2024 IEEE International Conference on Robotics and Automation (ICRA)
May 13-17, 2024. Yokohama, Japan

2024 IEEE International Conference on Robotics and Automation (ICRA) | 979-8-3503-8457-4/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICRA57147.2024.10610987

TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers

Khai Nguyen*, Sam Schoedel*, Anoushka Alavilli*, Brian Plancher, Zachary Manchester

***Abstract*— Model-predictive control (MPC) is a powerful tool for controlling highly dynamic robotic systems subject to complex constraints. However, MPC is computationally demanding, and is often impractical to implement on small, resource-constrained robotic platforms. We present TinyMPC, a high-speed MPC solver with a low memory footprint targeting the microcontrollers common on small robots. Our approach is based on the alternating direction method of multipliers (ADMM) and leverages the structure of the MPC problem for efficiency. We demonstrate TinyMPC's effectiveness by benchmarking against the state-of-the-art solver OSQP, achieving nearly an order of magnitude speed increase, as well as through hardware experiments on a 27 gram quadrotor, demonstrating high-speed trajectory tracking and dynamic obstacle avoidance. TinyMPC is publicly available at `https://tinympc.org`.**

I. Introduction

Model-predictive control (MPC) enables 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 [1], [2], [3], [4]. However, despite MPC's many successes, its practical application is often hindered by computational limitations, which can necessitate algorithmic simplifications [5], [6]. This challenge is amplified when dealing with systems that have fast or unstable open-loop dynamics, where high control rates are needed for safe and effective operation.

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 [7] to routine monitoring and maintenance of infrastructure and equipment [8], [9]. These robots are limited to low-power, resource-constrained microcontrollers (MCUs) for their computation [10], [11]. As shown in Figure 2, these microcontrollers feature orders of magnitude less RAM, flash memory, and processor speed compared to the CPUs and GPUs available on larger robots and historically were not able to support the real-time execution of computationally or memory-intensive algorithms [12], [13]. Consequently, many examples in the literature of intelligent robot behaviors executed on these tiny platforms rely on off-board computers [14], [15], [16], [17], [18], [19], [20].

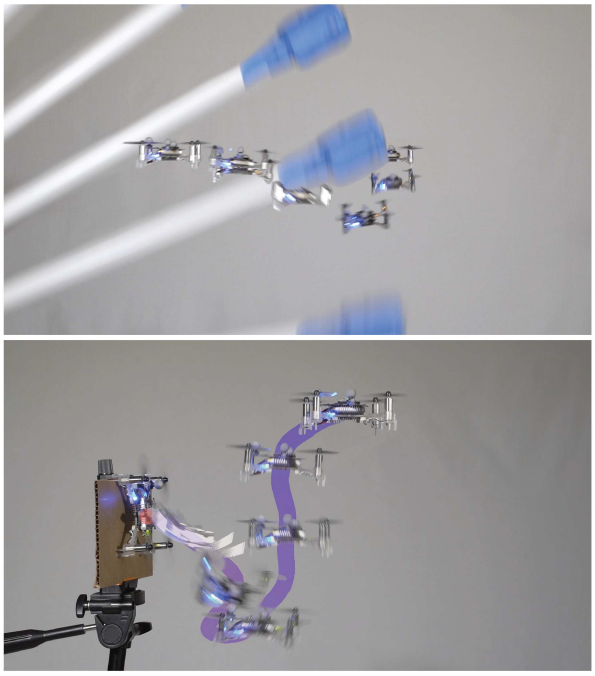

Fig. 1. TinyMPC is a fast convex model-predictive control solver that enables real-time optimal control on resource-constrained microcontrollers. We demonstrate its efficacy in dynamic obstacle avoidance (top) and recovery from 90° attitude errors (bottom) on a 27 gram Crazyflie quadrotor.

Several efficient optimization solvers suitable for embedded MPC have emerged in recent years, most notably OSQP [29] and CVXGEN [30]. Both of these solvers have code-generation tools that enable users to create dependency-free C code to solve quadratic programs (QPs) on embedded computers. However, they do not take full advantage of the unique structure of the MPC problem, generally making them too memory intensive and too computationally demanding to run within the resource constraints of many microcontrollers.

On the other hand, the recent success of "TinyML" has enabled the deployment of neural networks on microcontrollers [12]. Motivated by these results, we developed TinyMPC, an MCU-optimized implementation of convex MPC using the alternating direction method of multipliers (ADMM) algorithm. At its core, our solver is designed to *accelerate and compress* the ADMM algorithm by *exploiting the structure* of the MPC problem.

In particular, we precompute and cache expensive matrix factorizations, allowing TinyMPC to completely avoid online division and matrix inversion. This approach facilitates rapid

∗These authors contributed equally.
Khai Nguyen is with the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. xuankhan@andrew.cmu.edu
Anoushka Alavilli is with the Department of Electrical & Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. apalavil@andrew.cmu.edu
Samuel Schoedel and Zachary Manchester are with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. {sschoede, zmanches}@andrew.cmu.edu
Brian Plancher is with Barnard College, Columbia University, 3009 Broadway New York, NY 10027. bplancher@barnard.edu

	Micro Platforms		Tiny Platforms				Full-Scale Platforms	
	Robobee 	HAMR-F 	Crazyflie2.1 	DeepPicar Micro 	PIXHAWK PX4 	Petoi Bittle 	Snapdragon Flight 	Unitree Go1edu 
Processor	ATtiny20 4-8 MHz 8-bit MCU	ATmega1284RF2 16MHz 8-bit MUC	STM32F405 168 MHz 32-bit M4 MCU	RP2040 133 MHz Dual-Core 32-bit M0+ MCU	STM32F765 216 MHz Dual-Core 32-bit M7 MCU	ESP32-WROOM-32D 240MHz Dual-Core 32-bit LX7 MCU	Qualcomm Snapdragon 801 2.15 GHz Quad-Core 32-bit CPU 450 MHz 32-pipeline GPU	Jetson Nano (x3) 1.43 GHz Quad-Core 64-bit CPU 921 MHz 128-core GPU
RAM	128 B	16 kB	196 kB	264 kB	512 kB	512 kB	2 GB	4 GB (x3)
Flash	2 kB	128 kB	1 MB	2 MB	2 MB	16 MB	32 GB	64-256 GB (via SD card x3)
Processor Power	0.015 W	0.045 W (with RF)	0.15 W	0.15 W	0.5 W	0.5-1 W	3-10 W	5-10 W (x3)

Fig. 2. A comparison of micro, tiny, and full-scale robot platforms and their associated computational hardware. At the smallest scale, microrobots like the Robobee [21] and HAMR-F [22] use highly constrained 8-bit microcontrollers to execute pre-planned open-loop gaits or wing motions. At large scales, powerful embedded CPUs and GPUs, found onboard the Snapdragon Flight quadrotor [23] or Unitree Go1edu quadruped [24], enable high performance at the cost of high power requirements. In this work we target tiny robots such as the Crazyflie2.1 [25], DeepPiCarMicro [26], PIXHAWK PX4 [27], and Petoi Bittle [28] that leverage 32-bit microcontrollers for motion planning and control. These devices are capable of some onboard computation, but feature orders of magnitude less processor speed, RAM, and flash memory than full-scale robots.

computation with a very small memory footprint, enabling deployment on resource-constrained MCUs. 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. Our contributions include:

- A novel quadratic-programming algorithm that is optimized for MPC, is matrix-inversion free, and achieves high efficiency and a very low memory footprint. This combination makes it suitable for deployment on resource-constrained microcontrollers.
- An open-source implementation of TinyMPC in C++ that delivers state-of-the-art real-time performance for convex MPC problems on microcontrollers.
- Experimental demonstrations on a small, agile, resource-constrained quadrotor platform.

This paper proceeds as follows: Section II reviews linear-quadratic optimal control, convex optimization, and ADMM. Section III then derives the core TinyMPC solver algorithm. Benchmarking results and hardware experiments on a Crazyflie quadrotor are presented in Section IV. Finally, we summarize our results and conclusions in Section V.

II. Background

A. The Linear-Quadratic Regulator

The linear-quadratic regulator (LQR) [31] is a widely used approach for solving robotic control problems. LQR optimizes a quadratic cost function subject to a set of linear dynamics constraints:

$$\begin{aligned}\min_{x_{1:N},u_{1:N-1}} \; & J = \frac{1}{2}x_N^\intercal Q_f x_N + q_f^\intercal x_N + \\ & \sum_{k=1}^{N-1} \frac{1}{2}x_k^\intercal Q x_k + q_k^\intercal x_k + \frac{1}{2}u_k^\intercal R u_k + r_k^\intercal u_k \\ \text{subject to} \; & x_{k+1} = Ax_k + Bu_k \quad \forall k \in [1, N),\end{aligned} \tag{1}$$

where $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$ are the state and control input at time step k, N is the number of time steps (also referred to as the horizon), $A \in \mathbb{R}^{n\times n}$ and $B \in \mathbb{R}^{n\times m}$ define the system dynamics, $Q \succeq 0$, $R \succ 0$, and $Q_f \succeq 0$ are symmetric cost matrices and q and r are linear cost vectors.

Equation (1) has a closed-form solution in the form of an affine feedback controller [31]:

$$u_k^* = -K_k x_k - d_k. \tag{2}$$

The feedback gain K_k and feedforward term d_k can be obtained by solving the discrete Riccati equation backward in time, starting with $P_N = Q_f$ and $p_N = q_f$, where P_k and p_k are the quadratic and linear terms of the cost-to-go (or value) function [31]:

$$\begin{aligned} K_k &= (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal P_{k+1} A) \\ d_k &= (R + B^\intercal P_{k+1} B)^{-1}(B^\intercal p_{k+1} + r_k) \\ P_k &= Q + K_k^\intercal R K_k + (A - BK_k)^\intercal P_{k+1}(A - BK_k) \\ p_k &= q_k + (A - BK_k)^\intercal (p_{k+1} - P_{k+1} B d_k) + \\ &\quad K_k^\intercal (R d_k - r_k). \end{aligned} \tag{3}$$

B. Convex Model-Predictive Control

Convex MPC extends the LQR formulation to admit additional convex constraints on the system states and control inputs, such as joint and torque limits, hyperplanes for obstacle avoidance, and contact constraints:

$$\begin{aligned}\min_{x_{1:N},u_{1:N-1}} \; & J(x_{1:N}, u_{1:N-1}) \\ \text{subject to} \; & x_{k+1} = Ax_k + Bu_k, \\ & x_k \in \mathcal{X}, u_k \in \mathcal{U},\end{aligned} \tag{4}$$

where $\mathcal{X}$ and $\mathcal{U}$ are convex sets. The convexity of this problem means that it can be solved efficiently and reliably, enabling real-time deployment in a variety of control applications including the landing of rockets [32], legged locomotion [33], and autonomous driving [34].

When $\mathcal{X}$ and $\mathcal{U}$ can be expressed as linear constraints, (4) is a QP, and can be put into the standard form:

$$\begin{aligned}\min_{x\in\mathbb{R}^n} \; & \frac{1}{2}x^\intercal P x + q^\intercal x \\ \text{subject to} \; & Ax \le b, \\ & Cx = d.\end{aligned} \tag{5}$$

Further analysis, including theoretical guarantees regarding feasibility, stability, and design practices are elaborated upon in [35] and [36].

C. The Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) [37], [38], [39] is a popular and efficient approach for solving convex optimization problems, including QPs like (5). We provide a very brief summary here and refer readers to [40] for more details.

Given a generic problem:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & x \in \mathcal{C}, \end{aligned} \tag{6}$$

with f and $\mathcal{C}$ convex, we define the indicator function for the set $\mathcal{C}$:

$$I_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ \infty & \text{otherwise.} \end{cases} \tag{7}$$

We can now form the following equivalent problem by introducing the slack variable z:

$$\begin{aligned} \min_{x} \quad & f(x) + I_{\mathcal{C}}(z) \\ \text{subject to} \quad & x = z. \end{aligned} \tag{8}$$

The augmented Lagrangian of the transformed problem (8) is as follows, where λ is a Lagrange multiplier and ρ is a scalar penalty weight:

$$\mathcal{L}_A(x, z, \lambda) = f(x) + I_{\mathcal{C}}(z) + \lambda^\intercal (x - z) + \frac{\rho}{2} ||x - z||_2^2. \tag{9}$$

If we alternate minimization over x and z, rather than simultaneously minimizing over both, we arrive at the three-step ADMM iteration,

$$\text{primal update} : x^+ = \arg\min_x \mathcal{L}_A(x, z, \lambda), \tag{10}$$

$$\text{slack update} : z^+ = \arg\min_z \mathcal{L}_A(x^+, z, \lambda), \tag{11}$$

$$\text{dual update} : \lambda^+ = \lambda + \rho(x^+ - z^+), \tag{12}$$

the last step of which is a gradient-ascent update on the Lagrange multiplier [39]. These steps can be iterated until a desired convergence tolerance is achieved.

In the special case of a QP, 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 projection. ADMM-based QP solvers, like OSQP [29], have demonstrated state-of-the-art results.

III. The TinyMPC Solver

TinyMPC trades generality for speed by exploiting the special structure of the MPC problem. Specifically, we leverage the closed-form Riccati solution to the LQR problem to compute the primal update in (10). Pre-computing and caching this solution allows us to avoid online matrix factorizations and enables very fast performance while maintaining a small memory footprint.

A. Combining LQR and ADMM for MPC

We solve the following problem, introducing slack variables as in (9) and transforming (4) into the following:

$$\begin{aligned} \min_{\substack{x_{1:N}, z_{1:N}, \\ \lambda_{1:N}, u_{1:N-1}, \\ w_{1:N-1}, \mu_{1:N-1}}} \quad & \mathcal{L}_A(\cdot) = J(x_{1:N}, u_{1:N-1}) + \\ & I_{\mathcal{X}}(z_{1:N}) + I_{\mathcal{U}}(w_{1:N-1}) + \\ & \sum_{k=1}^{N} \frac{\rho}{2}(x_k - z_k)^\intercal (x_k - z_k) + \lambda_k^\intercal (x_k - z_k) + \\ & \sum_{k=1}^{N-1} \frac{\rho}{2}(u_k - w_k)^\intercal (u_k - w_k) + \mu_k^\intercal (u_k - w_k) \\ \text{subject to:} \quad & x_{k+1} = Ax_k + Bu_k, \end{aligned} \tag{13}$$

where z, w, λ, and μ are the state slack, input slack, state dual, and input dual variables over the entire horizon. State and input constraints are enforced through the indicator functions $I_{\mathcal{X}}$ and $I_{\mathcal{U}}$. We use the ADMM algorithm (10), (11), (12) to solve this optimal control problem. The primal update for (13) becomes an equality-constrained QP:

$$\begin{aligned} \min_{x_{1:N}, u_{1:N-1}} \quad & \frac{1}{2} x_N^\intercal \tilde{Q}_f x_N + \tilde{q}_f^\intercal x_N + \\ & \sum_{k=1}^{N-1} \frac{1}{2} x_k^\intercal \tilde{Q} x_k + \tilde{q}_k^\intercal x_k + \frac{1}{2} u_k^\intercal \tilde{R} x_k + \tilde{r}^\intercal u_k \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, \end{aligned} \tag{14}$$

where

$$\begin{aligned} \tilde{Q}_f &= Q_f + \rho I, & \tilde{q}_f &= q_f + \lambda_N - \rho z_N, \\ \tilde{Q} &= Q + \rho I, & \tilde{q}_k &= q_k + \lambda_k - \rho z_k, \\ \tilde{R} &= R + \rho I, & \tilde{r}_k &= r_k + \mu_k - \rho w_k. \end{aligned} \tag{15}$$

We reformulate (15) and introduce the scaled dual variables y and g for convenience [39]:

$$\begin{aligned} \tilde{q}_f &= q_f + \rho(\lambda_N/\rho - z_N) = q_f + \rho(y_N - z_N), \\ \tilde{q}_k &= q_k + \rho(\lambda_k/\rho - z_k) = q_k + \rho(y_k - z_k), \\ \tilde{r}_k &= r_k + \rho(\mu_k/\rho - w_k) = r_k + \rho(g_k - w_k). \end{aligned} \tag{16}$$

We observe that, because (14) exhibits the same LQR problem structure as in (1), it can be solved efficiently with the Riccati recursion in (3). The slack update for (13) becomes a simple linear projection onto the feasible set:

$$\begin{aligned} z_k^+ &= \text{proj}_{\mathcal{X}}(x_k^+ + y_k), \\ w_k^+ &= \text{proj}_{\mathcal{U}}(u_k^+ + g_k), \end{aligned} \tag{17}$$

where the superscript denotes the variable at the subsequent ADMM iteration. The dual update for (13) becomes:

$$\begin{aligned} y_k^+ &= y_k + x_k^+ - z_k^+, \\ g_k^+ &= g_k + u_k^+ - w_k^+. \end{aligned} \tag{18}$$

Finally, the algorithm terminates when the primal and dual residuals are within a set tolerance.

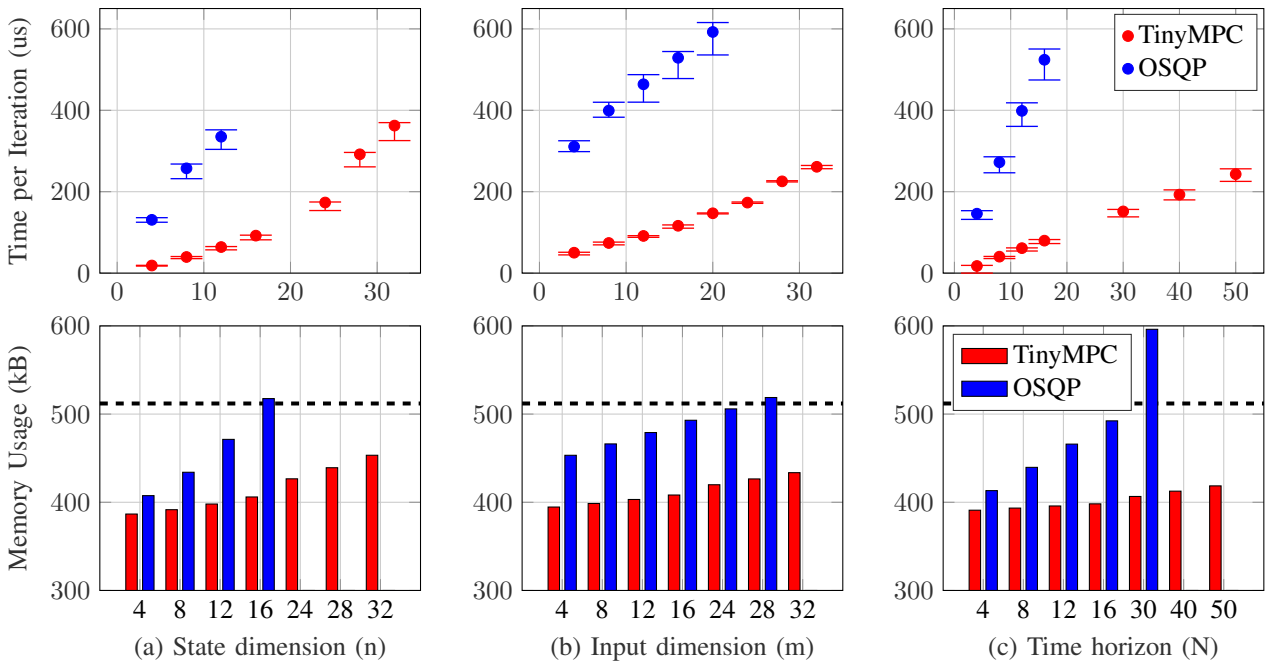

Fig. 3. Comparison of average iteration times (top) and memory usage (bottom) for OSQP and TinyMPC on randomly generated trajectory tracking problems on a Teensy 4.1 development board (ARM Cortex-M7 running at 600 MHz with 32-bit floating point support, 7.75 MB of flash, and 512 kB of tightly coupled RAM). Error bars show the maximum and minimum time per iteration over all MPC steps executed for a given problem. In (a), the input dimension and time horizon are held constant at $m = 4$ and $N = 10$ while the state dimension n varies from 4 to 32. In (b), $n = 10$ and $N = 10$ while the m varies from 4 to 32. In (c), $n = 10, m = 4$ and N varies from 4 to 50. The dotted black line indicates the memory limit of the Teensy 4.1.

B. Pre-Computation

Solving the linear system in each primal update is the most expensive step in each ADMM iteration. In our case, this is the solution to the Riccati equation, which has properties we can leverage to significantly reduce computation and memory usage. Given a long enough horizon, the Riccati recursion (3) converges to the constant solution of the infinite-horizon LQR problem [31]. Thus, we pre-compute a single LQR gain matrix K_{inf} and cost-to-go Hessian P_{inf}. We then cache the following matrices from (3):

$$\begin{aligned} C_1 &= (R + B^\intercal P_{\text{inf}} B)^{-1}, \\ C_2 &= (A - BK_{\text{inf}})^\intercal. \end{aligned} \tag{19}$$

A careful analysis of the Riccati equation then reveals that only the linear terms need to be updated as part of the ADMM iteration:

$$\begin{aligned} d_k &= C_1(B^\intercal p_{k+1} + r_k), \\ p_k &= q_k + C_2 p_{k+1} - K_{\text{inf}}^\intercal r_k. \end{aligned} \tag{20}$$

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

C. Penalty Scaling

ADMM is sensitive to the value of the penalty term ρ in (9). Solvers like OSQP [29] overcome this issue by adaptively scaling ρ. However, this requires performing additional matrix factorizations. To avoid this, we pre-compute and cache a set of matrices corresponding to several values of ρ. Online, we switch between these cached matrices based on the values of the primal and dual residual values in a scheme adapted from OSQP. The resulting TinyMPC algorithm is summarized in Algorithm 1.

Algorithm 1 TinyMPC

function TINY_SOLVE(input)
  while not converged **do**
    `//Primal update`
    $p_{1:N-1}, d_{1:N-1} \leftarrow$ Backward pass via (20)
    $x_{1:N}, u_{1:N-1} \leftarrow$ Forward pass via (2)
    `//Slack update`
    $z_{1:N}, w_{1:N-1} \leftarrow$ Project to feasible set (17)
    `//Dual update`
    $y_{1:N}, g_{1:N-1} \leftarrow$ Gradient ascent (18)
    $q_{1:N}, r_{1:N-1}, p_N \leftarrow$ Update linear cost terms
  return $x_{1:N}, u_{1:N-1}$

IV. Experiments

We evaluate TinyMPC through two sets of experiments: first, we benchmark our solver against the state-of-the-art OSQP [29] solver on a representative microcontroller, demonstrating improved computational speed and reduced memory footprint. We then test the efficacy of our solver on a resource-constrained nano-quadrotor platform, the Crazyflie 2.1. We show that TinyMPC enables the Crazyflie to track aggressive reference trajectories while satisfying control limits and time-varying state constraints.

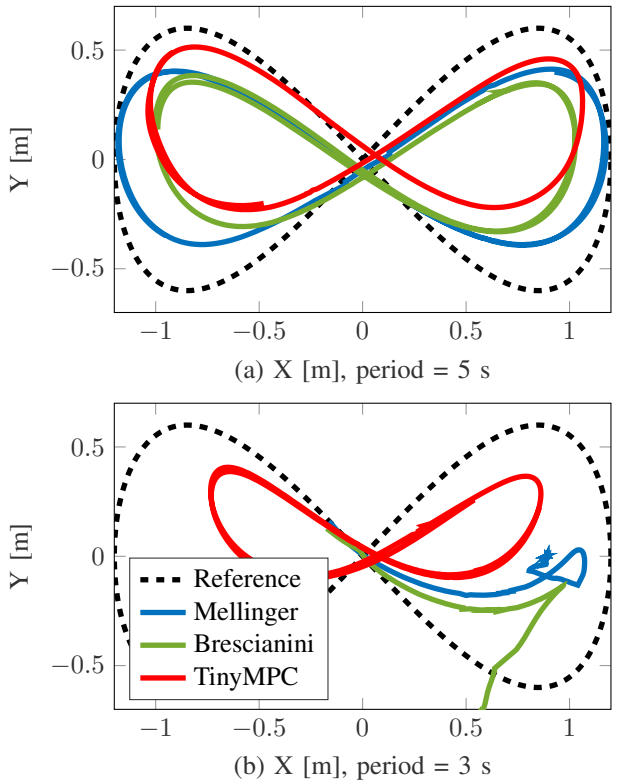

Fig. 4. Figure-eight tracking at low speed (top) and high speed (bottom) comparing TinyMPC with the two most performant controllers available on the Crazyflie. For slower trajectories, all three controllers resulted in similar performance. For faster trajectories, only TinyMPC was capable of maintaining tracking without crashing. The maximum velocity and attitude deviation from hover with TinyMPC reached 1.5 m/s and 20°, respectively.

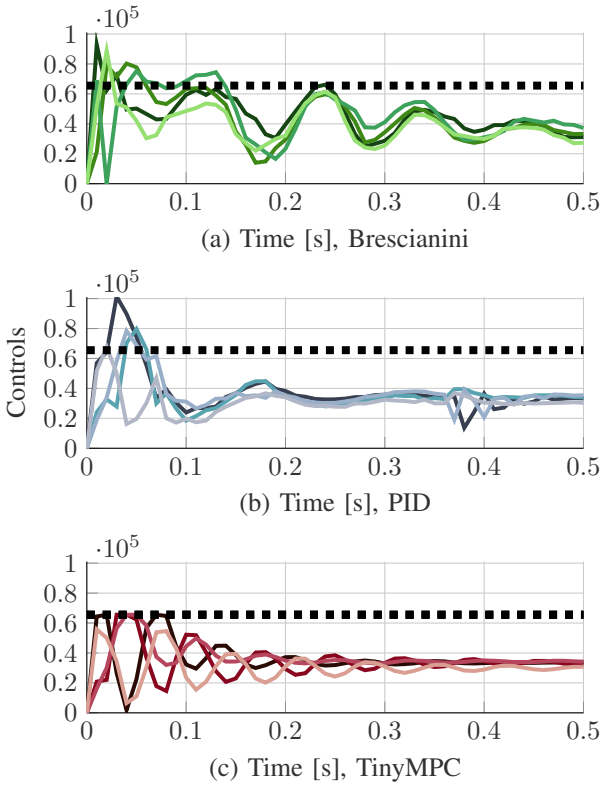

Fig. 5. Control trajectories during the Extreme Initial Poses experiment. Four pre-clipped PWM motor commands are shown for each controller. The black dotted line denotes the thrust limit, from 0 to 65535 PWM value. Among the three successful controllers, only TinyMPC could reason about control feasibility, exhibiting the maneuver shown in Fig. 1 (bottom).

A. Microcontroller Benchmarks

As shown in Fig. 3, we first compare TinyMPC and OSQP on random linear MPC problems while varying the state and input dimensions, as well as the horizon length.

1) Methodology: Experiments were performed on a Teensy 4.1 [41] development board, which has an ARM Cortex-M7 microcontroller operating at 600 MHz, 7.75 MB of flash memory, and 512 kB of RAM. TinyMPC is implemented in C++ using the Eigen matrix library [42]. We used OSQP's code-generation feature to generate a C implementation of each problem to run on the microcontroller. Objective tolerances were set to 10^{-3} and constraint tolerances to 10^{-4}. The maximum number of iterations for both solvers was set to 4000, and both utilized warm starting. OSQP's solution polishing was disabled to decrease solve time. Other parameters were set to equivalent values wherever possible.

Dynamics models A and B were randomly generated and checked to ensure controllability for all values of state dimension n, input dimension m, and time horizon N. The control input was constrained within fixed bounds over the entire horizon. During the microcontroller tests, noise was added to mimic imperfect state estimation. The largest problem instance involved 696 decision variables, 490 linear equality constraints, and 392 linear inequality constraints.

2) Evaluation: Fig. 3 shows the average execution times for both solvers, in which TinyMPC exhibits a maximum speed-up of 8.85x over OSQP. This speed-up allows TinyMPC to perform real-time trajectory tracking while handling input constraints. OSQP also quickly exceeded the memory limitations of the MCU, while TinyMPC was able to scale to much larger problem sizes. For example, for a fixed input dimension of $m = 4$ and time horizon of $N = 10$ (Fig. 3a), OSQP exceeded the 512 kB memory limit of the Teensy at a state dimension of only $n = 16$, while TinyMPC only used around 400 kB at a state dimension of $n = 32$.

B. Hardware Experiments

We demonstrate the efficacy of our solver for real-time execution of dynamic control tasks on a resource-constrained Crazyflie 2.1 quadrotor. We present three experiments: 1) figure-eight trajectory tracking at slow and fast speeds, 2) recovery from extreme initial attitudes, and 3) dynamic obstacle avoidance through online updating of state constraints.

1) Methodology: The Crazyflie 2.1 is a 27 gram quadrotor. Its primary MCU is an ARM Cortex-M4 (STM32F405) clocked at 168 MHz with 192 kB of SRAM and 1 MB of flash. OSQP could not fit within the memory available on this MCU, making it impossible to be used as an MPC baseline. Instead, we compare against the four controllers

included with the Crazyflie firmware: Cascaded PID [43], Mellinger [44], INDI [45], and Brescianini [46]. These are reactive controllers that often clip the control input to meet hardware constraints.

All experiments shown were performed in an OptiTrack motion-capture environment sending pose data to the Crazyflie at 100 Hz. TinyMPC ran at 500 Hz with a horizon length of $N = 15$ for the figure-eight tracking task and the attitude-recovery task. For the obstacle-avoidance task, we sent the location of the end of a stick to the Crazyflie using the onboard radio. Additionally, we reduced the MPC frequency to 100 Hz and increased N to 20.

In all experiments, we linearized the quadrotor's 6-DOF dynamics about a hover and represented its attitude with a quaternion using the formulation in [47]. This problem has state dimension $n = 12$ and $m = 4$ representing the quadrotor's full state and PWM motor commands. The largest problem was in the dynamic obstacle avoidance scenario, which was solved onboard at high frequency and consisted of 316 decision variables, 248 linear equality constraints, and 172 linear inequality constraints.

2) Evaluation—Figure-Eight Trajectory Tracking: We compare the tracking performance of TinyMPC and other controllers with a figure-eight trajectory, as shown in Fig. 4. For the faster trajectory, the maximum velocity and attitude deviation reached 1.5 m/s and 20°, respectively. Only TinyMPC could track the entire reference while respecting actuator limits, while the Mellinger and Brescianini controllers crashed almost immediately. TinyMPC converged at all steps within a maximum of 7 iterations and under the allotted 2 ms solve time defined by the 500 Hz control frequency.

3) Evaluation—Extreme Initial Poses: Fig. 1 (bottom) shows the performance of the Crazyflie when initialized with a 90° attitude error. TinyMPC displayed the best recovery performance with a maximum position error of 23 cm while respecting the input limits. The PID and Brescianini controllers achieved maximum errors of 40 cm and 65 cm, respectively, while violating input limits (Fig. 5). The other controllers, INDI and Mellinger, failed to stabilize the quadrotor, causing it to crash.

4) Evaluation—Dynamic Obstacle Avoidance: We demonstrate TinyMPC's ability to handle time-varying state constraints by avoiding a moving stick (Fig. 1 top). These experiments are more challenging because the constraints arbitrarily switch between inactive and active, requiring far more iterations to solve to convergence. The obstacle sphere was re-linearized about its updated position at each MPC step, allowing the drone to avoid the unplanned movements of the swinging stick. As illustrated, the quadrotor could move freely in space to avoid the dynamic obstacle and come back safely to the hovering position. As an additional challenge, we added a constraint such that the quadrotor must stay within a vertical plane defined by $x = 0$. The Crazyflie deviated a maximum of approximately 5 cm from this constraint plane while successfully avoiding the dynamic obstacle.

V. Conclusion and Future Work

We introduce TinyMPC, a model-predictive control solver for resource-constrained embedded systems. TinyMPC uses ADMM to handle state and input constraints while leveraging the structure of the MPC problem and insights from LQR to reduce memory footprint and speed up online execution compared to existing state-of-the-art solvers like OSQP. We demonstrated TinyMPC's practical performance on a Crazyflie nano-quadrotor performing highly dynamic tasks with input and obstacle constraints.

Several directions for future work remain. First, it should be straightforward to extend TinyMPC to handle second-order cone constraints, which are useful in many MPC applications for modeling thrust and friction. We also plan to further reduce TinyMPC's hardware requirements by developing a fixed-point version, since many small microcontrollers lack hardware floating-point support. Finally, to ease deployment and adoption, we plan to develop a code-generation wrapper for TinyMPC in a high-level language such as Julia or Python, similar to OSQP and CVXGEN.

For more information and to get started using our open-source solver, we recommend users visit our website, https://tinympc.org.

VI. Acknowledgments

The authors would like to thank Brian Jackson for insightful discussions and Professor Mark Bedillion for providing us with extra Crazyflies.

References

[1] P. M. Wensing, M. Posa, Y. Hu, A. Escande, N. Mansard, and A. Del Prete, "Optimization-based control for dynamic legged robots," *arXiv preprint arXiv:2211.11644*, 2022.

[2] J. Di Carlo, "Software and control design for the mit cheetah quadruped robots," Ph.D. dissertation, Massachusetts Institute of Technology, 2020.

[3] Z. Manchester, N. Doshi, R. J. Wood, and S. Kuindersma, "Contact-implicit trajectory optimization using variational integrators," *The International Journal of Robotics Research*, vol. 38, no. 12-13, pp. 1463–1476, 2019.

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

[5] B. Plancher and S. 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*. Springer, 2020, pp. 656–672.

[6] S. M. Neuman, B. Plancher, T. Bourgeat, T. Tambe, S. Devadas, and V. J. Reddi, "Robomorphic computing: A design methodology for domain-specific accelerators parameterized by robot morphology," ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 674–686. [Online]. Available: https://doi-org.ezp-prod1.hul.harvard.edu/10.1145/3445814.3446746

[7] K. McGuire, C. De Wagter, K. Tuyls, H. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics*, vol. 4, no. 35, p. eaaw9710, 2019.

[8] S. D. De Rivaz, B. Goldberg, N. Doshi, K. Jayaram, J. Zhou, and R. J. Wood, "Inverted and vertical climbing of a quadrupedal microrobot using electroadhesion," *Science Robotics*, vol. 3, no. 25, p. eaau3038, 2018.

[9] B. P. Duisterhof, S. Li, J. Burgués, V. J. Reddi, and G. C. 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)*. IEEE, 2021, pp. 9099–9106.

[10] W. G. et al. Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering. [Online]. Available: https://www.bitcraze.io/papers/giernacki_draft_crazyflie2.0.pdf

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

[12] S. M. Neuman, B. Plancher, B. P. Duisterhof, S. Krishnan, C. Banbury, M. Mazumder, S. Prakash, J. Jabbour, A. Faust, G. C. 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)*. IEEE, 2022, pp. 296–299.

[13] Z. Zhang, A. A. Suleiman, L. Carlone, V. Sze, and S. Karaman, "Visual-inertial odometry on chip: An algorithm-and-hardware co-design approach," 2017.

[14] V. Adajania, S. Zhou, S. Arun, and A. 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)*, 2023, pp. 1421–1427.

[15] P. Varshney, G. Nagar, and I. 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)*, 2019, pp. 43–50.

[16] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[17] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[18] L. Xi, X. Wang, L. Jiao, S. Lai, Z. Peng, and B. M. Chen, "Gto-mpc-based target chasing using a quadrotor in cluttered environments," *IEEE Transactions on Industrial Electronics*, vol. 69, no. 6, pp. 6026–6035, 2021.

[19] G. Torrente, E. Kaufmann, P. Föhn, and D. Scaramuzza, "Data-driven mpc for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3769–3776, 2021.

[20] K. Y. Chee, T. Z. Jiahao, and M. A. Hsieh, "Knode-mpc: A knowledge-based data-driven predictive control framework for aerial robots," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 2819–2826, 2022.

[21] N. T. Jafferis, E. F. Helbling, M. Karpelson, and R. J. Wood, "Untethered flight of an insect-sized flapping-wing microscale aerial vehicle," *Nature*, vol. 570, no. 7762, pp. 491–495, 2019.

[22] B. Goldberg, R. Zufferey, N. Doshi, E. F. Helbling, G. Whittredge, M. Kovac, and R. J. Wood, "Power and control autonomy for high-speed locomotion with an insect-scale legged robot," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 987–993, 2018.

[23] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and imu," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 404–411, 2016.

[24] Unitree, "Unitree go1," Accessed 2023, https://shop.unitree.com.

[25] Bitcraze, "Crazyflie 2.1," 2023. [Online]. Available: https://www.bitcraze.io/products/crazyflie-2-1/

[26] M. Bechtel, Q. Weng, and H. Yun, "Deeppicarmicro: Applying tinyml to autonomous cyber physical systems," in *2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. IEEE, 2022, pp. 120–127.

[27] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision," *Autonomous Robots*, vol. 33, pp. 21–39, 2012.

[28] Petoi, "Petoi bittle robot dog," Accessed 2023, https://www.petoi.com/products/petoi-bittle-robot-dog.

[29] 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, 2020.

[30] J. Mattingley and S. Boyd, "CVXGEN: A code generator for embedded convex optimization," in *Optimization Engineering*, pp. 1–27.

[31] F. L. Lewis, D. Vrabie, and V. Syrmos, "Optimal Control," 1 2012. [Online]. Available: https://doi.org/10.1002/9781118122631

[32] B. Açıkmeşe, J. M. Carson, and L. Blackmore, "Lossless convexification of nonconvex control bound and pointing constraints of the soft landing optimal control problem," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 6, pp. 2104–2113, 2013.

[33] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9.

[34] M. Babu, Y. Oza, A. K. Singh, K. M. Krishna, and S. Medasani, "Model predictive control for autonomous driving based on time scaled collision cone," in *2018 European Control Conference (ECC)*, 2018, pp. 641–648.

[35] A. Boccia, L. Grüne, and K. Worthmann, "Stability and feasibility of state constrained mpc without stabilizing terminal constraints," *Systems and Control Letters*, vol. 72, pp. 14–21, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167691114001595

[36] A. Mohammadi, H. Asadi, S. Mohamed, K. Nelson, and S. Nahavandi, "Optimizing model predictive control horizons using genetic algorithm for motion cueing algorithm," *Expert Systems with Applications*, vol. 92, pp. 73–81, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417417306000

[37] R. Glowinski and A. Marroco, "Sur l'approximation, par éléments finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problèmes de dirichlet non linéaires," *Revue française d'automatique, informatique, recherche opérationnelle. Analyse numérique*, vol. 9, no. R2, pp. 41–76, 1975.

[38] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximation," *Computers & mathematics with applications*, vol. 2, no. 1, pp. 17–40, 1976.

[39] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[40] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, 2011.

[41] "Teensy® 4.1." [Online]. Available: https://www.pjrc.com/store/teensy41.html

[42] G. Guennebaud, B. Jacob, *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[43] "Controllers in the Crazyflie — Bitcraze." [Online]. Available: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/

[44] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[45] E. J. J. Smeur, Q. Chu, and G. C. H. E. de Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, no. 3, pp. 450–461, 2016.

[46] D. Brescianini, M. Hehn, and R. D'Andrea, "Nonlinear quadrocopter attitude control," 2013.

[47] B. E. Jackson, K. Tracy, and Z. Manchester, "Planning with attitude," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5658–5664, 2021.