = f(x) + I_{\mathcal{C}}(z) + \lambda^\intercal(x-z) + \frac{\rho}{2}||x-z||_2^2. \tag{5}$$

Thus, if we alternate minimization over x and z, we arrive at the three-step ADMM iteration,

$$\text{primal update} : x^+ = \arg\min_x \mathcal{L}_A(x,z,\lambda), \tag{6}$$

$$\text{slack update} : z^+ = \arg\min_z \mathcal{L}_A(x^+,z,\lambda), \tag{7}$$

$$\text{dual update} : \lambda^+ = \lambda + \rho(x^+ - z^+), \tag{8}$$

where the last step is a gradient-ascent update on the Lagrange multiplier [6]. These steps can be iterated until a desired convergence tolerance is achieved.

In the special cases of quadratic programs (QP) or a second-order cone programs (SOCP), each step of the ADMM algorithm becomes very simple to compute: the primal update is the solution to a linear system, and the slack update is a linear or conic projection. ADMM-based QP and SOCP solvers such as OSQP [51] and SCS [39] are state-of-the-art.

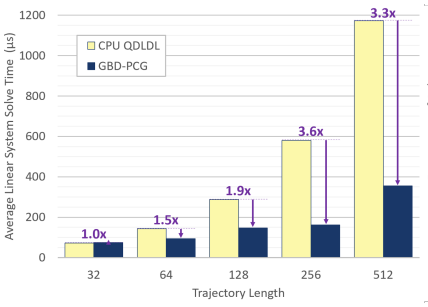
C. Iterative Linear System Solvers

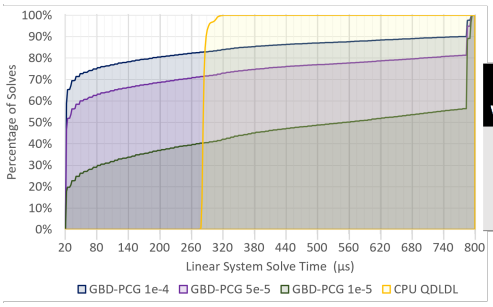
While solvers like OSQP leverage factorization-based approaches to solving their underlying linear systems (in particular the state-of-the-art QDLDL solver for OSQP), iterative methods solve the problem $Ax = b$ for a given A and b by iteratively refining an estimate for x up to some tolerance ϵ. The most popular of these methods is the conjugate gradient (CG) algorithm which has been used for state-of-the-art results on large-scale optimziation problems on the GPU [32, 49]. The convergence rate of CG is directly related to the spread of the eigenvalues of A [38]. Thus, a preconditoning matrix $\Phi \approx A$ is often applied to instead solve the equivalent problem with better numerical properties: $\Phi^{-1}Ax = \Phi^{-1}b$. To do so, the preconditioned conjugate gradient (PCG) algorithm leverages matrix-vector products with A and Φ^{-1}, as well as vector reductions, both parallel friendly operations.

III. MPCGPU

A. Design and Implementation

As shown in Figure 1, our approach is broken down into the three step process found in most direct methods [37], but optimized to expose GPU-friendly computational patterns. First, at each control step we construct a Taylor expansion of the original problem, forming a QP. To form a symmetric positive (semi-)definite linear system that we can solve with PCG, we leverage the Schur complement reformulation of the resulting KKT system. We form each block row of the Schur complement system, S and γ, as well as our preconditioner, Φ^{-1}, in parallel, by taking advantage of the structured sparsity of those matrices. To ensure efficient computation of the underlying dynamics and kinematic quantities, we leverage the GRiD library [45]. Next, we use our custom GPU-optimized, warm-started, block-tridiagonal PCG solver, GBD-PCG, to compute the optimal Lagrange multipliers, λ^*, and reconstruct the optimal trajectory update, $\delta X^*, \delta U^*$. Finally we leverage a parallel line search to compute the final trajectory X, U, which not only reduces latency of the update step, but can also





MPCGPU with GBD-PCG		Knot Points 32	64	128	256	512
Control Rate	250Hz	22.2	19.7	15.4	5.2	4.4
	500Hz	10.3	10.6	8.0	4.6	3.0
	1kHz	4.9	5.2	3.7	2.4	1.7

Fig. 2: (Left) Average linear system solve time ($N = 32 \dots 512$). (Center) Linear system solve cumulative distribution function ($N = 128$). (Right) Average number of trajectory optimization iterations of MPCGPU at each control step. Results were collected on high-performance workstation with a 3.2GHz 16-core Intel i9-12900K and a 2.2GHz NVIDIA GeForce RTX 4090 GPU which ran 100 NMPC trials of end-effector position tracking for a simulated Kuka IIWA-14 executing a 10 second, 5 goal, pick-and-place circuit, with thousands of underlying linear system solves.

improve the convergence of NMPC [42]. We send this to the (simulated) robot for execution and simultaneously measure the current state for our next control step.

B. Experiments and Results

As shown in Figure 2 (left), GBD-PCG outperforms the state-of-the-art CPU-based QDLDL solver [51] across most problem sizes, obtaining as much as a 3.6x average speedup.

In most cases, the speedup is much larger than this, as iterative methods can exit early when warm-started. Using the 128 knot point problem as a case study, as shown in Figure 2 (center), for $\epsilon = 1e^{-4}$, 65% of GBD-PCG solves are $\geq$10x faster than the fastest QDLDL solve, and the slowest GBD-PCG solve is only 2.5x slower than the slowest QDLDL solve (with only 10% $\geq$2x slower).

Figure 2 (right) shows the number of average trajectory optimization iterations MPCGPU can achieve while meeting the specified control rates and trajectory lengths. Our GPU-first approach enables us to solve 512 knot point trajectories at 1kHz and execute 8 iterations for 128 knot points at 500Hz, for a per-iteration rate of 4kHz. This compares favorably to previously reported results of 500hz to 1kHz per-iteration rates for trajectories of 30 to 120 knot points using state-of-the-art CPU-based [29, 19] and GPU-based [42] solvers.

IV. TinyMPC

A. Design and Implementation

TinyMPC trades generality for speed and low-memory utilization to enable real-time use on MCUs by exploiting the structure of the MPC problem. Specifically, we leverage the closed-form Riccati solution to the LQR problem to compute the primal update in (6), accounting for the standard dynamics constraints, and leave any additional constraints to be handled by the remainder of the ADMM algorithm.

In particular, given a long enough horizon, the Riccati recursion (9) converges to the constant solution of the infinite-horizon LQR problem [25]. Thus, we pre-compute a single LQR gain matrix K_∞ and cost-to-go Hessian P_∞. Then, instead of solving the full LQR update at each timestep:

$$\begin{aligned} K_k &= (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal P_{k+1} A) \\ d_k &= (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal p_{k+1} + r_k) \\ P_k &= Q + K_k^\intercal R K_k + (A - BK_k)^\intercal P_{k+1}(A - BK_k) \\ p_k &= q_k + (A - BK_k)^\intercal (p_{k+1} - P_{k+1} B d_k) + K_k^\intercal (R d_k - r_k). \end{aligned} \tag{9}$$

We instead cache the following matrices from (9):

$$\begin{aligned} C_1 &= (R + B^\intercal P_\infty B)^{-1}, \\ C_2 &= (A - BK_\infty)^\intercal, \end{aligned} \tag{10}$$

and then only need to update the linear terms during each ADMM iteration:

$$\begin{aligned} d_k &= C_1(B^\intercal p_{k+1} + r_k), \\ p_k &= q_k + C_2 p_{k+1} - K_\infty^\intercal r_k. \end{aligned} \tag{11}$$

As a result, we avoid online matrix factorization and only compute matrix-vector products. We also dramatically reduce memory footprint by only storing a few vectors per time step.

We also note that the slack update in (7) can be written as the operator Π that projects the slack variable onto the feasible space. For linear inequality constraints, the projection is onto a set of bounds defined by the element-wise operator

$$\Pi(z) = \max(z_l, \min(z_u, z)), \tag{12}$$

where z corresponds to the state and control input slack variables. As the structure of the ADMM algorithm inherently isolates the projection step, we can also replace the projection operator in the slack update (12) with the SOC projection:

$$\Pi_{\mathcal{K}}(z) = \begin{cases} 0, & \|v\|_2 \leq -a, \\ z, & \|v\|_2 \leq a, \\ \frac{1}{2}\left(1 + \frac{a}{\|v\|_2}\right) \begin{bmatrix} v \\ \|v\|_2 \end{bmatrix}, & \|v\|_2 > |a|, \end{cases} \tag{13}$$

where $v = [z_1, \dots, z_{n-1}]^\intercal$, $a = z_n$. Here, $z_i, i = 1, ..., n$ is any vector subset of the state or control slack variables.

B. Experiments and Results

We compare TinyMPC against state-of-the-art solvers for two problems while varying the number of states and the horizon length. The first is a predictive safety filtering problem with box constraints on the state and input. The second is

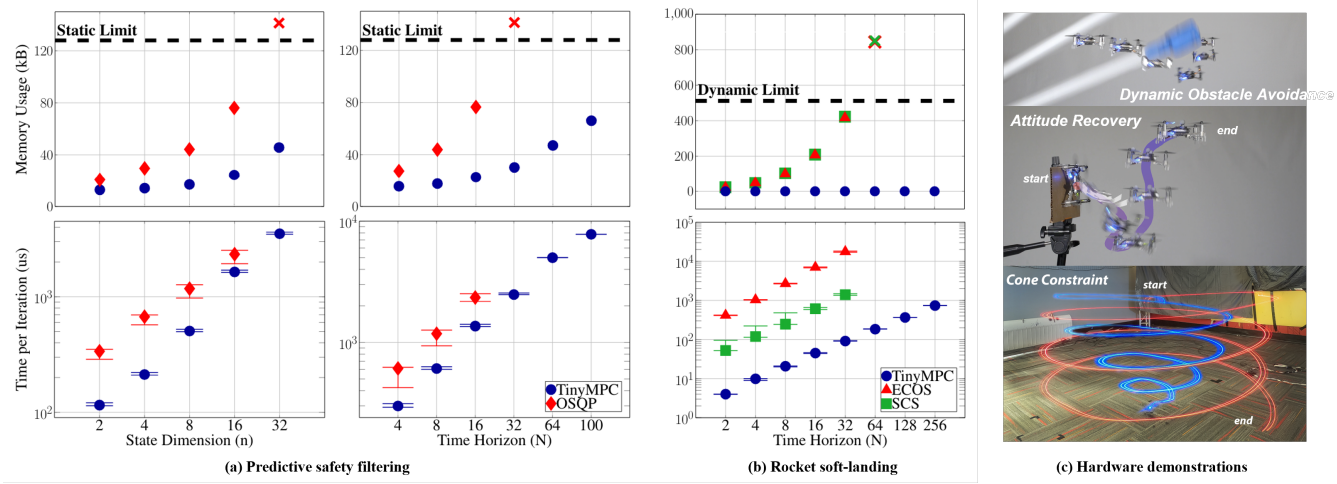


Fig. 3: (a) Compares memory usage (top) and average iteration times (bottom) for TinyMPC and OSQP on a QP-based predictive safety filtering task using a 168 MHz STM32F405 with 1 MB of Flash, and 128 kB of RAM. In the first column, the time horizon was kept constant at $N = 10$ while the state dimension n ranged from 2 to 32 and the input dimension was set to half of the state dimension. In the second column, the state and control input were held constant at $n = 10$ and $m = 5$ while N ranged from 4 to 100. (b) Compares average iteration times (top) and memory usage (bottom) for TinyMPC and ECOS and SCS on an SOCP-based rocket soft-landing using a 600 MHz Teensy 4.1 with 7.75 MB of flash, and 512 kB of tightly coupled RAM. In this experiment $n = 6$ and $m = 3$ while N varied from 2 to 256. The error bars represent the maximum and minimum time taken per iteration for all MPC steps performed for a specific problem. The black dotted lines denote memory thresholds. Across all problems, TinyMPC requires the least memory and is the fastest. (c) Shows the results of hardware experiments for real-time dynamic obstacle avoidance (top), recovery from a 90° attitude error (middle), and tracking a descending helical reference (red) with its position subject to a 45° second-order cone glideslope (blue).

a rocket soft-landing problem with a second-order cone constraint on the thrust vector. The safety filter QP is benchmarked against OSQP and the rocket soft-landing SOCP is benchmarked against ECOS and SCS. The microcontroller results are reported in Figure 3 (a) and (b). Across both problems we find that TinyMPC outperforms comparable state-of-the-art solvers both in terms of latency and in terms of memory usage. In fact, alternative solvers often exceed the memory limits available on our MCU hardware. For example, OSQP exceeds memory limits and cannot be run onboard the STM32F405 for $n, N \geq 32$, and ECOS and SCS exceed the limits of the higher resourced Teensy 4.1 at $N \geq 64$, while TinyMPC scales to $N, n = 100$ and $N = 256$ respectively.

Figure 3 (c) shows the results of hardware demonstrations showing real-time dynamic obstacle avoidance (top), recovery from a 90° attitude error (middle), and tracking a descending helical reference with its position subject to a 45° second-order cone glideslope (bottom), demonstrating the real-world, real-time applicability of TinyMPC.

V. Conclusion and Future Work

Throughout these works we show that in order to overcome computational challenges in robotics at all scales we need to look to the field of software performance engineering and leverage holistic algorithm-hardware co-design. With MPCGPU (https://github.com/a2r-lab/MPCGPU) [1, 7], we show that by leveraging structure sparsity and natural parallelism we can develop a faster-than-state-of-the-art nonlinear MPC solver on the GPU. With TinyMPC (https://tinympc.org) [36, 48], we show that through principled algorithmic simplifications, convex MPC can be run in real-time on a MCU, enabling highly dynamic, compute-limited robotic systems to perform real-time obstacle avoidance and support second-order cone constraints. We hope this work increases interest in co-design, performance engineering, and computational systems for robotics.

In future work we hope to leverage these performant solutions to begin to bridge the gap between optimal control and learning-based control and integrate both control stacks with additional onboard sensors to enable fully autonomous operation on real-world tasks. We are particularly interested in leveraging MPCGPU to support faster actor-critic approaches to MPC [16, 3, 31, 47, 46], and to add adaptive control [23], parameter learning [60], and model-based RL [22] approaches to TinyMPC, enabling it to adjust to a changing real-world environments despite caching certain computations offline.

Acknowledgments

This material is based upon work partially supported by the National Science Foundation (under Award 2246022). Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the funding organizations. The authors would also like to thank Brian Jackson for insightful discussions and Professor Mark Bedillion for providing us with extra Crazyflies for hardware experiments.

References

[1] Emre Adabag, Miloni Atal, William Gerard, and Brian Plancher. Mpcgpu: Real-time nonlinear model predictive control through preconditioned conjugate gradient on the gpu. In *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May 2024.

[2] Vivek Adajania, Siqi Zhou, Singh Arun, and Angela Schoellig. Amswarm: An alternating minimization approach for safe motion planning of quadrotor swarms in cluttered environments. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1421–1427, 2023.

[3] Elisa Alboni, Gianluigi Grandesso, Gastone Pietro Rosati Papini, Justin Carpentier, and Andrea Del Prete. Cacto-sl: Using sobolev learning to improve continuous actor-critic with trajectory optimization. *arXiv preprint arXiv:2312.10666*, 2023.

[4] Thomas Antony and Michael J. Grant. Rapid Indirect Trajectory Optimization on Highly Parallel Computing Architectures. 54(5):1081–1091. ISSN 0022-4650. doi: 10.2514/1.A33755.

[5] John T Betts. *Practical Methods for Optimal Control Using Nonlinear Programming*, volume 3 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM).

[6] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, Jonathan Eckstein, et al. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning*, 3(1):1–122, 2011.

[7] Xueyi Bu and Brian Plancher. Symmetric stair preconditioning of linear systems for parallel trajectory optimization. In *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May 2024.

[8] Kong Yao Chee, Tom Z Jiahao, and M Ani Hsieh. Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots. *IEEE Robotics and Automation Letters*, 7(2):2819–2826, 2022.

[9] Sébastien D De Rivaz, Benjamin Goldberg, Neel Doshi, Kaushik Jayaram, Jack Zhou, and Robert J Wood. Inverted and vertical climbing of a quadrupedal microrobot using electroadhesion. *Science Robotics*, 3(25):eaau3038, 2018.

[10] Jared Di Carlo. *Software and control design for the MIT Cheetah quadruped robots*. PhD thesis, Massachusetts Institute of Technology, 2020.

[11] Bardienus P Duisterhof, Shushuai Li, Javier Burgués, Vijay Janapa Reddi, and Guido CHE de Croon. Sniffy bug: A fully autonomous swarm of gas-seeking nano quadcopters in cluttered environments. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9099–9106. IEEE, 2021.

[12] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark Silicon and the End of Multicore Scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376. ACM. ISBN 978-1-4503-0472-6. doi: 10.1145/2000064.2000108.

[13] Wojciech Giernacki et al. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. URL https://www.bitcraze.io/papers/giernacki_draft_crazyflie2.0.pdf.

[14] Farbod Farshidian, Edo Jelavic, Asutosh Satapathy, Markus Giftthaler, and Jonas Buchli. Real-time motion planning of legged robots: A model predictive control approach. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 577–584. IEEE, 2017.

[15] Markus Giftthaler, Michael Neunert, Markus Stäuble, Jonas Buchli, and Moritz Diehl. A family of iterative gauss-newton shooting methods for nonlinear optimal control. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9. IEEE, 2018.

[16] Gianluigi Grandesso, Elisa Alboni, Gastone P Rosati Papini, Patrick M Wensing, and Andrea Del Prete. Cacto: Continuous actor-critic with trajectory optimization—towards global optimality. *IEEE Robotics and Automation Letters*, 2023.

[17] Francois Robert Hogan, Eudald Romo Grau, and Alberto Rodriguez. Reactive planar manipulation with convex hybrid mpc. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 247–253. IEEE, 2018.

[18] Yuning Jiang, Juraj Oravec, Boris Houska, and Michal Kvasnica. Parallel mpc for linear systems with input constraints. *IEEE Transactions on Automatic Control*, 66(7):3401–3408, 2020.

[19] Sébastien Kleff, Avadesh Meduri, Rohan Budhiraja, Nicolas Mansard, and Ludovic Righetti. High-frequency nonlinear model predictive control of a manipulator. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7330–7336. IEEE, 2021.

[20] Dimitris Kouzoupis, Rien Quirynen, Boris Houska, and Moritz Diehl. A block based aladin scheme for highly parallelizable direct optimal control. In *2016 American Control Conference (ACC)*, pages 1124–1129. IEEE, 2016.

[21] Scott Kuindersma. Taskable agility: Making useful dynamic behavior easier to create. Princeton Robotics Seminar, 4 2023.

[22] Nathan O Lambert, Daniel S Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer SJ Pister. Low-level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters*, 4(4):4224–4230, 2019.

[23] Ioan Doré Landau, Rogelio Lozano, Mohammed M'Saad, et al. *Adaptive control*, volume 51. Springer New York, 1998.

[24] Yeongseok Lee, Minsu Cho, and Kyung-Soo Kim. Gpu-parallelized iterative lqr with input constraints for fast

collision avoidance of autonomous vehicles. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4797–4804, 2022. doi: 10.1109/IROS47612.2022.9982026.

[25] Frank L. Lewis, Draguna Vrabie, and V.L. Syrmos. Optimal Control, 1 2012. URL https://doi.org/10.1002/9781118122631.

[26] He Li, Wenhao Yu, Tingnan Zhang, and Patrick M Wensing. A unified perspective on multiple shooting in differential dynamic programming. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9978–9985. IEEE, 2023.

[27] Carlos E Luis, Marijan Vukosavljev, and Angela P Schoellig. Online trajectory generation with distributed model predictive control for multi-robot motion planning. *IEEE Robotics and Automation Letters*, 5(2):604–611, 2020.

[28] Zachary Manchester, Neel Doshi, Robert J Wood, and Scott Kuindersma. Contact-implicit trajectory optimization using variational integrators. *The International Journal of Robotics Research*, 38(12-13):1463–1476, 2019.

[29] Carlos Mastalli, Rohan Budhiraja, Wolfgang Merkt, Guilhem Saurel, Bilal Hammoud, Maximilien Naveau, Justin Carpentier, Ludovic Righetti, Sethu Vijayakumar, and Nicolas Mansard. Crocoddyl: An efficient and versatile framework for multi-contact optimal control. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2536–2542. IEEE, 2020.

[30] KN McGuire, Christophe De Wagter, Karl Tuyls, HJ Kappen, and Guido CHE de Croon. Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment. *Science Robotics*, 4 (35):eaaw9710, 2019.

[31] Andrew S Morgan, Daljeet Nandha, Georgia Chalvatzaki, Carlo D'Eramo, Aaron M Dollar, and Jan Peters. Model predictive actor-critic: Accelerating robot skill acquisition with deep reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6672–6678. IEEE, 2021.

[32] Maxim Naumov. Incomplete-lu and cholesky preconditioned iterative methods using cusparse and cublas. *Nvidia white paper*, 3, 2011.

[33] Sabrina M. Neuman, Brian Plancher, Thomas Bourgeat, Thierry Tambe, Srinivas Devadas, and Vijay Janapa Reddi. Robomorphic computing: A design methodology for domain-specific accelerators parameterized by robot morphology. ASPLOS 2021, page 674–686, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi: 10.1145/3445814.3446746. URL https://doi-org.ezp-prod1.hul.harvard.edu/10.1145/3445814.3446746.

[34] Sabrina M Neuman, Brian Plancher, Bardienus P Duisterhof, Srivatsan Krishnan, Colby Banbury, Mark Mazumder, Shvetank Prakash, Jason Jabbour, Aleksandra Faust, Guido CHE de Croon, et al. Tiny robot learning: challenges and directions for machine learning in resource-constrained robots. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 296–299. IEEE, 2022.

[35] Sabrina M. Neuman, Radhika Ghosal, Thomas Bourgeat, Brian Plancher, and Vijay Janapa Reddi. Roboshape: Using topology patterns to scalably and flexibly deploy accelerators across robots. In *Proceedings of the 50th Annual International Symposium on Computer Architecture*, ISCA '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400700958. doi: 10.1145/3579371.3589104. URL https://doi.org/10.1145/3579371.3589104.

[36] Khai Nguyen, Sam Schoedel, Anoushka Alavilli, Brian Plancher, and Zachary Manchester. Tinympc: Model-predictive control on resource-constrained microcontrollers. In *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May 2024.

[37] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[38] Jorge Nocedal and Stephen J Wright. Conjugate gradient methods. *Numerical optimization*, pages 101–134, 2006.

[39] Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016. URL http://stanford.edu/~boyd/papers/scs.html.

[40] Zherong Pan, Bo Ren, and Dinesh Manocha. Gpu-based contact-aware trajectory optimization using a smooth force model. In *Proceedings of the 18th Annual ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '19, pages 4:1–4:12, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6677-9. doi: 10.1145/3309486.3340246.

[41] Petoi. Open source, programmable robot dog bittle. Available at https://www.petoi.com/pages/bittle-open-source-bionic-robot-dog (5.9.2023).

[42] Brian Plancher and Scott Kuindersma. A performance analysis of parallel differential dynamic programming on a gpu. In *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics 13*, pages 656–672. Springer, 2018.

[43] Brian Plancher and Scott Kuindersma. Realtime model predictive control using parallel ddp on a gpu. In *Toward Online Optimal Control of Dynamic Robots Workshop at the 2019 International Conference on Robotics and Automation (ICRA)*, Montreal, Canada, May. 2019.

[44] Brian Plancher, Sabrina M. Neuman, Thomas Bourgeat, Scott Kuindersma, Srinivas Devadas, and Vijay Janapa Reddi. Accelerating robot dynamics gradients on a cpu, gpu, and fpga. *IEEE Robotics and Automation Letters*, 6 (2):2335–2342, 2021. doi: 10.1109/LRA.2021.3057845.

[45] Brian Plancher, Sabrina M. Neuman, Radhika Ghosal, Scott Kuindersma, and Vijay Janapa Reddi. Grid: Gpu-accelerated rigid body dynamics with analytical gradi-

ents. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 2022.

[46] Rudolf Reiter, Andrea Ghezzi, Katrin Baumgärtner, Jasper Hoffmann, Robert D McAllister, and Moritz Diehl. Ac4mpc: Actor-critic reinforcement learning for nonlinear model predictive control. *arXiv preprint arXiv:2406.03995*, 2024.

[47] Angel Romero, Yunlong Song, and Davide Scaramuzza. Actor-critic model predictive control. In *IEEE International Conference on Robotics and Automation (ICRA)*, Yokohama, Japan, May 2024.

[48] Sam Schoedel, Khai Nguyen, Elakhya Nedumaran, Brian Plancher, and Zachary Manchester. Code generation for conic model-predictive control on microcontrollers with tinympc, 2024.

[49] Michel Schubiger, Goran Banjac, and John Lygeros. Gpu acceleration of admm for large-scale quadratic programming. *Journal of Parallel and Distributed Computing*, 144:55–67, 2020.

[50] Jean-Pierre Sleiman, Farbod Farshidian, Maria Vittoria Minniti, and Marco Hutter. A unified mpc framework for whole-body dynamic locomotion and manipulation. *IEEE Robotics and Automation Letters*, 6(3):4688–4695, 2021.

[51] Bartolomeo Stellato, Goran Banjac, Paul Goulart, Alberto Bemporad, and Stephen Boyd. Osqp: An operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4):637–672, 2020.

[52] Guillem Torrente, Elia Kaufmann, Philipp Föhn, and Davide Scaramuzza. Data-driven mpc for quadrotors. *IEEE Robotics and Automation Letters*, 6(2):3769–3776, 2021.

[53] Marco Tranzatto, Frank Mascarich, Lukas Bernreiter, Carolina Godinho, Marco Camurri, Shehryar Khattak, Tung Dang, Victor Reijgwart, Johannes Loeje, David Wisth, Samuel Zimmermann, Huan Nguyen, Marius Fehr, Lukas Solanka, Russell Buchanan, Marko Bjelonic, Nikhil Khedekar, Mathieu Valceschini, Fabian Jenelten, Mihir Dharmadhikari, Timon Homberger, Paolo De Petris, Lorenz Wellhausen, Mihir Kulkarni, Takahiro Miki, Satchel Hirsch, Markus Montenegro, Christos Papachristos, Fabian Tresoldi, Jan Carius, Giorgio Valsecchi, Joonho Lee, Konrad Meyer, Xiangyu Wu, Juan Nieto, Andy Smith, Marco Hutter, Roland Siegwart, Mark Mueller, Maurice Fallon, and Kostas Alexis. Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge. *arXiv preprint arXiv:2201.07067*, 2022.

[54] Pratyush Varshney, Gajendra Nagar, and Indranil Saha. Deepcontrol: Energy-efficient control of a quadrotor using a deep neural network. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 43–50, 2019. doi: 10.1109/IROS40897.2019.8968236.

[55] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation Cores: Reducing the Energy of Mature Computations. In *Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XV, pages 205–218. ACM. ISBN 978-1-60558-839-1. doi: 10.1145/1736020.1736044.

[56] Patrick M Wensing, Michael Posa, Yue Hu, Adrien Escande, Nicolas Mansard, and Andrea Del Prete. Optimization-based control for dynamic legged robots. *arXiv preprint arXiv:2211.11644*, 2022.

[57] Lele Xi, Xinyi Wang, Lei Jiao, Shupeng Lai, Zhihong Peng, and Ben M Chen. Gto-mpc-based target chasing using a quadrotor in cluttered environments. *IEEE Transactions on Industrial Electronics*, 69(6):6026–6035, 2021.

[58] Yuxin Yang, Xiaoming Chen, and Yinhe Han. Rbdcore: Robot rigid body dynamics accelerator with multifunctional pipelines. *arXiv preprint arXiv:2307.02274*, 2023.

[59] Zhengdong Zhang, Amr AbdulZahir Suleiman, Luca Carlone, Vivienne Sze, and Sertac Karaman. Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach. 2017.

[60] Xin Zhou, Chao Xu, and Fei Gao. Automatic parameter adaptation for quadrotor trajectory planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3348–3355. IEEE, 2022.