

# TinyMPC: Model-Predictive Control on Resource-Constrained Microcontrollers

Anoushka Alavilli\*, Khai Nguyen\*, Sam Schoedel\*, 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 both 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 g 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 processor speed, RAM ,and flash memory 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 of the examples

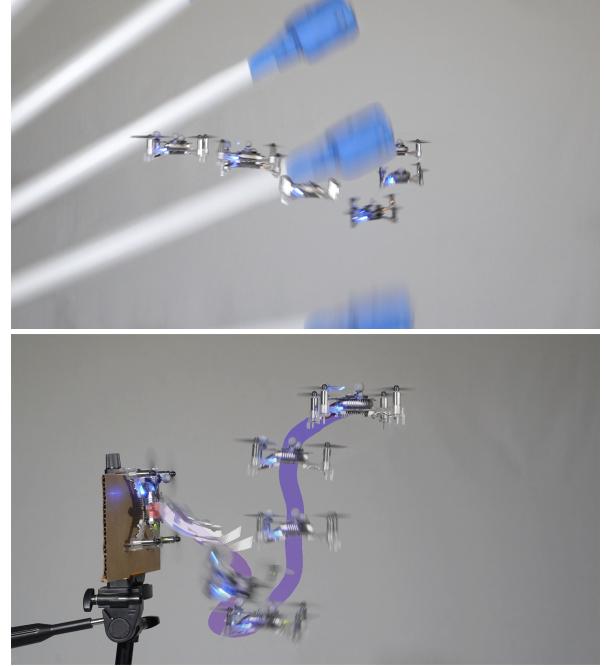


Fig. 1. TinyMPC is a fast convex MPC solver that enables real-time optimal control on resource-constrained microcontrollers. We demonstrate its efficacy by performing dynamic obstacle avoidance (top) and recover from 90° attitude errors (bottom) on a 27 g Crazyflie 2.1 quadrotor.

in the literature of intelligent robot behaviors being executed on these tiny platforms rely on off-board computers [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25].

Several efficient optimization solvers suitable for embedded MPC have emerged in recent years, most notably OSQP [34] and CVXGEN [35]. 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 and still have relatively large memory footprints, making them unable to run within the resource constraints of many microcontrollers.

Inspired by the recent success of “TinyML,” which has enabled the deployment of neural networks on microcontrollers [12], we introduce TinyMPC, an MCU-optimized implementation of convex MPC using the alternating direction method of multipliers (ADMM) algorithm. Our approach leverages the structure of the MPC problem by precomputing and caching as much as possible and completely avoiding divisions and matrix inversions online. This approach facilitates rapid computation and has a very small memory footprint, enabling deployment onto resource-constrained

\*These authors contributed equally and are ordered alphabetically.

Anoushka Alavilli is with the Department of Electrical & Computer Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. [apalavil@andrew.cmu.edu](mailto:apalavil@andrew.cmu.edu)

Khai Nguyen is with the Department of Mechanical Engineering, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. [xuankhan@andrew.cmu.edu](mailto:xuankhan@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](mailto:{sschoede, zmanches}@andrew.cmu.edu)

Brian Plancher is with Barnard College, Columbia University, 3009 Broadway New York, NY 10027. [bplancher@barnard.edu](mailto:bplancher@barnard.edu)

	Micro Platforms		Tiny Platforms				Full-Scale Platforms	
Processor	Robobee	HAMR-F	Crazyflie2.1	DeepPiCar Micro	PIXHAWK PX4	Petoi Bittle	Snapdragon Flight	Unitee GoTedu
RAM	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
Flash	2 kB	16 kB	196 kB	264 kB	512 kB	512 kB	2 GB	4 GB (x3) 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 [26] and HAMR-F [27] use highly constrained 8-bit microcontrollers to execute pre-planned open-loop cyclical gates and wing motions. At large scales, powerful embedded CPUs and GPUs, found onboard the Snapdragon Flight quadrotor [28] or Unitee GoTedu quadruped [29], enable high performance at the cost of high power requirements. In this work we target tiny robots such as the Crazyflie2.1 [30], DeepPiCarMicro [31], PIXHAWK PX4 [32], and Petoi Bittle [33] 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, as well as RAM and flash memory, than the powerful large-scale embedded CPUs and GPUs.

MCUs. To the best of the authors' knowledge, TinyMPC is the first MPC solver tailored for execution on 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 solver implementation of TinyMPC in C++ that delivers state-of-the-art real-time performance for convex MPC problems on microcontrollers.
- Experimental demonstration on a small, 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) [36] is a widely used approach for solving robotic control problems. LQR optimizes a quadratic cost function subject to a set of linear dynamics constraints:

$$\begin{aligned} \min_{x_{1:N}, u_{1:N-1}} J = & \frac{1}{2} x_N^\top Q_f x_N + q_f^\top x_N + \\ & \sum_{k=1}^{N-1} \frac{1}{2} x_k^\top Q x_k + q_k^\top x_k + \frac{1}{2} u_k^\top R u_k + r_k^\top u_k \end{aligned} \quad (1)$$

subject to  $x_{k+1} = Ax_k + Bu_k \quad \forall k \in [1, N]$ ,

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 weight matrices and  $q$  and  $r$  are the linear cost vectors.

Equation (1) has a closed-form solution in the form of a linear feedback controller [36]:

$$u_k^* = -K_k x_k - d_k, \quad (2)$$

$K_k$  and  $d_k$  can be obtained by solving the discrete Riccati equation backwards in time, starting with  $P_N = Q_f$  and  $p_N = q_f$ , where  $P_k$  and  $p_k$  are the Hessian and linear terms of the cost-to-go (or value) function [36]:

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

### 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:

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

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 [37], legged locomotion [38], and autonomous driving [39].

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

$$\begin{aligned} \min_{x \in \mathbb{R}^n} & \frac{1}{2} x^\top P x + q^\top x \\ \text{subject to } & Ax \leq b, \\ & Cx = d. \end{aligned} \quad (5)$$

### C. The Alternating Direction Method of Multipliers

The alternating direction method of multipliers (ADMM) [40], [41], [42] 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 [43] 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  $\lambda$  is a Lagrange multiplier and  $\rho$  is a scalar penalty weight:

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

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 dual-ascent update on the Lagrange multiplier [42]. 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 dual update is a linear projection. ADMM-based QP solvers, like OSQP [34], have demonstrated state-of-the-art results.

### III. THE TINY MPC 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 in the primal update of (10). Pre-computing and caching this solution allows us to avoid online matrix factorizations and enables very fast performance and a small memory footprint. We recommend users to check out our website, <https://tinympc.org>, to get started.

#### A. Combining LQR and ADMM for MPC

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

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

where  $z, w, \lambda, \mu$  are the state slack, input slack, state dual, and input dual variables over the entire horizon. The primal update for (13) becomes an equality-constrained QP:

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

$$\text{subject to } x_{k+1} = Ax_k + Bu_k,$$

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 leverage a scaled form of (17) by introducing the scaled dual variables  $y$  and  $g$  [42]:

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

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

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

Finally, the dual update for (13) simply becomes

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

#### B. Pre-Computation and Penalty Scaling

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 solution of the infinite-horizon LQR problem [36]. As such, we can pre-compute a single LQR gain matrix  $K_{\inf}$  and cost-to-go Hessian  $P_{\inf}$ . It is worth that the state weight  $Q$  is now enforced by  $P_{\inf}$  at each time step.

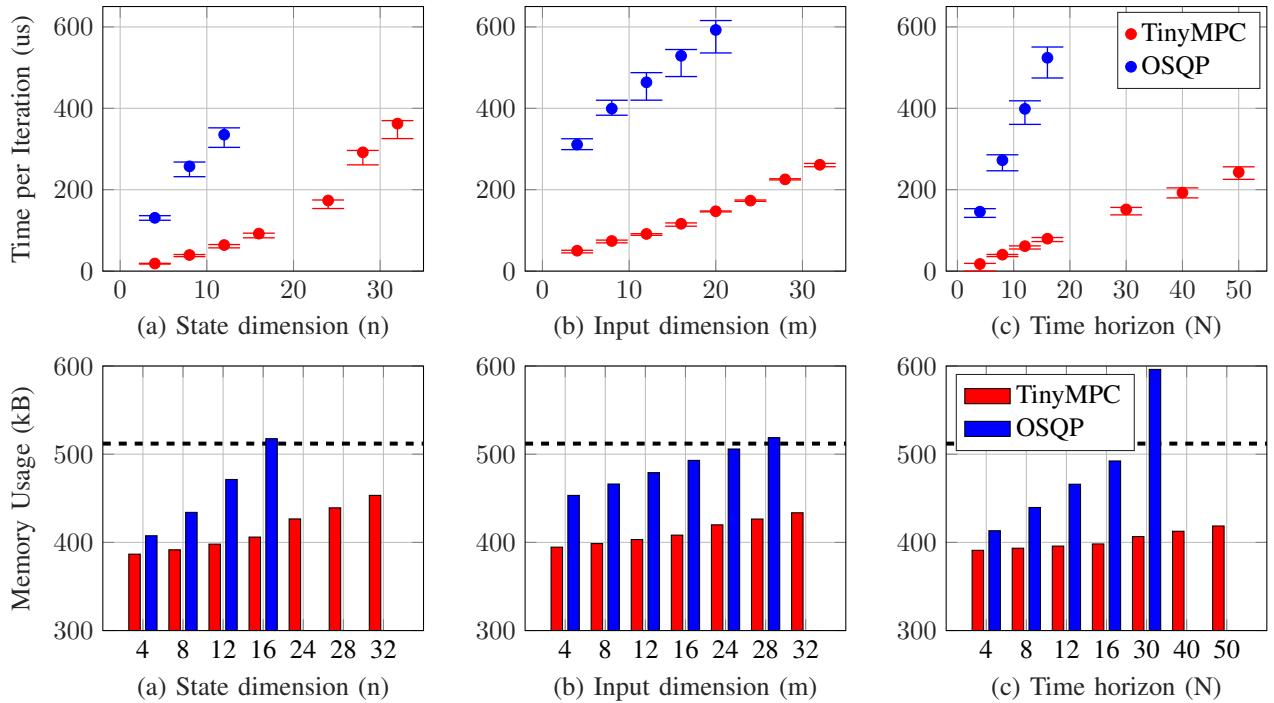


Fig. 3. Comparison of average iteration times (top) and memory usage (bottom) for OSQP and TinyMPC on randomly generated trajectory tracking problems on a Teensy 4.1 development board (ARM Cortex-M7 running at 600MHz with 32-bit floating point support, 7.75Mb of flash, and 512kB of tightly coupled RAM). 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.

We then cache the following matrices:

$$\begin{aligned} C_1 &= (R + B^\top P_{\text{inf}} B)^{-1} \\ C_2 &= (A - BK_{\text{inf}})^\top \\ C_3 &= K_{\text{inf}}^\top R - C_2 P_{\text{inf}} B \end{aligned} \quad (19)$$

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

$$\begin{aligned} d_k &= C_1(B^\top p_{k+1} + r_k) \\ p_k &= q_k + C_2 p_{k+1} - K_{\text{inf}}^\top r_k + C_3 d_k \end{aligned} \quad (20)$$

As a result, we can completely avoid matrix factorizations online and only compute matrix-vector products using the pre-computed matrices.

ADMM is also sensitive to the value of the penalty term  $\rho$  in (9). Adaptively scaling  $\rho$  is standard in solvers like OSQP [34]. However, this requires additional matrix factorizations that we are trying to avoid. Therefore, we pre-compute and cache a set of matrices corresponding to several values of  $\rho$ . Online, we switch between these cached matrices according to the primal and dual residual values, in a scheme adapted from OSQP. The resulting TinyMPC algorithm is summarized in Algorithm 1.

#### IV. EXPERIMENTS

We evaluate TinyMPC through two sets of experiments: First, we benchmark our solver against the state-of-the-art OSQP [34] solver on a representative microcontroller,

---

#### Algorithm 1 TinyMPC

---

```

1: function TINYMPG(input)
2:   while not converged do
3:     ▷Primal update
4:      $p_{1:N-1}, d_{1:N-1} \leftarrow$  Backward pass via (20)
5:      $x_{1:N}, u_{1:N-1} \leftarrow$  Forward pass via (2)
6:     ▷Slack update
7:      $z_{1:N}, w_{1:N-1} \leftarrow$  Projected to feasible set (17)
8:     ▷Dual update
9:      $y_{1:N}, g_{1:N-1} \leftarrow$  Gradient ascent (18)
10:     $q_{1:N}, r_{1:N-1}, p_N \leftarrow$  Update linear cost terms
return  $x_{1:N}, u_{1:N-1}$ 

```

---

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.

##### A. Microcontroller Benchmarks

We 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 are performed on a Teensy 4.1 [44] development board, which has an ARM Cortex-M7 microcontroller operating at 600MHz, 7.75MB of flash

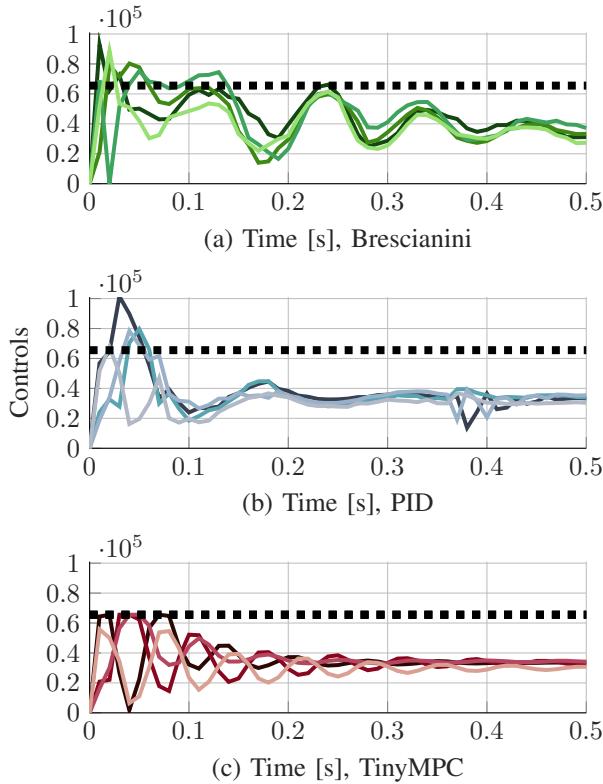


Fig. 4. Control trajectories during the Extreme Initial Poses experiment. Four sets of pre-clipped PWM Motor commands, ranging from 0 to 65535, are shown for each controller. The black dotted line is the thrust limit. Among the three successful controllers, only TinyMPC can guarantee feasible controls.

memory, and 512kB of RAM. TinyMPC is implemented in C++ using the Eigen matrix library [45]. We leverage OSQP’s code-generation feature to generate a C implementation of our problem to run on the microcontroller. Wherever possible, solver parameters were set to equivalent values. 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 make it faster. 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  $N$ .

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 and state 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  $N = 10$ , OSQP exceeds 512kB at only a state dimension of  $n = 16$ , while TinyMPC only used around 400kB at a state dimension of  $n = 32$ .

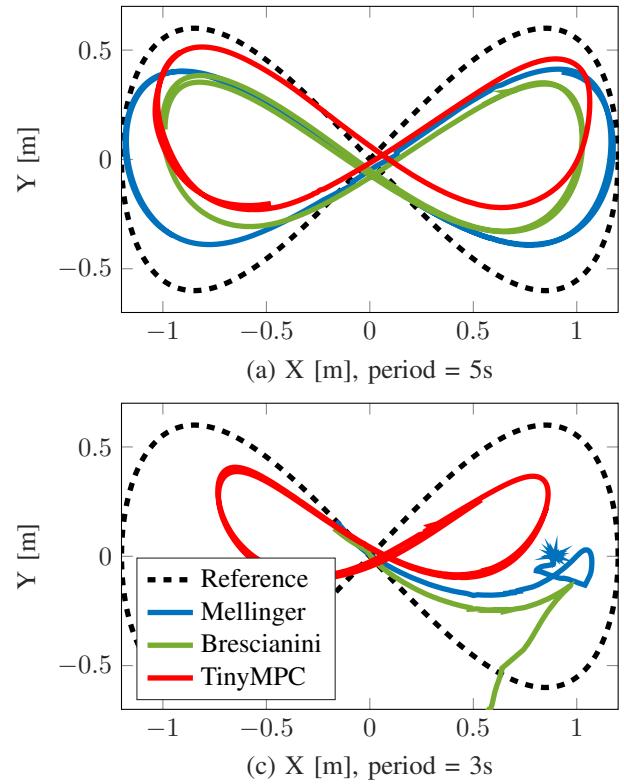


Fig. 5. Figure-eight tracking at low speed (top) and high speed (bottom) comparing TinyMPC with the two most performant available on the Crazyflie. For slower trajectories, all three controllers resulted in similar performance. For faster trajectories, only TinyMPC was capable of maintaining tracking without crashing. The maximum velocity and attitude deviation from hover with TinyMPC reached 1.5m/s and 20°, respectively.

## 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 g quadrotor. Its main MCU is an ARM Cortex-M4 (STM32F405) clocked at 168MHz with 192kB of SRAM and 1MB of flash. OSQP could not fit within the memory available on this MCU. Instead, we compare against the four controllers shipped with the Crazyflie firmware: Cascaded PID [46], Mellinger [47], INDI [48], and Brescianini [49]. These are reactive controllers that often clip the control input to meet hardware constraints.

All experiments shown were performed in an Opti-Track motion capture environment sending pose data to the Crazyflie at 100 Hz. We ran TinyMPC at 500Hz with the horizon length  $N = 15$  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 linearize the quadrotor’s dynamics about a hover and represent its attitude with a quaternion using the formulation in [50]. We solve a problem with state dimension  $n = 12$  and  $m = 4$  for the Crazyflie’s full state pose and four PWM motor control commands.

**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. 5. For the fast trajectory, the maximum velocity and attitude deviation reach 1.5 m/s and 20°, respectively. Only TinyMPC could track the entire reference, while the Mellinger and Brescianini controllers crashed almost immediately.

**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 achieved maximum errors of 40 cm and 65 cm, respectively, while violating input limits (Fig. 4). 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). The obstacle constraint was re-linearized about its updated position at each MPC step, thereby allowing the drone to avoid the unplanned movements of the swinging stick. To make it more challenging, we add an additional constraint of the quadrotor’s moving within a vertical plane. While avoiding the dynamic obstacle, the Crazyflie only makes a maximum deviation of approximately 5 cm from the vertical plane.

## V. CONCLUSIONS

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: It should be straight-forward to extend TinyMPC to handle second-order cone constraints, which are useful in many MPC applications for modeling thrust and friction cone constraints. 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, we plan to develop a code-generation wrapper for TinyMPC in a high-level language like Julia or Python, similar to OSQP and CVXGEN.

## VI. ACKNOWLEDGMENTS

The authors would like to thank Brian Jackson for insightful discussions during the initial stages of this work

and Professor Mark Bedillion for providing us with extra Crazyflies for the experiments.

## 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](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. Carbone, 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, “Knoden-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] A. Saviolo, G. Li, and G. Loianno, “Physics-inspired temporal learning of quadrotor dynamics for accurate model predictive trajectory tracking,” *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10 256–10 263, 2022.
- [22] T. Jin, X. Wang, H. Ji, J. Di, and H. Yan, “Collision avoidance for multiple quadrotors using elastic safety clearance based model predictive control,” in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 265–271.
- [23] R. M. Bena, S. Hossain, B. Chen, W. Wu, and Q. Nguyen, “A hybrid quadratic programming framework for real-time embedded safety-critical control,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 3418–3424.
- [24] C. Lerch, D. Dong, and I. Abraham, “Safety-critical ergodic exploration in cluttered environments via control barrier functions,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 10 205–10 211.
- [25] H.-T. Do and I. Prodan, “Experimental validation of an explicit flatness-based mpc design for quadcopter position tracking,” *arXiv preprint arXiv:2308.15946*, 2023.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] Unitree, “Unitree go1,” Accessed 2023, <https://shop.unitree.com>.
- [30] Bitcraze, “Crazyflie 2.1,” 2023. [Online]. Available: <https://www.bitcraze.io/products/crazyflie-2-1/>
- [31] 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.
- [32] L. Meier, P. Tanskanen, L. Heng, G. H. Lee, F. Fraundorfer, and M. Pollefeyns, “Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision,” *Autonomous Robots*, vol. 33, pp. 21–39, 2012.
- [33] Petoii, “Petoii bittle robot dog,” Accessed 2023, <https://www.petoii.com/products/petoii-bittle-robot-dog>.
- [34] 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.
- [35] J. Mattingley and S. Boyd, “CVXGEN: A code generator for embedded convex optimization,” in *Optimization Engineering*, pp. 1–27.
- [36] F. L. Lewis, D. Vrabie, and V. Syrmos, “Optimal Control,” 1 2012. [Online]. Available: <https://doi.org/10.1002/9781118122631>
- [37] B. Açıkmese, 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.
- [38] 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.
- [39] 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.
- [40] 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.
- [41] 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.
- [42] 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.
- [43] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, 2011.
- [44] “Teensy® 4.1.” [Online]. Available: <https://wwwpjrc.com/store/teensy41.html>
- [45] G. Guennebaud, B. Jacob, *et al.*, “Eigen v3,” <http://eigen.tuxfamily.org>, 2010.
- [46] “Controllers in the Crazyflie — Bitcraze.” [Online]. Available: <https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/>
- [47] 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.
- [48] 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.
- [49] D. Brescianini, M. Hehn, and R. D’Andrea, “Nonlinear quadrocopter attitude control,” 2013.
- [50] B. E. Jackson, K. Tracy, and Z. Manchester, “Planning with attitude,” *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5658–5664, 2021.