Draft: Zirbi Team Description Paper

Nils Mandischer Robert Jeutter Lukas Asam Daniel Gabler
Aleksander Michalak Sofia Öttl Jonas Platzer
Leonie Schmidt Peter Viechter Meruna Yugarajah

October 21, 2025

Abstract. This paper is a preliminary draft.

1 Introduction

This year marks the formation of a new team from the University of Augsburg participating in the RoboCup@Home competition. We work in collaboration with Aracom, whose expertise supported the team throughout the development process. Our team is comprised of Robert Jeutter from Aracom, Nils Mandischer from UniA, and the eight students in the Autonomous Robotics course. While the team is newly founded, it builds upon the technical expertise and academic background of our institution, aiming to contribute innovative approaches to robot perception, manipulation, and human–robot interaction.

For our first participation, we aim to establish a stable foundation for autonomous service robotics research. This means that we are not aiming for immediate top-level performance, but rather prioritize software architecture, maintainability, and usability. This foundation is intended to be further expanded by future students to steadily increase the robot's capabilities in the coming semesters.

We intend to lay the groundwork for long-term success in RoboCup@Home, both in terms of competition performance and in providing a valuable research and teaching environment at the University of Augsburg.

2 Description of the hardware

All components of the robot are mounted on a mobile platform. This platform is provided in collaboration with Aracom, whose robotic system forms the physical basis for our developments. This cooperation allows us to focus our efforts on building robust and flexible software components while leveraging reliable and proven robotic hardware. For the object manipulation we are using the UR5 arm from Universal Robotics that is mounted on the robot base. As an end-effector we are using a 2-finger gripper from Robotiq. In addition, the system is equipped with a variety of sensors and computing units to support perception, navigation, and human–robot interaction. This includes three NVIDIA Jetson Orin Nano modules for onboard processing, a LiDAR and a RealSense camera for environment sensing, as well as microphone, sonar, IMU, RTC, speaker, and an OLED display to facilitate multimodal interaction and precise timing.

3 Software architecture overview

3.1 Safety Approach

Safety in robotic systems must be addressed at multiple levels, ranging from local subsystem protection to global system-wide monitoring. In the proposed architecture, safety responsibilities are split between local safety nodes and a central monitoring node. Local safety nodes are embedded within each subsystem (e.g., manipulation, navigation, perception) and provide immediate protection against hazards specific to their domain, such as joint limit violations, sensor malfunctions, or collision risks.

The central safety node aggregates these local reports, maintaining a holistic view of system health. This is achieved through a cyclic publish–subscribe mechanism, where all safety-relevant modules periodically publish status updates. The central node monitors these heartbeats, allowing for the rapid detection of failures or communication breakdowns. This design enables both decentralized reaction to local hazards and centralized enforcement of global safety policies, ensuring scalability and resilience.

In the event of an anomaly, the central safety node evaluates the severity and context. Minor issues may trigger predefined mitigation strategies, such as slowing down movements or recalibrating sensors. Severe conditions (e.g., critical hardware faults or loss of localization) may escalate into a system-wide freeze, halting all robot activity to prevent accidents. This layered strategy balances responsiveness with robustness, ensuring that the robot can continue operating under non-critical disturbances while still guaranteeing a safe shutdown in emergencies.

3.2 Movement and Navigation

Movement and Navigation in a mobile service robot requires the integration of mapping, localization, path planning and aware navigation. For a solid base to develop on the navigation stack builds upon Nav2, a reliable and tested ROS2 library that provides tools for path planning, obstacle avoidance, and trajectory execution.

For generating and updating maps the SLAM-toolbox is used, which enables mapping of undiscovered areas, as well as localizing the robot in already mapped out environments. Leg tracking based on LiDAR data [1] lets the robot detect humans. A combination of this package and a computer vision model, that's trained on identifying different persons, shall enable the tracking of a selected one. The logic in the background should couple the identified position of a recognized human face to a pair of legs at roughly the same coordinates. This individual can then be tracked, even when leaving the field of view of the rgb camera, just by the above mentioned leg tracking algorithm. This shall provide a natural robot-human interaction when following.

The central intelligence of the robot, the global decision making, sends commands to a so-called movement orchestrator. These have a custom defined message type 'MovementAction'. It defines parameters like whether mapping and or localization should be activated or not, or a destination, in case the robot should move to a specific location. The orchestrator processes this request and forwards it to the navigation manager, which is responsible for properly starting and stopping the modules for localization, mapping and navigation/path-planning. When an executable trajectory is calculated, the according velocity is communicated to the omni-wheels hardware interfaces which translates into a physical motion.

After the task is concluded, the orchestrator provides feedback of success (or failure) to the global decision making and logs the outcome in the knowledge base.

3.3 Object Manipulation

Manipulation in a robotic system requires the seamless integration of kinematic control, motion planning, and end-effector coordination. In this project, a Universal Robots UR5 arm serves as the primary object manipulator, offering a versatile six-degree-of-freedom platform capable of executing both pick and place tasks. The UR5 is augmented with a Robotiq adaptive gripper, which provides robust and flexible gripping capabilities including a variation of the gripping force. This enables the system to handle a wide range of objects with varying shapes and surface properties.

The movement of the manipulator is coordinated through MoveIt2, which provides advanced capabilities for path planning, collision avoidance, and trajectory optimization. This framework allows the manipulator to compute feasible paths in real time and achieves modularity and interoperability of the system.

In this architecture, the global decision maker initiates a ManipulationAction on the Orchestrator, specifying both the ManipulationCommand, which is the task to be performed, and the identifier of the target object. To contextualize the request, the Orchestrator queries the Knowledge Base for object metadata, including semantic labels, bounding boxes, and point cloud representations. Once the property of the object is retrieved, the Orchestrator invokes the Gripping Force Service, which computes the required gripping force.

The Orchestrator subsequently transforms the object's point cloud and bounding box into the manipulator's base coordinate frame, so the system has a consistent reference frame for planning. From this representation, the gripping pose is derived. Currently, the position is estimated from the point cloud, while the orientation is inferred from the bounding box geometry.
The collective data is then sent to MoveIt2 where one of the 6 jobs is called according to the Orchestrator command. The manipulator is capable of executing four fundamental tasks: Home, Move, Open, and Close, as well as two higher-level tasks: Pick and Place. Invoking the Home task causes the manipulator to move to a predefined home position, while the Move task directs it to a specific pose provided by the orchestrator. The Open and Close tasks control the gripper, with the Close task additionally taking the gripping force into account. The Pick task involves grasping an object at a designated gripping pose, whereas the Place task executes the placement of an object at a specified placing location.
Finally, the Orchestrator reports the outcome, success or failure, back to the Decision Making module and logs the outcome as a RobotStatusEvent in the Knowledge Base.

This structured approach enables reliable and flexible manipulation, ensuring that the robot can adapt to a variety of objects and tasks. By integrating perception, planning, and execution in a modular framework, the system provides a foundation for further enhancements in autonomy, efficiency, and safe operation in complex, unstructured environments.

3.4 Speech Interaction

Speech is the main interface between a human operator and the robot.

To achieve natural and flexible voice interactions, we chose to employ a large language model that is deployed locally on one of the Jetson nanos. As LLM model, we use Qwen3-8b[2] that is executed by Ollama[3], a framework to build and run all kinds of LLMs. Qwen3-8b fits our needs, as it is reasonably small, supports reasoning, and has tool support.

This tool support allows us to define tools with parameters. These tools can then be triggered by the LLM in the course of an interaction. The triggered tool with its parameters is then validated and passed to the right subsystems for execution. For example, an operator tells the robot to pick up the green shopping bag. The LLM then triggers the pick_up_bag tool and inserts green as the color parameter. After that, feedback is given to the operator containing the subsystem status and potential errors.

For the speech-to-text input to the LLM we use a combination of silero-vad[4] as voice activity detection and pywhispercpp[5] as speech-to-text converter. Since this does not include any kind of wakeword detection (like "Hey Robot"), we check any text input for relevance before it is passed to the main chat. This is achieved by passing the text input to the LLM, but with the given task to just return if the text is relevant to the robot or not.

The text-to-speech output is handled by Tortoise TTS[6].

All stages of the speech interaction pipeline stream their output where possible (text tokens, audio samples), to achieve a low latency between operator request and an answer from the robot.

3.5 Knowledge-base

The Knowledge Base represents a central information hub within the robotic system, designed to store, organize, and provide access to heterogeneous data sources. It serves as the persistent memory of the system, enabling decision-making modules, orchestrators, and subsystems to operate with up-to-date and consistent information. All access is provided via multiple ROS services.

Entity Management The database supports entities: objects in the robot's environment such as people, furniture, or pickable items. Each entity contains structured metadata including semantic labels, spatial coordinates (x,y,z), orientation (quaternion), and object-specific properties. This enables the orchestrator to retrieve geometric and semantic information for manipulation tasks.

Event Logging The system maintains a comprehensive event log of robot operations, capturing timestamps, event types (Manipulation, Movement, Safety), and contextual metadata. Compared to the normal ROS log, all event types and attributes are fixed and not just message strings. This means the events can be used by the main decision maker to adapt the robot's behavior in real-time and give feedback to human operators.

SLAM Map Integration The Knowledge Base supports persistent storage of SLAM-generated maps. This makes it possible to map the robot's surroundings once, save the map, and then reuse the pre-mapped environment across sessions.

Technical Implementation Built with SQLAlchemy ORM[7] and PostgreSQL[8] backend, the database provides robust relational storage. The system uses async I/O with asyncpg[9] for PostgreSQL interactions, and all ROS 2 services are wrapped in a ReentrantCallbackGroup to enable concurrent request processing, ensuring low-latency access to the shared data model.

This architecture ensures a unified data model accessible via ROS 2 services, reducing redundancy while maintaining synchronization between perception, decision-making, and execution subsystems.

3.6 Computer Vision and Perception

The perception subsystem forms the robot's sensor interface and provides the raw data from the currently deployed sensors system-wide. At present, an RGB camera and a LiDAR are used; no explicit noise filtering of the data streams is performed. The LiDAR data is used in real time by the Movement subsystem to derive obstacles, free space, and local navigation cues, while the camera data is fed to the Computer Vision module.

In the Computer Vision module, camera images are used for real-time object detection and classification. For each detection, among other attributes, the object name and a 3D pose with fixed orientation are derived. The resulting information is stored as structured entities in the Knowledge Base, yielding a consistent, queryable representation of the world state. For object detection, we use YOLO-11n [10], as this model offers a convincing balance of accuracy and latency on edge hardware, performs robustly in everyday, partially occluded scenes, and scales within the YOLO family (from Nano to Large) without architectural changes. Pretrained weights facilitate coverage of numerous object categories and enable domain-specific fine-tuning. Overall, YOLO-11n meets our robot's real-time and versatility requirements while maintaining moderate resource demands [10].

4 Conclusions and future work

In the future, we plan to extend the manipulation parts of the system in several directions. First, all functions will be tested in a fully functional simulation and on the hardware, to ensure robust performance under varying conditions. Second, we aim to develop a placing point algorithm to realize precise object placement. Finally, the perception pipeline will be integrated to enable real-time collision avoidance, allowing the manipulator to operate safely and efficiently in dynamic and cluttered environments. Together, these advancements will increase the manipulator's autonomy, adaptability, and applicability to real-world scenarios.

The movement and navigation part has to harmonize it's inputs for the leg tracking function with computer vision. By this, and the combination with the leg tracking algorithm and extensive testing a robust human tracking should be made possible. Next to this, the hardware integration with the drivetrain and finetuning of the safety features have to be executed.

On the perception side, we will explore the integration of additional sensors to broaden coverage and improve robustness, and we will introduce principled preprocessing of sensor data—such as filtering, outlier handling, and temporal smoothing—to increase signal quality without altering the overall system architecture.

For computer vision, we plan to implement IoU-based tracking to maintain persistent identities across frames and to update existing entries in the Knowledge Base rather than re-adding them each frame. In addition, we will integrate depth-image handling where available to strengthen 3D localization, scale estimation, and occlusion reasoning, thereby improving downstream planning and interaction.

References

1. Puru Rastogi and Atharva Pusalkar. ros2_leg_detector. `https://github.com/mowito/ros2_leg_detector`, 2021. Commit: b4be35a2f540c0e72cb8718e99fd9a23dc31f501.
2. An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.
3. Ollama. Ollama: Get up and running with large language models. `https://github.com/ollama/ollama`, 2025.
4. Silero Team. Silero VAD: pre-trained enterprise-grade Voice Activity Detector (VAD), Number Detector and Language Classifier. `https://github.com/snakers4/silero-vad`, 2024.
5. Abdeladim Sadiki. pywhispercpp: Python bindings for whisper.cpp . `https://github.com/absadiki/pywhispercpp`, 2025.
6. James Betker. TorToiSe text-to-speech. `https://github.com/neonbjb/tortoise-tts`, 2022.
7. Michael Bayer. Sqlalchemy. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012.
8. The PostgreSQL Global Development Group. PostgreSQL: The World's Most Advanced Open Source Relational Database. `https://www.postgresql.org`, 2025.
9. The asyncpg authors and contributors. asyncpg: A fast PostgreSQL Database Client Library for Python/asyncio. `https://github.com/MagicStack/asyncpg`, 2025.
10. Ultralytics. Yolo11 — ultralytics yolo documentation. `https://docs.ultralytics.com/de/models/yolo11/`, 2025. Accessed: 2025-09-21.