Auto-Tuning of Controller and Online Trajectory Planner for Legged Robots

Alexander Schperberg, *Student Member, IEEE*, Stefano Di Cairano, *Senior Member, IEEE*, and Marcel Menner, *Member, IEEE*

***Abstract*—This letter presents an approach for auto-tuning feedback controllers and online trajectory planners to achieve robust locomotion of a legged robot. The auto-tuning approach uses an Unscented Kalman Filter (UKF) formulation, which adapts/calibrates control parameters online using a recursive implementation. In particular, this letter shows how to use the auto-tuning approach to calibrate cost function weights of a Model Predictive Control (MPC) stance controller and feedback gains of a swing controller for a quadruped robot. Furthermore, this letter extends the auto-tuning approach to calibrating parameters of an online trajectory planner, where the height of a swing leg and the robot's walking speed are optimized, while minimizing its energy consumption and foot slippage. This allows us to generate stable reference trajectories online and in real time. Results using a high-fidelity Unitree A1 robot simulator in Gazebo provided by the robot manufacturer show the advantages of using auto-tuning for calibrating feedback controllers and for computing reference trajectories online for reduced development time and improved tracking performance.**

***Index Terms*—Calibration and identification, integrated planning and learning, legged robots, machine learning for robot control, probability and statistical methods.**

I. Introduction

PLANNING and control of legged systems is challenging due to the tight coupling of reaction forces generated by foot contacts with the environment and the motion of the robot's base. This problem is further complicated as legged systems for real-world deployment are expected to traverse highly unstructured environments such as debris, obstacles, and rough terrain. Traditional control designs such as those employing Raibert or heuristic controllers [1] can only guarantee physical feasibility on smooth or flat surfaces. Optimization-based approaches often use a cost function to account for kinematic, dynamic, and friction-cone constraints. E.g., the Linear Inverted Pendulum approach optimizes over the Center of Mass (CoM) position, where the footsteps must be specified *a priori* to satisfy the Zero-Moment Point [2]. This approach has some drawbacks because pre-defining the footholds' locations on the ground may restrict the robot's range of achievable motion. Further, whole-body controllers that do not require pre-defined footholds, e.g., in [3], may be sub-optimal as they only consider the current joint torques.

On the other hand, trajectory Optimization (TO) is often used for predictive planning of feasible motions for legged systems [4]–[7]. TO methods typically employ some form of centroidal dynamics, optimizing over contact forces [4]. However, TO is computationally expensive and often not suitable for online re-planning [8]. Further, open-loop planners are susceptible to changes in terrain.

Due to the inverse relationship between the complexity of the dynamic model used for locomotion and the computation time required for an optimization solver, a popular approach is to employ Reinforcement Learning (RL) [9]–[14]. E.g., in [9], a stochastic policy is realized with neural networks to simulate a foothold and base motion controller using Trust-Region Policy Optimization. This is done by considering a reward function that consists of several physical parameters such as the error between actual and reference footstep position, penalizing foot slippage or large swing-leg velocities, or sudden changes in base orientation. An RL approach has also been successfully demonstrated in [10], which learns a neural network that acts only on a stream of proprioceptive signals for locomotion on uneven terrain.

Although tremendous progress has been made in the RL literature to achieve legged robot locomotion, this approach is highly complex and requires significant amounts of data. For example, in [10], a two-stage training process is required that involves a teacher/student policy in addition to particle-filtering, which maintains certain terrain parameters used to classify what is and is not traversable during training. In [9], the reward function and corresponding weighing parameters required intricate fine-tuning and assumes the user has extensive computational resources available during training. Another work can be found in [15], which uses RL to create adaptive trajectories based on following a reference trajectory calculated offline first through TO. Similar to our work, they also use step clearance and slippage as objectives to their calibration. However, different from ours, while we also use TO methods, we only rely on the reference trajectory from TO for the first few footsteps, while future footsteps are planned online. Overall, the complexity of formulating the correct reward function in RL and the effort

Manuscript received 24 February 2022; accepted 5 June 2022. Date of publication 23 June 2022; date of current version 5 July 2022. This letter was recommended for publication by Associate Editor L. Lanari and Editor A. Kheddar upon evaluation of the reviewers' comments. *(Corresponding author: Marcel Menner.)*

The authors are with the Mitsubishi Electric Research Laboratories (MERL), Cambridge, MA 02139 USA (e-mail: aschperb@gmail.com; dicairano@ieee.org; menner@ieee.org).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3185387, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3185387

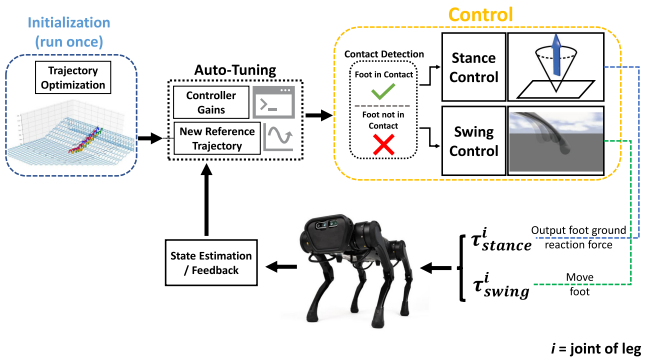

Fig. 1. Auto-tuning framework. We initialize the reference trajectory for the first few footsteps and CoM base using trajectory optimization. Then, auto-tuning is applied to estimate new reference trajectories which account for desired physical parameters. Further, auto-tuning is applied on cost function weights for a stance controller (MPC) and feedback gains of a swing controller (PD).

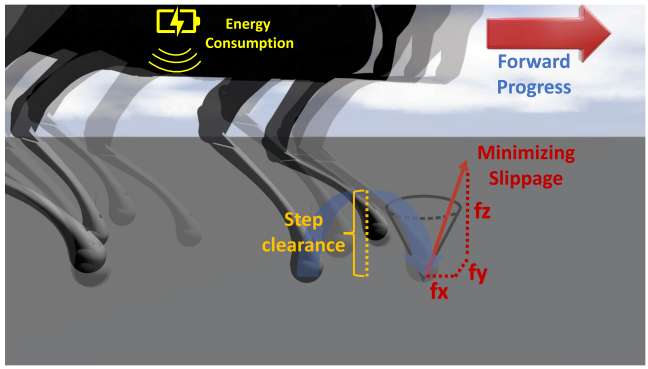

Fig. 2. Auto-Tuning reference trajectories in Gazebo. The robot follows a reference trajectory that is being tuned by the auto-tuning formulation. The figure shows the robot following a desired velocity (forward progress), a desired foot height (step clearance), minimizing foot slippage (ground reaction forces), and minimizing energy consumption.

required for tuning hyper-parameters may cause significant delay in development time. This also may make RL difficult to reproduce or generalize to a wide-variety of robotic platforms without significant amounts of tuning.

This letter proposes to use an alternative method, see Fig. 1, to the above approaches for achieving robust locomotion. In [16], [17], a Kalman filter technique was used to estimate control parameters with a training objective that evaluates the performance of a closed-loop system online using a recursive implementation. This approach was successfully used to automatically tune a variety of control architectures such as the weights of a neural network, cost function weights of an LQR, or gains of a PID controller to achieve stable motion of an autonomous vehicle. Although auto-tuning methods have been applied before to tune various controller gains, typically using some form of Bayesian optimization (BO) [18], [19], the applied method differs as we do not require a trial-and-error implementation, do not need a surrogate function, and can handle disturbances due to the recursive nature of the applied method [16], [17]. One of the issues with BO is that their performance decreases in higher dimensions [20]. In [21], this problem was overcome by including domain knowledge into BO for tuning walking controllers of humanoid robots. In [22], the parameters of running and jumping motions are tuned through a policy search of generic sets of motion primitives and their cost functions. Both [21], [22] require a trial-and-error implementation which requires extensive training using simulators before application to hardware. Our method however, is able to calibrate control parameters on a single run without relying on trial-and-error.

Additionally, while other algorithms have been proposed to tune controller gains [23], [24], these algorithms have never been applied (or evaluated) to directly tune reference trajectories without having to include a predefined set of reference trajectories [25].

This letter employs the method of auto-tuning initially proposed in [16], [17] on a legged system using an unscented Kalman filter (UKF) implementation and apply it not only to controller gains, but directly on physically-meaningful parameters such as 'step clearance,' 'forward progress,' 'energy efficiency,' and 'dynamic and kinematic gait stability,' see Fig. 2. This is achieved by online tuning the robot's future reference trajectories which uses a training objective to satisfy user-defined specifications or requirements. We demonstrate the method on a quadruped robot, the Unitree A1, in a high-fidelity physics simulation provided by the robot manufacturer. We demonstrate walking on both even and uneven terrain, and present several test-cases such as tuning reference trajectories that satisfy the physical parameters mentioned previously, and controller weights of a swing and stance controller. The presented method for automating the controller calibration and the trajectory planner can save development/deployment time as manually tuning control parameters can be tedious and time consuming, facilitates in abstracting the tuning problem into physically meaningful parameters, and enables greater autonomy of robotic systems.

II. Preliminaries

A. Notation

Given two integer indices n, m with $m < n$ and $\mathbf{x}_i \in \mathbf{R}^{n_x}$, we define $\mathbf{x}_{m|n} \in \mathbf{R}^{n_x(n-m+1)}$ as the vectorized sequence that comprises $\mathbf{x}_i$ from $i = m$ through $i = n$,

$$\mathbf{x}_{m|n} := \begin{bmatrix} \mathbf{x}_m \\ \vdots \\ \mathbf{x}_n \end{bmatrix}.$$

We define $\|\mathbf{x}\|_{\mathbf{\Sigma}} := \mathbf{x}^T \mathbf{\Sigma} \mathbf{x}$ and $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ as the Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$. The notation $\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$ means $\mathbf{x}$ sampled from $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$. Further, $\mathrm{diag}(\boldsymbol{\lambda}) \in \mathbf{R}^{n_\lambda \times n_\lambda}$ is a matrix, whose diagonal entries are the entries of a vector $\boldsymbol{\lambda} \in \mathbf{R}^{n_\lambda}$.

B. Unscented Kalman Filter for Control Parameter Tuning

This letter applies the auto-tuning method proposed in [16], [17] to calibrate control parameters of the Model Predictive Control (MPC) stance controller, the PD swing controller, as well as the reference trajectory. The auto-tuning method is based

on an UKF, which "estimates" the optimal control parameters measured with respect to a training objective. The UKF uses deterministic samples (called sigma points) around the mean, which are propagated and used to update the mean and covariance estimates [26]. Further, it uses a model of the system dynamics in order to obtain evaluations of the sigma points, which are then used to update the control parameters.

The method is model-based, i.e., it uses a model of a dynamical system (denoted by $\mathbf{dyn}(\mathbf{x}_k, \mathbf{u}_k)$),

$$\mathbf{x}_{k+1} = \mathbf{dyn}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k \tag{1}$$

with the state $\mathbf{x}_k \in \mathbb{R}^n$, the control input $\mathbf{u}_k \in \mathbb{R}^m$, and process noise or model mismatch $\mathbf{w}_k$ at time k. The method calibrates control parameters, $\boldsymbol{\theta} \in \mathbb{R}^L$, of a generic controller, $\mathbf{u}_k = \boldsymbol{\kappa}_\theta(\mathbf{x}_k)$ to minimize the training objective

$$\|\mathbf{y}_k - \mathbf{h}(\boldsymbol{\theta}_k)\|_{\mathbf{C}_y^{-1}} \tag{2}$$

with a positive definite $\mathbf{C}_y$, desired nominal values $\mathbf{y}_k$, and specification function $\mathbf{h}(\boldsymbol{\theta})$, where

$$\mathbf{h}(\boldsymbol{\theta}) := \mathbf{r}(\mathbf{x}_{k-N}, \mathbf{x}_{k-N+1}, \ldots, \mathbf{x}_k, \mathbf{u}_{k-N}, \ldots, \mathbf{u}_{k-1}),$$

i.e., $\mathbf{y}_k = \mathbf{h}(\boldsymbol{\theta}_k)$ when the dynamical system satisfies all the specifications in the training objective, exactly.

For the legged robot, the control parameters are tuned during operation (episodically, from time step $k-N$ to k) according to

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-N} + \Delta\boldsymbol{\theta}_k, \tag{3}$$

where the update $\Delta\boldsymbol{\theta}_k$ is computed based on sensor measurements, $\mathbf{x}_k$, $\mathbf{u}_k$, and the training objective in (2).

The idea of the auto-tuning method is to treat the control parameter adaptation problem as an optimal estimation problem with prior distributions $\Delta\boldsymbol{\theta}_k^{\text{prior}} \sim \mathcal{N}(\mathbf{0}, \mathbf{C}_\theta)$ and $\mathbf{y}_k \sim \mathcal{N}(\mathbf{h}(\boldsymbol{\theta}_k), \mathbf{C}_y)$. Thus, the parameter tuning law in (3) results from the corresponding posterior distribution,

$$\Delta\boldsymbol{\theta}_k = \mathbf{K}_k \left(\mathbf{y}_k - \hat{\mathbf{h}}_k\right) \tag{4a}$$

with the Kalman gain, $\mathbf{K}_k$, computed as

$$\hat{\boldsymbol{\theta}}_k = \textstyle\sum_{j=0}^{2L} v_j^a \boldsymbol{\theta}_k^{\text{sp},j} \tag{4b}$$

$$\hat{\boldsymbol{h}}_k = \textstyle\sum_{j=0}^{2L} v_i^a \mathbf{h}_k^{\text{sp},j} \tag{4c}$$

$$\mathbf{h}_k^{\text{sp},j} = \mathbf{h}(\boldsymbol{\theta}_k^{\text{sp},j}) \tag{4d}$$

$$\mathbf{S}_k = \mathbf{C}_v + \textstyle\sum_{j=0}^{2L} v_j^c (\mathbf{h}_k^{\text{sp},j} - \hat{\boldsymbol{h}}_k)(\mathbf{h}_k^{\text{sp},j} - \hat{\boldsymbol{h}}_k)^\top \tag{4e}$$

$$\mathbf{Z}_k = \textstyle\sum_{j=0}^{2L} v_j^c (\boldsymbol{\theta}_k^{\text{sp},j} - \hat{\boldsymbol{\theta}}_k)(\mathbf{h}_k^{\text{sp},j} - \hat{\boldsymbol{h}}_k)^\top \tag{4f}$$

$$\mathbf{K}_k = \mathbf{Z}_k \mathbf{S}_k^{-1} \tag{4g}$$

and the posterior covariance, which is used to generate the sigma points, computed as

$$\mathbf{P}_{k|k-N} = \mathbf{C}_\theta + \textstyle\sum_{j=0}^{2L} v_i^c (\boldsymbol{\theta}_k^{\text{sp},j} - \hat{\boldsymbol{\theta}}_k)(\boldsymbol{\theta}_k^{\text{sp},j} - \hat{\boldsymbol{\theta}}_k)^\top \tag{4h}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-N} - \mathbf{K}_k \mathbf{S}_k \mathbf{K}_k^\top, \tag{4i}$$

where $\boldsymbol{\theta}_k^{\text{sp},j}$ is the jth sigma point, v_j^c and v_j^a denote weights associated with the sigma points, $\mathbf{Z}_k$ is the cross-covariance matrix, $\mathbf{S}_k$ is the innovation covariance, and $\mathbf{P}_{k|k}$ is the estimate covariance. Hence, the UKF implementation uses $2L+1$ sigma points. The reader is referred to [16], [17] for more details.

Remark 1: For each sigma point evaluation (4d), the system dynamics (1) is simulated, where the model mismatch, $\mathbf{w}_k$, is calculated using measured data, $\mathbf{x}_{k-N}$ through $\mathbf{x}_k$.

Remark 2: We choose the weights $v_0^a = v_0^c = 0$, $v_i^a = v_i^c = (1 - v_0^a)/(2L)$ and the sigma points as $\boldsymbol{\theta}_k^{\text{sp},0} = \boldsymbol{\theta}_k$, $\boldsymbol{\theta}_k^{\text{sp},j} = \boldsymbol{\theta}_k + \sqrt{L/(1-v_0^a)}\boldsymbol{\Gamma}^j$ for $j = 1, \ldots, L$, and $\boldsymbol{\theta}_k^{\text{sp},j} = \boldsymbol{\theta}_k - \sqrt{L/(1-v_0^a)}\boldsymbol{\Gamma}^j$ for $j = L+1, \ldots, 2L$ with $\boldsymbol{\Gamma}^j$ being the jth column of $\boldsymbol{\Gamma}$ and $\mathbf{P}_{k-N|k-N} = \boldsymbol{\Gamma}\boldsymbol{\Gamma}^\top$, i.e., $\boldsymbol{\Gamma}$ is calculated using the Cholesky decomposition.

III. Robot Control Architecture

A. Swing Controller

The swing controller in this letter is similar to [27], which is used to compute the torque for each foot i for all three joints of the robot as (time step k omitted for simplicity)

$$\begin{aligned}\boldsymbol{\tau}_i = \mathbf{J}_i^\top \left[\mathbf{K}_p \left(\mathbf{p}_{i,\text{ ref}}^b - \mathbf{p}_i^b\right) + \mathbf{K}_d \left(\mathbf{v}_{i,\text{ ref}}^b - \mathbf{v}_i^b\right)\right] \\ + \mathbf{J}_i \boldsymbol{\Lambda}_i \left(\mathbf{a}_{i,\text{ ref}}^b - \dot{\mathbf{J}}_i^\top \dot{\mathbf{q}}_i\right) + \mathbf{V}_i \dot{\mathbf{q}}_i + \mathbf{G}_i\end{aligned} \tag{5}$$

where $\boldsymbol{\tau}_i \in \mathbb{R}^3$ is the joint torque, $\mathbf{q}_i \in \mathbb{R}^3$ and $\dot{\mathbf{q}}_i \in \mathbb{R}^3$ are the current joint position and velocity of foot i, $\mathbf{J}_i \in \mathbb{R}^{3\times3}$ is the foot Jacobian, $\mathbf{K}_p$ and $\mathbf{K}_d$ are the proportional and derivative (PD) gain matrices (3×3 diagonal positive semi-definite), $\mathbf{p}_{i,\text{ ref}}^b \in \mathbb{R}^3$ and $\mathbf{p}_i^b \in \mathbb{R}^3$ are the reference and current footstep positions in the body frame, $\mathbf{v}_{i,\text{ ref}}^b \in \mathbb{R}^3$ and $\mathbf{v}_i^b \in \mathbb{R}^3$ are the reference and current footstep velocities in the body frame, $\mathbf{a}_{i,\text{ ref}}^b \in \mathbb{R}^3$ is the reference footstep acceleration in the body frame, $\mathbf{V}_i \in \mathbb{R}^3$ is the torque due to the coriolis and centrifugal forces, $\mathbf{G}_i \in \mathbb{R}^3$ is the torque due to gravity, and $\boldsymbol{\Lambda}_i \in \mathbb{R}^{3\times3}$ is the operational mass matrix.

B. Stance Controller

The stance MPC calculates ground reaction forces $\mathbf{f}_i$ for all feet i and is formulated as in [27],

$$\min_{\mathbf{x},\mathbf{f}} \sum_{k=0}^{N_{\text{MPC}}} \|\mathbf{x}_k - \mathbf{x}_{k,\text{ref}}\|_{\mathbf{Q}} + \|\mathbf{f}_k\|_{\mathbf{R}} \tag{6a}$$

$$\text{subject to} \quad \begin{aligned} f_{k,\min} \le f_{k,z} \le f_{k,\max} \\ -\mu f_{k,z} \le \pm f_{k,x} \le \mu f_{k,z} \\ -\mu f_{k,z} \le \pm f_{k,y} \le \mu f_{k,z} \\ \mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{f}_k \\ \mathbf{D}_k \mathbf{f}_k = 0 \end{aligned} \tag{6b}$$

with the state

$$\mathbf{x}_k = \begin{bmatrix} \boldsymbol{\Theta}_k \\ \mathbf{r}_k \\ \boldsymbol{\omega}_k \\ \mathbf{v}_k \end{bmatrix}, \tag{6c}$$

where $\boldsymbol{\Theta}$ is the robot's orientation, $\mathbf{r}$ is the CoM base position, $\boldsymbol{\omega}$ is the angular velocity, $\mathbf{v}$ is the linear velocity, μ is the friction coefficient, $\mathbf{A}$ and $\mathbf{B}$ are the dynamic matrices for state propagation, $\mathbf{D}_k$ is a force selection matrix (selecting forces

that are not in contact to be equal to zero), and $\mathbf{Q} \in \mathbb{R}^{12\times12}$ and $\mathbf{R} \in \mathbb{R}^{12\times12}$ are diagonal positive semi-definite cost matrices, see [27] for further details. The joint torques, which are the input to the torque-controlled motors, are obtained using the forces $\mathbf{f}_k$ resulting from (6) as

$$\boldsymbol{\tau}_{i,k} = \mathbf{J}_{i,k}^{\top} \mathbf{R}_{b,i,k}^{w,\top} \left(-\mathbf{f}_{i,k}\right), \tag{7}$$

where $\mathbf{f}_{i,k} \in \mathbb{R}^3$ is the force vector associated with leg i as subset of $\mathbf{f}_k$, and $\mathbf{R}_{b,i,k}^{w}$ is the rotation matrix from world to body frame of leg i at time step k.

C. Problem Definition and Contributions

In this letter, we implement model-based controllers and auto-tune their control parameters. Model-based control offers the advantages that the physics of a dynamical system are utilized to make informed decisions, less data are needed to improve the robot's performance, safety guarantees may be provided, and that the computational structure remains fixed. This control philosophy stands in contrast to black-box controllers and reduces the amount of required tuning to a few selected control parameters.

Problem A: Auto-tune the gains of the swing controller in (5), i.e., $\mathbf{K}_p$, $\mathbf{K}_d$. Section IV-A addresses this problem by adjusting the controller calibration method in [16], [17] to a swing controller of a legged robot. This is achieved by defining a training objective suited for a swing controller as well as deriving a motion model suited to model the swing motion of a legged robot.

Problem B: Auto-tune the cost function weights of the stance controller in (6), i.e., $\mathbf{Q}, \mathbf{R}$. Section IV-B addresses this problem by adjusting the controller calibration method in [16], [17] to a stance controller of a legged robot. This is achieved by defining a training objective suited for a stance controller as well as deriving a motion model suited to model the stance phase of a legged robot.

Problem C: Develop concept for online computation of reference trajectories. Section IV-C addresses this problem by continuously generating reference trajectories for continuous operation of a legged robot.

Problem D: Develop concept for adaptively refining reference trajectory to the current task and the current environment. Section IV-D addresses this problem by adjusting the method in [16], [17] to calibrate parameters of reference trajectories. This is achieved by parameterizing the reference trajectories, defining a suitable training objective for the operation of the legged robot, and deriving a motion model for the robot's movements.

Finally, Section V presents the application of the proposed algorithms to a third-party high-fidelity simulator of the Unitree A1 in Gazebo. The high-fidelity simulator is provided by Unitree [28] and it includes contact dynamics, multi-body dynamics, sensor noise, etc.

IV. Auto-Tuning Controller and Reference Trajectories of Legged Robot

The overall implementation of the auto-tuning method is as follows: (1) we generate a feasible motion plan for the first few footsteps using TO; (2) we use this initialization as part of the reference-generator function to estimate new trajectories without optimization; (3) the auto-tuning formulation is employed for both improved trajectory tracking (i.e., auto-tuning the swing or stance controller) and modifying the trajectories directly in order to minimize energy consumption, produce forces that minimize slippage, or satisfy a desired step clearance (i.e., step height) or forward progress (i.e., base velocity), see Fig. 2

A. Training Objectives for Auto-Tuning Swing Controller

For auto-tuning the swing controller in (5), we parametrize the gains, i.e., $\mathbf{K}_p = \mathbf{K}_p(\boldsymbol{\theta})$, $\mathbf{K}_d = \mathbf{K}_d(\boldsymbol{\theta})$, and we use the dynamical model using the notation in (1) for leg i,

$$\mathbf{x}_{i,k} = \begin{bmatrix} \mathbf{q}_{i,k} \\ \dot{\mathbf{q}}_{i,k} \end{bmatrix} \tag{8a}$$

$$\mathbf{dyn}(\mathbf{x}_{i,k}, \boldsymbol{\tau}_{i,k}) = \begin{bmatrix} \mathbf{q}_{i,k} + dt\dot{\mathbf{q}}_{i,k} + dt^2\ddot{\mathbf{q}}_{i,k} \\ \dot{\mathbf{q}}_{i,k} + dt\ddot{\mathbf{q}}_{i,k} \end{bmatrix} \tag{8b}$$

$$\ddot{\mathbf{q}}_i = \mathbf{M}_i(\mathbf{q}_i)^{-1}(\boldsymbol{\tau}_i - \mathbf{V}_i(\mathbf{q}_i, \dot{\mathbf{q}}_i) - \mathbf{G}_i(\mathbf{q}_i)), \tag{8c}$$

where $\mathbf{M}_i \in \mathbb{R}^{3\times3}$ is the mass matrix in the joint space, and $\ddot{\boldsymbol{q}}_i \in \mathbb{R}^{3\times3}$ is the current joint acceleration of foot i. The training objective (2) is defined to improve reference trajectory tracking as

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{p}_{i,k-N|k}^{\mathrm{ref}} \\ \mathbf{v}_{i,k-N|k}^{\mathrm{ref}} \end{bmatrix}, \quad \mathbf{h}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{p}_{i,k-N|k} \\ \mathbf{v}_{i,k-N|k} \end{bmatrix}, \tag{8d}$$

where $\mathbf{p}_{i,k-N|k}^{\mathrm{ref}}$ and $\mathbf{v}_{i,k-N|k}^{\mathrm{ref}}$ are positions and velocities of foot i from time step $k-N$ through k as in (10) and (11).

The model mismatch $\mathbf{w}_{k-N|k} \in \mathbb{R}^6$ for the swing controller is computed using (8) and the measured states for the past swing execution. The UKF-based control parameter adaptation method uses (5) and (8) to "simulate" an execution of a swing using the gains defined by the sigma points, $\mathbf{K}_p(\boldsymbol{\theta}^{\mathrm{sp},j})$ and $\mathbf{K}_d(\boldsymbol{\theta}^{\mathrm{sp},j})$, considering the model mismatch, $\mathbf{w}_{k-N|k}$. Note that $\mathbf{p}_{i,k}^{b}$ and $\mathbf{v}_{i,k}^{b}$ in (5) can be calculated using the robot forward kinematics, which uses $\mathbf{q}_{i,k}$ as input, and the footstep Jacobian by $\mathbf{v}_{i,k}^{b} = \mathbf{J}_i(\mathbf{q}_{i,k})\dot{\mathbf{q}}_{i,k}$.

B. Training Objectives for Auto-Tuning Stance Controller

For auto-tuning the stance controller cost function weights in (6), i.e., $\mathbf{Q} = \mathbf{Q}(\boldsymbol{\theta})$, $\mathbf{R} = \mathbf{R}(\boldsymbol{\theta})$, we use the dynamical model (1) as

$$\mathbf{dyn}(\mathbf{x}_k, \mathbf{f}_k) = \begin{bmatrix} \boldsymbol{\Theta}_k + \mathbf{R}_b^w \boldsymbol{\omega}_k \\ \mathbf{r}_k + dt\mathbf{v}_k \\ \boldsymbol{\omega}_k + dt\left(\sum_{i=1}^{4} \hat{\mathbf{I}}^{-1} \left[\mathbf{p}_{i,k}^{b}\right]_{\times} \mathbf{f}_k^i\right) \\ \mathbf{v_k} + dt(\sum_{i=1}^{4} \frac{\mathbf{f}_k^i}{m} + \mathbf{g}) \end{bmatrix} \tag{9a}$$

with the state $\mathbf{x}_k$ defined as in (6), where $\hat{\mathbf{I}}$ represents the inertia tensor in the world frame ($\times$ indicates a skew matrix) [27], $\mathbf{R}_b^w$ is the rotation matrix from world to body frame, and $\mathbf{g}$ is the gravity vector. The training objective in (2) is defined to improve reference trajectory tracking as

$$\mathbf{y}_k = \begin{bmatrix} \mathbf{x}^{\text{ref}}_{k-N|k} \end{bmatrix}, \quad \mathbf{h}(\boldsymbol{\theta}) = \begin{bmatrix} \mathbf{x}_{k-N|k} \end{bmatrix} \tag{9b}$$

Here, too, the UKF-based control parameter adaptation method uses (6) and (9) to "simulate" the MPC using the cost function weights defined by the sigma points $\mathbf{Q}(\boldsymbol{\theta}^{\text{sp},j})$, considering the model mismatch $\mathbf{w}_{k-N|k}$.

C. Generating Reference Trajectories

To ensure kinematic and dynamically feasible motion for the first few footsteps, we apply the methods proposed in [4], which uses trajectory optimization.

1) Reference Trajectory for Feet: In order to continuously generate new reference trajectories, we use

$$\mathbf{p}^{\text{ref}}_{i,k} = \mathbf{p}^{\text{ref}}_{i,k-N} + \mathbf{v}^{\text{ref}}_{i,k-N}\Delta T \tag{10a}$$

$$\mathbf{p}^{\text{ref}}_{i,k} = \begin{bmatrix} p^{x,\text{ref}}_{i,k} \\ p^{y,\text{ref}}_{i,k} \\ p^{z,\text{ref}}_{i,k} \end{bmatrix}, \quad \mathbf{v}^{\text{ref}}_{i,k} = \begin{bmatrix} v^{x,\text{ref}}_{i,k} \\ v^{y,\text{ref}}_{i,k} \\ v^{z,\text{ref}}_{i,k} \end{bmatrix} \tag{10b}$$

where $\mathbf{p}^{\text{ref}}_{i,k-N}$ denotes the position of foot i on the ground at time step $k-N$, $\mathbf{p}^{\text{ref}}_{i,k}$ is the next position of foot i on the ground at time step k, which are determined by a desired CoM body velocity, $\mathbf{v}^{\text{ref}}_{i,k}$, and the phase time ΔT, i.e., the time that the foot is in stance or swing phase. Throughout, we assume $v^{z,\text{ref}}_{i,k} = 0$.

Then, we use $s(t) = 2\pi\frac{N-k+t}{N}$ with $t = k-N, k-N+1,..., k$ to create cycloidal footstep reference trajectories in between the ground positions in (10) with

$$p^{x,\text{ref}}_{i,t} = p^{x,\text{ref}}_{i,k-N} + \left(p^{x,\text{ref}}_{i,k} - p^{x,\text{ref}}_{i,k-N}\right)\frac{s(t) - \sin(s(t))}{2\pi} \tag{11a}$$

$$p^{y,\text{ref}}_{i,t} = p^{y,\text{ref}}_{i,k-N} + \left(p^{y,\text{ref}}_{i,k} - p^{y,\text{ref}}_{i,k-N}\right)\frac{s(t) - \sin(s(t))}{2\pi} \tag{11b}$$

and

$$p^{z,\text{ref}}_{i,t} = \begin{cases} p^{z,\text{ref}}_{i,k-N} + p^{z}_{\text{max}}\frac{1-\cos(s(t))}{2} \\ \quad \text{if } s(t) \leq \pi \\ p^{z,\text{ref}}_{i,k} + \left(p^{z}_{\text{max}} + p^{z,\text{ref}}_{i,k-N} - p^{z,\text{ref}}_{i,k}\right)\frac{1-\cos(s(t))}{2} \\ \quad \text{if } s(t) > \pi \end{cases} \tag{11c}$$

where p^{z}_{max} is the step clearance/apex height of the swing trajectory, i.e., the maximum p^z component of the footstep in swing. If the foot is in stance, $p^{z}_{\text{max}} = 0$.

2) Reference Trajectory for Center of Mass: The reference trajectory of the body's CoM is specified by

$$\begin{bmatrix} \mathbf{r}^{\text{ref}}_{k} \\ \boldsymbol{\Theta}^{\text{ref}}_{k} \end{bmatrix} = \begin{bmatrix} \mathbf{r}^{\text{ref}}_{k-N} \\ \boldsymbol{\Theta}^{\text{ref}}_{k-N} \end{bmatrix} + \begin{bmatrix} \mathbf{v}^{\text{ref}}_{k-N} \\ \boldsymbol{\omega}^{\text{ref}}_{k-N} \end{bmatrix}\Delta T \tag{12}$$

with the current body position, $\mathbf{r}^{\text{ref}}_{k-N} \in \mathbb{R}^3$, and orientation, $\boldsymbol{\Theta}^{\text{ref}}_{k-N} \in \mathbb{R}^3$, and $\mathbf{r}^{\text{ref}}_{k}, \boldsymbol{\Theta}^{\text{ref}}_{k}$ are composed of the next desired body position, and $\mathbf{v}^{\text{ref}}_{k-N}, \boldsymbol{\omega}^{\text{ref}}_{k-N}$ is the desired velocity (linear and angular) over the phase duration ΔT. Here, we use $\bar{s}(t) = \frac{N-k+t}{N}$ with $t = k-N, k-N+1,..., k$ to linearly interpolate between current and desired states with

$$\begin{bmatrix} \mathbf{r}^{\text{ref}}_{t} \\ \boldsymbol{\Theta}^{\text{ref}}_{t} \end{bmatrix} = \bar{s}(t)\begin{bmatrix} \mathbf{r}^{\text{ref}}_{k} \\ \boldsymbol{\Theta}^{\text{ref}}_{k} \end{bmatrix} + (1 - \bar{s}(t))\begin{bmatrix} \mathbf{r}^{\text{ref}}_{k-N} \\ \boldsymbol{\Theta}^{\text{ref}}_{k-N} \end{bmatrix}.$$

D. Training Objectives for Auto-Tuning Reference Trajectory

For auto-tuning the reference trajectory, we use the same states and dynamic model as for the swing controller in (8). The main difference is in the implementation of the training objective and control parameters, i.e., we do not auto-tune the $\mathbf{K}_p$ and $\mathbf{K}_d$ gain matrices. Instead, we tune two parameters often desirable for legged robots, which are to achieve a desired step clearance in order to step over an obstacle and reach a desired velocity. Hence for leg i, we parametrize the step clearance and the desired forward velocity,

$$p^{z}_{\text{max}} = \theta_z, \quad v^{x,\text{ref}}_{i,k} = \theta_v, \quad \boldsymbol{\theta} = \begin{bmatrix} \theta_z \\ \theta_v \end{bmatrix}$$

and the training objective for adapting such parameters is

$$\mathbf{y}_k = \begin{bmatrix} p^{z}_{\text{des}} \\ v^{x}_{\text{des}} \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{h}(\boldsymbol{\theta}) = \begin{bmatrix} h_z(\boldsymbol{\theta}) \\ h_v(\boldsymbol{\theta}) \\ h_f(\boldsymbol{\theta}) \\ h_e(\boldsymbol{\theta}) \end{bmatrix}, \tag{13}$$

where $h_z(\boldsymbol{\theta})$ is the achieved step clearance, $h_v(\boldsymbol{\theta})$ is the achieved velocity, $h_f(\boldsymbol{\theta})$ computes lateral forces of the robot and is used to reduce slippage, and $h_e(\boldsymbol{\theta})$ is the energy consumption. Note that p^{z}_{des} measures the achieved step clearance in closed loop, whereas p^{z}_{max} is the control parameter that defines the reference trajectory. Thus, p^{z}_{des} and p^{z}_{max} may be different, e.g., due to a model mismatch of the physical robot and the dynamical system model.

1) Step Clearance Optimization $h_z(\boldsymbol{\theta})$: To propagate the states for optimizing the step clearance, we use the same method as in Section IV-A. However, here we "simulate" the various reference trajectories defined using the sigma points for the swing leg, where we do not change the controller gains of $\mathbf{K}_p$, $\mathbf{K}_d$. Hence, we modify the reference trajectory of the footstep using the reference generator function described in Section IV-C. Thus, $h_z(\boldsymbol{\theta}) = \max(p^{z}_{k-N|k})$, i.e., the maximum value of the z component of the footstep trajectory (after propagation with $\mathbf{w}_{k-N|k}$).

2) Forward Progress Optimization $h_v(\boldsymbol{\theta})$: For forward progress, we propagate the CoM position using the desired velocity, v^{x}_{des}, with (12), where the initial state is the actual state.

3) Slippage Optimization $h_f(\boldsymbol{\theta})$: Representing slippage of the foot first requires the propagation of ground reaction forces along the trajectory, which can be achieved by rearranging the MPC state-space equations in (6),

$$\mathbf{f}_k = \mathbf{B}^{+}(\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k), \tag{14}$$

where $\mathbf{B}^{+}$ is the Moore-Penrose inverse and $\mathbf{x}_k$ is propagated using the sigma points. We chose (14) due to its similarity to (6), but other slippage formulations are possible, too, e.g., as in [29]. If the foot is not in contact, $\mathbf{f}_k = \mathbf{0}$. Then, the foot slippage is

computed using similar ideas to friction cone constraints (see, e.g., [4]) as

$$h_f(\boldsymbol{\theta}) = \sum_{i=1}^{4} \frac{\sqrt{f^x_{i,\text{avg}} + f^y_{i,\text{avg}}}}{f^z_{i,\text{avg}}},$$

$$\begin{bmatrix} f^x_{i,\text{avg}} \\ f^y_{i,\text{avg}} \\ f^z_{i,\text{avg}} \end{bmatrix} = \frac{1}{N} \sum_{t=k-N}^{k} \begin{bmatrix} |f^x_{i,t}| \\ |f^y_{i,t}| \\ |f^z_{i,t}| \end{bmatrix}.$$

4) Energy Consumption Optimization $h_e(\boldsymbol{\theta})$: The energy consumption is included as it may be desirable to find the best balance between maximizing step clearance and forward progress, while simultaneously minimizing the energy consumption required. The energy consumption can be computed using the joint velocity $\dot{\mathbf{q}}$, torque $\boldsymbol{\tau}_t$, and phase time ΔT, as

$$h_e(\boldsymbol{\theta}) = \sum_{t=k-N}^{k} |\dot{\mathbf{q}}_t|^\top |\boldsymbol{\tau}_t| \Delta T.$$

Remark 3: To ensure we do not violate kinematic or dynamic constraints as we simulate new trajectories using $\boldsymbol{\theta}$, we can use $|\mathbf{R}^b_w[\mathbf{p}_{i,k} - \mathbf{r}_k] - \overline{\mathbf{p}}^b_i| < \mathbf{b}$, where $\mathbf{R}^b_w$ is the rotation matrix from base to world frame, and $\overline{\mathbf{p}}^b_i$ is the nominal footstep position centered in the kinematic bounding box specified by $\mathbf{b} \in \mathbb{R}^3$. If the bounding box constraint does not hold, we can impose a cost for that particular simulation guiding the auto-tuner away from such control parameters.

V. Results Using Physics-Based Simulator

We implemented the auto-tuning algorithms for the Unitree A1 robot [28] within a realistic simulation using Gazebo with the bullet physics engine. The Unitree A1 simulator is made available by the robot manufacturer. The simulation considers not only localization noise but also friction within each torque-controlled motor, where torque commands are sent using the robotic operating system (ROS). Further, the Gazebo simulation environment is executed in a parallel thread, and hence the controller and the auto-tuning algorithm need to be executed during run-time of the simulation. Hence, being able to execute the applied algorithms using this high-fidelity simulator indicates their applicability to the physical robot. The A1 Unitree is a 12.7 kg quadruped robot with 3 degrees of freedom per leg. For obtaining an initial trajectory as in [4], we use the nonlinear programming solver IPOPT [30]. Optimization of the MPC stance controller was done using the quadratic programming solver qpOASES [31]. Constraints and overall problem formulation were setup using the CasADi [32] software. The phase time was chosen as 0.2 s, which is also the update frequency of the parameters. The swing and stance controller are executed at sampling frequency at $dt = 0.01$ s, and $N_{\text{MPC}} = 5$ in (6).

A. Auto-Tuning for Optimizing Step Clearance and Forward Progress

The first test case is to auto-tune reference trajectories aiming at providing a desired forward progress and step clearance. Here, we consider only the first and second element in (13), where we use $p^z_{\text{des}} = 0.15$ m and $v^x_{\text{des}} = 0.45$ m/s. The results are shown in Fig. 3-**A)**, **B)**, **C)**. In **A)**, we show the cost defined as $||\mathbf{y}_k - \mathbf{h}(\boldsymbol{\theta})||_{C_y^{-1}} = \|[p^z_{\text{des}} - h_z(\boldsymbol{\theta})\ v^x_{\text{des}} - h_v(\boldsymbol{\theta})]\|_{C_y^{-1}}$ as in (2). Thus, as the cost decreases, the closer the actual step clearance and forward progress get to the desired value. We decompose the results in **A)** and show how the actual velocity reaches the desired velocity in **B)**, and how the actual step clearance reaches the desired step clearance in **C)**.

B. Auto-Tuning for Optimizing Step Clearance, Forward Progress, Slippage, and Energy Consumption

In **D)**, we show the cost when setting a desired step clearance and forward progress with $p^z_{\text{des}} = 0.1$ m and $v^x_{\text{des}} = 0.4$ m/s, while minimizing energy consumption and foot slippage as in (13). The cost decreases quickly within a few time steps, which indicates that the auto-tuning method was able to find a good balance between reaching its desired step clearance and forward progress, while ensuring a minimization of both energy consumption and foot slippage.

C. Auto-Tuning Stance Controller

Next, we demonstrate the auto-tuning method to calibrate the gains of the stance controller. For simplicity, here we auto-tune only the diagonal elements of $\mathbf{Q} = \mathbf{Q}(\boldsymbol{\theta})$ in (6), but $\mathbf{R}$ can similarly be tuned. We initialize $\mathbf{Q} = \text{diag}([1, 1, Q_3, Q_4, Q_5, Q_6, 1000, 1, 1, 1, 1, 1])$, where Q_3, Q_4, Q_5, Q_6 are initialized to 300, and will be further calibrated, i.e., $\mathbf{Q}(\boldsymbol{\theta})$ with $\boldsymbol{\theta} \in \mathbb{R}^4$. The cost function is tuned online for 20 s of locomotion using trot gait. In Fig. 3-**E)**, we present the cost difference between the auto-tuning case and the no auto-tuning case. The difference decreases over time, which shows that the auto-tuning improves the cost using auto-tuning decreased significantly compared to the cost without using auto-tuning. Additionally, without auto-tuning, the robot eventually falls after about 8 seconds of locomotion, because the initial gains of the MPC controller have not been properly hand-tuned. Note that some hand-tuning was necessary to ensure initial gains that at least allow the robot to start moving a few steps without falling. After auto-tuning the stance controller for 20 s, $Q_3 = 5245.53$, $Q_4 = 1172.34$, $Q_5 = 662.45$, and $Q_6 = 1172.25$. Further, we applied the tuned controller along with the tuned swing controller in Section V-D and demonstrated robust locomotion on uneven terrain, see Fig. 4.

D. Auto-Tuning Swing Controller

Lastly, we apply the auto-tuning method to calibrate the gains of the swing controller, $\mathbf{K}_p$ and $\mathbf{K}_d$. We choose $\boldsymbol{\theta} \in \mathbb{R}^6$ consisting of the three diagonal elements of the proportional gain matrix, $\mathbf{K}_p$, and the three diagonal elements of the derivative gain matrix, $\mathbf{K}_d$. We initialized $\mathbf{K}_d = \text{diag}([0.1, 10, 10])$ and $\mathbf{K}_p = \text{diag}([150.11, 16.11, 10.11])$ to be equal to the leg's natural frequency using the inverted pendulum model multiplied by the operational mass matrix, see [27] for more detail. Fig. 3-**F)**, shows the difference of the cost when not auto-tuning with the cost when auto-tuning for over 20 seconds of locomotion. The

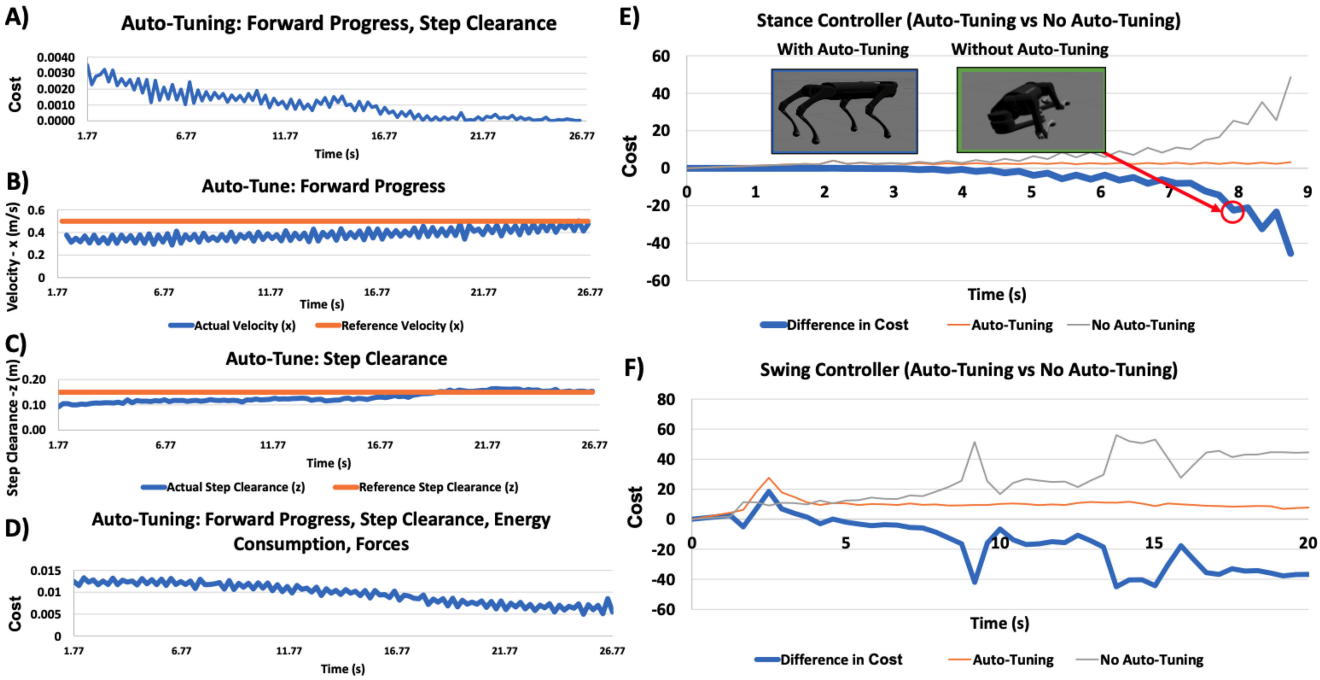

Fig. 3. Auto-Tuning results. The auto-tuning method successfully calibrated control parameters that generate reference trajectories (**A-D**), and controller gains (**E-F**). To make this evaluation, we used the cost as calculated using (2). The cost decreases (**A** and **D**) as the control parameters (e.g., forward progress and step clearance) get to their desired reference values. In $\mathbf{E}-\mathbf{F}$, we show the difference between the cost when not auto-tuning with the cost when auto-tuning.

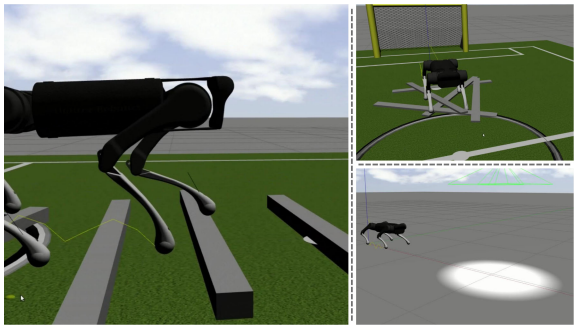

Fig. 4. Results for robust locomotion on uneven terrain. After auto-tuning the stance and swing controllers, we employ the robot on several test cases to demonstrate robust locomotion. We show the robot traversing over and on large beams and planks in the left and top right with a trot gait, respectively, and demonstrate a successful jumping gait in the bottom right. The robot is not aware of the obstacles and must overcome them by using the auto-tuned controllers.

cost curve follows a downward trend, demonstrating that the cost when auto-tuning is smaller than without auto-tuning. The gains using auto-tuning outperformed the gain selection without auto-tuning, which was based on the current natural frequency calculation and was shown to perform well in [27]. The final gains after tuning was $\mathbf{K}_d = \text{diag}([13.82, 12.42, 17.80])$ and $\mathbf{K}_p = \text{diag}([166.26, 22.75, 14.03])$.

Fig. 4 shows the robot walking on uneven ground using auto-tuning of the swing controller gains and the stance controller cost function.

Remark 4: While this letter presents one simulation run, we have obtained similar results for repeated test cases, which is expected as the auto-tuning formulation is robust to noise/disturbances due to its filter-based design.

TABLE I
AUTO-TUNING COMPUTATION TIMES

Auto-Tune	Min. Time	Median Time	Max. Time
Reference Traj. **A)**	0.0108s	0.0143s	0.0370s
Reference Traj. **B)**	0.0072s	0.0089s	0.0121s
Reference Traj. **C)**	0.0086s	0.0093s	0.0235s
Reference Traj. **D)**	0.0632s	0.0662s	0.0667s
Stance Controller **E)**	0.0409s	0.0465s	0.0807s
Swing Controller **F)**	0.0078s	0.0569s	0.1006s

E. Computation Times

The computation times of auto-tuning the swing and stance controllers, as well as the reference trajectory are shown in Table I. Table I lists the maximum, minimum, and median computation times of the auto-tuning algorithm for tuning the gains of the swing controller, the MPC weights of the stance controllers and the parameters of the reference trajectory (Reference Traj. $\mathbf{A}-\mathbf{D}$ in Table I refers to the training objectives specified by $\mathbf{A}-\mathbf{D}$ in Fig. 3). The computation times were obtained while running on a laptop using 4 CPU cores (Intel core i7-8850H CPU at 2.60 Ghz) with a Quadro P3200 GPU. Further, the controller ran in parallel with a Gazebo simulation, which requires more computational resources than an implementation on hardware would. As the computation times are below the update rate of the auto-tuner updating at the end of the phase time with 0.2 s, we can conclude that the algorithms in this letter can be executed in real time on comparable computational resources.

Remark 5: The computation times scale linearly with the amount of parameters to be tuned. Although the applied algorithms can be executed online, depending on user specifications, there is the option to execute the algorithms offline or in parallel. E.g., the auto-tuner for the swing and stance controllers can update the gains on a different thread, which runs in parallel to the low-level/high-level controllers of the robot. Additionally, the methods could also be implemented offline if computational resources are limited.

VI. Conclusion

In this work we successfully demonstrated robust locomotion. An auto-tuning method was implemented, which is based on an unscented Kalman filter formulation, to calibrate the gains of the swing controller and the cost function weights of a stance controller. We also demonstrated that the method can be applied for directly auto-tuning the reference trajectory, i.e., reference trajectories are calibrated to make the robot achieve a desired forward progress and step clearance, while also minimizing energy consumption and foot slippage. We showed that the method can be easily generalized to consider diverse control parameter by demonstrating the auto-tuning on both controller gains and on physically meaningful parameters of a reference trajectory. Future work may include experimental validation on the robot platform and extending the method to consider tuning for optimal footstep timings of various gait sequences, consider dynamic environments that require the auto-tuner to calibrate changing step heights and base velocities, and also adapt the aggressiveness of the tuning through the unscented Kalman filter's covariance matrices based the robot's environment or desired task. Future work may also include more sudden changes in the robot's motion, e.g., by leveraging the "prediction model" of the Kalman filter in addition to the "measurement model" used in this letter.

References

[1] M. H. Raibert, *Legged Robots That Balance*. Cambridge, MA, USA: MIT Press, 1986.

[2] S. Kajita *et al.*, "Biped walking pattern generation by using preview control of zero-moment point," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2003, pp. 1620–1626.

[3] S. Kuindersma, F. Permenter, and R. Tedrake, "An efficiently solvable quadratic program for stabilizing dynamic locomotion," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 2589–2594.

[4] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robot. Automat. Lett.*, vol. 3, no. 3, pp. 1560–1567, Jul. 2018.

[5] Y. Zhao, H.-C. Lin, and M. Tomizuka, "Efficient trajectory optimization for robot motion planning," in *Proc. 15th Int. Conf. Control, Automat. Robot. Vis.*, 2018, pp. 260–265.

[6] J. Norby and A. M. Johnson, "Fast global motion planning for dynamic legged robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 3829–3836.

[7] A. Bratta, R. Orsolino, M. Focchi, V. Barasuol, G. G. Muscolo, and C. Semini, "On the hardware feasibility of nonlinear trajectory optimization for legged locomotion based on a simplified dynamics," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 1417–1423.

[8] N. Rathod *et al.*, "Model predictive control with environment adaptation for legged locomotion," *IEEE Access*, vol. 9, pp. 145710–145727, 2021.

[9] V. Tsounis, M. Alge, J. Lee, F. Farshidian, and M. Hutter, "DeepGait: Planning and control of quadrupedal gaits using deep reinforcement learning," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 3699–3706, Apr. 2020.

[10] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning quadrupedal locomotion over challenging terrain," *Sci. Robot.*, vol. 5, no. 47, 2020, Art. no. eabc5986.

[11] K. Ito and F. Matsuno, "A study of reinforcement learning for the robot with many degrees of freedom - acquisition of locomotion patterns for multi-legged robot," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2002, pp. 3392–3397.

[12] Z. Liang, S. Zhu, and X. Jin, "Walking parameters design of biped robots based on reinforcement learning," in *Proc. 30th Chin. Control Conf.*, 2011, pp. 4017–4022.

[13] H. Kimura, T. Yamashita, and S. Kobayashi, "Reinforcement learning of walking behavior for a four-legged robot," in *Proc. 40th IEEE Conf. Decis. Control*, 2002, pp. 330–337.

[14] A. Schperberg, S. Tsuei, S. Soatto, and D. Hong, "SABER: Data-driven motion planner for autonomously navigating heterogeneous robots," *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 8086–8093, Oct. 2021.

[15] S. Gangapurwala, M. Geisert, R. Orsolino, M. Fallon, and I. Havoutis, "Real-time trajectory adaptation for quadrupedal locomotion using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 5973–5979.

[16] M. Menner, K. Berntorp, and S. Di Cairano, "A Kalman filter for online calibration of optimal controllers," in *Proc. IEEE Conf. Control Technol. Appl.*, 2021, pp. 441–446.

[17] M. Menner, K. Berntorp, and S. Di Cairano, "Automated controller calibration by Kalman filtering," 2021, *arXiv:2111.10832*.

[18] A. Marco, P. Hennig, J. Bohg, S. Schaal, and S. Trimpe, "Automatic LQR tuning based on gaussian process global optimization," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 270–277.

[19] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-efficient autotuning with bayesian optimization: An industrial control study," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 730–740, May 2020.

[20] R. Calandra, A. Seyfarth, J. Peters, and M. P. Deisenroth, "Bayesian optimization for learning gaits under uncertainty," *Ann. Math. Artif. Intell.*, vol. 76, no. 1, pp. 5–23, 2016.

[21] A. Rai, R. Antonova, S. Song, W. Martin, H. Geyer, and C. Atkeson, "Bayesian optimization using domain knowledge on the atrias biped," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 1771–1778.

[22] C. Gehring *et al.*, "Practice makes perfect: An optimization-based approach to controlling agile motions for a quadruped robot," *IEEE Robot. Automat. Mag.*, vol. 23, no. 1, pp. 34–43, Mar. 2016.

[23] Y. Rahman, A. Xie, J. B. Hoagg, and D. S. Bernstein, "A tutorial and overview of retrospective cost adaptive control," in *Proc. Amer. Control Conf.*, 2016, pp. 3386–3409.

[24] D. Clever, R. Malin Schemschat, M. L. Felis, and K. Mombaur, "Inverse optimal control based identification of optimality criteria in whole-body human walking on level ground," in *Proc. 6th IEEE Int. Conf. Biomed. Robot. Biomechatronics*, 2016, pp. 1192–1199.

[25] F. L. Haufe, S. Maggioni, and A. Melendez-Calderon, "Reference trajectory adaptation to improve human-robot interaction: A database-driven approach," in *Proc. 40th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, 2018, pp. 1727–1730.

[26] G. A. Terejanu, "Unscented Kalman filter tutorial," University at Buffalo, Buffalo, 2011.

[27] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[28] *Unitree Robot.*, "GitHub repository for Unitree A1," Accessed: Sep. 2021. [Online]. Available:https://github.com/unitreerobotics

[29] G. Bledt, P. M. Wensing, S. Ingersoll, and S. Kim, "Contact model fusion for event-based locomotion in unstructured terrains," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4399–4406.

[30] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Prog.*, vol. 106, no. 1, pp. 25–57, 2006.

[31] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Math. Program. Comput.*, vol. 6, no. 4, pp. 327–363, 2014.

[32] J. Andersson, J. Gillis, G. Horn, J. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, 2019.