End-to-End Hardware-Algorithm Co-Design for High-Rate FPGA-Accelerated Model Predictive Control on Tiny Drones

Andrea Grillo[1,2], Brian Plancher[1]

***Abstract*— Model Predictive Control (MPC) enables high-performance feedback control by explicitly handling system dynamics and constraints but is often computationally prohibitive for resource-constrained robots. We overcome this challenge via an end-to-end co-design approach that jointly optimizes the solver algorithm, hardware architecture, and physical integration. Our system features a solver based on the alternating direction method of multipliers (ADMM) framework, tailored for FPGA implementation using reduced-precision arithmetic, paired with a custom-designed lightweight, 6g PCB to ensure high-bandwidth communication for deployment on a 35g Crazyflie quadrotor. This tight hardware-algorithm integration enables 500 Hz constrained control over long horizons, a regime previously unattainable. Hardware experiments demonstrate up to 16× faster solve times and a 244× improvement in energy-delay product over state-of-the-art, embedded, microcontroller-based solvers, proving that co-design is essential for sophisticated constrained control on highly compute-limited platforms.**

I. Introduction

Tiny aerial robots, such as the 35g Crazyflie quadrotor [1], pose an acute control challenge. Their fast, unstable dynamics demand feedback rates of several hundred hertz, yet their onboard computation, energy, weight, and size budgets are severely constrained. Model Predictive Control (MPC) is an attractive solution as it can deliver high-performance feedback control while explicitly accounting for stringent system dynamics and constraints [2], [3], [4]. Recent advances in embedded MPC solvers have made progress toward real-time rates on such limited compute platforms. Solvers such as TinyMPC [5] have exploited the alternating direction method of multipliers (ADMM) framework [6] to enable deployment onboard the Crazyflie's microcontroller. However, despite these advances, at fast rates, TinyMPC is limited to only one or two ADMM iterations per control step. This is far too few for reliable convergence when constraints are active and has required deployed control rates to be reduced [5], [7], [8], leaving a fundamental gap between the performance MPC promises and what current embedded hardware can deliver.

We close this gap through end-to-end hardware-algorithm co-design (Fig. 1), jointly optimizing solver structure, numerical representation, hardware architecture, and physical integration onboard our target tiny drone platform. Our system pairs a co-designed FPGA-accelerated ADMM-based MPC solver, with a custom 6g PCB providing high-bandwidth communication for deployment on a Crazyflie (Fig. 4). This tight co-design enables up to 16× faster solve times and a 244× improvement in energy-delay product over state-of-the-art embedded solvers (e.g., [5]), unlocking a larger iteration budget for constrained control at 500 Hz, and, for the first time, fully onboard FPGA-based constrained trajectory tracking on a tiny quadrotor (Fig. 2).

[1] Dartmouth College
[2] École Polytechnique Fédérale de Lausanne (EPFL)
Correspondence to: plancher@dartmouth.edu
We thank Roy Xing for experimental help and Colin Jones for advising.

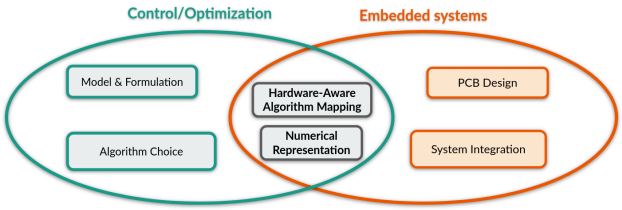

Fig. 1: End-to-end co-design for embedded MPC systems. The approach jointly considers control and optimization aspects (formulation and algorithm choice) and embedded system constraints (PCB design and system integration).

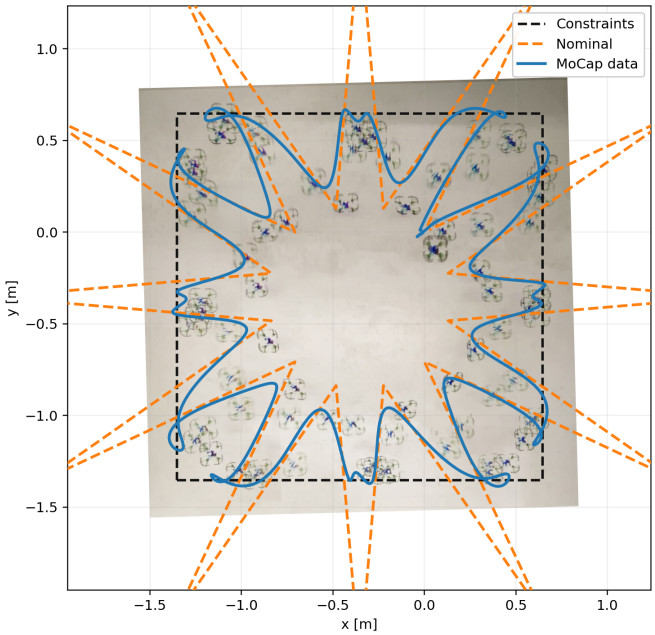

Fig. 2: Hardware demonstration of constrained trajectory tracking at 500 Hz. Box constraints on x and y are enforced fully onboard by the FPGA-accelerated MPC controller ($H = 40, k = 10$). The trajectory remains within the admissible region despite a constraint-violating reference.

II. Background

We consider a linear MPC problem over a prediction horizon H, for a system with states, x_k, and controls, u_k,

where A and B define the system dynamics, and $Q \succeq 0$, $Q_f \succeq 0$, and $R \succ 0$ define the cost function. The constraint sets $\mathcal{X}$ and $\mathcal{U}$ are assumed polyhedral:

$$\begin{aligned} \min_{\{x_k,u_k\}} \sum_{k=0}^{H-1} & \big[(x_k - x_{\text{ref}})^\top Q (x_k - x_{\text{ref}}) \\ & + (u_k - u_{\text{ref}})^\top R (u_k - u_{\text{ref}})\big] \\ & + (x_H - x_{\text{ref}})^\top Q_f (x_H - x_{\text{ref}}) \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k, \quad k = 0, \dots, H-1, \\ & x_0 = \hat{x}, \quad x_k \in \mathcal{X}, \quad u_k \in \mathcal{U}. \end{aligned} \tag{1}$$

By stacking states and inputs into an interleaved vector,

$$w = [x_0^\top, u_0^\top, x_1^\top, u_1^\top, \dots, x_{H-1}^\top, u_{H-1}^\top, x_H^\top]^\top, \tag{2}$$

we can write (1) as a sparse convex quadratic program:

$$\min_w \ \tfrac{1}{2} w^\top P w + c^\top w \quad \text{s.t.} \quad l \le Mw \le u, \tag{3}$$

where $P \succeq 0$ is block-diagonal and M is sparse and block-banded due to the stage-wise dynamics coupling.

A. ADMM for solving MPC QP

We briefly review ADMM [6], as applied to the MPC QP in (3). First, we introduce an auxiliary variable y:

$$\min_{w,y} \ \tfrac{1}{2} w^\top P w + c^\top w + I_{\mathcal{C}}(y) \quad \text{s.t.} \quad Mw = y. \tag{4}$$

where $I_{\mathcal{C}}(y)$ is the indicator function of the convex set,

$$\mathcal{C} = \{y \mid l \le y \le u\}. \tag{5}$$

The augmented Lagrangian of this problem is:

$$\begin{aligned} \mathcal{L}_\rho(w, y, \lambda) = & \tfrac{1}{2} w^\top P w + c^\top w + I_{\mathcal{C}}(y) \\ & + \tfrac{\rho}{2} \|Mw - y\|_2^2 + \lambda^\top (Mw - y). \end{aligned} \tag{6}$$

It is then solved via the three-step ADMM iterations:

$$w^+ = \arg\min_w \mathcal{L}_\rho(w, y, \lambda), \tag{7}$$

$$y^+ = \Pi_{[l,u]}\Big(Mw^+ + \tfrac{1}{\rho}\lambda\Big), \tag{8}$$

$$\lambda^+ = \lambda + \rho(Mw^+ - y^+). \tag{9}$$

where $^+$ denotes the next iterate, $\Pi_{[l,u]}(\cdot)$ is the projection onto the box $[l, u]$ and the w-update reduces to the structured linear system solve:

$$(P + \rho M^\top M)\, w^+ = -c + \rho M^\top y - M^\top \lambda, \tag{10}$$

which, as discussed in IV-A, is particularly amenable to hardware acceleration.

III. Related Work

A. Embedded and Edge MPC Solvers

Given the structured QP formulation described in Sec. II, a wide range of embedded and edge MPC solvers have been proposed with different trade-offs.

Code-generation tools such as CVXGEN [9] produce solvers tailored to a specific QP structure, enabling efficient execution on edge CPUs. More general-purpose approaches, such as OSQP [10], rely on operator-splitting methods with warm-starting and have been widely adopted in robotics. Structured solvers such as HPIPM [11] further exploit problem structure and optimized linear algebra to achieve high performance on embedded platforms. Another thread has investigated GPU-accelerated MPC, exploiting data-level parallelism for high-performance [12], [13], [14].

However, importantly, in both cases, tiny quadrotors like the Crazyflie, cannot support such power-hungry and heavy edge CPUs and GPUs. Instead, they rely on resource-constrained microcontrollers. To the best of our knowledge, TinyMPC [5], [15] is the only solver capable of achieving real-time MPC on-board the Crazyflie platform. It does so by reformulating the primal update as an unconstrained LQR problem and caching the associated Riccati factorizations offline, enabling efficient QP solves within tight memory and timing constraints. This approach limits its ability to reliably handle constraints at high update rates. As a result, when constraints become active, the controller is typically operated at lower rates to maintain stability and feasibility.

B. FPGA-Based Linear MPC

A significant body of work has explored FPGA acceleration of linear MPC: Jerez *et al.* [16] demonstrated sparse QP solvers on large Kintex-7 boards; Hartley *et al.* [17] targeted aircraft control with FPGA-in-the-loop simulation; Shukla *et al.* [18] proposed a code-generation framework for FPGA splitting methods; Jeong *et al.* [19] implemented ADMM for power electronics. Critically, all prior FPGA-based MPC implementations either target *explicit* MPC [20], which pre-computes control laws offline and does not run an online optimization, use development boards too large or power-hungry for airborne deployment, or validate only in simulation or hardware-in-the-loop. As such, none have been deployed and flown onboard an aerial robot. Similarly, while the neural thrust controller of Azem *et al.* [21] showed that an FPGA can be flown on a Crazyflie, it does not perform online optimization.

IV. Hardware-Algorithm Codesign

To the best of our knowledge, no prior work combines FPGA-accelerated *online* MPC with onboard deployment and closed-loop flight validation. Our end-to-end hardware-algorithm co-designed system fills this gap.

A. Solver Algorithm Design and Implementation

To enable high-rate constrained control on a lightweight airborne FPGA platform, our solver algorithm had to satisfy the following co-design requirements:

- *Pipeline-friendly structure*: support for pipelined execution to achieve throughput and deterministic latency;
- *Regular computational structure*: limited branching and predictable data flow;
- *Robustness to reduced precision*: stable convergence under fixed-point or low-precision arithmetic.

The ADMM algorithm presented in Sec. II-A satisfies these requirements due to its structured computational patterns and known robustness to noise [6], [22].

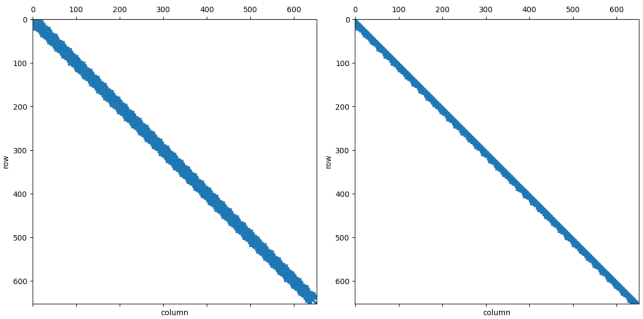

Fig. 3: Left: coefficient matrix of the w-update. Right: corresponding Cholesky factor. The banded sparsity structure is preserved, enabling efficient triangular solves.

The dominant cost within each ADMM iteration is the structured linear solve in (10). Unlike prior hardware-accelerated solutions on large GPUs [23], our embedded FPGA does not have sufficient memory to precompute the entire matrix inverse. Instead, following the spirit of Riccati-based approaches [5], we precompute and store the Cholesky factorization, which exploits the banded structure of our problem (Fig. 3), reducing the online work to forward and backward substitution passes.

The Cholesky factor is stored in a banded-row format. That is, if the bandwidth is h, each row stores only h values, with the diagonal reciprocal in the last slot to avoid online divisions. Because h is determined primarily by the coupling structure and is nearly independent of the horizon length, the resulting forward and backward substitution can be implemented with an efficient *sliding-window shift register* of depth h. Here, newly computed values enter the window while old ones shift out, so only small local registers are accessed rather than large BRAMs. This yields perfectly regular access patterns that are readily pipelined by high-level synthesis frameworks (HLS), like Vitis HLS [24] used in this work. Thus, the cost per iteration scales as $\mathcal{O}(N_{\text{var}})$, linear in the number of decision variables.

The underlying matrix-vector products used in these passes exploit problem structure and are generated on-the-fly from system dynamics matrices, reducing memory usage and enabling regular, pipeline-friendly computation.

The y-update is an elementwise projection onto the box constraint $[l, u]$ and the dual update is a fused multiply–add, both trivially pipelined.

All constant matrices are generated offline and compiled as header constants into the bitstream, enabling an efficient and division-free runtime.

The penalty parameter ρ is constrained to a power of two, so scaling by ρ reduces to a bit shift (fixed-point) or exponent adjustment (`ldexp`, floating-point).

Finally, we note that the HLS implementation is designed to be modular on the numeric datatype, allowing the same source to be synthesized in multiple formats (e.g., IEEE 754 single-precision, 32-bit fixed-point).

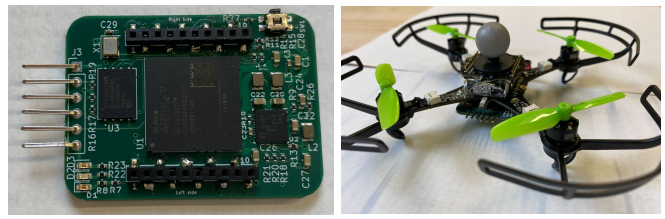

Fig. 4: The $4\times3\,\text{cm}$ four-layer PCB hosts an AMD Artix-7 100T FPGA and mounts directly on the Crazyflie via the standard expansion connector. Left: fabricated FPGA board. Right: integrated onboard the tiny quadrotor.

B. Hardware Integration

We found that existing solutions to mount an FPGA on the Crazyflie were unsuitable for our needs. Available decks either provide insufficient FPGA resources (e.g., [25]) or rely on low-speed interfaces such as I^2C (e.g., [21]), which limit closed-loop control bandwidth. As such, we designed and fabricated a custom four-layer PCB expansion board (Fig. 4).

The board hosts an AMD Artix-7 100T FPGA, powered by a dedicated PMIC generating the required core and I/O voltage rails with sequencing. High-speed SPI communication is used to interface with the Crazyflie, enabling low-latency data exchange for closed-loop control. A 100 MHz oscillator provides the clock, while an onboard SPI flash stores the FPGA configuration. Additional components include JTAG access and user LEDs for debugging and development.

We modified the Crazyflie firmware (FreeRTOS-based) to compute the state deviation from the EKF estimate to the reference, transmit it to the FPGA via SPI to trigger the optimization, and retrieve the resulting control input for actuation. The ADMM computation is fully offloaded to the FPGA, while state estimation and the remaining flight-stack tasks remain on the MCU.

Design files are open-sourced at `github.com/A2R-Lab`, including PCB (`Crazyflie_FPGA_Deck`), firmware (`crazyflie_fpga_firmware`), and the FPGA project (`ADMM_FPGA`).

V. Experimental Evaluation

A. Solver Benchmarking

Methodology. We sweep prediction horizon $H \in \{10, 20, 30, 40, 50, 60, 70, 80\}$ for three implementations: (i) TinyMPC on the Crazyflie MCU (STM32F405, 168 MHz), measured inside the FreeRTOS flight stack over 80 samples per operating point; (ii) FPGA floating-point; (iii) FPGA fixed-point. FPGA timings are measured with one sample per configuration as execution is cycle-deterministic.

Latency. Fig. 5 shows solve time versus H at fixed k=10. All three implementations scale approximately linearly with horizon, but with clearly separated slopes. The median time per ADMM iteration per horizon step is: TinyMPC: 52.3 µs/(iter·step); FPGA float: 8.6 µs/(iter·step); FPGA fixed: 3.3 µs/(iter·step). Relative to TinyMPC, the median speedup is **6.1**× (float) and **16**× (fixed-point).

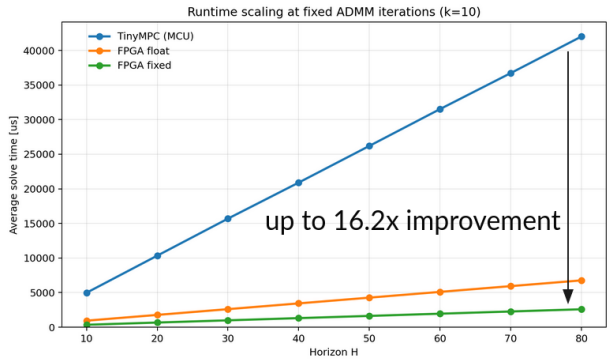

Fig. 5: Average solve time versus prediction horizon at k=10 ADMM iterations. The fixed-point FPGA solver achieves nearly 16× lower latency than TinyMPC (MCU).

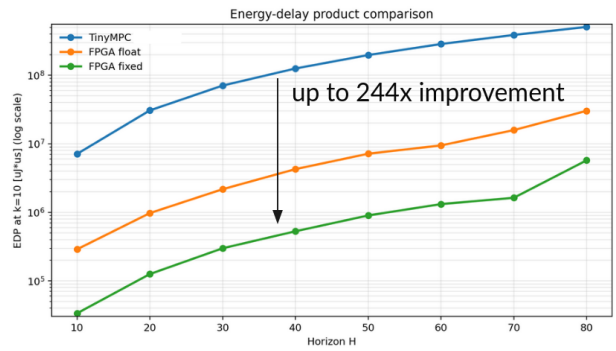

Fig. 6: Energy–delay product at k=10 ADMM iterations, shown on a log scale. The fixed-point FPGA solver achieves up to 244× lower EDP than TinyMPC.

	Prediction Horizon H							
Solver	10	20	30	40	50	60	70	80
TinyMPC (MCU)	2	1	1	–	–	–	–	–
FPGA float	20	10	7	5	4	3	3	2
FPGA fixed	53	28	19	14	11	9	8	7

TABLE I: Maximum feasible ADMM iterations within a 500 Hz control period. FPGA includes 100 µs SPI overhead.

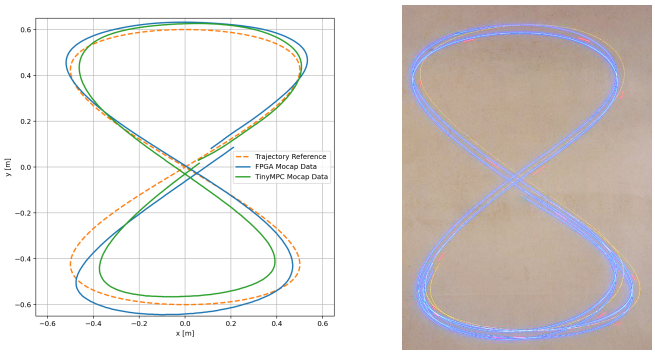

Fig. 7: Unconstrained figure-eight trajectory tracking. Both TinyMPC and the FPGA-based controller achieve similar tracking performance. Left: trajectory overlay in the xy plane. Right: long-exposure flight photograph.

Real-time feasibility at 500 Hz. Table I reports the maximum feasible ADMM iteration budget within the 2 ms control period (1.9 ms for FPGA after 100 µs SPI overhead). TinyMPC is limited to $k \leq 2$ at $H \leq 30$; for longer horizons, it lacks sufficient time to perform even a single iteration, rendering the solver effectively non-operational. The FPGA fixed-point implementation sustains k=19 at H=30 and k=7 even at H=80, opening a dramatically larger region of practically deployable MPC configurations.

Energy and energy–delay product (EDP). For TinyMPC we estimate power from the STM32F405 datasheet active-mode current (87 mA at 168 MHz, 3.3 V). FPGA power is from post-route analysis combined with measured solve time. The fixed-point FPGA implementation achieves **5.5–15.9**× lower energy per solve and **88–244**× lower EDP than TinyMPC (Fig. 6). The floating-point variant still improves by 2.7–5.4× in energy and 16.7–32.5× in EDP.

B. Onboard Flight Experiments

All experiments use the Crazyflie Brushless 2.1 in an OptiTrack eight-camera arena (120 Hz position stream), with the EKF and flight stack on the MCU and the ADMM solver on the FPGA. The control loop runs at 500 Hz. Only the fixed-point FPGA implementation is used in flight.

Experiment 1: Unconstrained figure-eight tracking. We compare TinyMPC (H=15, k=2) with the FPGA controller (H=20, k=28) on a smooth figure-eight path in the horizontal plane at constant altitude. Because no constraints are active, both configurations solve tractable QPs and achieve similar closed-loop tracking performance, as shown in Fig. 7.

Experiment 2: Constrained trajectory tracking. Box constraints are imposed on the x and y coordinates, restricting the workspace to a square region. The reference trajectory intentionally violates these bounds (a star-shaped path), requiring active constraint enforcement. The controller uses H=40, k=10. TinyMPC cannot reliably enforce constraints at 500 Hz because its budget of one to two ADMM iterations is insufficient for convergence; prior deployments therefore reduce the rate to ≤100 Hz for constrained tasks.

Fig. 2 shows the closed-loop trajectory remaining within the square region in a real flight. The larger optimization budget enabled by end-to-end co-design translates directly into a new onboard control capability: high-rate constrained trajectory tracking on a tiny quadrotor platform.

VI. Conclusion

We presented an FPGA-accelerated linear MPC system that achieves a 16× speedup and 244× better energy–delay product compared to the state-of-the-art TinyMPC [5]. By co-designing an ADMM-based solver for fixed-point execution with a custom PCB for fast communication, we enable 500 Hz constrained trajectory tracking on a tiny quadrotor.

REFERENCES

[1] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, and P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," in *2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 2017, pp. 37–42.

[2] J. Rawlings, D. Mayne, and M. Diehl, *Model Predictive Control: Theory, Computation, and Design*, Jan. 2017.

[3] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, June 2000.

[4] E. F. Camacho and C. Bordons, *Model Predictive Control*, ser. Advanced Textbooks in Control and Signal Processing, M. J. Grimble and M. A. Johnson, Eds. London: Springer, 2007.

[5] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 1–7.

[6] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers," *Foundations and Trends in Machine Learning*, vol. 3, pp. 1–122, Jan. 2011.

[7] I. Mahajan, K. Nguyen, S. Schoedel, E. Nedumaran, M. Mata, B. Plancher, and Z. Manchester, "Code generation and conic constraints for model-predictive control on microcontrollers with conic-tinympc," in *IEEE International Conference on Robotics and Automation (ICRA)*, June 2026.

[8] I. Mahajan and B. Plancher, "Robust and efficient embedded convex optimization through first-order adaptive caching," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

[9] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," *Optimization and Engineering*, vol. 13, Mar. 2012.

[10] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, Dec. 2020.

[11] G. Frison and M. Diehl, "Hpipm: a high-performance quadratic programming framework for model predictive control∗∗this research was supported by the german federal ministry for economic affairs and energy (bmwi) via eco4wind (0324125b) and dyconpv (0324166b), and by dfg via research unit for 2401." *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020, 21st IFAC World Congress. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320303293

[12] M. Schubiger, G. Banjac, and J. Lygeros, "GPU acceleration of ADMM for large-scale quadratic programming," *Journal of Parallel and Distributed Computing*, vol. 144, pp. 55–67, Oct. 2020.

[13] E. Adabag, M. Atal, W. Gerard, and B. Plancher, "MPCGPU: Real-Time Nonlinear Model Predictive Control through Preconditioned Conjugate Gradient on the GPU," in *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*. IEEE, 2024, pp. 9787–9794.

[14] L. Amatucci, J. Sousa-Pinto, G. Turrisi, D. Orban, V. Barasuol, and C. Semini, "Primal-dual ilqr for gpu-accelerated learning and control in legged robots," *IEEE Robotics and Automation Letters*, vol. 11, no. 1, pp. 1010–1017, 2025.

[15] S. Schoedel, K. Nguyen, E. Nedumaran, B. Plancher, and Z. Manchester, "Code Generation for Conic Model-Predictive Control on Microcontrollers with TinyMPC," *CoRR*, vol. abs/2403.18149, 2024.

[16] J. L. Jerez, P. J. Goulart, S. Richter, G. A. Constantinides, E. C. Kerrigan, and M. Morari, "Embedded Online Optimization for Model Predictive Control at Megahertz Rates," *IEEE Transactions on Automatic Control*, vol. 59, no. 12, pp. 3238–3251, Dec. 2014.

[17] E. N. Hartley, J. L. Jerez, A. Suardi, J. M. Maciejowski, E. C. Kerrigan, and G. A. Constantinides, "Predictive Control Using an FPGA With Application to Aircraft Control," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 1006–1017, May 2014.

[18] H. A. Shukla, B. Khusainov, E. C. Kerrigan, and C. N. Jones, "Software and Hardware Code Generation for Predictive Control Using Splitting Methods*," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 386–14 391, July 2017.

[19] M. Jeong, M. Schoen, and J. Biela, "When FPGAs Meet ADMM with High-level Synthesis (HLS): A Real-time Implementation of Long-horizon MPC for Power Electronic Systems," in *2023 11th International Conference on Power Electronics and ECCE Asia (ICPE 2023 - ECCE Asia)*, May 2023, pp. 1704–1711.

[20] P. Tondel, T. Johansen, and A. Bemporad, "Computation and approximation of piecewise affine control laws via binary search trees," in *Proceedings of the 41st IEEE Conference on Decision and Control, 2002.*, vol. 3, Dec. 2002, pp. 3144–3149 vol.3.

[21] S. Azem, D. Scheunert, M. Li, J. Gehrunger, K. Cui, C. Hochberger, and H. Koeppl, "FPGA-Based Neural Thrust Controller for UAVs," https://arxiv.org/abs/2403.18703v2, Mar. 2024.

[22] J. Eckstein and D. P. Bertsekas, "On the douglas—rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical programming*, vol. 55, no. 1, pp. 293–318, 1992.

[23] A. L. Bishop, J. Z. Zhang, S. Gurumurthy, K. Tracy, and Z. Manchester, "Relu-qp: A gpu-accelerated quadratic programming solver for model-predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 13 285–13 292.

[24] AMD, Inc., *Vitis High-Level Synthesis User Guide (UG1399)*, Advanced Micro Devices (AMD), 2026, version 2025.2. [Online]. Available: https://docs.amd.com/r/en-US/ug1399-vitis-hls

[25] Bitcraze AB, "Lighthouse positioning deck," https://store.bitcraze.io/products/lighthouse-positioning-deck, accessed: 2026-04-08. Features an onboard FPGA for real-time signal processing, Lattice iCE40UP5K device.