IEEE 21st International Conference on Automation Science and Engineering (CASE), Los Angeles, USA, 2025.

Integration of the TIAGo Robot into Isaac Sim with Mecanum Drive Modeling and Learned S-Curve Velocity Profiles

Vincent Schoenbach[1], Marvin Wiedemann[2], Raphael Memmesheimer[1], Malte Mosbach[1], and Sven Behnke[1]

***Abstract*— Efficient physics simulation has significantly accelerated research progress in robotics applications such as grasping and assembly. The advent of GPU-accelerated simulation frameworks like Isaac Sim has particularly empowered learning-based methods, enabling them to tackle increasingly complex tasks. The PAL Robotics TIAGo++ Omni is a versatile mobile manipulator equipped with a mecanum-wheeled base, allowing omnidirectional movement and a wide range of task capabilities. However, until now, no model of the robot has been available in Isaac Sim. In this paper, we introduce such a model, calibrated to approximate the behavior of the real robot, with a focus on its omnidirectional drive dynamics. We present two control models for the omnidirectional drive: a physically accurate model that replicates real-world wheel dynamics and a lightweight velocity-based model optimized for learning-based applications. With these models, we introduce a learning-based calibration approach to approximate the real robot's S-shaped velocity profile using minimal trajectory data recordings. This simulation should allow researchers to experiment with the robot and perform efficient learning-based control in diverse environments. We provide the integration publicly at `https://github.com/AIS-Bonn/tiago_isaac`.**

I. Introduction

General-purpose mobile manipulators hold great potential for automating a wide variety of tasks. By combining mobility with universal manipulation capabilities, such robotic platforms can perform various tasks, including human-robot collaboration and object retrieval. The resulting applications span industries from logistics and service robotics to healthcare. However, realizing the full potential of mobile manipulators remains a challenge due to the complexities of integrating robust perception and control systems and the countless diverse environments such robots might face.

Physics simulation has become an essential tool in robotics research and development, as it provides a cost-effective and scalable means to train, evaluate, and refine robotic approaches before real-world deployment. Modern simulation pipelines allow for accelerated faster-than-real-time physics simulation, unlocking new magnitudes of data that can be generated, which is a critical component for learning-based methods on both the perception and control side. Additionally, simulation offers a safe environment for training and testing control algorithms, minimizing the risk of damage to both the robot and its surroundings. These advantages have made physics simulation a cornerstone of robotic development in both academia and industry.

[1]V. Schoenbach, R. Memmesheimer, M. Mosbach, and S. Behnke are with the Autonomous Intelligent Systems group of University of Bonn, Germany; vschoenb@uni-bonn.de

[2]M. Wiedemann is with the Department Robotics and Cognitive Systems, Fraunhofer Institute for Material Flow and Logistics, Dortmund, Germany.

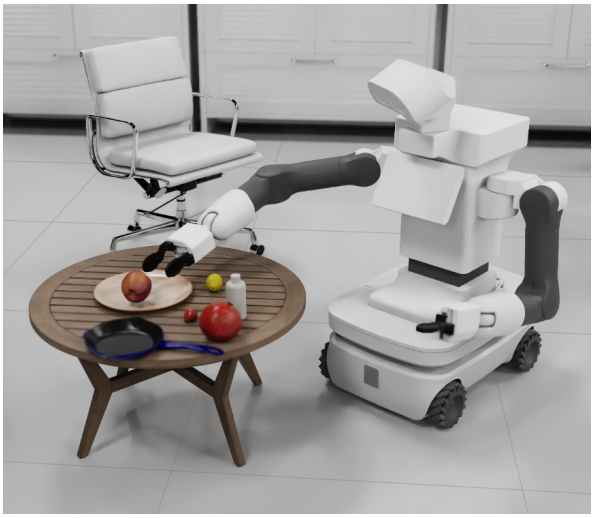


Fig. 1. Sample view of the TIAGo++ Omni integrated into Isaac Sim.

The PAL Robotics TIAGo robot [1] is a widely adopted general-purpose mobile manipulator in the research community, valued for its versatility and close integration with the ROS framework. However, like many robots, TIAGo is primarily simulated in Gazebo [2]. While Gazebo has long been a standard in robotics simulation [3], its limited physics accuracy and graphical realism pose challenges for applications requiring precise modeling of both robot dynamics and environment perception.

Modern simulation tools, such as NVIDIA's Isaac Sim [4], offer significant advantages in computational efficiency, realism, and compatibility with modern machine learning pipelines. Its underlying physics engine leverages GPU-based parallelizationwidely used in deep learningto enable massively parallel simulations, making it well-suited for reinforcement learning (RL) and large-scale data generation. Furthermore, tools such as Isaac Lab [5], built on Isaac Sim, provide seamless integration with RL frameworks, allowing researchers to apply state-of-the-art algorithms to their specific learning problems.

Despite these advantages, TIAGo has not yet been integrated into Isaac Sim due to several challenges. First, the provided Gazebo model is not directly compatible with Isaac Sim. Second, accurately simulating TIAGo's omnidirectional base requires precise modeling of mecanum wheels, which involves computationally expensive physics calculations. Third, replicating the mecanum wheel controller in simulation is difficult due to the lack of access to its internal

code, making it challenging to predict how the wheels should accelerate. Additionally, we found that tuning PID controllers in Isaac Sim does not accurately reflect the real robot's behavior, necessitating an alternative approach.

In this work, we present an integration of the TIAGo++ Omni platform into Isaac Sim, enabling its use in modern simulation environments. Specifically, we contribute the following:

- A physically accurate model of the TIAGo++ Omni, including a high-fidelity simulation of its mecanum wheels for omnidirectional driving.
- As an alternative, a lightweight velocity-based control model that approximates the real robot's motion while reducing computational overhead.
- A neural network-based calibration approach that approximately aligns both control models with the S-shaped velocity profile of the TIAGo++ using real-world movement data.

By making our integration publicly available, we aim to facilitate research in mobile manipulation, reinforcement learning (RL), and perception. To the best of our knowledge, no publicly available Isaac Sim model currently exists for a dual-arm omnidirectional robot. Furthermore, we hope this integration effort serves as a guide for future robotic projects involving omnidirectional robots in Isaac Sim.

II. Related Work

Several studies have addressed the modeling of omni-wheeled platforms, though most utilize simulation tools other than Isaac Sim. For instance, [6] employs the Matlab-Simulink SimScape Toolbox and SolidWorks to model an omni-wheeled platform, while [7] uses RecurDyn for a similar purpose. Additionally, [8], [9] rely on Gazebo for their modeling. In contrast, [10] leverages Isaac Sim to simulate a highly dynamic omnidirectional robot. Most of these studies primarily focus on kinematic modeling and validate their simulations by comparing the motion of the simulated and real robots.

Ensuring that a simulation model accurately reflects the real robot's behavior is crucial for successful sim-to-real transfer. Real2sim approaches [11] optimize simulation models using real-world data. This data is often used as a reference for manual model tuning; for example, [10] enhances model accuracy by incorporating motion capture data. Other studies go further by employing data-driven simulation models. Research on mathematical dynamic models, for instance, identifies model parameters using real-world data [12], [13], even accounting for effects such as slipping [14]. [15] applies machine learning to model an omnidirectional mobile robot, simulating its control unit, PID controllers, motors, and wheelsall trained on real-world measurements.

However, many of these studies do not account for physics engines, as common robotics simulators like Gazebo and Isaac Sim are non-differentiable. This non-differentiability prevents gradient-based estimation of model parameters, requiring trial-and-error strategies instead. For example, Evolutionary Algorithms are employed in [16] to optimize simulation models of manipulators.

To the best of our knowledge, no prior work models the S-shaped velocity profile of mecanum wheels using a neural network trained on limited, simple trajectory data recordings. Most existing research focuses on system identification or control using neural networks, or both, achieving strong performance in closed-loop trajectory tracking [17], [18]. However, such system identification methods are not directly compatible with physics simulators like Isaac Sim. In contrast, our approach does not address closed-loop trajectory tracking but instead learns a lower-level velocity model-specifically, how the wheels accelerate when transitioning between motion commands. This allows seamless integration of our velocity model into the Isaac Sim robot simulation via the Isaac Sim API.

III. Building the Robot Integration

This section describes the integration of the *TIAGo++ Omni* robot into Isaac Sim, focusing on the simulation of its omnidirectional movement. We implement two controllers: a physically accurate controller that models full wheel dynamics and a lightweight controller that directly sets the base velocity for efficiency. Both controllers are implemented as an Isaac Sim extension, incorporating a small neural network trained to predict the velocity curves of individual wheels.

The *TIAGo++ Omni* robot is a mobile manipulator with two gripper-equipped arms, an omnidirectional mecanum wheel drive, and multiple sensors, including an RGB-D camera and LiDAR.

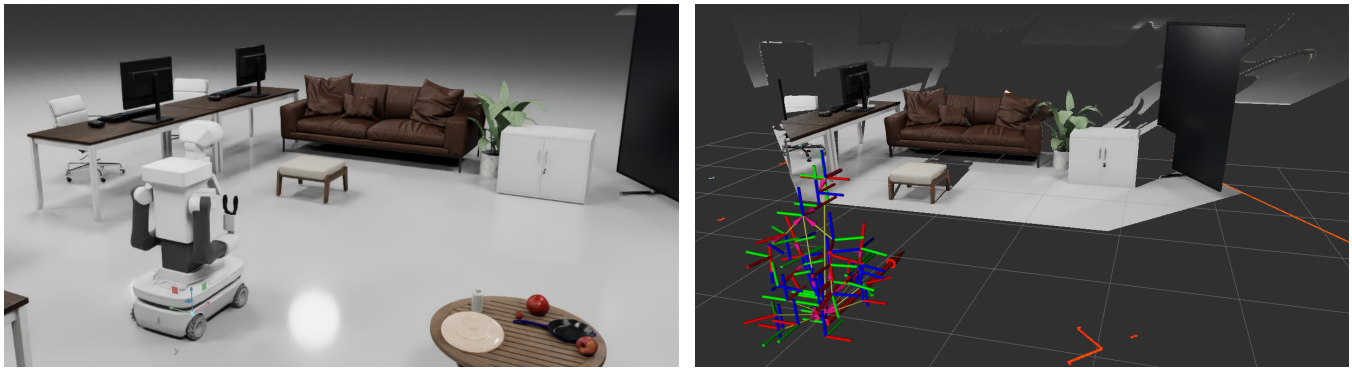
In the following subsections, we detail the integration of the omnidirectional drive, which posed a significant challenge due to the unique mecanum wheel mechanism.

Beyond the wheels, the remaining jointsincluding the torso, arms, fingers, and headwere successfully imported using the Isaac Sim URDF importer. Joint control is managed via ROS 2 and the Isaac Sim API, where joint state messages are sent and received. To synchronize joint trajectory commands between the simulated and real robot, practitioners must convert trajectory commands into a sequence of joint state messages.

Integrating sensors such as LiDAR and cameras into Isaac Sim is straightforward. The simulator provides various built-in sensors, and using the Isaac Sim API, sensor data can be recorded and published to ROS 2 topics. Example sensor data and the corresponding transform tree visualization in RViz 2 are shown in Fig. 2.

A. Integration of the Omnidirectional Drive by Full Physical Simulation

In the Gazebo simulation provided by PAL Robotics, the TIAGo robot's omnidirectional drive is approximated rather than physically simulated [19]. While some efforts in Gazebo aim to improve omnidirectional drive modeling [9], mecanum and similar wheels are typically not simulated with full physical accuracy. As a result, direct omnidirectional



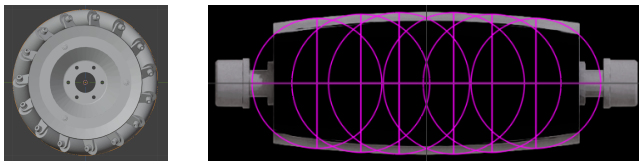
(a) Isaac Sim (b) RViz Visualization

Fig. 2. (a) Simulation in Isaac Sim and (b) corresponding sensor data visualization in RViz 2.

driving in Isaac Sim is not feasible using the imported `.urdf` file and requires a custom extension.

We address this by implementing a physically accurate simulation based on [10], following three key steps: (i) wheel modeling, (ii) roller collider modeling, and (iii) implementation of the holonomic controller in Isaac Sim.

For mecanum wheel modeling, we replaced the wheels from the `.urdf` file with custom ones (Fig. 3a) generated using the script from [20]. These wheels feature 15 free-spinning rollers angled at 45 degrees and are scaled to match the original size. Proper roller alignment and a precisely round wheel shape are crucial, as misalignment can lead to loss of ground contact and unintended jumps in driving behavior [21]. The generated wheels in our simulation ensure these requirements are met.



(a) Procedurally generated mecanum wheel. (b) Roller collider modeled with six spheres, balancing computational efficiency and physical accuracy.

Fig. 3. Model of (a) the mecanum wheel and (b) the roller colliders.

A crucial aspect of the simulation is the roller collider design, as ground contact directly affects the robot's motion, particularly for lateral driving. Following the methodology in [10], we modeled each roller collider using six spheres (Fig. 3b).

This approach ensures smooth motion in Isaac Sim while being significantly more computationally efficient than mesh-based or low-poly collision models [9].

To control the robot omnidirectional, the wheel velocities for all four wheels must be computed from the movement command, which is provided as a `Twist` message. A `Twist` command specifies linear velocities v_x, v_y along the x- and y-axes, as well as rotational velocity v_θ around the robot's center.

If r denotes the wheel radius, and L_x, L_y represent the distances from the robot's center to the wheels along the x- and y-axes, then the relationship between the base velocity (v_x, v_y, v_θ) and the four angular wheel velocities ω_i can be derived to be (see e.g., [22]):

$$\begin{bmatrix}\omega_1\\ \omega_2\\ \omega_3\\ \omega_4\end{bmatrix} = \frac{1}{r}\begin{bmatrix}1 & 1 & -(L_x+L_y)\\ 1 & -1 & (L_x+L_y)\\ 1 & -1 & -(L_x+L_y)\\ 1 & 1 & (L_x+L_y)\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ v_\theta\end{bmatrix}. \quad (1)$$

While this formulation allows the robot to move in the desired direction, it does not fully replicate the real robot's base movement, which behaves as a black boxi.e., the exact mapping from velocity commands to wheel accelerations is unknown. Notably, the real robot's controller exhibits S-shaped velocity curves (Fig. 4), but since we lack access to the internal controller, we cannot directly inspect its behavior.

As the matrix formulation alone does not account for these acceleration dynamics, we recorded real-world trajectory data and trained a small neural network to predict the required acceleration profiles for given movement commands, as described in the following section.

B. Predicting the Wheel Velocity Profile from Data

To approximate the smooth acceleration profile of the real robot for any given target wheel velocity, we train a neural network to predict an S-shaped velocity curve that closely matches the recorded data (Fig. 4).

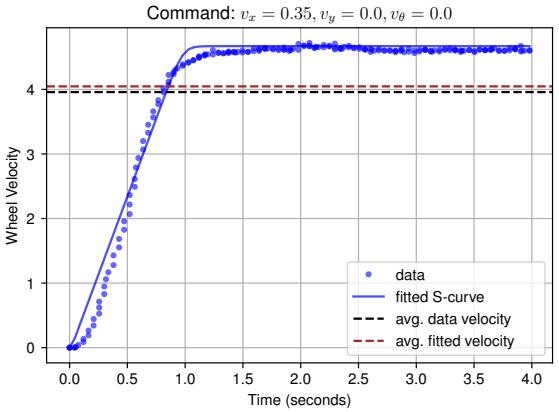


Fig. 4. Example of an S-shaped velocity curve fitted to wheel velocity data. The velocity command is to move in the x-direction at 0.35 m/s. Since all wheels behave identically in this scenario, only one wheel's velocity is shown.

A detailed explanation of the mathematical model that best matches the recorded data, identified through trial and error, is provided in Appendix A. Setting aside the intricacies, we predict the parameters Θ of an S-shaped function $S_\Theta : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ based on a given target wheel velocity ω. This function maps elapsed time to velocity according to the learned S-shaped velocity profile.

To achieve this, we train a small neural network with weights W to approximate the mapping $\Theta = f_W(\omega)$ using data from transitions starting at zero velocity. At runtime, we apply the model to arbitrary transitions by feeding in the wheel velocity difference $\Delta\omega$ between the current and target states. The network then outputs the corresponding S-curve parameters Θ, allowing us to smoothly interpolate between any two commands. Because different velocity changes result in distinct acceleration profiles, the goal is for the network to generalize from the training data and predict suitable S-curve parameters for any given target velocity ω, or more generally, any velocity change $\Delta\omega$.

The training data consists of recorded wheel velocity profiles from movements in the x-direction, y-direction, and rotational motion around the z-axis, ensuring coverage of the robot's basic degrees of freedom. Each command generates a unique velocity profile for the four wheels. For details on the recorded trajectories, see Section IV.

We observe that collecting individual wheel acceleration profiles for various target velocities results in a relatively consistent family of S-curves (Fig. 5(a)). Notably, even for significantly different commandssuch as x-direction movement versus rotational movementsome wheel velocity profiles align on up to two wheels due to the kinematics of mecanum-wheeled driving.

Ideally, on a mecanum-wheeled robot, the velocity profile for a given `Twist` command should ensure proportional acceleration across all wheels. That is, each wheel should follow an acceleration curve scaled to its final target velocity, preserving the correct velocity ratios throughout the transition. This ensures that the robot moves in the intended direction not only once all wheels reach their target speeds, but also during the acceleration phase itself. Deviations from proportional acceleration can result in unintended drift or curvature during motion. This observation suggests that the acceleration behavior of individual wheels should be effectively modeled based on their target velocity change, independent of the full `Twist` commanda principle we follow in our approach above.

However, in practice, we found that PAL Robotics, the manufacturer of the TIAGo robot, did not fully adhere to this principle, resulting in non-proportional velocity profiles for certain commands, such as movements combining x- and y-directional motion (Fig. 6). Consequently, our S-curve predictions for such movements exhibit deviations due to the suboptimal design of the original robot controller. Additionally, we observed that the real robot exhibits slight deviations from the intended direction during acceleration.

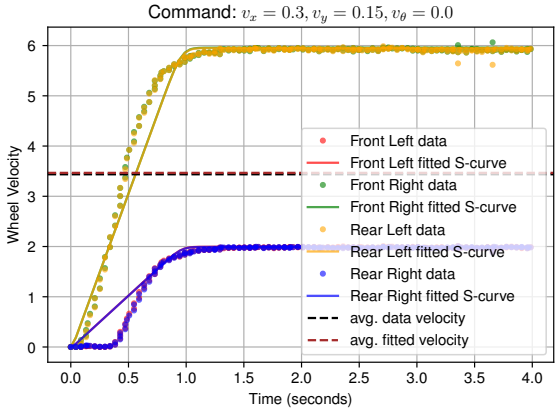


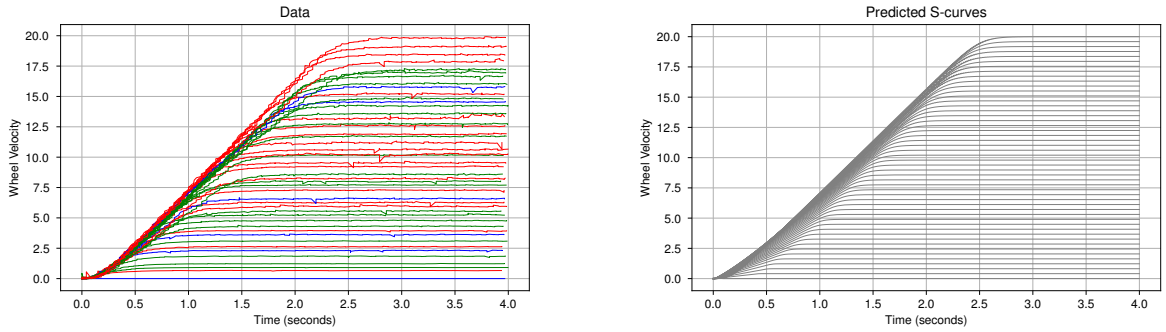
Fig. 6. For non-axis-aligned movements, such as a skewed trajectory in the direction $x = 2y$, the predicted acceleration behavior of the four wheels deviates from the real robot due to the original controller not enforcing proportional acceleration. However, the average velocity of our learned model remains close to the expected value.

The most effective solution would be to reimplement the original robot controller to enforce proportional acceleration. However, since we do not have access to this level of hardware control, this task would fall to PAL Robotics. More broadly, this highlights an important consideration when designing controllers for mecanum-wheeled robots.

Our strategy for simulating a movement command is as follows: Suppose the robot is currently at velocity $T = (v_x, v_y, v_\theta)$ and the desired velocity is $T' = (v'_x, v'_y, v'_\theta)$. To ensure a smooth transition, we interpolate the commands received by the simulated robot between T and T'.

To do this, first, we compute the total required velocity change $\Delta\omega$ for the wheels (see Appendix B for details) and use this as input to our model to obtain the velocity S-curve, parameterized by $\Theta = f(\Delta\omega)$. This curve is then used to update the target `Twist` command at each timestep:

$$T_t = T + \mathrm{p}_t \cdot (T' - T), \tag{2}$$



(a) Selection of recorded data

(b) Predicted S-curves for various target wheel velocities

Fig. 5. (a) Wheel velocity curves for different commands in the x-direction (red curves), y-direction (blue curves), and rotational motion (green curves). The S-curve shape exhibits some inconsistencies, likely due to noise from the real robot's PID controller. (b) Predicted S-curves from the fitted model.

where

$$\mathrm{p}_t = \frac{S_\Theta(t - t_0)}{\Delta\omega} \tag{3}$$

represents the fraction of the command executed at time t, with t_0 denoting the command start time.

The wheel controllers then receive target velocities derived from the updated `Twist` command T_t, which is converted using Equation 1.

Overall, this heuristic approach for interpolating between the current velocity T and the desired velocity T' ensures a smooth transition, replicating the S-shaped acceleration profiles observed in the real robot's basic movements. However, while our approach provides a realistically looking controller for simulation, the real robot does not exhibit identical transitional behavior and may even accelerate more rapidly. For example, the corners of a square trajectorywhere the robot undergoes a 90 change in movement directionare navigated much more smoothly in simulation than on the real robot (Fig. 7(a)). Consequently, the simulated robot is slightly harder to control than the real robot.

Nevertheless, we argue that this discrepancy is not a major concern for two reasons. First, making the simulation more challenging to control than the real robot introduces artificial noise, which can improve robustness. Second, since our controller operates in an open-loop fashion, a higher-level closed-loop trajectory controller should still be able to utilize our open-loop movement controller effectively.

We also hypothesize that accurately modeling the acceleration profile of an arbitrary black-box mecanum robot controller may be nearly impossible, or at least highly challenging, as it would require collecting extensive real-world data for transitions between all possible velocity states and fitting a model to predict the resulting acceleration behavior for each wheel. We leave this question open for future research but, as we argued, believe that our approach is already sufficient for most use cases. However, future work using our model would have to confirm whether this is true.

C. Integration of the Omnidirectional Drive by Directly Setting the Base Velocity

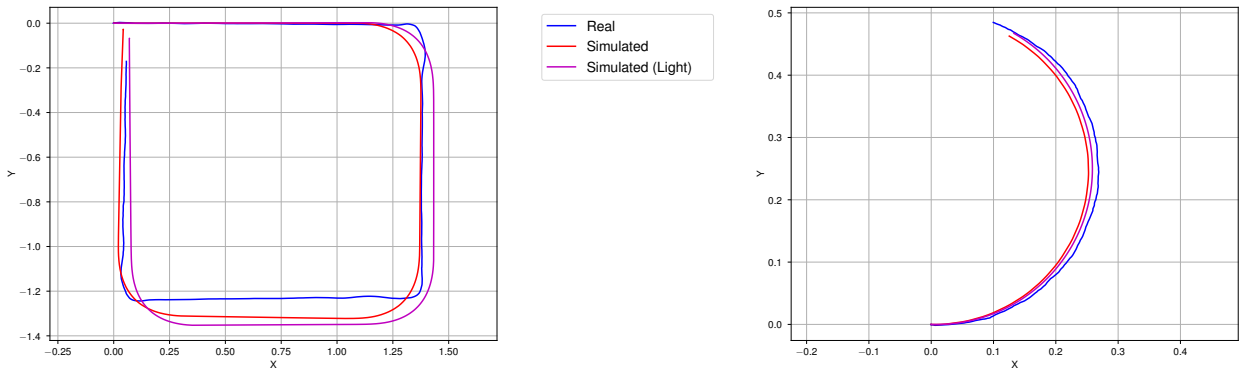
As an alternative to the physically accurate simulation of mecanum wheels, we provide a lightweight approach to simulating omnidirectional movement. Instead of modeling wheel-ground interactions, this method directly sets the robot's base velocity to the desired `Twist` command using the Isaac Sim API. While we still apply the S-shaped velocity profile for smooth acceleration, individual wheel velocities are no longer explicitly controlled.

This method is less physically accurate, as it neglects forces such as friction and does not even require knowledge of how mecanum wheels function. However, as demonstrated in the Experiments section, it remains sufficiently accurate while significantly improving simulation efficiency. By eliminating roller collision computations and treating the wheels as dummy components, we achieve a notable performance gain, e.g., reducing the required physics steps per second from 360 to 60. In practice, this leads to a clear reduction in computational cost, which is noticeably reflected in higher simulation frame rates (FPS).

A key advantage of this approach is its generalizability — it can be applied to any holonomic robot, regardless of the real PID controller's implementation. Moreover, it eliminates the need to tune wheel joint physics parameters in Isaac Sim. Additionally, if only the lightweight controller is used, the mecanum wheels do not need to be modeled at all.

IV. Experiments

In this section, we compare the behavior of the simulated robot with that of the real robot. To do so, we recorded multiple simple open-loop trajectories in both environments and analyzed their differences. Specifically, we compared the simulation's odometry data to motion tracking data from the real robot executing the same trajectories. Motion tracking was conducted using OpenVR with a VIVE tracker mounted at the robot's rotational center, while the robot's pose (position and orientation) was triangulated using two VIVE lighthouses.



(a) Square trajectory

(b) Circular trajectory

Fig. 7. Comparison of simulated and real robot trajectories: (a) Square trajectory consisting of 3-second movements in forward, rightward, backward, and leftward directions with a target velocity of 0.45 m/s, and (b) Circular trajectory with 0.19 m/s movement in the y-direction and 0.78 rad/s rotational movement.
Note that our experiments do not use closed-loop control, meaning the robot has no feedback on its actual position during execution.

To evaluate simulation accuracy, we recorded various simple trajectories for both simulated versions and the real robot.

Among our experiments, we tested 20 different target velocities ranging from 0.05 to 1.00 m/s in the x-, y-, and x-y- (diagonal) directions, as well as target rotational velocities around the z-axis from 0.05 to 1.5 rad/s. Each command lasted four seconds, during which we recorded the resulting wheel velocity profiles using the robot's onboard encoders. To ensure smooth and consistent data, each command was repeated three times, and the resulting profiles were averaged. The relative errors for each case are summarized in Table I. Overall, these simulated simple trajectories closely follow the intended paths of the real robot's controller.

TABLE I
MEAN RELATIVE ERROR (MRE) AND STANDARD DEVIATION (STDRE) FOR 4-SECOND VELOCITY COMMANDS

MRE $\pm$ STDRE	**% Value**	**% Value (Light)**
x-direction	8.24 ± 1.37	7.36 ± 3.71
y-direction	4.61 ± 5.54	3.89 ± 1.49
x-y-direction	5.68 ± 2.71	5.16 ± 3.07
Rotation	4.30 ± 1.62	2.97 ± 1.52

The table shows mean relative error (MRE) and standard deviation (STDRE) for 4-second velocity commands in the x-, y-, and x-y-directions, as well as for rotation. The relative error is defined as $\frac{|\Delta p_{\text{real}} - \Delta p_{\text{sim}}|}{\Delta p_{\text{real}}}$, where Δp_{real} and Δp_{sim} denote the total traveled distances for linear motion (or total rotation angles for rotational motion), measured on the real and simulated robots, respectively. The MRE and STDRE represent the mean and standard deviation of this error across all tested velocities.

Note that diagonal (x-y) commands were used for evaluation only and were not included in the training data, allowing us to assess generalization to combined motions.

Figure 7 presents two example trajectories. In the first, the robot moves forward, rightward, backward, and leftward, forming an approximate square. In the second, the robot moves forward while rotating leftward, tracing a circular path.

For longer and more complex trajectories, accumulated errors become more significant, particularly when certain acceleration patterns, such as simultaneous x-y directional movement, are not modeled as accurately (as we saw before in Fig. 6). However, despite these limitations, the S-curve approximation remains sufficiently accurate for pure movements, as indicated by the relatively low error for x-y-directional motion in Table I.

In practice, the slight error accumulation in the open-loop trajectories is unlikely to be a major issue, as real-time closed-loop trajectory controllers continuously correct errors and adjust movement commands to maintain the intended path.

V. CONCLUSION

In this work, we presented the integration of the TIAGo++ Omni robot into Isaac Sim, enabling its use in modern simulation environments. While the primary focus was the modeling of mecanum wheel dynamics, we also implemented a full robot simulation, including sensor and joint control integration with ROS 2, ensuring compatibility with existing frameworks.

For the omnidirectional drive, we introduced two simulation models: a physically accurate model that closely replicates real-world wheel dynamics and a lightweight velocity-based model that significantly improves simulation efficiency. The lightweight model is well-suited for high-throughput tasks such as reinforcement learning, while the physically accurate model can be used for fine-tuning and validation when precise behavior is required. Additionally, our learning-based calibration improves simulation accuracy by approximating the S-shaped velocity profile of the real robot using straightforward data recordings.

A key insight from our work is that ensuring proportional acceleration across all wheels is crucial for smooth

omnidirectional movement in mecanum-wheeled robots. Our findings suggest that non-proportional acceleration patterns can lead to inconsistencies and driving behavior that is difficult to model and replicate in simulation, highlighting an important design consideration for controllers of mecanum-wheeled bases.

Future work may focus on further validating the simulation framework for tasks such as reinforcement learning and trajectory tracking. Additionally, alternative approaches could be explored to improve the modeling of S-shaped velocity profiles for mecanum-wheeled robots.

ACKNOWLEDGMENT

This research has been partially funded by the Federal Ministry of Education and Research of Germany under grants no. 16ME0999 RIG and 16SV8683 RimA. The authors acknowledge the use of OpenAI's ChatGPT-4o for language refinement, including grammar, clarity, and conciseness improvements. All technical content, results, and interpretations remain the sole work of the authors.

REFERENCES

[1] J. Pages, L. Marchionni, and F. Ferro, "Tiago: The modular robot that adapts to different research needs," in *International Workshop on Robot Modularity, IROS*, vol. 290, 2016.

[2] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, Ieee, 2004, pp. 2149–2154.

[3] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A Review of Physics Simulators for Robotic Applications," *IEEE access : practical innovations, open solutions*, vol. 9, pp. 51 416–51 431, 2021, ISSN: 2169-3536.

[4] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," in *Conference on Robot Learning*, PMLR, 2018, pp. 270–282.

[5] M. Mittal, C. Yu, Q. Yu, J. Liu, N. Rudin, D. Hoeller, J. L. Yuan, R. Singh, Y. Guo, H. Mazhar, A. Mandlekar, B. Babich, G. State, M. Hutter, and A. Garg, "Orbit: A unified simulation framework for interactive robot learning environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3740–3747, 2023.

[6] G. Bayar and S. Ozturk, "Investigation of the effects of contact forces acting on rollers of a mecanum wheeled robot," *Mechatronics*, vol. 72, 2020-12-01, 2020, ISSN: 0957-4158.

[7] Y. Li, S. Dai, Y. Zheng, F. Tian, and X. Yan, "Modeling and Kinematics Simulation of a Mecanum Wheel Platform in RecurDyn," *Journal of Robotics*, vol. 2018, Jan. 2018, ISSN: 1687-9600.

[8] Y. Okada, K. Oguma, K. Gunji, Y. Yokota, H. Aryadi, S. Kojima, R. Bezerra, M. Konyo, K. Ohno, and S. Tadokoro, "Fast and accurate simulation of mecanum wheels with passive rollers emulated by fixed joints and anisotropic friction," in *2023 21st International Conference on Advanced Robotics (ICAR)*, 2023, pp. 592–598.

[9] A. Apurin, B. Abbyasov, A. Dobrokvashina, Y. Bai, M. Svinin, and E. Magid, "Omniwheel Chassis' Model and Plugin for Gazebo Simulator," *Proceedings of International Conference on Artificial Life and Robotics*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259886517 (visited on 10/14/2025).

[10] M. Wiedemann, O. Ahmed, A. Dieckhoefer, R. Gasoto, and S. Kerner, "Simulation Modeling of Highly Dynamic Omnidirectional Mobile Robots Based on Real-World Data," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, May 2024, pp. 16 923–16 929. [Online]. Available: https://ieeexplore.ieee.org/document/10611459 (visited on 08/24/2024).

[11] S. Hoefer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Karen Liu, J. Peters, S. Song, P. Welinder, and M. White, "Sim2Real in Robotics and Automation: Applications and Challenges," *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 2, pp. 398–400, Apr. 2021, ISSN: 1558-3783.

[12] N. Tlale and M. de Villiers, "Kinematics and dynamics modelling of a mecanum wheeled mobile platform," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, IEEE, 2008, pp. 657–662.

[13] Z. Hendzel and. Rykaa, "Modelling of Dynamics of a Wheeled Mobile Robot with Mecanum Wheels with the use of Lagrange Equations of the Second Kind," *International Journal of Applied Mechanics and Engineering*, vol. 22, no. 1, pp. 81–99, Feb. 1, 2017, ISSN: 1734-4492, 2353-9003.

[14] R. Williams, B. Carter, P. Gallina, and G. Rosati, "Dynamic model with slip for wheeled omnidirectional robots," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 3, pp. 285–293, Jun. 2002, ISSN: 2374-958X. [Online]. Available: https://ieeexplore.ieee.org/document/1019459 (visited on 07/10/2024).

[15] A. Kanwischer and O. Urbann, "A Machine Learning Approach to Minimization of the Sim-To-Real Gap via Precise Dynamics Modeling of a Fast Moving Robot," in *2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Dec. 2022, pp. 349–354. [Online]. Available: https://ieeexplore.ieee.org/document/10004376 (visited on 01/19/2024).

[16] J. Collins, R. Brown, J. Leitner, and D. Howard. "Traversing the Reality Gap via Simulator Tuning." arXiv: 2003.01369 [cs]. (Mar. 3, 2020), [Online]. Available: http://arxiv.org/abs/2003.01369 (visited on 07/10/2024), pre-published.

[17] M. Abdalnasser, M. Elsamanty, and A. Ibrahim, "Trajectory tracking of wheeled mobile robot through system identification and control using deep neural network," *Engineering Research Journal*, vol. 183, no. 3, pp. 1–17, 2024, ISSN: 1110-5615. eprint: https://erj.journals.ekb.eg/article_376858_cb06eacecbae554a94e2b824a1ba310e.pdf. [Online]. Available: https://erj.journals.ekb.eg/article_376858.html (visited on 05/01/2025).

[18] T. T. K. Ly, N. T. Thanh, H. Thien, and T. Nguyen, "A neural network controller design for the mecanum wheel mobile robot," *Engineering, Technology & Applied Science Research*, vol. 13, no. 2, pp. 10 541–10 547, 2023.

[19] PAL_Robotics, *Tiago_simulation*. [Online]. Available: https://github.com/pal-robotics/tiago_simulation (visited on 08/24/2024).

[20] DaiGuard, *Fuji mecanum wheels*. [Online]. Available: https://github.com/DaiGuard/fuji_mecanum (visited on 08/24/2024).

[21] S. Dickerson and B. Lapin, "Control of an omni-directional robotic vehicle with Mecanum wheels," in *NTC '91 - National Telesystems Conference Proceedings*, Mar. 1991, pp. 323–328.

[22] N. G. Hamid Taheri Bing Qiao, "Kinematic model of a four mecanum wheeled mobile robot," *International Journal of Computer Applications*, vol. 113, no. 3, pp. 6–9, Mar. 2015, ISSN: 0975-8887. [Online]. Available: https://ijcaonline.org/archives/volume113/number3/19804-1586/ (visited on 05/03/2025).

rger elements draw more attention than smaller

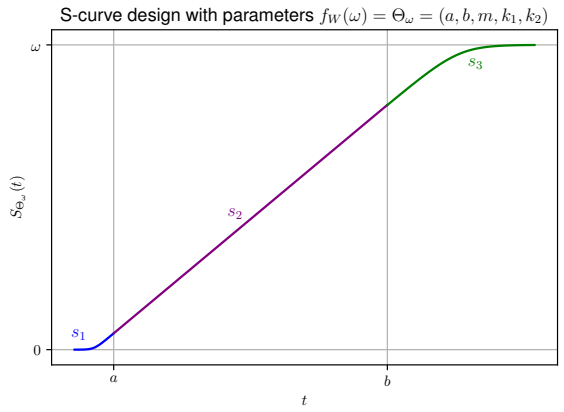


Fig. 8. Structure of the learned S-curve model S_{Θ_ω} for an example target velocity $\omega = 11.48$, using the actual learned weights W.

APPENDIX

A. Model of the Learned S-Curves

Here, we describe the selection of our S-curve model S_Θ, used in Section III-B. Our goal was to choose a function that best aligns with the inductive bias observed in the data, i.e., an S-curve that closely matches the velocity profiles of the robot.

In general, an S-curve can be described by a set of parameters $\Theta \in \mathbb{R}^d$. A common example is the logistic function, parameterized by its growth rate and midpoint. A small neural network, e.g., a multi-layer perceptron (MLP), $f : \mathbb{R} \to \mathbb{R}^d$, learns to predict the parameters Θ_ω for each target wheel velocity ω. As long as the S-curve is differentiable, we can backpropagate through it, effectively treating it as the final layer of the network.

However, after experimenting with various known S-curves, such as the logistic function and others, we found that none captured the observed behavior as effectively as the custom-assembled S-curve we describe next. The reason is that the velocity profiles exhibit asymmetrical sharpness, with a steeper initial rise, a smoother asymptotic approach to the final velocity, and a linear transition in between.

Our S-curve model S_{Θ_ω} (Figure 8) is parameterized by five variables, $\Theta_\omega = (a, b, m, k_1, k_2)$, and consists of three segments:

1) Initial phase: A smooth-ramp-like function s_{1a,k_1} in the interval $[0, a]$.
2) Linear transition: A linear function $s_{2a,b,m}$ in the interval $[a, b]$.
3) Final phase: A flipped smooth-ramp-like function s_{3b,k_2} in the interval $[b, \infty)$.

The parameters a and b determine the transition points between these segments, k_1 and k_2 control the sharpness of the functions s_1 and s_3, and m defines the slope of the linear segment s_2.

The ramp-like functions s_1 and s_3 are implemented as softplus functions of the form:

$$s(x) = \frac{\log(1 + \exp(x \cdot \text{sharpness}))}{\text{sharpness}} \tag{4}$$

However, any other ramp-like functions could also have been used.

During training, we enforce continuity by ensuring that:

1) s_1 and s_2 meet at a.
2) s_2 and s_3 meet at b.
3) s_1 and s_3 asymptotically approach the same slope m as s_2.
4) $s_1(0) = 0$ and $s_3(\infty) = \omega$.

The implementation details of enforcing continuity constraints are available through our open-source code, though alternative implementations could achieve similar results.

We designed our neural network architecture as a small MLP with layers sized $[1, 35, 15, 5]$, employing softplus activation functions between layers. The network predicts the five parameters defining the S-curve for a given target wheel velocity. This compact network structure was chosen because it provides a good balance between fitting accuracy and inference efficiency.

We chose the Adam optimizer arbitrarily for training. To maximize performance, we trained 100 network instances with different random initializations and selected the best-performing model based on the lowest validation error. We do not claim that our model architecture or training procedure is optimal; rather, we just want to highlight that our chosen approach achieves good performance.

B. Computation of Total Velocity Change Bound

The quantity Δv_{bound} represents a conservative upper bound on the velocity change at any of the wheel contact points due to both translational and rotational motion. Since the mecanum wheels are positioned at fixed offsets L_x and L_y from the robots center, any change in the chassis velocities affects the wheel speeds differently depending on their location. By the triangle inequality, each signed component of the true perwheel velocity change is bounded in magnitude by the sum of the magnitudes of its translational and rotational contributions.

Let v_x, v_y, v_θ be the current chassis velocities in the x, y, and rotational directions, respectively, and let v'_x, v'_y, v'_θ be the commanded velocities. We define

$$\Delta v_{x,\text{bound}} = \left| v'_x - v_x \right| + \left| v'_\theta - v_\theta \right| L_y, \tag{5}$$

$$\Delta v_{y,\text{bound}} = \left| v'_y - v_y \right| + \left| v'_\theta - v_\theta \right| L_x. \tag{6}$$

Then the combined chassiswide bound is

$$\Delta v_{\text{bound}} = \sqrt{\Delta v^2_{x,\text{bound}} + \Delta v^2_{y,\text{bound}}} \tag{7}$$

and normalizing by the wheel radius r gives the angularspeed bound

$$\Delta\omega := \frac{\Delta v_{\text{bound}}}{r}. \tag{8}$$

In our approach, we then use $\Delta\omega$ as the single feedforward input to our Scurve model.