Bittle MPC: Model Predictive Control for Quadrupedal Gait Design without Force Sensing

Reid Graves
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: rgraves@andrew.cmu.edu

Brandon Woodward
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: bwoodwar@andrew.cmu.edu

Aatish Gupta
Mechanical Engineering
Carnegie Mellon University
Pittsburgh, PA 15213
Email: aatishg@andrew.cmu.edu

Fig. 1: The Petoi Bittle robot. A small, hand sized robot that is entirely open source, equipped with an ESP32 microcontroller. [4]

***Abstract*—This project investigates the feasibility of achieving stable quadrupedal locomotion without relying on force sensors, under strict computational constraints typical of microcontroller platforms such as the ESP32. We implement and evaluate two approaches: the MuJoCo Model Predictive Control (MJPC) framework for offline trajectory optimization and TinyMPC for efficient online control suitable for embedded deployment. Preliminary results show that stable locomotion is achievable without force sensing, and gaits optimized in MJPC will be executable in real time on Bittle's ESP32 hardware.**

I. Introduction

Quadrupedal robots often rely on force sensing and significant computational power to achieve stable gaits [1]–[3]. However, platforms such as the Bittle quadruped robot [4], which use microcontrollers like the ESP32, require lightweight and computationally efficient control strategies. The core question this research addresses is whether stable locomotion can be achieved without force sensing and with limited computational resources. Our Mujoco MJPC environment for the Bittle Robot is publicly available on GitHub at Bittle Mujoco MJPC

II. System Description

A. Robot Platform

The Petoi Bittle quadruped robot is equipped with eight actuated joints (four hips and four knees), giving it eight degrees of freedom (DoF). Many quadruped platforms support twelve DoF by adding four hip abduction joints that enable movement in three spatial directions, increasing agility and foot placement flexibility. In the Bittle robot, control is limited to two planes perpendicular to the robot's center of mass. Joint servos on the hardware are position-controlled via internal PID controllers. The ESP32 microcontroller sends pulse width modulation (PWM) commands over a one wire bus to the servos, which maintain the commanded angles.

The robot includes open loop built in gaits defined as arrays of predefined joint angles sent continuously until a new command is received. Locomotion gaits (e.g., crawling, walking) loop angle sequences, whereas stationary positions are executed once, with servos maintaining the target angle with ±1 degree precision within a $[-120, 120]$ degree range. These gaits rely on sliding feet along the ground to preserve a support polygon. On rough surfaces (e.g., concrete, carpet), foot catch can destabilize the robot and lead to tumbles.

In simulation, we model servos as PID controlled position actuators with absolute position feedback. While physical servos can measure position, one wire communication requires alternating control and feedback reads, introducing latency that is intractable for dynamic gait control.

III. Approach

A. Getting the Robot into Simulation

The Bittle Robot is an open source project, which means Petoi does not provide native simulation import files. Since our ultimate goal is to use Mujoco MJPC, we need the robot description in a Mujoco XML format. To accomplish this, we utilize a community created URDF representation of the robot developed by AIWintermuteAI [5], and employ the K-Scale [6] Python package for URDF to MJCF conversion. We then made several manual adjustments, particularly correcting object file scaling factors to ensure physically realistic dynamics during simulation.

B. MuJoCo MPC Framework

We utilize the MuJoCo Model Predictive Control (MJPC) framework [7] to simulate and optimize gait behaviors for the Bittle quadruped robot. MJPC provides a robust environment for predictive control by integrating physics based dynamics simulations with shooting based trajectory optimization methods. Specifically, our implementation employs a derivative based Gradient Descent planner to optimize joint trajectories.

MJPC is a powerful interactive software framework based on MuJoCo physics, which enables the rapid synthesis of complex robotic behaviors. It supports various trajectory optimization techniques, including derivative based methods like Gradient Descent and iLQG (Iterative Linear Quadratic Gaussian), and derivative free methods such as Predictive Sampling. MJPC's flexibility allows users to define complex tasks through intuitive cost functions based on tunable residual terms, enabling the detailed specification of desired robot behaviors.

These residuals are explicitly defined within the MJPC XML model, incorporating sensors for joint positions, velocities, orientation, and angular momentum. The Gradient Descent planner we employ leverages a spline based representation of control sequences, significantly reducing computational complexity by minimizing the search space. Optimization is performed by iteratively updating spline parameters based on computed gradients of the defined residuals.

Additionally, MJPC's asynchronous design allows the planner to operate faster than real time, making it suitable for simulation based optimization on limited computational hardware. By using parallel rollouts and spline based control parameterizations, MJPC effectively balances computational demands with the quality of the generated trajectories.

Our implementation extends the MJPC framework with a custom task for the Bittle quadruped. We based our formulation off of the Mujoco MJPC framework for the Unitree A1 robot [3], [8]. Our implementation required definition of the Bittle Robot XML file, the Task Description XML file, and the C++ and header file for the task. The task definition includes:

1) Residual Function: We defined multiple residual terms to guide the optimization:

- **Upright term**: Encourages maintaining a specified robot orientation during locomotion
- **Height term**: Maintains a target height between the torso and feet
- **Position term**: Tracks desired positions for trajectory following
- **Gait phase term**: Coordinates limb movements according to predefined gait patterns
- **Balance term**: Leverages capture point dynamics to maintain stability
- **Effort term**: Minimizes control effort to promote energy efficiency
- **Posture term**: Encourages configuration close to a reference posture
- **Yaw term**: Controls heading direction

2) Gait Parameterization: We introduced parameters that define different gait types (stand, walk) with properties including:

- **Duty ratio**: The proportion of time each foot remains in contact with the ground
- **Cadence**: The frequency of the stepping pattern
- **Amplitude**: The height of foot lifting during swing phase

3) Data Collection: We implemented joint angle logging capabilities to facilitate analysis of the generated motions:

$$\mathcal{D} = \{(t_i, \theta_i) \mid i = 1, \dots, N\} \tag{1}$$

where t_i represents time points and θ_i represents the corresponding joint angles across all eight actuated joints.

4) Cost Function Design: We define the optimization objective using MJPC's composable cost structure:

$$J(x_{0:T}, u_{0:T}) = \sum_{t=0}^{T} \sum_{i=0}^{M} w_i \cdot \|r_i(x_t, u_t)\| \tag{2}$$

where w_i are weights for each residual component r_i, allowing us to balance competing objectives. The residuals are computed based on sensor data from the MuJoCo simulation forward rollout, including joint positions, body orientation, and computed quantities like capture point.

5) Spline Based Control: We utilize MJPC's spline representation to parameterize the control sequence:

$$u(t) = s(t; (\tau_{0:P}, \theta)) \tag{3}$$

where $\tau_{0:P}$ are time points and θ are spline parameters. This approach reduces the dimensionality of the optimization problem, which is particularly important for our microcontroller deployment target.

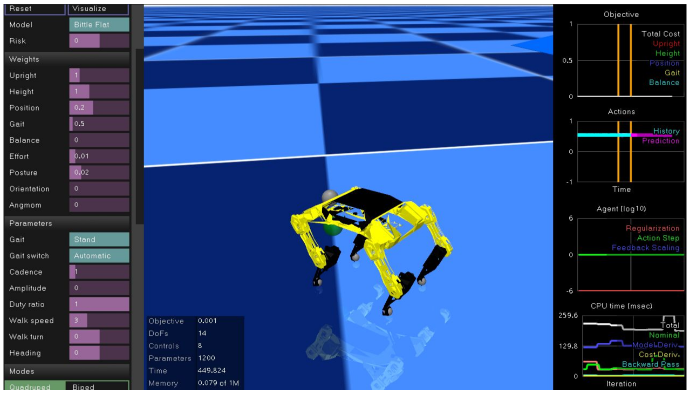
6) Visualization: In addition to rendering task specific geometries such as support polygons and capture points, the MJPC interface includes interactive panels for adjusting residual weights in real time. Users can tune objectives like uprightness, gait phase timing, and control effort while observing their effect on behavior. Diagnostic charts display cost function evolution, joint trajectories, and optimization progress, offering immediate insight into the performance and stability of planned gaits.

C. From MJPC to Embedded Deployment with TinyMPC

While MJPC provides an excellent framework for trajectory optimization and controller design, deploying these controllers on microcontroller platforms requires additional considerations. Using the time history of the joint angles in simulation, we formulate a sequence of 64 frames of joint angles to run open loop on the ESP32. While open-loop playback of these gaits is computationally efficient, it lacks robustness to disturbances or modeling errors. In order to mitigate this, we leverage TinyMPC [9], an efficient implementation of model predictive control designed for embedded systems, to deploy the gaits optimized in MJPC. This allows for real time re-optimization based on observed deviations.

Our workflow consists of two phases:

1) **Offline Trajectory Optimization**: Using MJPC, we design and optimize gait patterns that achieve stable



Reset | Visualize

Model: Bittle Flat

Risk: 0

Weights

Upright: 1

Height: 1

Position: 0.2

Gait: 0.5

Balance: 0

Effort: 0.01

Posture: 0.02

Orientation: 0

Angmom: 0

Parameters

Gait: Stand

Gait switch: Automatic

Cadence: 1

Amplitude: 0

Duty ratio: 1

Walk speed: 3

Walk turn: 0

Heading: 0

Modes

Quadruped | Biped

Objective	0.001
DoFs	14
Controls	8
Parameters	1200
Time	449.824
Memory	0.079 of 1M

Fig. 2: MJPC optimization framework visualizing a quadrupedal robot navigation task. The Bittle robot model (yellow quadruped) is simulated in the MuJoCo physics environment with the objective of reaching the green spherical target. The interface displays real time optimization metrics and allows interactive tuning of MPC parameters.

locomotion under various conditions. The optimization utilizes full order dynamics and complex cost terms that would be prohibitively expensive to compute on an ESP32.

2) **Online Simplified Control**: We extract the essential patterns and parameters from the MJPC optimized gaits and implement a reduced order model that can run efficiently on the ESP32. By solving this reduced MPC problem over the simplified dynamic model, TinyMPC can generate real time control inputs for joint level actuation.

This approach allows us to leverage the power of MJPC for designing sophisticated behaviors while enabling real time execution on resource constrained hardware without force sensing.

The modular nature of the MJPC framework enables us to compare different optimization methods (Gradient Descent vs. Predictive Sampling) under identical task definitions, providing insight into the trade offs between optimization quality and computational efficiency.

Due to the absence of force sensors, we rely on the error between commanded and actual joint angles (from servo encoders in simulation) as a proxy for estimating interaction forces. Future work will explore how to use this for contact estimation or stance detection.

D. TinyMPC

The control structure is represented in figure 3. Its structure is as such: An angular reference gait along with a center of mass reference are input into a dynamics model which translates them into a ground force reaction reference. The dynamics model used by TinyMPC takes ground force reaction as inputs and the state described in equation 8 as its state vector. Based on a linerization around the standing pose, TinyMPC generates an optimal gait which is translated by the PID controller into motion. The initial conditions are updated once the trajectory is executed and TinyMPC solves again without any additional linearization. This reduces computational overhead. To further reducec computational overhead, TinyMPC is not used only to produce the next step: the entire control horizon is executed by the PID controller. This enables TinyMPC to update less frequently. The math used is detaield in the following subsection.

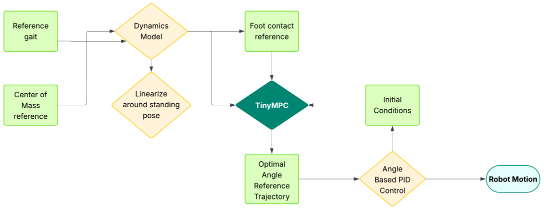


Fig. 3: Process diagram outlining information flow in TinyMPC control structure. An angular reference gait along with a center of mass reference are input into a dynamics model which translates them into a ground force reaction reference. The dynamics model used by TinyMPC takes ground force reaction as inputs and the state described in equation 8 as its state vector. Based on a linerization around the standing pose, TinyMPC generates an optimal gait which is translated by the PID controller into motion. The initial conditions are updated once the trajectory is executed and TinyMPC solves again without any additional linearization.

1) Dynamics Model and Linearization: For each leg joint angles α, β and link lengths L_1, L_2,

$$x_{\text{foot}}(\alpha, \beta) = L_1 \sin\alpha + L_2 \sin(\alpha+\beta) \tag{4}$$

$$y_{\text{foot}}(\alpha, \beta) = -\,L_1 \cos\alpha - L_2 \cos(\alpha+\beta) \tag{5}$$

—

The Jacobian mapping joint-space velocities $\dot{\alpha}, \dot{\beta}$ to foot-velocities is

$$\mathrm{J}(\alpha,\beta) = \frac{\partial}{\partial(\alpha,\beta)} \begin{pmatrix} x_{\text{foot}} \\ y_{\text{foot}} \end{pmatrix} = \begin{pmatrix} L_1 \cos\alpha + L_2 \cos(\alpha+\beta) & L_2 \cos(\alpha+\beta) \\ L_1 \sin\alpha + L_2 \sin(\alpha+\beta) & L_2 \sin(\alpha+\beta) \end{pmatrix} \tag{6}$$

—

Given joint torques $\tau = [\tau_\alpha, \tau_\beta]^T$, the GRF $F = [f_x, f_y]^T$ is computed via the Moore–Penrose pseudoinverse of J^T:

$$F \;=\; \left(J(\alpha,\beta)^T\right)^{\dagger} \tau. \tag{7}$$

—

Let the full state be

$$x = \begin{pmatrix} x_{\text{com}} \\ z_{\text{com}} \\ \psi \\ \dot{x}_{\text{com}} \\ \dot{z}_{\text{com}} \\ \dot{\psi} \\ \theta_1 \\ \vdots \\ \theta_8 \\ \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_8 \end{pmatrix} \tag{8}$$

and the control u pack the four leg forces $\left[f_{x,i}, f_{y,i}\right]_{i=1\ldots4}$. Then:

$$\tau_{2i-1:2i} = J(\alpha_i, \beta_i)^T \, F_i, \quad \tau \in R^8 \tag{9}$$

$$\sum_i F_i =: \mathbf{F}_{\text{tot}} \tag{10}$$

$$\tau_{\text{tot}} = \sum_{i=1}^{4} \bigl(x_{\text{foot},i} \, f_{y,i} - y_{\text{foot},i} \, f_{x,i}\bigr) \tag{11}$$

$$\ddot{x}_{\text{com}} = \frac{F_{\text{tot},x}}{m}, \quad \ddot{z}_{\text{com}} = \frac{F_{\text{tot},z}}{m} - g, \quad \ddot{\psi} = \frac{\tau_{\text{tot}}}{I} \tag{12}$$

Joint Dynamics (with damping c and inertia I_j):

$$\ddot{\theta} = \bigl(\tau - c\,\dot{\theta}\bigr) \,/\, I_j \tag{13}$$

$$\dot{x} = \begin{pmatrix} \dot{x}_{\text{com}} \\ \ddot{x}_{\text{com}} \\ \dot{\theta} \\ \ddot{\theta} \end{pmatrix} \in R^{22} \tag{14}$$

—

With step size Δt, letting $f(x,u)$ denote the continuous right-hand side,

$$\begin{aligned} k_1 &= f(x_k, u_k) \\ k_2 &= f\bigl(x_k + \tfrac{\Delta t}{2} k_1,\ u_k\bigr) \\ k_3 &= f\bigl(x_k + \tfrac{\Delta t}{2} k_2,\ u_k\bigr) \\ k_4 &= f\bigl(x_k + \Delta t \, k_3,\ u_k\bigr) \\ x_{k+1} &= x_k + \tfrac{\Delta t}{6}\bigl(k_1 + 2k_2 + 2k_3 + k_4\bigr) \end{aligned} \tag{15}$$

These dynamics were used as detailed before to optimize the reference trajectory using TinyMPC.

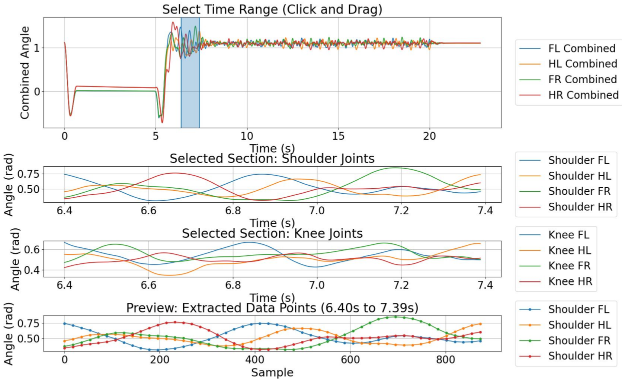


Fig. 4: Figure showing the joint angle history of the Bittle robot during the MJPC simulation rollout, with all 8 joint angles shown in degrees on the y-axis, and the time stamp in seconds on the x-axis. The first 5 seconds shows the joint angles during the flipping motion to correct the robot from laying on its back to right side up. The remaining portion shows the joint angles during normal walking locomotion.

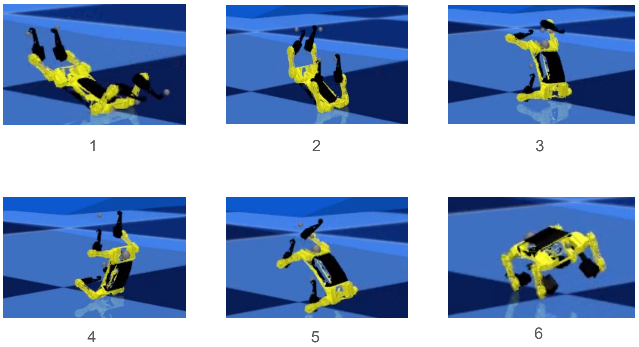
IV. Results

A. Joint Trajectory Generation

We recorded the MJPC joint angle trajectories during successful locomotion. The MJPC framework allowed us to generate stable gaits without force sensing by utilizing the built in parameter and weight adjustment finetuning during simulation. During simulation, we logged joint angle histories at each timestep using a dataset collection mechanism defined in equation (1).

Figure 4 Shows a sample of such recorded joint angle histories, from a single simulation. At the simulation beginning, the robot was placed on its back, and the MJPC algorithm automatically corrects the robot to upright itself. This is evident in the first 5 seconds of Figure 4, with all 8 limbs operating synchronously, with the two distinct spikes in joint angles. After the robot righted itself, the joint angles displayed a sinusoidal pattern, which was expected for quadruped locomotion. The amplitude of the joint angles was initially greater, from $\sim$ 5-8 seconds in the simulation, as the controller refined the gait based on residual and parameter weights. From this point to $\sim$ 20 seconds of the simulation, the joint angles exhibited a regular sinusoidal pattern as the robot traversed to the goal location, and the joint angles remained constant for the remaining simulation time once the robot reached the goal location, as expected. This regular sinusoidal pattern was desireable, as our end goal was to capture a sample of the logged joint angles to run open loop on the robot, as well use as a reference trajectory for the TinyMPC implementation.

Aside from our initial aim to develop optimal gaits for the Bittle robot using trajectory optimization in MJPC, it is also of interest to develop strategies to correct the robot orientation if it falls on its side or back. Fig. 5 shows several sample frames of the MJPC viewer during the beginning of the



h

Fig. 5: Sequential frames demonstrating autonomous recovery behavior using Model Predictive Control in MJPC. The Bittle robot, initially positioned on its back (frame 1), executes a self righting maneuver (frames 2-5) to restore its operational stance (frame 6). This sequence illustrates the MJPC algorithm's capability to synthesize complex recovery trajectories by optimizing joint configurations in real time, without pre programmed recovery behaviors.

simulation as the algorithm automatically finds the joint angles to recover the robot from its back to upright configuration. Notably, the algorithm leverages the mass of the thigh and calf joints to rock the robot on its back to flip to right side up, in a somewhat acrobatic maneuver. This highlights the versatility of this method as a comprehensive technique for quadruped skill discovery in recovery maneuvers as well as gait generation.

B. Joint Sequence Extraction for Hardware Deployment

To transfer the MPC generated trajectories to the physical Bittle platform, we developed a two stage process for extracting and formatting joint sequences:

1) Data Collection and Processing: The controller logs joint angles during simulation, producing large datasets with thousands of datapoints. To make this data usable on the ESP32 microcontroller, we implemented a joint sequence extraction process. The high sampling rate of the simulation (approximately 1000 Hz) produced trajectories with far more frames than could be efficiently stored and executed on the physical robot.

2) Trajectory Resampling: For efficient hardware deployment, we needed to reduce the dimensionality of the joint trajectories while preserving the essential movement characteristics. Our conversion pipeline consists of:

1) **Interactive Selection**: Using a custom visualization function we developed in Python, we selected the most stable gait cycles from the MPC generated data.
2) **Resampling**: We resampled the selected trajectories to a fixed length of 64 frames, which matched the expected format of the robot's firmware.
3) **Cyclical Adjustment**: To ensure smooth continuous execution, we applied a cyclical correction that guaranteed the start and end points of the trajectory matched:

$$\|\theta_0 - \theta_{63}\| < \epsilon \tag{16}$$

4) **Format Conversion**: The resampled trajectory was converted to a C++ array format for deployment on the physical robot.

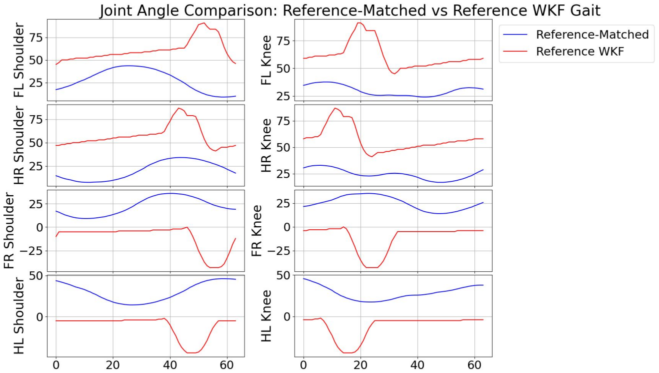


Fig. 6: Plots showing the joint angle sequence for the reference build in gait (red lines) and MJPC generated gait (blue lines).

In the top plot of Fig. 4, a sample window of joint angles is shown to be captured for extraction and conversion to a sequence of joint angles to run on the Bittle Robot. The second and third plots show the sequence of joint angles for the shoulder and knee joints, respectively, while the fourth plot shows the angle sample points for the shoulder joints, highlighting that even for this small sample of the joint histories, there were more than the desired 64 joint angle snap shots for this selection. From this selection, there were 855 timeshots of joint angles, hence our development of the resampling algorithm to enable running MJPC gaits on the robot.

Figure 6 illustrates the difference between our MJPC generated gait and the reference gait provided with the Bittle firmware. While both gaits exhibit the expected periodic patterns, the MJPC generated gait showed several differences:

- Smoother transitions between swing and stance phases
- More coordinated inter joint coupling
- Lower peak joint velocities, potentially reducing actuator strain

However, we also observed that the MJPC-generated gait had lower foot clearance during swing phases. This resulted from the algorithm's tendency to minimize energy expenditure when not explicitly penalized for low ground clearance, which became a limiting factor in hardware deployment.

C. Simulation to Reality Challenges

When deploying the extracted sequences on the physical Bittle robot, we encountered several challenges that highlight the well known simulation to reality gap in robotic control:

1) Gait Performance: The MJPC optimized gaits did not show the expected improvements over the reference gaits when deployed on hardware. Key observations included:

- **Insufficient Foot Clearance**: The trajectories produced by MJPC often kept the feet close to the ground to minimize energy expenditure and maximize stability, which worked well in simulation with perfect contact modeling but caused the robot to stumble on real world surfaces with minor irregularities.
- **Servo Response Limitations**: The simulation assumed ideal actuator response, while the physical servos exhibited latency and tracking errors under dynamic loads.
- **Model Discrepancies**: Small differences between the simulated and physical robot mass distribution affected the dynamic balance, particularly during transitions between different support configurations.

These findings suggest that additional constraints are needed in the cost function to explicitly reward behaviors that are robust to model uncertainty and hardware limitations. Specifically, enforcing minimum foot clearance during swing phases would likely improve performance on varied surfaces.

2) Recovery Behavior Transfer: The self righting behaviors discovered in simulation transferred partially to hardware, with mixed results:

- The recovery from a 90 degree tipped position was successfully replicated on hardware.
- The full 180 degree back to feet recovery maneuver failed to execute correctly on the physical robot.

This discrepancy can be attributed to several factors:

- The impulsive nature of the recovery behavior relies on precise timing of contact events that differ between simulation and reality.
- The simulation may not accurately capture the compliance and friction characteristics of the real robot's materials.
- Joint torque limitations that were not fully modeled in simulation constrained the physical robot's ability to execute the required motions.

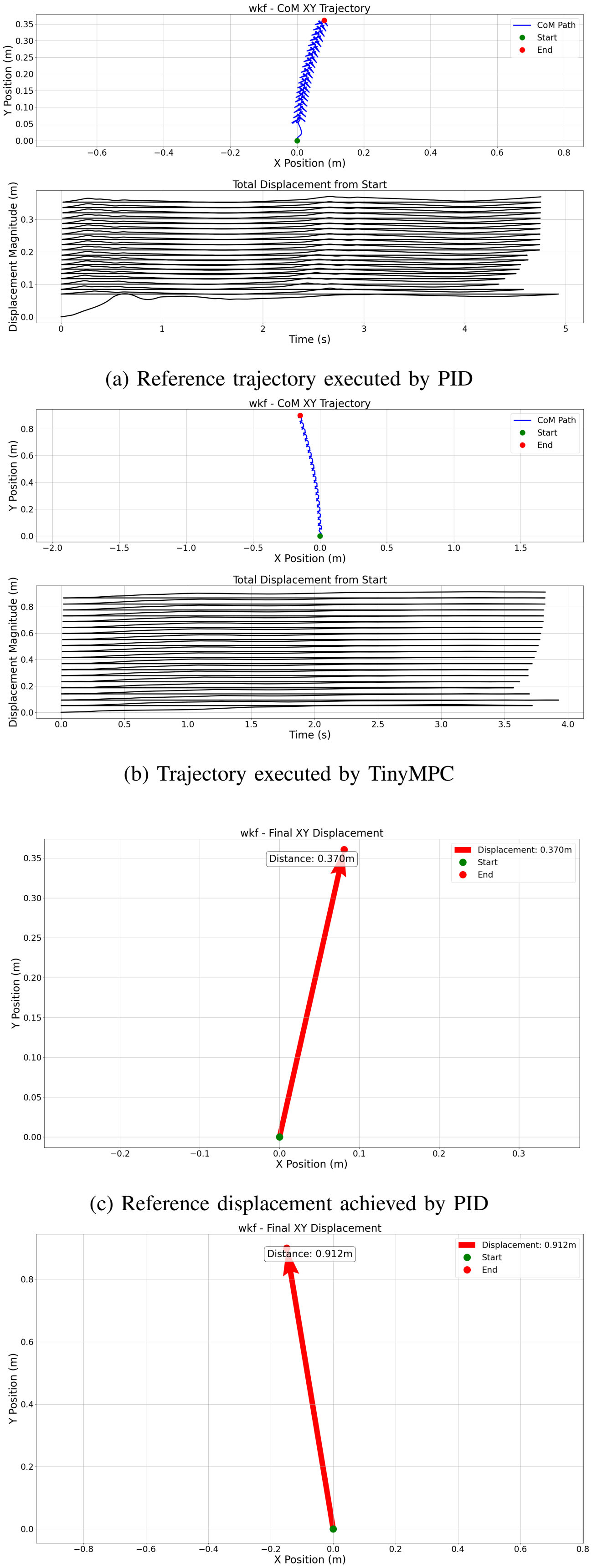
These experiments provide valuable insights for future work on bridging the simulation to reality gap for dynamic locomotion tasks.

D. *TinyMPC Results*

We used the ”wkf” angle trajectory reference provided as a default walk cycle by Petoi. To test the performance of TinyMPC, we compared our framework of optimizing the reference using TinyMPC against simply executing the reference trajectory using the low level PID controller. The results, shown in 7, show that the MPC controller achieves approximately 3x the XY displacement, showcasing the viability of our method. A video of the gait can be shown here: https://youtu.be/AegWtIFUd98.

V. Future Work

Our experiments with MuJoCo MJPC for quadrupedal locomotion reveal several promising directions for further research:



(a) Reference trajectory executed by PID

(b) Trajectory executed by TinyMPC

(c) Reference displacement achieved by PID

(d) Displacement achieved by TinyMPC

Fig. 7: Comparison of PID and MPC trajectories and displacements. The greater displacement achieved by TinyMPC is indicative that it creates trajectories that are more viable than the base PID executing the reference trajectory.

A. Enhanced Simulation Fidelity

To address the simulation-to-reality gap observed in our hardware deployment, future work should focus on improving simulation fidelity:

- **Actuator Dynamics Modeling**: Incorporating more accurate models of servo response characteristics, including torque limits, velocity dependent behavior, and temperature effects.
- **Contact Parameter Identification**: Developing systematic methods to identify contact parameters (friction, restitution, compliance) that better match real world interactions.
- **Uncertainty Aware MJPC**: Extending the MJPC framework to explicitly account for model uncertainty, perhaps through techniques like robust MPC or by incorporating disturbances during trajectory optimization.

B. Cost Function Refinement

Our results indicate that the cost function design significantly impacts the transferability of behaviors to hardware:

- **Explicit Foot Clearance Terms**: Adding residual terms that explicitly reward minimum foot height during swing phases to ensure robust traversal of irregular surfaces.
- **Energy Aware Formulations**: Developing more nuanced energy cost terms that better reflect the actual power consumption characteristics of the servos.
- **Automated Cost Weight Tuning**: Investigating methods for automatically tuning cost weights, such as Bayesian optimization or evolutionary strategies, to reduce the manual effort required.

C. Learning Enhanced Approaches

The limitations of purely model based approaches suggest several directions for incorporating learning:

- **Learned Dynamics Adaptation**: Training neural networks to predict and compensate for the simulation to reality gap, enabling adaptation of MPC policies to real world conditions.
- **Hybrid MPC Reinforcement Learning**: Combining the interpretability and performance guarantees of MPC with the adaptability of reinforcement learning, as proposed by Howell et al. [7].
- **Residual Policy Learning**: Using MPC generated behaviors as a starting point and learning corrective terms that address model discrepancies.

D. Efficient Embedded Implementation

To fully leverage the computational capabilities of the ESP32 platform:

- **TinyMPC Integration**: Completing the implementation of simplified online MPC using the TinyMPC framework [9], which could enable real-time feedback control on the ESP32. We would like to implement this on hardware.
- **Efficient State Estimation**: Developing lightweight algorithms for estimating robot state using only onboard IMU and joint encoders, enabling closed loop execution of optimized trajectories.
- **Hardware Acceleration**: Investigating the use of ESP32's digital signal processing capabilities to accelerate critical computations in the control loop.

E. Validation and Benchmarking

To quantify the benefits of our approach:

- **Systematic Hardware Evaluation**: Conducting controlled experiments to measure and compare the performance of different gait generation methods across varied terrains and disturbances.
- **Energy Efficiency Analysis**: Quantifying the power consumption of different gaits to determine if MPC optimized trajectories offer efficiency advantages despite the current performance limitations.
- **Open Benchmarks**: Contributing to open benchmarks for legged locomotion on resource constrained platforms to facilitate comparison with other approaches.

By addressing these research directions, we believe that model predictive control approaches like MJPC can be successfully adapted to resource constrained platforms like the Bittle robot, enabling more sophisticated locomotion behaviors without requiring additional sensing hardware.

References

[1] B. Katz, J. D. Carlo, and S. Kim, "Mini cheetah: A platform for pushing the limits of dynamic quadruped control," *2019 International Conference on Robotics and Automation (ICRA)*, vol. 00, pp. 6295–6301, 2019.

[2] W. Bosworth, S. Kim, and N. Hogan, "The MIT super mini cheetah: A small, low-cost quadrupedal robot for dynamic locomotion," *2015 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–8, 2015.

[3] U. Robotics, "Unitree A1 quadruped robot," 2020. [Online]. Available: https://www.unitree.com/products/a1/

[4] Petoi, "Bittle / bittle x — petoi doc center," 2025, accessed: 2025-04-30. [Online]. Available: https://docs.petoi.com/desktop-app/joint-calibrator/bittle-bittle-x

[5] AIWintermuteAI, "Bittle urdf," https://github.com/AIWintermuteAI/Bittle_URDF.git, 2025. [Online]. Available: https://github.com/AIWintermuteAI/Bittle_URDF.git

[6] KScale, "Urdf to mujoco converter," 2025, utility for converting from URDF format to Mujoco format, Version 0.2.16. [Online]. Available: https://docs.kscale.dev/docs/urdf2mjcf

[7] T. Howell, N. Gileadi, S. Tunyasuvunakool, K. Zakka, T. Erez, and Y. Tassa, "Predictive sampling: Real-time behaviour synthesis with mujoco," 2022. [Online]. Available: https://arxiv.org/abs/2212.00541

[8] google deepmind, "Mujoco mpc: Real-time behaviour synthesis with mujoco, using predictive control," 2022. [Online]. Available: https://github.com/google-deepmind/mujoco_mpc

[9] K. Nguyen, S. Schoedel, A. Alavilli, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2024, arXiv:2310.16985.