Vision-Based Multi-Camera Robotics Simulation

The SIMLAN Open-Source Framework for Industrial Environments

Hamid Ebadi[1][0000−0003−2944−8223] and Pär Aronsson[1]

Infotiv research, Gothenburg, Sweden
{hamid.ebadi,par.aronsson}@infotiv.se

Abstract. We present SIMLAN (**SIM**ulation for **I**ndoor **M**ulti-Camera **L**ocalization **A**nd **N**avigation), an open-source [1] simulation environment designed for rapid prototyping and testing of multi-camera robotic algorithms. Inspired by Volvo Group's Generic Photo-based Sensor System (GPSS) [5,13] the SIMLAN project models ceiling-mounted camera networks and factory layouts to enable global, vision-based monitoring, localization, and navigation. The system includes utilities for deep-learning-based visual human pose capture and replay, bird's-eye image stitching, geofencing, ArUco-based localization and navigation, all packaged in reproducible Docker containers. SIMLAN reduces the barriers to prototyping and research on the monitoring, verification, and validation of vision-based robotics systems.

Keywords: Simulation environments · Robotic · Computer vision · Multi-camera systems · Human–robot interaction · Verification and validation

1 Introduction

The deployment of autonomous robots in human workspaces introduces complexities that traditional methods do not address, often resulting in a lack of the necessary reliability and flexibility. A key limitation of these conventional approaches is their sole reliance on low-dimensional sensor data such as the Global Positioning System (GPS) for localization and LiDAR for perception. These systems struggle to manage complex environments and dynamic scenarios in which human workers and robots are collocated and collaborating.

This study focuses on the Tuve pilot plant use case in Gothenburg, Sweden, illustrated in Fig. 1, where there are pallet truck transporters responsible for carrying loads while navigating and avoiding static obstacles (e.g., shelves) and dynamic obstacles such as human-operated forklifts and workers.

The GPSS use case offers an alternative logistics system by shifting the localization burden of pallet truck transporters to a network of ceiling-mounted cameras and navigation to a central processing node, enabling simpler robot designs and global coordination that allows efficient path planning and improves

[1] https://github.com/infotiv-research/SIMLAN

control and monitoring. In contrast, traditional robotic solutions rely on a set of sensors on board, including LiDAR, IMU, force/torque sensors, and GPS. This reliance leads to complex onboard processing and high maintenance costs, especially as challenges increase and higher processing demands are placed on the system.

The strengths of GPSS can be framed as the direct weaknesses of the Simultaneous Localization and Mapping (SLAM) algorithm [8,1]. SLAM often struggles where the GPS signals are unreachable, such as indoor environments. In addition, in areas where physical landmarks are absent, repetitive, or constantly changing, such as corridors, warehouses with movable shelving can confuse an onboard SLAM-based system, leading to unstable and uncertain localization. Furthermore, onboard sensors only perceive what is directly within their line of sight. When occlusions occur, localization becomes slow or unreliable. SLAM also relies heavily on LiDAR, which has a limited field of view and is susceptible to cross-talk interference with other LiDAR sensors if multiple robots operate nearby.

Orchestrating and synchronizing multiple robotic agents remains challenging without a reliable global view and centralized coordination. This difficulty persists across tasks ranging from simple right-of-way resolution at intersections and scheduling access to shared resources (e.g., chargers and docks) to avoiding gridlocks in narrow aisles and making complex logistical decisions.

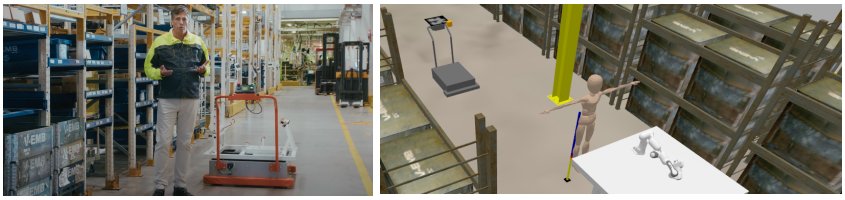
GPSS addresses the difficulty of interpreting non-visual decisions derived from SLAM point clouds. This lack of interpretability often complicates explainability and accountability, both of which are essential in safety-critical industrial environments and legal contexts. Furthermore, many facilities already have installed camera infrastructure for manual security monitoring; leveraging these existing assets for machine-learning systems can efficiently introduce novel features and address emerging use cases.

2 Motivation

SIMLAN is developed to support the GPSS concept by providing a simulation environment that closely resembles the Volvo Trucks warehouse (Fig. 1). It enables researchers and practitioners to prototype robotic algorithms, train machine learning models, and test algorithms that rely on off-board cameras without the complexities and costs associated with physical deployments.

The platform further supports the verification and validation of complex systems by enabling reproducible experiments, ensuring that the results can be consistently tested and compared with the base scenarios. SIMLAN reduces the barrier to entry through its hardware-agnostic and open source choices maintained by the Open Source Robotics Foundation[2]. Unlike resource-intensive alternatives such as NVIDIA Isaac Sim [23], which require dedicated NVIDIA GPUs; SIMLAN enables experimentation and rapid prototyping on standard workstations.

[2] https://www.openrobotics.org



(a) Automated transporters introduced at Volvo Group [14].

(b) Volvo warehouse within SIMLAN simulation

Fig. 1. The GPSS concept: (a) real-world and (b) SIMLAN simulation.

Another major challenge in current robotics simulators is that human workers are often modeled as simple, static objects with rigid movements. This lack of detail is a significant problem for vision-based machine learning algorithms, which cannot effectively work without realistic human shapes and behavior. To solve this, the SIMLAN project models humans as articulated humanoid agents driven by deep learning motion capture. By mapping real human landmarks into simulated joint poses, we provide more detailed and realistic human agents necessary to validate complex vision-based systems.

Beyond development and testing, SIMLAN also facilitates the generation of synthetic data to train machine learning models, addressing the common challenge of acquiring large, high-quality datasets in real-world industrial environments. The simulation environment can be used to generate vast amounts of synthetic data, including camera images, sensor readings, and ground-truth position data, at a fraction of the cost and time required for real-world data collection. This is particularly valuable for algorithms that require a high volume of interaction with the environment during training, such as reinforcement learning (RL) or genetic algorithms (GA).

Finally, SIMLAN offers a safe and privacy-friendly framework for evaluating new algorithms and system configurations without the risk of damaging physical assets or harming real human workers, making it a valuable tool for safety testing.

3 System Description

SIMLAN [15] is built on several established tools in the robotics community, ROS 2 (Robot Operating System 2) [21], an open-source robotics middleware that provides the essential tools and libraries for robotic applications; Gazebo Harmonic [19], a powerful 3D simulator that offers accurate physics and realistic sensor models, and RViz [18], a 3D visualization tool for debugging and monitoring a robot's internal state and perception. By leveraging both ROS 2 and Gazebo, maintained by the Open Source Robotics Foundation; SIMLAN ensures a standardized environment for the seamless transition of algorithms from simulation to physical hardware.

For consistency and reproducibility, the entire Developer Environment is containerized using Docker and Visual Studio Code development containers [22],

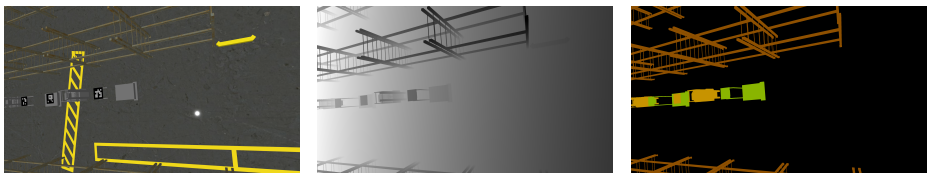
ensuring reliable deployment and execution across diverse host machines. Finally, comprehensive documentation and presentation materials are provided to facilitate onboarding and collaboration.

3.1 Asset and Environment Modeling

Warehouse and object models are based on real-world blueprints and measurement tables, ensuring that simulations are physically accurate and reproducible. The asset library is built entirely with open source and free software, including FreeCAD and Blender for 3D mesh creation. The assets modeled include warehouses and common objects such as shelves, EUR-pallets, traffic cones, and boxes (see Fig. 1). The simulation environment includes realistic physics parameters for contact and collision. Users can configure parameters such as mass, inertia, friction, damping, and contact properties for each object in the Simulation Description Format (SDF) [24].

3.2 Sensor Configuration

The simulation environment supports a diverse suite of sensors, facilitating multi-modal outputs such as RGB imagery, depth maps, and semantic segmentation as seen in Fig. 2.



(a) RGB camera (b) Depth sensor (c) Semantic segmentation

Fig. 2. Sensors

The camera configuration used in the warehouse is defined through its intrinsic and extrinsic parameters. The intrinsic and extrinsic properties of the camera are provided by Volvo Trucks, which defines the setup of the camera in the warehouse. These camera configurations and a library to load them are available in the `camera_utility/` package. Inside the library, there are utility functions to convert rotation matrices and Rodrigues vectors used by OpenCV into quaternions used by the Gazebo simulator.

The *intrinsic parameters* describe the internal geometry of the camera and are represented by the camera matrix.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where f_x and f_y are the focal lengths in pixel units, and (c_x, c_y) is the principal point (usually near the center of the image). Intrinsics also include distortion coefficients that correct lens imperfections, as well as image resolution. The intrinsic calibration files, therefore, contain the camera matrix K, distortion coefficients D and image resolution.

The *extrinsic parameters* describe the pose of the camera in the world using a rotation matrix R and a translation vector t, which together form the world-to-camera transformation matrix $[R|t]$. While the intrinsics map camera coordinates to pixel coordinates, the extrinsic properties transform world coordinates into the camera coordinate system. The extrinsic calibration files additionally contain the rotation matrix R, translation vector t, camera projection matrix P (computed as $P = K[R|t]$) and the position of the camera in 3D space (factory coordinate frame). These parameters are essential for camera projection and for ArUco localization, which will be discussed in the next two sections.

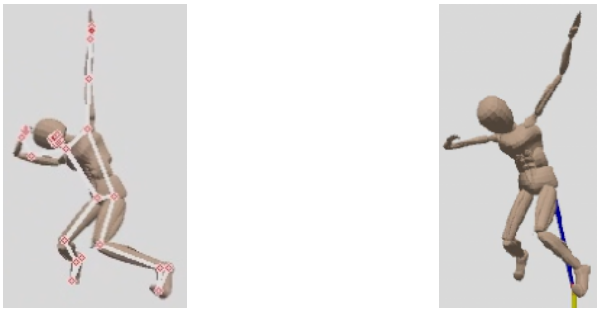
Finally, a collision sensor installed on each agent serves as the primary fail-safe mechanism, functioning as the final layer of a robot's safety for emergency halting upon contact with other agents or objects.

3.3 Humanoid Agent

While Gazebo's actor [11] supports advanced visual features like skeleton animation from COLLADA or BVH files and scripted trajectories, their primary limitation is that they are static and cannot interact dynamically with the simulation's physical world, limiting their behavior to what they are strictly scripted for. SIMLAN addresses the lack of a flexible human worker model by incorporating humanoid robots into its simulator. The humanoid robots are based on the Unified Robot Description Format (URDF) models that are used with the *"Human Dynamics Estimation software suite"* [12] and MoveIt2 [6] which are used for motion planning and control of the humanoid. Google Mediapipe [20] is used [4] for estimating human pose landmarks. The extracted landmarks are then processed by a custom neural network, which maps them to the joint controls of humanoid robots, managed by MoveIt2.

This project presents a deep learning-based system for translating human posture from 2D imagery into humanoid robot configurations. By leveraging synthetic data generation within a simulated environment, the system circumvents the necessity for labor-intensive manual labeling. The architecture bridges the gap between human pose estimation and robotic control by utilizing Google MediaPipe for the extraction of 33 spatial landmarks (P) from video streams, mapping these coordinates to 47 robot joint parameters (M). The implementation utilizes MoveIt2 for kinematic execution and motion planning.

Synthetic Data Generation: A primary contribution of this work is the automated pipeline for generating paired datasets $\langle P, M \rangle$. Instead of empirical human data collection, the system utilizes a simulator as a "ground truth" generator:



(a) MediaPipe landmarks (b) Predicted motion

Fig. 3. Kinematic mapping

1. Stochastic Sampling: Random joint configurations M are dispatched to the humanoid model within the Gazebo simulator.
2. Observation: Multiple cameras capture the humanoid from diverse perspectives to ensure spatial robustness.
3. Pose Extraction: The MediaPipe pose estimator processes these synthetic images I_s to extract landmark coordinates P.
4. Data Alignment: The resulting dataset consists of multi-view pose inputs aligned with the original kinematic command, facilitating multi-camera fusion during training.

Kinematic Mapping: The translation of human landmarks to robot joint states is treated as a high-dimensional regression problem, evaluated through two distinct modeling approaches. First, a Multilayer Perceptron (MLP) developed in PyTorch utilizes a custom neural network architecture optimized via the Optuna framework for hyperparameter tuning; this model processes flattened landmark arrays to predict a 47-element joint vector. Second, an Ensemble Regression approach using AutoGluon employs a tabular ensemble method to train specialized predictors for individual joints, subsequently aggregating these outputs to form a complete predicted motion sequence.

The mathematical foundation of the system relies on the assumption of Pose Estimator Domain Invariance [2,16]. Let I_s represent a simulated image and I_r represent a real-world image. The relationship can be defined as follows:

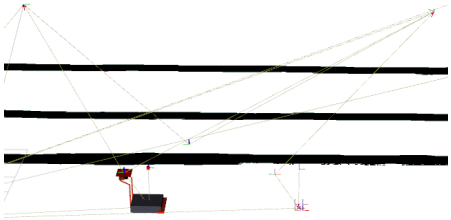
- Forward Simulation: $SIM(M) \rightarrow I_s$ (The simulator generates an image from motion).
- Feature Extraction: $PE(I) \rightarrow P$ (The Pose Estimator extracts landmarks from an image).
- Learned Mapping (Q): The objective is to derive a function Q such that $Q(P) \approx M$.

By training on synthetic pairs $\langle P_s, M \rangle$, we derive the mapping function Q. Under the assumption that the pose estimator PE performs consistently across both domains ($P_r \approx P_s$), and then executing the pipeline $Q(PE(I_r))$.

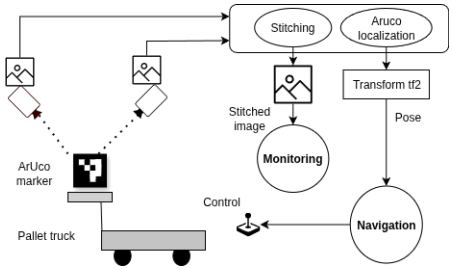
3.4 Robot Agent

To demonstrate the utility of the SIMLAN simulated environment for image processing and control algorithms, we implemented a lightweight GPSS system within the SIMLAN project. This implementation includes ArUco-based [10] localization, agent navigation, and a bird's eye view visualization for centralized monitoring. The implementation of these subsystems is provided in `aruco_localization/` and `camera_bird_eye_view/` packages.

Localization: The OpenCV [3] ArUco library [10] is used to detect and process ArUco markers that are mounted on the agents, publishing their precise 3D position and orientation in space to the transform tree [9] (`tf2`). ROS 2's transform system (`tf2`) keeps track of how different coordinate frames move and relate to each other over time, organized in a tree-like structure of parent-child relationships (the `tf2` tree) to support consistent spatial understanding. In `tf2`, each detected ArUco marker is represented as a child frame of its corresponding camera(s). If multiple cameras observe the same marker, the system merges the detections to provide a robust estimate of the marker's position. The localization system is tightly integrated with the navigation stack (see Fig. 4a), enabling accurate and centralized control of all agents.



(a) Localization of an ArUco marker on a pallet truck by two cameras, visualized in RViz.



(b) Flow from camera images to ArUco-based localization and Nav2 navigation.

Fig. 4. Camera network system

The `aruco_localization` package runs the ArUco-based localization, supplying the necessary `tf2` chain from the origin link to the robot (e.g., pallet truck). By simple re-mapping of topics and without changing the navigation system, the navigation system utilizes this localization information for the planning and execution of the path.

Navigation: Nav2 is a modular, behavior-tree-based autonomous navigation framework for ROS 2 that enables mobile robots to perform reliable localization, path planning, and obstacle avoidance through a plugin-based architecture. The

Nav2 framework depends on a consistent `tf2` chain `map`, `odom`, and `base_link` to be able to perform its internal path planning. Typically, the `map` to `odom` chain represents global localization drift correction and the `odom` to `base_link` chain represents local odometry. In our setup, the `tf2` transform structure is organized such that the `map` frame is static at origin, `robot_agent_X/odom` frame is placed at the robot's agent initial pose, and the distance between the `robot_agent_X/odom` and `robot_agent/base_link` frames is calculated using ArUco localization. Figure 4b illustrates this data flow.

Monitoring: Images from different cameras (Fig. 6b) can be projected [25] onto a common canvas by applying the same sequence of transformations that map 3D world points to 2D pixel coordinates as illustrated in Fig. 5, while accounting for each camera's unique parameters. The world coordinates are transformed into the camera coordinate system using the extrinsic parameters $[R|t]$, which define the orientation and position of the camera in space and are projected onto the image plane using the intrinsic calibration matrix K.

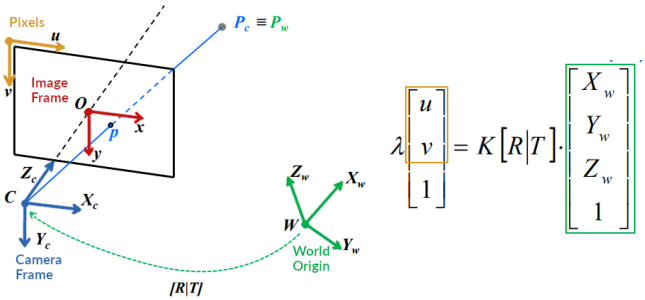
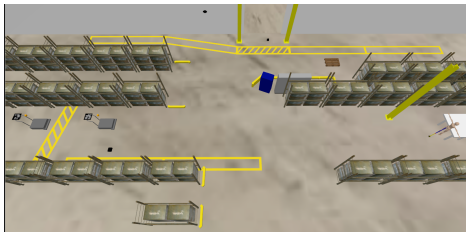


Fig. 5. Convert from world to pixel coordinates [17]

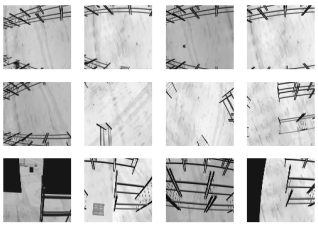
The `camera_bird_eye_view/` package initializes the node responsible for image projection and final stitching (Fig. 6). This node then publishes the respective processed images to a new topic.

Geofencing: There is a geofencing mechanism that utilizes Behavior Tree [7] condition node, which, when triggered by predefined safety issues, executes a Behavior Tree action to immediately stop the truck's navigation. Current safety triggers monitored by this system include the activation of a collision sensor and the loss of observability of a pallet truck or when a robot reaches a restricted area.

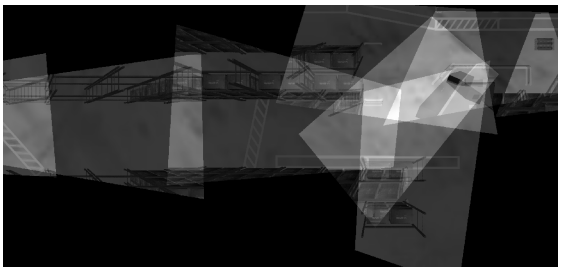
This design enables the navigation stack to seamlessly utilize the localization data for robust and accurate multi-agent navigation, without requiring onboard sensors (except a collision sensor as the safety feature) on the robot agent.



(a) Side-view perspective for context.



(b) Multi-camera view of the environment.



(c) Stitching images from (b).

Fig. 6. Bird's eye view of the simulation environment

3.5 Multi-Agent

In Robot Operating System 2 (ROS 2), the concept of namespaces provides a structured mechanism for organizing and isolating communication entities such as nodes, topics, services, and actions and for avoiding naming conflicts. In this setting, each robot agent is assigned a unique namespace and an identification number corresponding to its ArUco ID, enabling scalable multi-agent navigation and individualized control. The simulation supports dynamic agent spawning, with configuration options for initial positions, robot types (e.g., pallet truck, forklift, jackal robots, human actor, panda arm), and sensor assignments. The use of namespaces ensures that all robot agents can be visualized and controlled independently in ROS2 and RViz, and that navigation stacks can be launched for each agent in isolation.

4 Future Work

At present, agent localization and orientation are based exclusively on ArUco markers. While this method is fast and accurate, it requires the placement of specific visual markers on every agent. Future iterations can integrate existing vision pipelines to infer an agent's position and orientation directly from its visual appearance. This transition will make the system more robust to challenges such as occlusions when parts of the ArUco marker are partially masked.

5 Experimental Results

The current version of SIMLAN was evaluated using ROS 2 Jazzy Jalisco and Gazebo Harmonic on a workstation running Linux (Ubuntu 24.04 LTS). The system was equipped with an Intel Core i9-10900KF CPU @ 3.70 GHz (10 cores, 20 threads), 32 GB of RAM, and an NVIDIA GeForce RTX 2070 Super GPU with 8 GB of VRAM.

The efficiency of the simulation is evaluated using the Real-Time Factor (RTF), a metric that represents the ratio of simulated time to actual wall-clock time. As shown in Fig. 7, the introduction of additional camera sensors increases the computational burden, causing the RTF to drop, indicating a degradation in processing performance.

A key technical advantage of using simulation is the ability to adjust the RTF. By decoupling the simulation clock from the wall-clock time, complex algorithms can be evaluated on modest hardware without the risk of desynchronization, effectively removing the requirement for expensive high-performance computing clusters while maintaining high fidelity in experimental results.

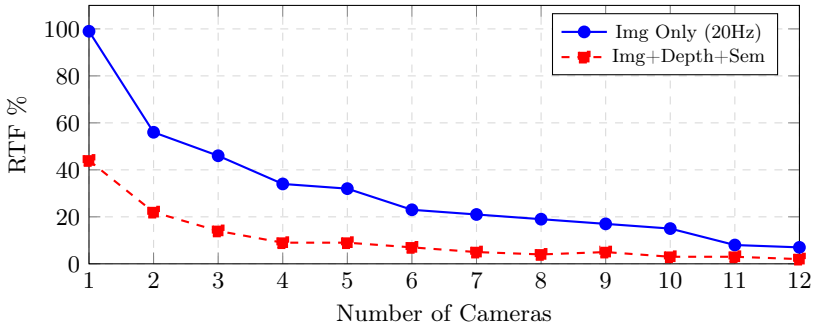


Fig. 7. Impact of camera count and feed type on RTF.

6 Conclusion

This paper introduced SIMLAN, an open-source simulation environment tailored for the development, testing, and validation of multi-camera robotic systems in industrial settings. By leveraging a realistic digital twin of a Volvo truck warehouse and GPSS system, integrating robust frameworks like ROS 2 and Gazebo, and emphasizing reproducibility and ease of use, SIMLAN effectively bridges the gap between theoretical research and industrial deployment. The project also integrates human operators into the simulation environment by representing them as humanoid agents. These models are driven by deep-learning-based visual motion capture, which maps human landmarks onto simulated joint configurations to enable realistic kinematic behavior.

Beyond its functional capabilities, SIMLAN is distinguished by its commitment to open science, as it is released under the Apache License 2.0. This framework ensures that the tool remains accessible for both academic and industrial advancement, establishing SIMLAN as a valuable, forward-looking resource for the robotics and AI communities.

Acknowledgments. This work was supported by the SMILE IV project (Vinnova grant 2023-00789) and the EUREKA ITEA4 ArtWork project (Vinnova grant 2023-00970). We would like to thank the teams from Infotiv AB, RISE Research Institutes of Sweden, Volvo GTO, Dyno-Robotics, and Chalmers University of Technology for their contributions and their work on the open-source project.

References

1. Bailey, T., Durrant-Whyte, H.: Simultaneous localization and mapping (slam): part ii. IEEE Robotics & Automation Magazine **13**(3), 108–117 (2006). https://doi.org/10.1109/MRA.2006.1678144
2. Ben-David, S., Blitzer, J., Crammer, K., Kulesza, A., Pereira, F., Vaughan, J.W.: A theory of learning from different domains. Mach. Learn. **79**(1–2), 151–175 (May 2010). https://doi.org/10.1007/s10994-009-5152-4
3. Bradski, G.: The opencv library. Dr. Dobb's Journal of Software Tools **25**, 120–125 (2000)
4. Casparsson, T., Yi, S.: Human Motion Replay on a Simulated Humanoid Robot Using Pose Estimation. Master's thesis in systems, control and mechatronics, Chalmers University of Technology (2025)
5. Chalmers University of Technology: In the factory of the future, humans and robots work together on equal terms (2024), https://www.chalmers.se/en/current/news/e2-in-the-factory-of-the-future-humans-and-robots-work-together-on-equal-terms/
6. Coleman, D., Sucan, I., Chitta, S., Correll, N.: Reducing the barrier to entry of complex robotic software: a moveit! case study (2014), https://arxiv.org/abs/1404.3785
7. Colledanchise, M., Ögren, P.: Behavior trees in robotics and ai (Jul 2018). https://doi.org/10.1201/9780429489105
8. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part i. IEEE Robotics & Automation Magazine **13**(2), 99–110 (2006). https://doi.org/10.1109/MRA.2006.1638022
9. Foote, T.: tf: The transform library. In: 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA). pp. 1–6. IEEE, Piscataway, NJ (2013). https://doi.org/10.1109/TePRA.2013.6556373
10. Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition **47**(6), 2280–2292 (2014), https://www.sciencedirect.com/science/article/pii/S0031320314000235
11. Gazebo Documentation: Tutorial : Make an animated model (actor). Gazebo Classic Documentation (2025), https://classic.gazebosim.org/tutorials?tut=actor, Accessed: September 29, 2025
12. Gorbani, D., Sartore, Carlotta, R.: human-gazebo: URDF models of humans. https://github.com/robotology/human-gazebo (2024), version retrieved: September 29, 2025

13. Group, V.: AI transporters push the boundaries of modern manufacturing. https://www.volvogroup.com/en/news-and-media/news/2024/nov/ai-modern-manufacturing.html (Nov 2024)
14. Group, V.: Automated transporters introduced at Volvo Group powered by AI and computer vision. https://www.youtube.com/watch?v=DA7lKiCdkCc (2024), https://www.youtube.com/watch?v=DA7lKiCdkCc, accessed: 2025-08-22
15. Infotiv Research: SIMLAN Project. https://github.com/infotiv-research/SIMLAN (2024), accessed on 22 August 2025
16. Jakab, T., Gupta, A., Bilen, H., Vedaldi, A.: Unsupervised learning of object landmarks through conditional image generation (2018), https://arxiv.org/abs/1806.07823
17. Joel P. Barmettler: Chapter 3 – Camera Projection [Jupyter Notebook]. https://github.com/polygon-software/python-visual-odometry/blob/master/Chapter%203%20-%20Camera%20Projection.ipynb (2019), accessed: 2025-08-27
18. Kam, H.R., Lee, S.H., Park, T., Kim, C.H.: Rviz: a toolkit for real domain data visualization. Telecommun. Syst. **60**(2), 337–345 (Oct 2015). https://doi.org/10.1007/s11235-015-0034-5
19. Koenig, N., Howard, A.: 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, pp. 2149–2154 vol.3. IEEE, Piscataway, NJ (2004). https://doi.org/10.1109/IROS.2004.1389727
20. Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.L., Yong, M.G., Lee, J., Chang, W.T., Hua, W., Georg, M., Grundmann, M.: Mediapipe: A framework for building perception pipelines (2019), https://arxiv.org/abs/1906.08172
21. Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W.: Robot operating system 2: Design, architecture, and uses in the wild. Science Robotics **7**(66), eabm6074 (2022), https://www.science.org/doi/abs/10.1126/scirobotics.abm6074
22. Microsoft: Developing inside a container. https://code.visualstudio.com/docs/devcontainers/containers (2025), accessed: 2025-09-01
23. Mittal, M., Roth, P., Tigue, J., Richard, A., et al.: Isaac lab: A GPU-accelerated simulation framework for multi-modal robot learning. arXiv preprint arXiv:2511.04831 (2025), https://arxiv.org/abs/2511.04831
24. Open Source Robotics Foundation: SDFormat: Simulation Description Format. http://sdformat.org/spec (2024), Accessed: 2025-08-22
25. Scaramuzza, D., Bauersfeld, L., Romero, A.: Lecture slides for vision algorithms for mobile robotics. Online lecture materials (2019), https://rpg.ifi.uzh.ch/docs/teaching/2019/02_image_formation_1.pdf, robotics and Perception Group, University of Zurich