Simulation-Driven Optimization of Stanley Controller Gains for Enhanced Tracking in Autonomous Navigation Robots

Héctor Pérez-Villeda[1], Clemens Mühlbacher[1], Konstantin Mautner-Lassnig[1]

***Abstract*— Fine-tuning controllers for robotic systems is a tedious process that often requires significant time for convergence and can lead to mechanical component wear. Having an accurate simulation of the robotic system and its environment can help reduce this effort and accelerate the tuning process.**

This work presents an optimization-based approach that leverages simulations to optimize control parameters before transferring them to a real mobile robot, significantly reducing fine-tuning effort and the need for extensive real-world testing. The method follows a two-stage process: first, calibrating the simulator to closely replicate the mobile robot's trajectory, and second, using the refined simulation to optimize the Stanley controller's gains. By aligning the simulator's behavior with real-world performance, we ensure that control tuning is both effective and time-efficient, allowing optimized parameters to be directly applied to the real system.

The methodology is validated through experiments comparing simulated and real-world trajectories, demonstrating that the optimized gains improve tracking accuracy. Additionally, we provide an estimation of the achieved improvements, including tracking error reduction, time savings, and energy consumption minimized by our approach, highlighting its efficiency in the fine-tuning process.

***Index Terms*— Autonomous navigation, Stanley controller, simulator optimization, control tuning, simulation-to-reality transfer, parameter optimization.**

I. INTRODUCTION

Autonomous navigation is a critical capability for mobile robots operating in dynamic environments. A key challenge in this domain is ensuring accurate trajectory tracking, which is essential for applications such as autonomous vehicles [6], warehouse automation [9], and field robotics [5]. Stanley controller is widely used due to its effectiveness in minimizing lateral errors and maintaining stability during navigation [11]. However, achieving optimal tracking performance requires careful tuning of the controller's gains, a process that is often time-consuming and tedious when performed directly on a physical robot.

To address this challenge, we propose a simulation-driven optimization framework that enhances the efficiency of Stanley controller gain tuning. Instead of manually adjusting control parameters in real-world experiments, our approach leverages automated hyperparameter optimization techniques to systematically refine both the simulator parameters and controller gains. Specifically, we employ Optuna [2], an efficient framework for hyperparameter search, to explore optimal configurations while minimizing trajectory tracking errors. By optimizing in simulation before transferring the learned parameters to the real system, our method reduces the need for extensive physical testing while maintaining real-world applicability [7], [8], [14].

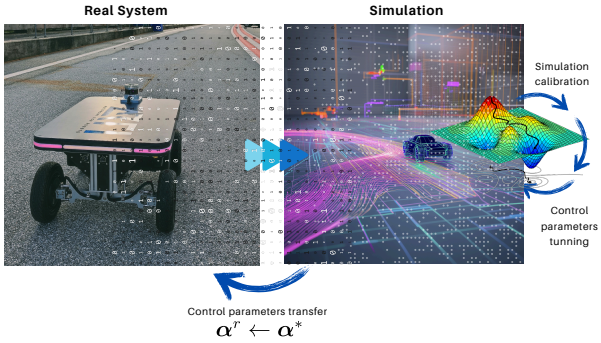

Fig. 1. On the left: CHASI, the ARTI robotic platform used in the experiments. Our method aims to calibrate the simulation environment to accurately replicate the real-world robot behavior. Once calibrated, control parameters are fine-tuned in simulation before transferring the optimized parameters to the real robot. This approach accelerates the tuning process while minimizing mechanical wear on the physical system.

The remainder of this paper is structured as follows: Section 2 describes the robot model used in this work. Section 3 defines the environment and the mathematical framework used throughout the paper. Section 4 details the proposed methodology, while Section 5 presents the experimental results along with their analysis. Finally, Section 6 concludes the paper and discusses directions for future work.

II. RELATED WORK

Accurate control tuning for mobile robots is a critical challenge, particularly in scenarios where real-world testing is expensive, risky, or time-consuming. A common strategy to address these challenges involves the use of high-fidelity simulators to replicate the behavior of robotic systems. However, discrepancies between simulation and actual performance, commonly referred to as the sim-to-real gap, can significantly reduce the reliability of control policies if the simulator is not properly calibrated.

Sim-to-real transfer techniques have received increasing attention in robotics. Domain randomization [13] and system identification methods [10] are often employed to bridge the simulation-reality gap by either increasing robustness

* This project is funded by the Austrian Research Promotion Agency (FFG). www.ffg.at, under the project: AVASI (Autonomous Vehicle Advanced Simulation)

[1]ARTI — AI software solutions for autonomous robots, Website:https://arti-robots.com, Emails: h.villeda@arti-robots.com, c.muehlbacher@arti-robots.com, k.ml@arti-robots.com

to environmental variation or aligning simulator dynamics with the real system. DROPO [12] focus on estimating domain randomization ranges for improving transferability of optimized policies. These methods often assume that the simulation environment and its physical model provides an adequate representation of the real-world system.

Hybrid learning strategies have been proposed to incorporate real-world data for improving simulation fidelity and learning efficiency [4]. Similarly, [1] evaluate various simulators, revealing limitations in modeling elastic impacts and complex motions, even with contact parameter tuning—highlighting the need for more accurate physical calibration.

While these works typically address either simulation calibration or control optimization in isolation, our method introduces an integrated three-stage approach: first refining simulation parameters to match real-world robot behavior, then optimizing control gains within the calibrated environment and finally transfer the learned parameter to the real robot. This ensures that the resulting policies are both physically grounded and readily transferable, minimizing reliance on real-world trials.

III. Robot Model

A. Ackerman Kinematic Model

The Ackerman kinematic model is widely used to describe the motion of wheeled vehicles with nonholonomic constraints [3]. It assumes no lateral slip and is based on the geometry of steering. The vehicle's motion is governed by the following equations:

$$\begin{aligned} \dot{x} &= v\cos(\theta), \\ \dot{y} &= v\sin(\theta), \\ \dot{\theta} &= \frac{v}{L}\tan(\delta). \end{aligned} \tag{1}$$

where x and y represent the vehicle's position coordinates, θ is the heading angle, v denotes the velocity, L is the wheelbase length, and δ corresponds to the steering angle.

B. ARTI-Controller

The ARTI-Controller is based on the Stanley method [11], a widely used approach for autonomous vehicle path following that minimizes cross-track and heading errors to ensure smooth trajectory convergence.

The steering control law is defined as:

$$\delta_v = \theta_e + \tan^{-1}\left(\frac{k e_{fa}}{v + v_{\min}}\right) \tag{2}$$

where δ_v is the steering output, $\theta_e = \theta - \theta_p$ is the heading error, and e_{fa} is the cross-track error from the front axle to the closest path point (c_x, c_y). The gain k adjusts the influence of the cross-track error, while v is the vehicle speed, and $v_{\min}$ ensures stability at low speeds.

To enhance robustness and adaptability at different speeds, the ARTI-Controller applies gain scheduling for k across velocity ranges, enabling dynamic tuning of the control response. The gain values are summarized in Table III-B. The geometric relationship of the control variables is illustrated in Figure 2.

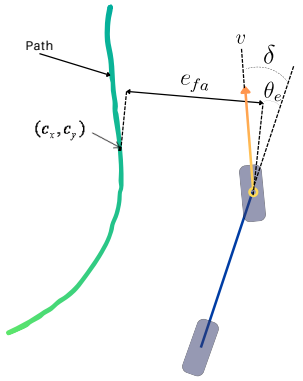

Fig. 2. Diagram illustrating the variables used by the Stanley controller to compute the control output

TABLE I
Stanley control Gains for different velocity ranges.

Velocity	$0 \le v < 0.5$	$0.5 \le v < 0.8$	$0.8 \le v \le 1.0$
k	k_1	k_2	k_3

IV. Environment Definition

A. Real-World (Physical) System Representation

The real system is modeled as a discrete-time nonlinear state-space system:

$$\mathbf{x}^r_{i+1} = f^r(\mathbf{x}^r_i, \mathbf{u}^r_i, \boldsymbol{\alpha}^r, \mathbf{T}^d) \tag{3}$$

where $\mathbf{x}^r_i = [x^r_i, y^r_i, \theta^r_i]^T$ represents the robot's position and orientation at time step i, and $\mathbf{u}^r_i = [\delta^r_{v_i}, v^r_i]^T$ denotes the steering and linear velocity control inputs. The dynamics function $f^r(\cdot)$ is defined according to the Ackermann model in Equation 1.

The closed-loop controller is parameterized by $\boldsymbol{\alpha}^r = \{k^r_1, k^r_2, k^r_3\}$, which affect the robot's tracking behavior. The desired trajectory is defined as:

$$\mathbf{T}^d = \left\{ \mathbf{x}^d_i = [c_{x_i}, c_{y_i}]^T \mid i = 1, \ldots, N^d \right\} \tag{4}$$

where $\mathbf{x}^d_i$ are the target waypoints in Cartesian coordinates.

A sampled trajectory from the real system consists of a discrete sequence of the robot's states, defined as:

$$\mathbf{T}^r_{\boldsymbol{\alpha}^r} = \{\mathbf{x}^r_i \mid i = 1, \ldots, N^r\} \tag{5}$$

where each $\mathbf{x}^r_i$ is a recorded state of the robot under the controller parameters $\boldsymbol{\alpha}^r$, and N^r is the number of sampled steps.

B. Simulated System Representation

The simulated system is represented as:

$$\mathbf{x}^s_{i+1} = f^s(\mathbf{x}^s_i, \mathbf{u}^s_i, \boldsymbol{\alpha}^s, \boldsymbol{\beta}, \mathbf{T}^d) \tag{6}$$

where $\mathbf{x}^s_i = [x^s_i, y^s_i, \theta^s_i]^T$ is the simulated state, and $\mathbf{u}^s_i = [\delta^s_{v_i}, v^s_i]^T$ is the control input, with $\delta^s_{v_i}$ as the steering angle

https://doi.org/10.34749/3061-0710.2025.14

and v_i^s as the linear velocity. The parameters $\boldsymbol{\alpha}^s = \{k_1^s, k_2^s, k_3^s\}$ configure the Stanley controller in simulation, while $\boldsymbol{\beta} = \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5\}$ defines configurable simulation parameters defined in the Table II used to more accurately capture the interaction between the robot and the real system. The function $f^s(\cdot)$ approximates the simulated system's transition dynamics.

A simulated trajectory consists of a sequence of discrete states:

$$\mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}} = \{\mathbf{x}_i^s \mid i = 1, \ldots, N^s\} \tag{7}$$

where N^s is the number of recorded samples. The trajectory $\mathbf{T}^s$ depends on the control parameters in the simulation environment $\boldsymbol{\alpha}^s$ and simulation parameters $\boldsymbol{\beta}$.

C. System Discrepancy Measure

To quantify the difference between two trajectories $\mathbf{T}_1 = [\mathbf{T}_{1_x}, \mathbf{T}_{1_y}]$ and $\mathbf{T}_2 = [\mathbf{T}_{2_x}, \mathbf{T}_{2_y}]$, we define a cost function that integrates well-known metrics: Mean Square Error (MSE) and Dynamic Time Warping (DTW) .

$$\mathscr{L}(\mathbf{T}_1, \mathbf{T}_2) = \mathrm{MSE}\left(\mathrm{DTW}(\mathbf{T}_{1_x}, \mathbf{T}_{2_x}), \mathrm{DTW}\left(\mathbf{T}_{1_y}, \mathbf{T}_{2_y}\right)\right) \tag{8}$$

DTW first aligns the x- and y-coordinate sequences to account for temporal variations; MSE then measures the deviation between the aligned pairs, yielding a robust similarity metric even under phase shifts or speed differences.

D. Problem Definition

Our objective is to minimize the tracking error between the real trajectory (5) of the physical robot (3) and the desired trajectory (4). This is achieved by fine-tuning the control parameters $\boldsymbol{\alpha}$. We formulate the optimization problem as:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \mathscr{L}(\mathbf{T}^d, \mathbf{T}^r_{\boldsymbol{\alpha}^r}) \tag{9}$$

where $\mathscr{L}(\mathbf{T}^d, \mathbf{T}^r_{\boldsymbol{\alpha}^r})$ represents the discrepancy between the desired trajectory T^d and the real trajectory $\mathbf{T}^r_{\boldsymbol{\alpha}^r}$.

To achieve this, we first use a simulator to capture the robot's initial real-world behavior through a calibration process. Then, we fine-tune the controller gains in the calibrated simulation before transferring these optimized gains to the real system.

V. Method

This section provides a detailed description of our method. Figure 3 illustrates the overall process, while Table 1 summarizes the key steps for clarity. Our approach is divided into four main steps:

1) **Real-World Data Collection**: Gather the robot's trajectory $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ following the desired trajectory $\mathbf{T}^d$ using the initial control parameters $\boldsymbol{\alpha}^r_0$.
2) **Simulation Calibration**: The goal of this step is to ensure that the robot's simulated trajectory $\mathbf{T}^d$ closely resembles the real trajectory $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ obtained using the initial control parameters. To achieve this, we transfer the initial control parameter values from the real robot to the simulation, i.e., $\boldsymbol{\alpha}^s_0 \leftarrow \boldsymbol{\alpha}^r_0$. Afterwards, we calibrate the simulation to ensure that the simulated robot's behavior closely approximates the real-world system, i.e., $\mathbf{x}_i^s \approx \mathbf{x}_i^r$ by fixing the values of the initial control parameters $\boldsymbol{\alpha}^s_0$ and optimizing the set of configurable simulation parameters $\boldsymbol{\beta}$ through the following minimization problem:

$$\begin{aligned} \boldsymbol{\beta}^* &= \arg\min_{\boldsymbol{\beta}} \mathscr{L}(\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}, \mathbf{T}^s_{\boldsymbol{\alpha}^s_0,\boldsymbol{\beta}}) \\ &\Rightarrow \mathbf{x}_i^s \approx \mathbf{x}_i^r, \quad \forall i \in \{1, \ldots, N\} \end{aligned} \tag{10}$$

The function $\mathscr{L}(\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}, \mathbf{T}^s_{\boldsymbol{\alpha}^s_0,\boldsymbol{\beta}})$, introduced in (8), quantifies the discrepancy between the simulated and real trajectories.

This process involves executing the path-following task in simulation using the same desired trajectory $\mathbf{T}^d$ from (4) and iteratively adjusting $\boldsymbol{\beta}$ until the optimal parameters $\boldsymbol{\beta}^*$ are obtained.
3) **Fine-tuning of control parameters**: Once the simulated robot accurately replicates the real robot's behavior, the objective of this step is to optimize the control parameters to ensure the simulated robot closely follows the desired trajectory $\mathbf{T}^d$. To achieve this, after calibrating the simulation state $\mathbf{x}_i^s$, we fix the optimal simulation parameters $\boldsymbol{\beta}^*$ obtained in the previous step and fine-tune the control gains $\boldsymbol{\alpha}^s$ by solving the following minimization problem:

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}^s} \mathscr{L}\left(\mathbf{T}^d, \mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}^*}\right), \tag{11}$$

This optimization process improves tracking accuracy, ensuring that the simulated trajectory $\mathbf{T}^s$ closely aligns with the desired trajectory $\mathbf{T}^d$.
4) **Transfer to Real System**: After optimizing the control parameters to ensure the simulated robot closely follows the desired trajectory, we transfer the tuned parameters $\boldsymbol{\alpha}^r \leftarrow \boldsymbol{\alpha}^*$ to the physical system (3) and validate their performance under real-world conditions.

TABLE II
Notation and name of variables

Symbol	Description
$\mathbf{T}^d$	Desired trajectory
$\mathbf{T}^r_{\boldsymbol{\alpha}^r}$	Real trajectory from the robot with control parameters $\boldsymbol{\alpha}^r$
$\mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}}$	Simulated trajectory with parameters $\boldsymbol{\alpha}^s, \boldsymbol{\beta}$
$\boldsymbol{\alpha}^r$, $\boldsymbol{\alpha}^s$	Control parameters (real-world & sim.)
$\boldsymbol{\alpha}^r_0$, $\boldsymbol{\alpha}^s_0$	Initial control parameters (real-world & sim.)
$\boldsymbol{\alpha}^*$	Optimized control parameters
$\boldsymbol{\beta}$	Set of simulation parameters
$\boldsymbol{\beta}_0$	Initial simulation parameters
$\boldsymbol{\beta}^*$	Optimized simulation parameters
β_1	Delay velocity
β_2	Delay steering
β_3	Max. allowed acceleration
β_4	Max. allowed angular velocity
β_5	Angular acceleration

https://doi.org/10.34749/3061-0710.2025.14

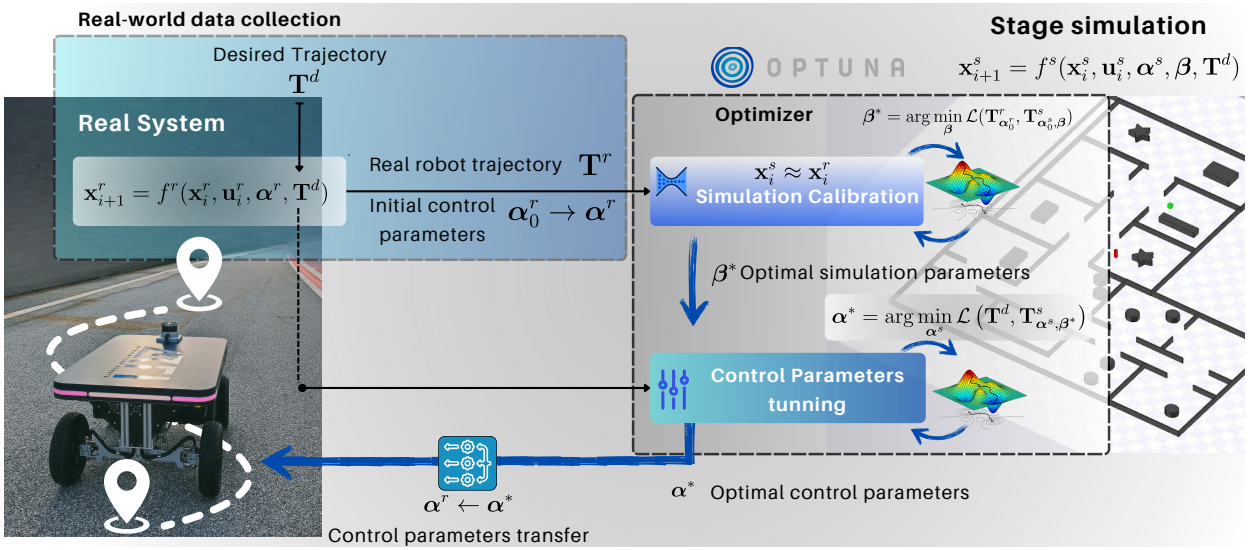

Fig. 3. Pipeline of our method: The process begins by collecting the real robot's trajectory while following the desired path. This trajectory is then used to calibrate the simulation environment iteratively until the simulated behavior closely matches the real-world performance. Once calibrated, control parameters are fine-tuned in simulation and subsequently transferred to the real robot to enhance its tracking accuracy.

Algorithm 1 Simulation-Based Control Gain Optimization

Input: Desired trajectory $\mathbf{T}^d$
Output: Optimized control parameters $\boldsymbol{\alpha}^*$
Step 1: Real-World Data Collection
$\mathbf{T}^r_{\boldsymbol{\alpha}^r} = \{\mathbf{x}^r_i \mid i = 1, \ldots, N^r\}$
Step 2: Simulation Calibration $\mathbf{x}^{sl}_i \approx \mathbf{x}^r_i$
Initialize $\boldsymbol{\alpha}^s_0 \leftarrow \boldsymbol{\alpha}^r_0$; $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}_0$ ▷ Load simulation default values
while $\mathscr{L}(\mathbf{T}^r_{\boldsymbol{\alpha}^r}, \mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}}) \geq \varepsilon$ **do**
    $\mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}} = \{\mathbf{x}^s_i \mid i = 1, \ldots, N^s\}$
    $\boldsymbol{\beta} = \boldsymbol{\beta} - \eta \nabla_{\boldsymbol{\beta}} \mathscr{L}(\mathbf{T}^r_{\boldsymbol{\alpha}^r}, \mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}})$
end while
Return $\boldsymbol{\beta}^* \leftarrow \boldsymbol{\beta}$ ▷ Final optimized parameters
Step 3: Fine-Tuning of Control Parameters in sim.
$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta}^*$ ▷ Load optimized simulation parameters
while $\mathscr{L}(\mathbf{T}^d, \mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}}) \geq \varepsilon$ **do**
    $\mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}} = \{\mathbf{x}^s_i \mid i = 1, \ldots, N^s\}$
    $\boldsymbol{\alpha}^s = \boldsymbol{\alpha}^s - \eta \nabla_{\boldsymbol{\alpha}^s} \mathscr{L}(\mathbf{T}^d, \mathbf{T}^s_{\boldsymbol{\alpha}^s,\boldsymbol{\beta}})$
end while
Return $\boldsymbol{\alpha}^* \leftarrow \boldsymbol{\alpha}^s$ ▷ Final optimized control parameters
Step 4: Control parameters transfer
$\boldsymbol{\alpha}^r \leftarrow \boldsymbol{\alpha}^*$

VI. Experiments and Results

We evaluated our approach using the CHASI robotic platform, a mobile robot with an Ackermann steering configuration (0.8 m × 0.6 m × 0.45 m). Simulations were conducted in the Stage simulator, a lightweight 2D tool that efficiently models sensor data and robot motion. Our navigation stack was fully integrated into Stage, enabling controlled and repeatable testing.

For real-world validation, we collected trajectory data in a 14 m × 2 m test area and compared it with the simulated results. Simulation and control parameter tuning were optimized using Optuna, a widely used hyperparameter optimization framework.

A. Real-World Collected Data

For this experiment, we used the desired trajectory illustrated in Figure 4 (a), denoted as $\mathbf{T}^d$. This trajectory consists of both straight-line segments and two sharp curves, designed to assess and optimize the robot's performance in both linear and curved path-following scenarios.

The actual trajectory followed by the robot, denoted as $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ using the initial default control parameters $\boldsymbol{\alpha}^r_0$, is also shown in Figure 4 (a). It can be observed that the robot successfully tracks the straight-line segments of the trajectory. However, when navigating the curved sections, the robot struggles to maintain accurate path tracking, exhibiting a noticeable error gap between the desired and actual trajectories.

B. Simulation calibration

This step aimed to align the simulator with the real robot's behavior. Figure 4 (b) shows the simulated trajectory $\mathbf{T}^s_{\boldsymbol{\alpha}^s_0,\boldsymbol{\beta}_0}$ using default control parameters $\boldsymbol{\alpha}^s_0$ and simulation parameters $\boldsymbol{\beta}^s_0$. While the trajectory appears to follow $\mathbf{T}^d$, a discrepancy with the real trajectory $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ reveals inaccuracies in the simulation.

To improve fidelity, we used the real robot's trajectory $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ as a reference, aiming to match its deviations while tracking $\mathbf{T}^d$. Keeping the control parameters fixed at $\boldsymbol{\alpha}^s_0$, we optimized the simulation parameters $\boldsymbol{\beta}$ using Optuna. This adjustment resulted in a refined simulated trajectory, $\mathbf{T}^s_{\boldsymbol{\alpha}^r_0,\boldsymbol{\beta}^*}$, which more closely aligned with the real-world trajectory. As shown in

https://doi.org/10.34749/3061-0710.2025.14

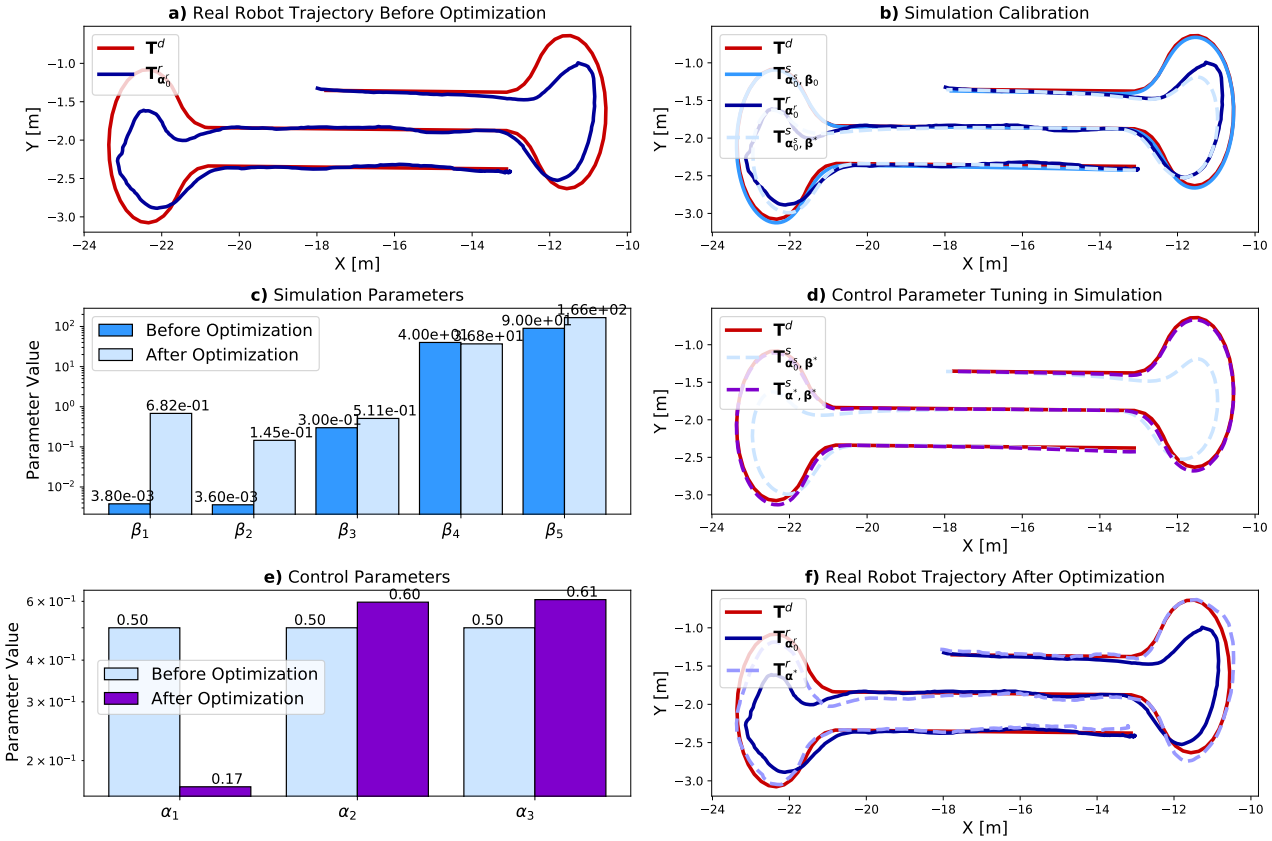

Fig. 4. (a) Robot performance before control parameter optimization. (b) Simulation inaccuracy before calibration and the improvement afterward. (c) Changes in simulation parameters after calibration. (d) Simulation improvement after tuning the control parameters. (e) Adjustments in control parameters after fine-tuning. (f) Final improvement in real-robot tracking using the fine-tuned control parameters.

Figure 4 (b), this calibrated trajectory better represents the real robot's performance.

Figure 4 c) presents a comparison between the default initial simulation parameters $\boldsymbol{\beta}_0$ and the optimized parameters $\boldsymbol{\beta}^*$, highlighting the changes introduced through the calibration process. The most significant adjustments can be observed in the velocity delay and steering delay, suggesting that the real robot experiences inherent delays when executing both velocity and steering commands. Additionally, the maximum allowable acceleration was increased, while the maximum angular velocity was slightly reduced. Conversely, the steering angle acceleration was increased. These adjustments enable the simulated robot to better replicate the real-world robot's behavior, effectively capturing hardware-induced limitations and response delays.

C. Control Parameter Tunning

In this step, we fine-tuned the control parameters in simulation, $\boldsymbol{\alpha}^s$, while keeping fixed the optimized simulation parameters $\boldsymbol{\beta}^*$ from the previous stage. Since $\boldsymbol{\beta}^*$ already captures the real robot's characteristics, the goal was to adjust $\boldsymbol{\alpha}^s$ to ensure the simulated robot closely follows the desired trajectory, $\mathbf{T}^d$. After optimization, we obtained an improved trajectory, $\mathbf{T}^s_{\boldsymbol{\alpha}^*,\boldsymbol{\beta}^*}$, which better aligns with the reference trajectory. The resulting trajectories are shown in Figure 4 (d), while Figure 4 (e) compares the initial control parameters, $\boldsymbol{\alpha}^s_0$, with the optimized parameters, $\boldsymbol{\alpha}^*$.

D. Transfer Learning

In this step, the optimized control parameters are transferred to the real robot $\boldsymbol{\alpha}^r \leftarrow \boldsymbol{\alpha}^*$. The robot is then tested again in the same environment, following the initial reference trajectory to evaluate its performance.

Figure 4 f) illustrates the desired trajectory $\mathbf{T}^d$, the real robot's trajectory before optimization, $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$, and the trajectory obtained after applying the optimized control parameters, $\mathbf{T}^r_{\boldsymbol{\alpha}^*}$. As observed, the optimized trajectory follows the desired path more closely. While the robot already performed well on straight-line segments, the most significant improvement is evident in tracking the curved sections, demonstrating the effectiveness of our method for this trajectory.

E. Achieved Improvements

This section presents the improvements in three key aspects: tracking error, time savings, and energy consumption. The results are summarized in Table III, with the corresponding calculations detailed in Appendix VIII. These estimations are based on approximate data.

https://doi.org/10.34749/3061-0710.2025.14

The table shows that our method achieved a 51.08% reduction in tracking error, saved approximately 3.84 hours of execution time, and reduced energy consumption by 9.175 kWh.

TABLE III
PERFORMANCE AND RESOURCE CONSUMPTION IMPROVEMENTS

Metric	Original Gains	Optimal Gains	Improved
Tracking Error (2D-DTW)	3.72	1.82	51.08%
	Estimated Real System Consumption	Optimized Method Consumption	Reduction
Time (hours)	9.34	5.50	3.84
Energy Consumption (kWh)	9.34	0.165	9.175

VII. CONCLUSION

We presented a simulation-driven framework to optimize the Stanley controller for autonomous navigation. The approach follows a two-stage process: first calibrating simulation parameters to reflect real-world dynamics, then optimizing controller gains within the calibrated simulator. This method reduces real-world experimentation, lowers mechanical wear, and improves trajectory tracking. The resulting control parameters transferred effectively to the physical robot, validating the framework's reliability.

Future work will explore broader trajectory variations to derive more generalized gains and incorporate sensor noise modeling—particularly from laser scans—as additional tuning parameters, given their significant impact on navigation in dynamic environments.

VIII. APPENDIX

A. Tracking Error Improvement Calculation

The tracking error improvement is computed using the following equation:

$$\text{Improvement}\% = \frac{e_{ig} - e_{og}}{e_{ig}} \times 100 \tag{12}$$

where $e_{ig} = 3.72$ represents the tracking error with the initial gains, whereas $e_{ig} = 1.82$ represents the tracking error with the optimized gains obtained using our method. This formulation quantifies the relative improvement achieved through optimization.

B. Time Savings Calculation

To estimate the time required for real-system optimization, we define:

- t_a = time to complete one trajectory (170 sec). This value has been obtained directly from $\mathbf{T}^r_{\boldsymbol{\alpha}^r_0}$ during the data collection.
- σ_a = repositioning time before a new trial (120 sec). This time was obtained experimentally in past experiments.
- N_o = total trials required for the control parameter optimization (116)

The total estimated time required on the real system is:

$$\text{Estimated Real System Time} = \frac{(t_a + \sigma_a) \cdot N_o}{3600}$$

Using our method, the time per simulation trial was $t_s = 45$ sec, with $N_s = 324$ trials needed for simulation calibration. The total time spent in simulation is:

$$\text{Estimated Simulation Time} = \frac{(t_s \cdot N_o) + (t_s \cdot N_s)}{3600}$$

C. Energy Consumption Calculation

To estimate the energy consumption required for tuning the control parameters on the real robot, we use $p_r \cdot t_{op}$, where $p_r = 1kW$ is the robot's power consumption, obtained from technical datasheet; $t_{op} = 9.34h$ is the estimated time required to complete the tuning process on the real robot.

To compute the energy consumption using our method, we consider a standard laptop's power consumption of $p_c = 0.030kW$. Given that the total simulation time is $t_s = 5.5h$, the energy consumption is calculated as $p_c \cdot t_s$

REFERENCES

[1] B. Acosta, W. Yang, and M. Posa, "Validating robotics simulators on real world impacts," *CoRR*, vol. abs/2110.00541, 2021. [Online]. Available: https://arxiv.org/abs/2110.00541

[2] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[3] T. D. Gillespie, *Fundamentals of Vehicle Dynamics*. SAE International, 1992.

[4] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," *CoRR*, vol. abs/1902.03701, 2019. [Online]. Available: http://arxiv.org/abs/1902.03701

[5] J. U. Kreber, J. Schlechtriemen, J. Boedecker, and W. Burgard, "Learning a terrain- and robot-aware dynamics model for autonomous mobile robot navigation," *Robotics and Autonomous Systems*, 2024, https://arxiv.org/html/2409.11452v1.

[6] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[7] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 3803–3810.

[8] F. Ramos, R. C. Possas, and D. Fox, "Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators," 2019. [Online]. Available: https://arxiv.org/abs/1906.01728

[9] E. O. Sodiya, U. J. Umoga, O. O. Amoo, and A. Atadoga, "Ai-driven warehouse automation: A comprehensive review of systems," *GSC Advanced Research and Reviews*, vol. 18, no. 02, pp. 272–282, 2024. [Online]. Available: https://gsconlinepress.com/journals/gscarr/sites/default/files/GSCARR-2024-0063.pdf

[10] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," 2018. [Online]. Available: https://arxiv.org/abs/1804.10332

[11] S. Thrun, M. Montemerlo, H. Dahlkamp, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[12] G. Tiboni, K. Arndt, and V. Kyrki, "Dropo: Sim-to-real transfer with offline domain randomization," *Robotics and Autonomous Systems*, p. 104432, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889023000714

[13] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017. [Online]. Available: https://arxiv.org/abs/1703.06907

[14] Z. Wang, X. Li, J. Yang, Y. Liu, and S. Jiang, "Sim-to-real transfer via 3d feature fields for vision-and-language navigation," 2024. [Online]. Available: https://arxiv.org/abs/2406.09798

https://doi.org/10.34749/3061-0710.2025.14