Jornadas de Automática

ADAMSim: PyBullet-Based Simulation Environment for Research on Domestic Mobile Manipulator Robots

Prados, Adrian [1,*], Espinoza, Gonzalo [1], Mendez, Alberto, Mora, Alicia, Garrido, Santiago, Barber, Ramon

[a] RoboticsLab, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, Av. Universidad, nº30, 28911, Leganés, España.

To cite this article: Prados, Adrian, Espinoza, Gonzalo, Mendez, Alberto, Mora, Alicia, Garrido, Santiago, Barber, Ramon, 2025. ADAMSim: PyBullet-Based Simulation Environment for Research on Domestic Mobile Manipulator Robots. Jornadas de Automática, 46. https://doi.org/10.17979/ja-cea.2025.46.12217

Resumen

Este artículo presenta ADAMSim, un entorno de simulación basado en PyBullet diseñado específicamente para el Ambidextrous Domestic Autonomous Manipulator (ADAM), desarrollado para apoyar la investigación en navegación, manipulación y aprendizaje en robótica doméstica. El simulador replica con precisión la estructura y el comportamiento del robot físico, lo que permite una transferencia robusta de algoritmos entre simulación y mundo real. ADAMSim sigue un diseño modular, que incluye navegación, cinemática de brazos y manos, percepción y comunicación mediante ROS. Esta arquitectura permite la operación sincronizada entre el robot real y su gemelo digital. Se desarrollaron diversos ejemplos, que abarcan desde tareas de visión y agarre, hasta navegación y teleoperación, incluyendo experimentos ejecutados simultáneamente en el robot simulado y el real. Su diseño de código abierto y flexible convierte a ADAMSim en una herramienta poderosa para el desarrollo seguro y reproducible de algoritmos y experimentos en robótica doméstica. La plataforma también está pensada para apoyar investigaciones en mapeo de interiores, aprendizaje de manipulación avanzada y proyectos educativos, como banco de pruebas.

Palabras clave: Simulador de Robots, Gemelo Digital, PyBullet, Real-to-Sim, Sim-to-Real, Manipulador Móvil.

Abstract

This paper introduces ADAMSim, a PyBullet-based simulation environment tailored for Ambidextrous Domestic Autonomous Manipulator (ADAM), developed to support research in navigation, manipulation, and learning for domestic robotics. The simulator accurately replicates the structure and behavior of the physical robot, enabling robust sim-to-real and real-to-sim algorithm transfer. ADAMSim follows a modular design, including navigation, arm and hand kinematics, perception, and ROS communication. This architecture allows synchronized operation between the real robot and its digital twin. Several example applications were developed, ranging from vision and grasping tasks to navigation and teleoperation, including experiments running both simulated and real robots simultaneously. Its open-source and flexible design makes ADAMSim a powerful tool for safe and reproducible algorithm development and experimentation in household robotics. The platform is also intended to support future research in indoor mapping, advanced manipulation learning, and educational projects, serving as a test bed.

Keywords: Robot Simulation, Digital Twin, PyBullet, Real-to-Sim, Sim-to-Real, Mobile Manipulator.

1. Introduction

In recent years, the field of robotics has witnessed a significant surge in the development of mobile manipulators, especially for domestic assistance applications. These are typically mobile humanoid robots that perform household tasks in diverse human-centred indoor environments that they must adapt to and understand (Fernandez et al., 2025). Unlike industrial environments, which are typically structured and organized, domestic environments are usually unstructured, dy-

[*]Correspondence author: aprados@ing.uc3m.es

[1]Both authors contributed equally

namic, and primarily designed for human use, involving cluttered objects in unpredictable locations and constantly being rearranged, and impose spatial limitations for the robot. Hence, domestic assistant robots must be versatile and dexterous when navigating through the environment (Mora et al., 2022) and interacting with the objects in it (Mendez et al., 2024; Prados et al., 2024b). In this context, research on domestic mobile manipulators relies heavily on simulators, where algorithms can be tested safely, repeatedly, and within controlled virtual environments. This allows real-world processes to be evaluated in advance, ensuring greater reliability and avoiding the limitation of being tested solely in controlled laboratory settings. Instead, a variety of indoor environment scenarios, many of which may not always be physically accessible, can be modelled. For this purpose, a range of simulation tools are available for modelling and simulating robotic platforms, as well as for configuring custom environments with objects and obstacles. Gazebo, PyBullet (Coumans and Bai, 2021), MuJoCo (Todorov et al., 2012), and Isaac Lab (Makoviychuk et al., 2021) are among the most popular simulation tools and have been extensively compared in various studies (Collins et al., 2021).

In this paper, we introduce *ADAMSim*, a complete simulation platform based on PyBullet for the *Ambidextrous Domestic Autonomous Manipulator* robot (ADAM) (Mora et al., 2024; Barber et al., 2022), a mobile manipulator robot designed to assist the elderly at home. This PyBullet-based platform integrates ADAM in simulation and contains different modules for navigation, manipulation, grasping, and vision. This setup allows us to design and test our algorithms on a virtual model of the robot and to send and retrieve information from the real robot via *Robot Operating System* (ROS), allowing real-time sim-to-real and real-to-sim communications, effectively functioning as a digital twin.

2. Related Work

Robotic simulators have become indispensable tools in modern robotics. By providing a virtual environment that accurately models physical interactions, these platforms enable rapid prototyping, algorithm testing, and risk-free experimentation (Erez et al., 2015). Moreover, they improve safety during early-stage trials, and facilitate reproducibility across research groups. According to the needs of each application, this means preference for high visual fidelity, advanced physics or scalability. Thus, simulators are widely used in various aspects of mobile manipulators, including manipulation, navigation, and perception. They enable testing of software components, robot behaviours, and control algorithms in diverse environments. As digital twins, they maintain a synchronized representation of the physical robot and its surroundings, enabling real-time diagnostics and safe human-robot collaboration (Choi et al., 2022). Currently, the use of simulators has increased within the field of learning. Simulators are widely used for "sim-to-real" transfer, where control policies are trained under various virtual conditions before being deployed on real hardware, allowing consideration of a wide range of parameters and environmental conditions (Dan et al., 2025). Within learning, notable uses include Reinforcement Learning (Wang et al., 2020), where different conditions are added so robots can learn in safe environments; Imitation Learning (Prados et al., 2024a, 2023), where human data are used to carry out the learning process; and even the direct use for data collection (Lopez et al., 2023) or the generation of synthetic datasets (Fernandez et al., 2024), where simulators are used to improve environment knowledge.

Self-developed simulators play a crucial role in ensuring accurate behaviour replication and fostering reproducibility between research groups. HomeRobot (Yenamandra et al., 2023) integrates multiple modules designed to maintain consistency with real-world robotic behaviour while allowing external researchers to experiment under identical conditions. By its API, HomeRobot facilitates research on complex tasks, ensuring that algorithms are evaluated fairly and reproducibly. Another example is the iCub platform (Tikhanoff et al., 2008), whose developers opted to build a custom simulation environment rather than rely on existing ones. As an open-source, academic robot with a compliant musculoskeletal structure, iCub demands highly accurate modelling of its joints and actuation mechanisms. Additionally, since the robot operates on the YARP middleware, a dedicated simulator ensures full compatibility with its control architecture. These customized digital twins not only capture the robots intricate dynamics but also promotes reproducibility by offering a unified, self-contained software stack. However, achieving reproducibility in robotics does not necessarily require the development of custom simulators. An example of this is the BestMan framework (Yang et al., 2024), which builds on the widely used PyBullet to address the challenges of bridging simulation and real-world deployment. It uses 10 modular components including perception, task planning, navigation and manipulation, each designed for flexible integration and reuse. By exposing a unified simulation-to-hardware API, BestMan allows algorithms trained or tested in simulation to be transferred directly to physical robots with minimal adaptation.

In conclusion, simulators enable the development and testing of algorithms in varying environments in a simple and safe manner, allowing the use of different platforms and settings. For this reason, this work aims to develop a PyBullet-based simulation platform that allows us to obtain these advantages with our ADAM robot, following:

- **Modular Design**: Each generated module has standardized interfaces and logic, which allows for easy development of control algorithms, ensuring modularity as well as the ability to expand or extend these modules to enhance the robot's capabilities.
- **Multilevel Skill Coordination Model**: Our simulation environment allows connection with multiple integrated layers. The simulator supports tasks focused on navigation, perception, manipulation, and grasping, as well as the interconnection of each module with the others reducing the complexity of component integration and improving the efficiency of algorithm development.
- **ROS–Simulation Interoperability**: A bidirectional communication layer that synchronizes real-world sensor data and control commands with the simulator, enabling continuous real-time feedback and gradual sim-to-real transfer.

- **Teleoperation and Robust Learning via Digital Twin**: A high-fidelity visualization and control interface that enables remote operation of ADAM in simulation, facilitating data collection and user-in-the-loop testing without risking the robot. This is integrated into a Real-Sim-Real learning pipeline that augments real-world datasets with simulated data, leveraging domain randomization and synthetic annotations to train models that reliably generalize to the physical robot.

3. PyBullet-based simulator for ADAM

We present ADAMSim, our PyBullet based simulator for the ADAM robot. It is structured into a main class *ADAM*, which is divided into several modules for each of the robot's main functionalities (navigation, manipulation, perception, and ROS communication) which inherit the main robot's properties and configurations (Fig. 1).

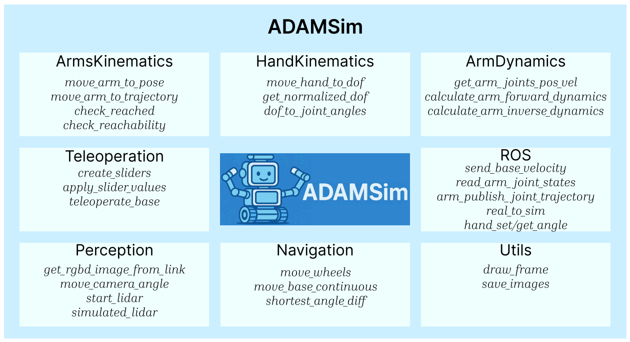

Figure 1: General integration of the modules developed for ADAMSim.

3.1. PyBullet framework

In the field of physical simulators for robotics applications, there is no single platform that is superior in all aspects. Studies focused on comparing simulators (Collins et al., 2021) conclude that it is up to the engineer to choose the implementation that best meets their specifications when developing the required tasks. In our case, we have chosen to use the PyBullet simulator. This simulator is a physics and robotics engine based on the Bullet Physics SDK, which allows integration with a wide variety of sensors, robotic platforms, terrains, and objects. It also supports loading models using different formats, such as URDF or XML definitions.

Several criteria were followed for the selection of this simulator. First, we sought a simulator that was flexible for modifications, accessible for use across the various areas in which the ADAM robot is involved, offered good performance, and compatible with learning tools. PyBullet meets all these requirements, as it features an intuitive and functional Python-based API, making it compatible with various environments and widely used algorithms. This flexibility contrasts with other simulators like Gazebo or MuJoCo, where accessing and modifying the underlying code is more complex. PyBullet was designed from the beginning as a free and open-source tool, which makes it widely adopted in academic and research settings. In comparison, simulators like MuJoCo were initially proprietary, and Isaac Lab, developed by NVIDIA, often requires specific hardware to achieve optimal performance. Additionally, PyBullet supports custom robots (such as ADAM), which has been built using an URDF model that allows for easy hardware improvements (e.g., adding the robot's camera, hands ...). Finally, one of the most valuable advantages in research is that the PyBullet community is large and well-established, with extensive documentation and numerous implementation examples available. This broad usage also means that many of PyBullet's limitations have gradually been addressed through ongoing research and development using the simulator. A more detailed comparison is presented in Table 1, which outlines the strengths and weaknesses of each simulator according to standardized criteria derived from various simulator review studies[2].

3.2. Navigation module

ADAM is a bimanual mobile robot, and one of its essential components is the navigation module responsible for moving ADAM through the simulated environment. The robot's base is an RB-1 platform developed by the company Robotnik, designed for indoor environments. It has a diameter of 50 cm and five wheels, two of which are motorized (highlighted in red in Fig. 2a), with a radius of 7.62 cm. This base operates similarly to many other robotic platforms on the market, following the typical structure of a differential drive robot. To simulate the base, we developed a custom navigation module capable of estimating each wheel's contribution to the robot's overall velocity.

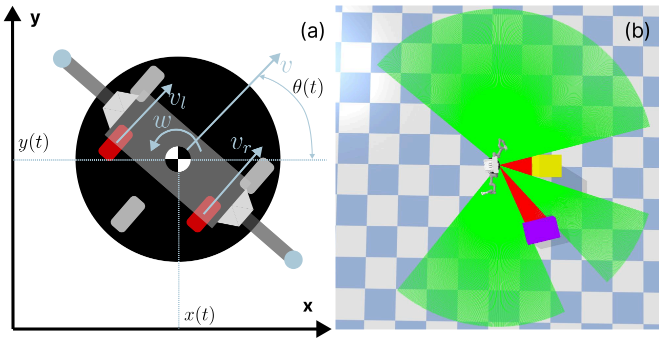

Figure 2: (a) ADAM's motion is defined by the linear velocity v, angular velocity ω, and wheel velocities v_r and v_l. θ determines the robot's orientation, (b) Example of use of the Lidar information for navigation. Red colour represent obstacles (purple and yellow boxes) in the ray casting.

The navigation control is based on converting a desired linear velocity v and angular velocity ω of the robot's base into the individual angular velocities of the left and right wheels, denoted as v_l and v_r respectively. Given the wheel radius r and the distance between the wheels d (red wheels from Fig. 2a and whit a value of $d = 43.6$ cm) expressed as:

$$v_l = \frac{2v - \omega \cdot d}{2r}, \quad v_r = \frac{2v + \omega \cdot d}{2r} \tag{1}$$

[2]Features with * in the table indicate that there are implementations addressing these shortcomings, but they are not integrated by default in the simulator

Table 1: Feature comparison between popular robotic simulator used for indoor Mobile Manipulator robots

Simulator	**Wheels**	**Legs**	**Height Map**	**Path Planning**	**ROS**	**RGBD**	**Lidar**	**Realistic Rendering**	**Force Sensor**	**Deformable Objects**	**Inverse Kinematics**	**Inverse Dynamics**
MuJoCo	✓	✓	✓	×	×	✓	✓	×	✓	✓	×	✓
Gazebo	✓	✓	✓	✓	✓	✓	✓	×	✓	×	✓	✓
IsaacLab	✓	✓	×	✓	×*	✓	✓	✓	✓	✓	✓	×
CoppeliaSim	✓	✓	✓	✓	✓	✓	✓	×	✓	×*	✓	×
PyBullet	✓	✓	✓	✓	×*	✓	✓	×*	✓	✓	✓	✓

where the term $2v$ accounts for the combined linear velocity contribution of both wheels, and $\omega \cdot d$ represents the differential component needed to achieve the desired angular velocity. Dividing by $2r$ converts the linear velocities at the wheels' circumference into angular velocities. The navigation module includes a continuous motion control function, `move_base_continuous`, which guides the robot base toward a goal pose (x_g, y_g, θ_g), maintaining specified tolerances for both position and orientation. At each control step, the system first retrieves the robot's current position (x, y) and orientation θ. Based on this state, it computes the positional error vector as $dx = x_g - x$, $dy = y_g - y$, and calculates the Euclidean distance to the target as $dist = \sqrt{dx^2 + dy^2}$. It also determines the angle to the target via $path\_angle = \arctan 2(dy, dx)$, and then derives both the heading error $\alpha = \text{shortest\_angle\_diff}(\theta, path\_angle)$, and the final orientation error $\Delta\theta_f = \text{shortest\_angle\_diff}(\theta, \theta_g)$. With these errors, the control law computes the linear and angular velocities needed to correct the robot's motion. These velocities are then translated into individual wheel velocities v_l and v_r using the kinematic equations of a differential drive platform (see Eq. 1). Finally, these velocities are applied as commands to the robot's motors, and the simulation advances one step. This loop continues until the robot is within the specified tolerances, that is, when $dist < pos\_tolerance$ and $|\Delta\theta_f| < angle\_tolerance$, ensuring accurate and smooth convergence to the desired pose.

In navigation, it's important to be aware of the obstacles. This is why the navigation module typically uses information from the laser scanner integrated into the robot's base (see Section 3.5). In Fig. 2b, an example is shown where the field of view (FOV) and range of the laser are illustrated using the UST-10LX model integrated into ADAM. This sensor allows for obstacle detection during navigation, enabling obstacle avoidance to be performed while moving.

3.3. Arms module

Once it is capable of navigating through the environment, the simulated robot requries a series of tools that leverage the capabilities of the upper limbs to interact with the objects in it. Particularly, it is important to define forward kinematics (FK) and inverse kinematics (IK) computations, as well as more complex functionalities that help to ensure correct arm trajectory planning and execution, and to capture information from the arm joint states during simulation that can be later use in machine learning algorithms.

Firstly, the arms module defined in the *ArmKinematics* class focuses on the FK and IK computations using PyBullet's default solver (Fig. 3). For example, the function *calculate_arm_inverse_kinematics* receives a target pose, which can be referenced with respect to the arm's wrist, the hand's base or a predefined tool-centre-point (TCP), and returns the arm joints that better achieves the pose on the referenced link. However, the built-in PyBullet functions have certain limitations when iteratively computing the FK, as they estimate this value only after the robot has moved to the desired position. Therefore, the arm needs to be moved in order to calculate the FK value — it cannot be done analytically. To complement these functions, we integrate PyKDL, a wrapper for the Kinematics and Dynamics Library, which parses the robot's URDF files to generate a kinematic chain. Thus, we develop a *calculate_forward_kinemtics* function which receives a joint configuration $q_{R/L} \in \mathbb{R}^6$ and returns a pose $p \in SE(3)$. PyKDL is also used to compute pre-grasp or approach poses given a grasp pose and an offset vector which considers the morphology of the RH56DFX hands. Lastly, two functions are included to ensure the correct execution of an arm pose, which is essential to ensure that an IK solution is valid. The first function *check_reached* takes a specific arm (R/L), and end link (end effector/hand base/TCP) and checks if the error between the link's pose and the target pose is smaller than certain position and angle thresholds. In a similar way, *check_reachability* checks if a target pose is reachable by a certain link, considering only the FK of the robot. These are based on a *get_pose_error* function that considers the error in position and orientation between two poses.

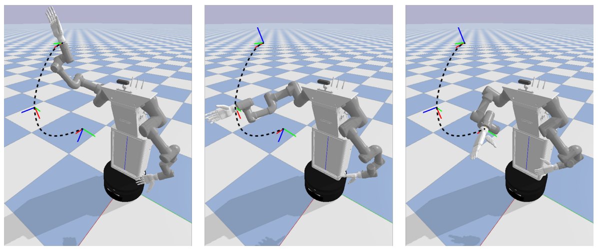

Figure 3: Sequence of movements of the right arm passing through different waypoints with specific position and orientation.

3.4. Hands module

Similarly to the arms module, the kinematic chain configuration of the hand models determines the range of grasp types that they can perform. To ensure effective grasp, it is essential that the simulated hands replicate the behaviour of the physical devices. This alignment allows grasp poses synthesised in simulation to be reliably transferred to the real robot.

Hands functionalities encapsulated within the *HandsKinematics* class, which adapts the virtual instances of the RH56DFX right and left hand models, developed by Inspire Robots, to reflect the behaviour of the real devices assembled on the robot. Each hand is actuated by 6 motors that control the flexion movement of the individual fingers and the abduction movement of the thumb. These actuators are driven by normalized input values ranging from 0 to 1000, corresponding to the minimum and maximum angles of each individual joint as specified in the URDF. To translate the normalized actuator commands to their corresponding joint angles in the simulated hand models, we define a linear mapping function $f : \mathbb{R}^6 \rightarrow \mathbb{R}^{12}$. The function *move_hand_to_joints* uses this mapping to convert normalized input values into corresponding joint angles, thereby controlling the flexion and abduction movements of the simulated hands for them to achieve desired grasp poses and configurations. The hand models are also considered in the collision detection, which is vital for obstacle detection algorithms involving object manipulation.

3.5. Perception module

An important aspect of mobile manipulator robots is the ability to detect changes in their environment. To capture such information, we must integrate sensors that provide visual and depth data from the robot's surroundings.

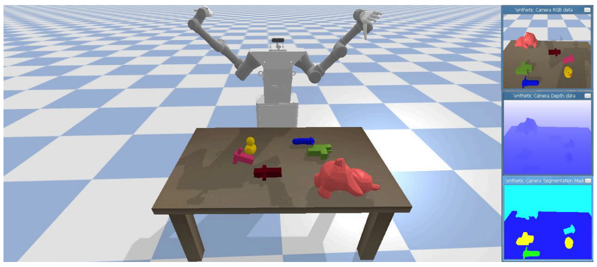

Figure 4: Example of vision with the RGB-D sensor. ADAMSim provides RGB information, depth, and segmented masks in a synthetic manner.

In our *Perception* module, two vision sensors are integrated. First, an external RGB-D sensor mounted on the robot's head (Fig. 4). In simulation, the RGB-D camera is assembled onto the robot through a revolute joint, which can be controlled to orient the camera within an angle of ±45°relative to the horizontal. This camera is configured to match the FOV and resolution parameters of the RealSense D435i mounted on the real robot. From the simulator, we obtain the RGB and depth information of the image, as well as pre-segmented information of the elements in the simulated environment. This is highly useful for manipulation tasks with the robot, as well as for generating synthetic datasets. Secondly, the 2D LiDAR located at the base of the robot has been implemented. This laser sensor is a UST-10LX, which features a 270°FOV and a 10 m range with a resolution of 0.25°. The sensor emits simulated laser beams that provide information about whether the space is free (green beams) or if there are obstacles in between (red beams), as is represented in Fig. 2b. This information is extensively used for navigation tasks, allowing the robot to detect objects along its path. It can also be used for map generation or simultaneous localization and mapping (SLAM) processes.

3.6. ROS module

To take advantage of PyBullet's capabilities in our simulator, we decided to command and receive information from the real robot directly within the simulation. Since PyBullet lacks a built-in implementation for connecting with real robots, we implemented a bridge between the two using ROS (see Fig. 5). For this, we created various connections for each of the robot's modules, aligning them with the corresponding modules in the simulator. The simulator allows for the ROS connection to be enabled or disabled, making it unnecessary to install ROS if the real robot is not available, and avoiding direct ROS dependencies. This ensures independence between the simulator and the communication layer. This module primarily focuses on reading and commanding the robot's arms, base and hands. For the arms, we have state reading functions such as *arm_real_to_sim*, which retrieves the joint positions of the robotic arms in real time. To perform the opposite operation, the function *arm_publish_joint_trajectory* is used, which sends the trajectory information generated in simulation to the real arms. This process is delicate since it requires converting the data to the message format understood by the arms. This is handled by the *convert_joints_to_msg* function, which reorders and structures the information according to the arms' specific message format. This process works similarly for the mobile base, where the function *send_velocity* can be used to command the robot's velocity. This allows, for example, teleoperation of the real base directly from the simulator, moving both the simulated model and the real robot. Additionally, we implemented direct communication with the robot's hands. Unlike the base and the arms, the hands are independent components, so they have their own ROS communication service. This enables simultaneous control of the hands in both the real robot and the simulator. To work with all components simultaneously, it is necessary to coordinate the different sampling times. For this purpose, a *wait* function has been created, which waits to receive data from the various ROS modules, thereby avoiding delays between the different elements. Our ROS module also enables connection with components that are not part of the robot itself, such as the camera or additional sensors like the Lidars we have available.

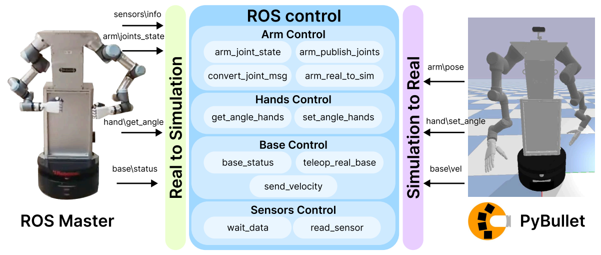

Figure 5: Diagram of the connections between the real model and the simulator through the created ROS bridge.

4. Examples of simulator use cases

To verify the functionality of the simulator, several examples have been prepared, all of which are available for use in the project's GitHub repository (`https://github.com/Mobile-Robots-Group-UC3M/AdamSim`). In this case, experiments have been carried out both using only the simulator and also by connecting it to the real robot. The tests conducted with the simulator are organized into small demos that showcase the modular functionality and how to apply the functions created for the robot. For this purpose, different examples have been developed. One focuses on vision, where real-time information captured by the simulated camera can be viewed. Another example demonstrates how to perform arm movement following a series of points, reaching the required position and orientation without error. To test the use of the robotic hands, an example was created showing how the algorithm is capable of grasping a simulated object and lifting it in place. Finally, two teleoperation tests were performed: one using sliders to control the robotic arms, and another teleoperating the robot's base while simulating a 2D LiDAR to detect obstacles.

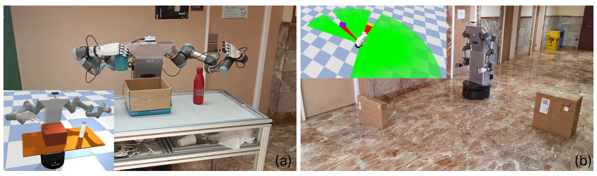

Figure 6: (a) Learning a manipulation task from demonstrations using a real-sim-real pipeline, (b) Simultaneous real and simulated navigation

For the experiments with both the real and simulated robot (Fig. 6), we focused on navigation and manipulation, executing both the real and simulated robots simultaneously. For navigation, we placed obstacles in the environment, which were detected by the simulator, and a set of waypoints was generated for the robot to follow while avoiding collisions with the boxes. For the manipulation task, we collected data and generated solutions using our own LfD algorithm (Prados et al., 2024b,c). Using two newly parameterized points for the task, we successfully placed two bottles into a box. All these examples can be seen in detail on the project's website (`https://mobile-robots-group-uc3m.github.io/AdamSim/`).

5. Conclusions and Future Implementations

In this work, we presented ADAMSim, a simulation framework for the bimanual mobile robot ADAM based on PyBullet. The framework includes various modules that allow for full simulation of the real robot. A modular block structure was developed to facilitate fast coding of different simulator capabilities. Additionally, a bridge was created between the simulated and real robot using ROS, enabling the use of simulation functions on the real robot and thus creating a digital twin that supports Real-to-Sim and Sim-to-Real transmission. The simulator is open-source, and various usage examples have been provided, all of which are available in the code associated with this paper. Future work includes applying the simulator for the development of indoor mapping algorithms, environment understanding, navigation, manipulation via Learning from Demonstration, optimal grasp generation, and as a development platform for student projects such as Bachelor's theses related to our research lines, or for use in simulation courses at the undergraduate or master's level.

Acknowledgments

Work supported by Advanced Mobile dual-arm manipulator for Elderly People Attendance (AMME) (PID2022-139227OB-I00) by Ministerio de Ciencia e Innovacion and by iRoboCity2030-CM (TEC-2024/TEC-62) by Programas de Actividades I+D en tecnologías de la Comunidad de Madrid.

References

Barber, R., Ortiz, F. J., Garrido, S., Calatrava-Nicolás, F. M., Mora, A., Prados, A., Vera-Repullo, J. A., Roca-González, J., Méndez, I., Mozos, Ó. M., 2022. A multirobot system in an assisted home environment to support the elderly in their daily lives. Sensors 22 (20), 7983.

Choi, S. H., Park, K.-B., Roh, D. H., Lee, J. Y., Mohammed, M., Ghasemi, Y., Jeong, H., 2022. An integrated mixed reality system for safety-aware human-robot collaboration using deep learning and digital twin generation. Robotics and Computer-Integrated Manufacturing 73, 102258.

Collins, J., Chand, S., Vanderkop, A., Howard, D., 2021. A review of physics simulators for robotic applications. IEEE Access 9, 51416–51431.

Coumans, E., Bai, Y., 2021. Pybullet quickstart guide. ed: PyBullet Quickstart Guide. https://docs. google. com/document/u/1/d.

Dan, P., Kedia, K., Chao, A., Duan, E. W., Pace, M. A., Ma, W.-C., Choudhury, S., 2025. X-sim: Cross-embodiment learning via real-to-sim-to-real. arXiv preprint arXiv:2505.07096.

Erez, T., Tassa, Y., Todorov, E., 2015. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx. In: International conference on robotics and automation. IEEE, pp. 4397–4404.

Fernandez, N., Espinoza, G., Mendez, A., Prados, A., Mora, A., Barber, R., 2024. Data generation in simulated domestic environments for assistive robots. In: 7th Iberian Robotics Conference (ROBOT). IEEE, pp. 1–6.

Fernandez, N., Espinoza, G., Mendez, A., Prados, A., Mora, A., Barber, R., 2025. Perspective chapter: Advanced environment modelling techniques for mobile manipulators. IntechOpen.

Lopez, B., Prados, A., Moreno, L., Barber, R., 2023. Taichi algorithm: Human-like arm data generation applied on non-anthropomorphic robotic manipulators for demonstration. In: 2023 European Conference on Mobile Robots (ECMR). IEEE, pp. 1–7.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al., 2021. Isaac gym: High performance gpu-based physics simulation for robot learning. arXiv preprint arXiv:2108.10470.

Mendez, A., Prados, A., Menendez, E., Barber, R., 2024. Everyday objects rearrangement in a human-like manner via robotic imagination and learning from demonstration. IEEE Access.

Mora, A., Prados, A., Mendez, A., Barber, R., Garrido, S., 2022. Sensor fusion for social navigation on a mobile robot based on fast marching square and gaussian mixture model. Sensors 22 (22), 8728.

Mora, A., Prados, A., Mendez, A., Espinoza, G., Gonzalez, P., Lopez, B., Muñoz, V., Moreno, L., Garrido, S., Barber, R., 2024. Adam: a robotic companion for enhanced quality of life in aging populations. Frontiers in Neurorobotics 18, 1337608.

Prados, A., Espinoza, G., Fernandez, N., Barber, R., 2024a. Imitation learning for low-cost dexterous hand using rgbd camera. In: 2024 7th Iberian Robotics Conference (ROBOT). IEEE, pp. 1–6.

Prados, A., Garrido, S., Barber, R., 2024b. Learning and generalization of task-parameterized skills through few human demonstrations. Engineering Applications of Artificial Intelligence 133, 108310.

Prados, A., Mendez, A., Espinoza, G., Fernandez, N., Barber, R., 2024c. f-divergence optimization for task-parameterized learning from demonstrations algorithm. In: 2024 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC). IEEE, pp. 9–14.

Prados, A., Mora, A., López, B., Muñoz, J., Garrido, S., Barber, R., 2023. Kinesthetic learning based on fast marching square method for manipulation. Applied Sciences 13 (4), 2028.

Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., Nori, F., 2008. An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator. In: Proceedings of the 8th workshop on performance metrics for intelligent systems. pp. 57–61.

Todorov, E., Erez, T., Tassa, Y., 2012. Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. IEEE, pp. 5026–5033.

Wang, C., Zhang, Q., Tian, Q., Li, S., Wang, X., Lane, D., Petillot, Y., Wang, S., 2020. Learning mobile manipulation through deep reinforcement learning. Sensors 20 (3), 939.

Yang, K., Cao, N., Ding, Y., Chen, C., 2024. Bestman: A modular mobile manipulator platform for embodied ai with unified simulation-hardware apis. arXiv preprint arXiv:2410.13407.

Yenamandra, S., Ramachandran, A., Yadav, K., Wang, A., Khanna, M., Gervet, T., Yang, T.-Y., Jain, V., Clegg, A. W., Turner, J., et al., 2023. Homerobot: Open-vocabulary mobile manipulation. arXiv preprint arXiv:2306.11565.