AutonOhm@Work
Team Description Paper 2025

Sina Steinmueller, Hannes Haag, Hannes Duske, Denis Trescher, Tim Lachmann, Usukh-Erdene Batbold, Timon Schindler, Stefan Seitz, Katharina Nätscher, Zhongyao Zhang, Lukas Fuchs, Silvana Gosewehr and Marco Masannek

Technische Hochschule Nürnberg Georg-Simon-Ohm
Kesslerplatz 12, 90489 Nuremberg, Germany
steinmuellersi89050@th-nuernberg.de
https://www.th-nuernberg.de/en/faculties/efi/research/laboratories-actively-involved-in-research/mobile-robotics/robocupwork/

Abstract. This paper presents the team AutonOhm and their solutions to the challenges of the RoboCup@Work league. The hardware section covers the robot setup of Ωmnibot - V4 (Ωmeg4), which was developed using knowledge from previous robots used by the team. Custom solution approaches for the @Work navigation, perception, and manipulation tasks are discussed in the software section, as well as the control architecture for the autonomous task completion.

1 AutonOhm@Work

History The AutonOhm@Work team from the Nuremberg Institute of Technology was founded in September 2014 and has since participated in various tournaments. The team was able to win 3 local and 4 global events* (see tab. 1). It consists of a mix of bachelor, master and phd students with various fields of research (see tab. 2).

Robot Since the initially used robot model, the KUKA Youbot, was discontinued and the last remaining one broke down at the World Cup in 2018, the team started developing a custom robotic platform called Ωnibot, which has been continously improved and reworked since 2019.

Contributions to the League We've started to bundle our knowledge in a repository [23], where many aspects of our solution are be publicly available. With this repository, we want to share our knowledge about autonomous robots and task execution.

Fig. 1: Team AutonOhm 2025 with Ωmeg4 (Room SH8)

Table 1: History of AutonOHM@Work

Year	Location	Event	Result	Robot
2015	Magdeburg (Germany)	RC German Open	5th	Youbot V1.2
2016	Leipzig (Germany)	RC World Cup	5th	Youbot V1.3
2017	Magdeburg (Germany)	RC German Open	1st*	Youbot V1.4
2017	Nagoya (Japan)	RC World Cup	1st*	Youbot V1.5
2018	Magdeburg (Germany)	RC German Open	1st*	Youbot V1.6
2018	Montreal (Canada)	RC World Cup	1st*	Youbot V1.6
2019	Magdeburg (Germany)	RC German Open	6th	Ωmnibot V1
2019	Sydney (Australia)	RC World Cup	5th	Ωmnibot V1.2
2020	Virtual Online	RC Asia Pacific	3rd	Ωmnibot V2
2021	Virtual Online	RC World Cup	1st*	Ωmnibot V2.1
2021	Bologna (Italy)	SciRoc Ep5	1st*	Ωmnibot V2.2
2022	Bangkok (Thailand)	RC World Cup	1st*	Ωmn3
2023	Bordeaux (France)	RC World Cup	4th	Ωmeg4
2024	Eindhoven (Netherlands)	RC World Cup	5th	Ωmeg4

Table 2: Team AutonOHM@Work 2024

Member	Field of Study	Tasks
Marco Masannek	Cognitive Robotics (Phd)	Project Lead Mentor Tech Lead
Sina Steinmüller	Media Engineering (B-Eng)	Team Lead Marketing Software
Hannes Haag	Scientific Engineering (Phd)	Hardware Lead
Hannes Duske	Perception for mobile robots (Phd)	Software Lead
Dennis Trescher	Applied Research (M-Sc)	Object Perception
Tim Lachmann	Media Engineering (B-Eng)	Arm Controller
Usukh-Erdene Batbold	Mechatronics (B-Eng)	Hardware Dev
Lukas Fuchs	Mechatronics (B-Eng)	April Tags
Timon Schindler	Mechatronics (B-Eng)	Software, Orga
Stefan Seitz	Electrical Engineering and Information Technology (B-Eng)	Sensors
Katharina Nätscher	Mechatronic Systems (M-Eng)	Gripper
Zhongyao Zhang	Mechatronics (B-Eng)	Gripper
Silvana Gosewehr	Electrical Engineering and Information Technology (B-Eng)	Simulation

2 League Description

The RoboCup@Work league, established in 2012, focuses on the use of mobile manipulators and their integration with automation equipment for performing industrial-relevant tasks [1]. Participating robots must be able to navigate in a previously known arena, reach different service areas and perform manipulation tasks. Therefore, each robot must be able to correctly identify a requested object on various background surfaces and pick and place the objects that vary in shape, appearance and weight. With low participation numbers of teams after the COVID-19 pandemic in the first on-site robocup at the world cup 2023 in Bangkok, Thailand, major changes to the structure of the competition have been made with the intention to ensure the future of the league. The main idea is to make setting foot in the league more easy for newer teams, while still keeping the ambition to target scientific problems in robotics in the more advanced stages of the competition. Therefore, the competition has been split into two main sections, the beginner and the advanced section. The Beginner Section isolates or simplifies some of the requirements for the robots so teams without much experience can successfully participate without having a competitive setup in relation to the more advanced teams. The Advanced Section introduces a new (more difficult) set of objects, aswell as incorporating Precise Placement and Rotating Table tasks into the newly introduced Advanced Transportation Tests. The standalone Tests have therefore been removed from the competition schedule. As these changes increase the upper treshold of skill required to do all tasks without making mistakes, the option to replace the real objects with april-tagged cubes has been introduced to relax the problem of object detection and manipulation.

3 Hardware Description

Version 4 of our Ωnibot, Ωmeg4, is built on a customized Evocortex[3] R&D platform. The platform is equipped with an omnidirectional mecanum drive, an aluminum chassis capable of carrying loads up to 100 kg and a Li-Ion Battery with a nominal voltage of 48V. Our platform was issued as a bare chassis without any sensors, power management or computation units. It serves as our base and all further components needed for the competition are mounted in or on this chassis by the team.

3.1 Sensors

Lidar Mapping, navigation and the detection of physical obstacles all depend on the perception of the environment. Three 2D LiDAR sensors from Sick were used in the previous version of the robot platform (Ωmn3). The current robot uses a single Rplidar S2E from Slamtec, a 360°2D LiDAR sensor, which is mounted in the robots center below its floor.

Time-of-Flight sensors The centrally mounted lidar has blind spots in the four corners of the robot beyond its mecanum wheels. For this purpose, two custom 3D Time-of-Flight (ToF) sensors boards provided by EduArt Robotik are attached in each corner to fill the blind spots (see fig. 4). The sensor boards use the ST VL53L8CX 3D Time-of-Flight sensors with a resolution of 8×8 distance measurements and a square field of view of $45° \times 45°$. The sensors are connected in a ring topology on a CAN FD Bus, called the sensor ring.

Cameras We use an Intel RealSense D435 3D-camera for the object detection. It is attached to the sensor head that it can be positioned above the workstations and detect the surface as well as the pose of the objects. For barrier tape detection, multiple ELP USB fisheye cameras can be mounted around the robot, which enable 360° view. During the competition, we usually use a single fisheye camera mounted on the sensor head as this solution reduces complexity is sufficient to reliably detect barrier tape.

Fig. 2: Image of our robot Ωmeg4

Fig. 3: Robot floor

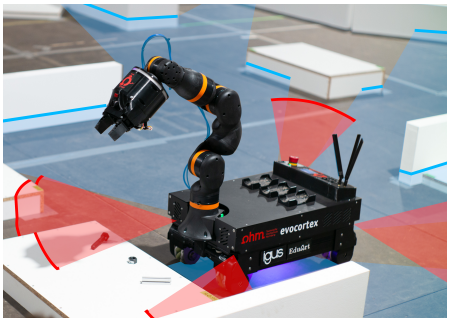


Fig. 4: Filling Lidar blind spots with Time-of-Flight sensors

3.2 Computer

The newly introduced neural networks require a GPU for computation onboard of the robot. As embedded GPU chips such as the Nvidia Jetson TX2 do not provide enough processing power for the task optimization and navigation algorithms, we designed a custom solution consisting of an AMD Ryzen 7 5700X

processor, a mini-ATX mainboard and a low power Nvidia A2000 graphics card. The GPU is connected to the mainboard with a riser cable which enabled us to build a flat case with both the mainboard and the graphics card safely mounted inside the robot. The form factor of the case makes it possible to slide it into the robot's back, similar to a server rack.

3.3 Power supply

We developed a custom PSU circuit board containing emergency switches for the actuators, a main power switch and voltage controllers for 5V, 12V and 24V rails. It is equipped with a custom designed plug system with selectable voltage, so every peripheral device can be connected using the same plug type. In addition to that, we use an additional DC-DC controller for the power supply of the manipulator, as its power consumption exceeds the limits of the onboard controllers. For the custom computer system, we use a standard 250W automotive ATX power supply.

3.4 Manipulator

Arm We use the 6-DoF Cobot ReBeL from igus as the arm for our robot (see fig. 2). We choose this arm because it is budget friendly and has no external control box. This saves space on the platform itself and reduces the total weight of the robot.

Sensor head Our sensor head concept is a plug-in solution for a variety of robotic arms. Our all-in-one design includes a customized parallel gripper with tilting fingers, a 3D camera for object detection, a fisheye camera for barrier tape detection, a status led ringlight and customized electronics. It can be connected to any robot with just one USB and one power connection.

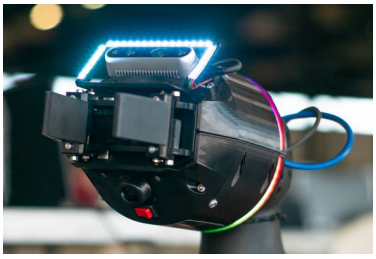


Fig. 5: Sensor head

Gripper Our customized parallel gripper is equipped with a guide rail for the fingers which allows for a circular tilting motion towards the maximum open angle, as well as a specially developed linear gear drive that converts the circular motion of a Dynamixel MX28 motor to a linear motion of the finger sleds. We use customized fin-ray fingers which are printed with TPU and lined with foam rubber to firmly grasp the objects. The force feedback from the fingers is transferred to the motor via the linear gears and can be used for grip detection and for calculating the grasping force.

4 Software Description

We use Linux Ubuntu 20.04 and ROS Noetic [4] as our operating systems. A custom software architecture was created to simplify the overall structure and to regain system flexibility. Our new design is displayed in fig. 6.

The idea derives from the Model-View-Controller software design pattern, which is adjusted to be used with the ROS framework. Regarding the frequent changes of hardware, an additional driver layer is added below the model layer. Models that need data from hardware, e.g. sensor data, can get them from the individual driver programs. The view layer is realized with each program using interfaces to RVIZ or simple console logging, which makes custom implementations obsolete. Components that require additional control features, such as the robot arm, have dedicated controllers providing simple interfaces for the brain layer, which is responsible for the actual task interpretation and execution. The individual layer components will be explained in the following sections.

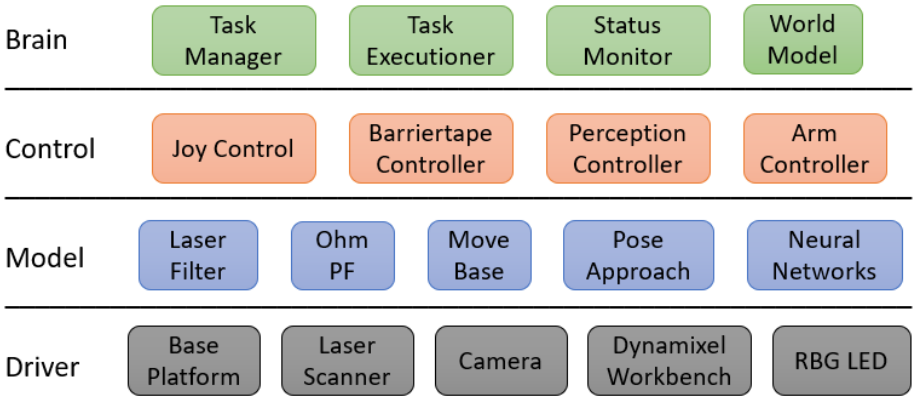


Fig. 6: Software Architecture - BCMD

4.1 Driver

The driver layer only contains actual hardware control programs, such as the sensor interfaces. The idea here is that the whole layer can be replaced with simulation tools such as Gazebo.

Base Platform The base platform driver converts incoming cmd_vel messages into wheel rpm and calculates the odometry from obtained rpm. It stops the robot automatically if the incoming commands time out to prevent uncontrolled movements. An additional twist_mux node throttles incoming commands from the joy_controller, move_base and the pose_approach.

Laser Scanner The rplidar node provides the interface to the scanner with given IP address and scan area configuration. However, as the Lidar is prone to measurement errors such as shadows or reflections, custom laser filters are applied to the raw data for later computation.

Time-of-Flight sensors The edu_sensorring driver [20] from our sponsor EduArt Robotik enables real time data-reading from the Time-of-Flight sensors via CAN FD. The measurements are published as a 3d point cloud.

Camera We use the Intel Realsense SDK with the provided ROS wrapper. The fisheye cameras are accessed via the ROS usb_cam package[21].

Dynamixel Workbench The parallel gripper is controlled with a controller instance of the dynamixel_workbench package. It provides access to motor variables (e.g. position, torque, ..) which are used for precise grip controls.

Igus Rebel We use the igus rebel [26] 6-DoF driver to control the robot arm. The driver follows a given trajectory and provides feedback about its position, velocity and accelerations during the trajectory execution.

4.2 Model

Our models contain all algorithms used to challenge the problems of the tasks in the @Work league. This includes localization, navigation and perception. The task planner is not included as a model but in the brain layer because it is more convenient to attach it directly to the task manager, as discussed in section 4.4.

Laser Filter As mentioned in section 4.1, we filter the raw laser data before computing. The first filters are simple area filters to delete the robot's wheels from the scan. The second filter is a custom jumping point filter implementation. We faced problems with reflections of the alu profile rails used for the walls of the arena, which caused the robot to mark free space as occupied. The filter calculates the x- and y-position for each scan point and checks if there are enough neighbors in close range to mark a point as valid. All points with less than n neighbors in the given range will be handled as measurement errors and are therefore deleted.

Ohm PF For localization in the arena, we use our own particle filter algorithm. Its functionality is close to amcl localization, as described in [5] and [13]. The algorithm is capable of using multiple laser scanners and an omnidirectional movement model. Due to the Monte Carlo filtering approach, the localization is robust and accurate enough to provide useful positioning data to the navigation system. The positioning error with the particle filter is about 6 cm, depending on the complexity and speed of the actual movement.

Time-of-Flight Merger To fill in the blind spots of the Lidar with the measurements from the Time-of-Flight sensors, we use a custom merger algorithm. First, the 3D points from the ToF sensors are filtered, which eliminates all points on the arena floor and all points with a high variance. Next, a 2D slice parallel to the Lidar plane is taken from the 3D ToF point cloud. This 2D slice can't be directly combined with the Lidar measurements, because Ros represents LaserScans as arrays of distance measurements with constant angle increments between the points. This is not the case for the ToF points and therefore an interpolation step is performed to generate points in the same polar coordinate system as the Lidar measurements. To further improve the interpolation, lines are detected and sampled in the 2D slice from the ToF measurements. The interpolated and sampled points are then finally combined with the Lidar data.

Move Base We use the ROS navigation stack [10] for global path planning and the local path control loops. Path cost calculations are performed by using the costmap 2D plugins. The base layer is a 2D laser map created with gmapping [11,12]. On top of that, we use a barrier tape map layer which contains all detected barrier tape points. For local obstacle avoidance, we added an obstacle layer which includes laser data from the lidar and the points from the Time-of-Flight sensors. All layers are combined in the final inflation layer. Global path planning is computed with the mcr_global_planner [17] while the path is executed using the TEB local planner [6,7,8,9]. As the local planner is not able to precisely navigate to a given goal pose, we set the goal tolerance relatively high. Once we reached our goal with move_base, we continue exact positioning with our custom controller, the pose_approach.

MoveIt We use the MoveIt [27] motion planning library to calculate inverse kinematics and trajectories for the robot arm and connect it to the igus driver.

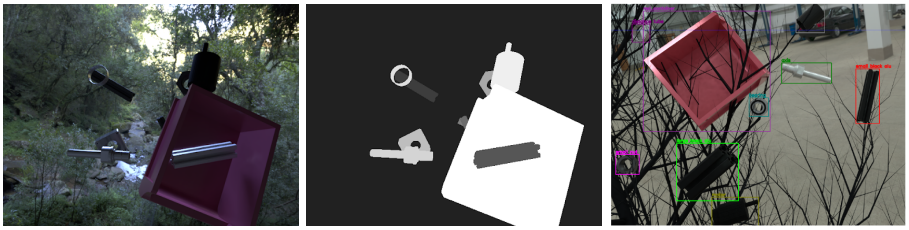
Pose Approach The pose_approach package utilizes a simple PID controller to move the robot to a given pose. It utilizes the robot's localization pose as input and the target pose as reference. As the controller does not consider costmap obstacles, the maximum distance to the target is 20 cm to prevent collisions. A laser monitor algorithm checks for obstacles in the current scan and stops the robot if necessary.

Fisheye rectification The raw fisheye images need to be rectified to be used as input for the detection network. A specific image_pipeline fork [22] is used, which contains this functionality.

NN - Barrier tape For the barrier tape detection, we use a U-Net with manually labelled datasets. The ROS node receives raw input images and returns a masked binary image. We have ported the network node from Python to C++ to increase the detection rate from around 5Hz up to 20Hz.

NN - Objects The detection and classification of objects is done with a fine-tuned yolo-v7 network [24]. The node receives a raw input image and returns a vector with the ID, bounding box and confidence of all objects that were found. As the creation of a dataset specifically for the competition requires more than 10,000 labelled images, we have implemented an automated dataset creation method using Blender and Python as an time-efficient alternative to manual creation. It basically changes environments, illumination, camera and object pose as well as object appearance in pre-defined bounds. The script creates rendered images as well as bounding boxes, segmentations and 6DoF labels. With this data generation method, data similar to the original scene can be created, as well as rather abstract data (Figure 7).

Using an original to artificial image ratio of 1:10, we achieved a detection reliability of over 90% for most scenes. Our data generation scripts are public and free to use [15]. The trained network is converted to TRT-Engine using code from the TRT-YOLO-App from the Deepstream Reference Apps [16]. This increases performance by making more efficient use of the CUDA cores and enables a detection rate of up to 60Hz. In the future, other network types such as segmentation networks and 6DoF networks will be explored.



(a) (b) (c)

Fig. 7: abstract image (a) corresponding mask label (b) abstract image with bounding box label (c)

As an alternative to detecting real objects we have also implemented AprilTag detection with a ROS wrapper of the AprilTag 3 visual fiducial detection algorithm [25]. It provides full access to the core AprilTag 3 algorithm's customisations and makes the tag detection image and the detected tags' poses available via ROS topics to detect objects via apriltags. Apriltag detection is a useful simplification for developing and debugging systems that don't involve real objects.

4.3 Controller

Model nodes that require additional control features are connected to control nodes, which then provide interfaces for the brain layer. They use our robot_custom_msgs interface to share information about the subtask, workstation, or

objects. Individual nodes may be created for specific subtask types for optimized behaviour.

Joy Control We use a PS5 joystick to move our robot manually (e.g. for mapping). For this, we have implemented a custom teleop_joy_node with similar functionality. We plan to implement the PS5 feedback functions such as rumble to enhance the usability.

Barrier tape Control The barrier tape controller is a custom mapping implementation for visual obstacles. It throttles the input images for the barrier tape network and computes masked images. Looping through multiple cameras enables us to perform 360°barrier tape detection.

Received masked images are converted into a point cloud with a predefined density. The point cloud is then transformed from the individual camera frame into the global map frame. Afterwards, all new points are compared to the existing points in the map. New barrier tape points that are already occupied are ignored to reduce computation. In response to problems with image blur and resulting inaccurate barrier tape detection, we also calculate pixels that mark free space (no barrier tape detected). The free space points are compared to existing points, which get deleted if they overlap. Finally, the detected points are included in the occupancy grid which is included in the costmap of the move_base node 4.2.

Arm Control The arm is controlled via ROS services or a development GUI for debugging. To calculate the inverse kinematics we use MoveIt in conjunction with a custom arm controller. The arm controller receives the target pose via service calls and checks whether the trajectory requires the arm to pass through pre-defined safe positions. MoveIt performs the calculations and passes the trajectories to the igus arm controller. In case of a pick task, it moves the TCP to the object position, closes the gripper, and places the object in the inventory. Upon completion, the success status is returned to the caller.

Perception Control The perception control node is responsible for the workstation analysis and object detection from the 3D point cloud and RGB image of a given scene. First, the surface of the workstation is estimated with a RANSAC [14] algorithm. If a valid result is obtained, raw images are sent to the object perception network (4.2). All detected objects are then localized using the pinhole camera model, the workstation surface and the bounding box pixels. Finally, the object poses are transformed into the workstation frame and saved in the world model. For moving objects, multiple positions are recorded and then used to estimate their movement with RANSAC.

4.4 Brain

The brain layer contains nodes that provide intelligence to the robot. This includes tracking the robot's state, the environment and the tasks.

World model The world model database stores all information about the robot's environment. This includes the map, all workstation positions and all detected objects on the workstations. The data can be accessed using service calls.

Status Monitor The status monitor keeps track of the current robot state, including the current pose and inventory. A color code associated with the current state is sent to the RGB LED driver node.

Task Manager The robot can receive tasks from multiple sources, such as the RefBox or voice commands. To handle different input formats, different parsers are used to standardise the input to the task manager. When the robot receives a new task, it is analysed and scheduled before execution it. All extracted subtasks are managed by the task_manager node, which optimizes the order of the subtasks. With increasing number of tasks, high efficiency is crucial to achieve perfect runs. The score of a single subtask is calculated considering expected duration, points, and the risk of failure. These factors may change depending on internal and external conditions. For example, the navigation time is set to zero if the robot already is at a given position.

Before a subtask is processed, it is checked to see if it can be successfully executed. A task is considered impossible if a target workstation is unknown or unreachable, or if an object is lost. Any subtasks that cannot be executed are moved to a deletion vector.

A custom planning algorithm calculates the raw score of the remaining subtask vector, followed by a simple nearest neighbour (NN) search. The result is then passed to a recursive tree solver, which further optimises the solution. A branch is only fully computed if the sum of the scores does not exceed the best solution found by the NN. In this way, we achieve a planning time of about 10s for a challenge with 14 subtasks. For challenges with less than 12 subtasks, the planning time is only 2s. When the number of subtasks exceeds 14, the recursive strategy is skipped because the planning time grows exponentially and cannot produce results fast enough for the time frame of a run.

After planning, the subtasks are sent to the task executioner (section 4.4). If the execution fails, the task is moved to a failed subtask vector and deleted from the current working subtask vector. The short planning time allows us to replan each time a subtask fails, or new data becomes available. This is necessary because even simple changes can seriously affect the initial plan. If certain paths are blocked, the navigation time for transport tasks can increase dramatically, resulting in a huge loss of efficiency. A final garbage collection checks all deleted and failed subtasks for plausibility and reschedules subtasks that might succeed on a second attempt.

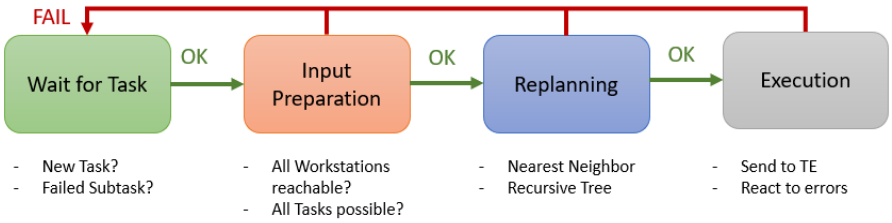


Fig. 8: Task Manager States

Task Executioner Subtasks that are sent to the Task Executioner are run through an interpreter to extract the actions required to complete the task. All actions are performed in custom states, which can be set via parameters at creation. The interpreter uses information from the status monitor, the world model and the given subtask to create substates accordingly. The resulting state vector is iterated until completion or failure. During execution, the node reads and modifies the data in the status monitor and world model packages to make changes immediately available to all other nodes.

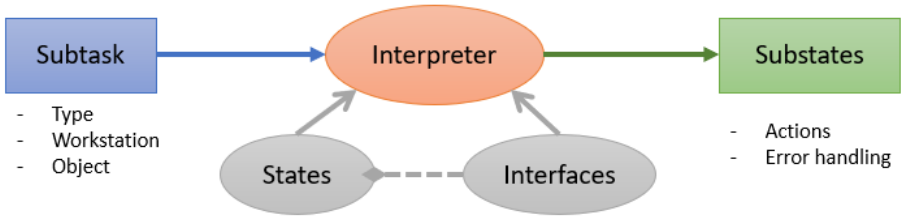


Fig. 9: Task Executioner - Subtask Interpretation

5 Conclusion and Future Work

After the 2022 season, we optimized our robot concept and built a completely new robot (Ωmeg4). The robot arm concept has been reworked and improved, a new operation interface and sensor concept has been implemented, as well as the step up at to 48V base voltage.

In the coming season we plan on improving our object detection by using other network architectures such as segmentation, 6DoF and grasp detection networks. We are also reworking our gripper concept to enable the use of force feedback, which may benefit our task management system. We also want to introduce performance monitoring, allowing us to identify bottlenecks and plan future improvements. Finally, we plan to extend the above-mentioned repository by adding more documentation as well as some theses from our students.

We are very much looking forward to the upcoming season, where we aim to benchmark our systems performance at the RoboCup 2025 in Salvador.

References

1. RoboCup Atwork Website, https://atwork.robocup.org/. Last accessed 20 Jan 2021
2. A. Norouzi, B. Schnieders, S. Zug, J. Martin, D. Nair, C. Steup, G. Kraetzschmar: RoboCup@Work 2019 - Rulebook, https://atwork.robocup.org/rules/, 2019
3. EvoCortex Homepage, https://evocortex.org/. Last accessed 20 Jan 2021
4. Stanford Artificial Intelligence Laboratory et al., 2018. Robotic Operating System, Available at: https://www.ros.org.
5. F. Dellaert, D. Fox, W. Burgard and S. Thrun, "Monte Carlo localization for mobile robots," Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), Detroit, MI, USA, 1999, pp. 1322-1328 vol.2, doi: 10.1109/ROBOT.1999.772544.
6. C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram: Trajectory modification considering dynamic constraints of autonomous robots. Proc. 7th German Conference on Robotics, Germany, Munich, 2012, pp 74–79.
7. C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann and T. Bertram: Efficient trajectory optimization using a sparse model. Proc. IEEE European Conference on Mobile Robots, Spain, Barcelona, 2013, pp. 138–143
8. C. Rösmann, F. Hoffmann and T. Bertram: Integrated online trajectory planning and optimization in distinctive topologies, Robotics and Autonomous Systems, Vol. 88, 2017, pp. 142–153
9. C. Rösmann, F. Hoffmann and T. Bertram: Planning of Multiple Robot Trajectories in Distinctive Topologies, Proc. IEEE European Conference on Mobile Robots, UK, Lincoln, Sept. 2015
10. ROS navigation, http://wiki.ros.org/navigation. Last accessed 20 Jan 2021
11. Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters, IEEE Transactions on Robotics, Volume 23, pages 34-46, 2007
12. Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard: Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling, In Proc. of the IEEE International Conference on Robotics and Automation (ICRA), 2005
13. S. Thrun, W. Burgard, D. Fox: Probabilistic Robotics, Massachusetts Institute of Technology (2006)
14. Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM 24, 6 (June 1981), 381–395. DOI:https://doi.org/10.1145/358669.358692
15. GitHub: DataGeneration, https://github.com/ItsMeTheBee/DataGeneration. Last accessed 22 Jan 2021
16. GitHub: NVIDIA deepstream reference apps, https://github.com/NVIDIA-AI-IOT/deepstream_reference_apps. Last accessed 20 Dec 2019
17. GitHub: bitbots mas_navigation, https://github.com/b-it-bots/mas_navigation. Last accessed 20 Jan 2021
18. Alphacei Vosk, https://alphacephei.com/vosk/. Last accessed 20 Jan 2021
19. D. Dadswell, "E05: Pick & Pack," SciRoc, 17-May-2021, https://sciroc.org/e05-pick-pack/. Last accessed 13 Jan 2022
20. GitHub: edu_lib_sensorring, https://github.com/EduArt-Robotik/edu_lib_sensorring. Last accessed 19 Feb 2025

21. GitHub: usb_cam, https://github.com/ros-drivers/usb_cam. Last accessed 13 Jan 2022
22. GitHub: image_pipeline, https://github.com/DavidTorresOcana/image_pipeline. Last accessed 13 Jan 2022
23. GitHub:docs_atwork, https://github.com/autonohm/docs_atwork. Last accessed 20 Jan 2022
24. GitHub: WongKinYiu, https://github.com/WongKinYiu/yolov7. Last accessed 2 Mar 2024
25. AprilTagROS, https://wiki.ros.org/apriltag_ros. Last accessed 2 Mar 2024
26. bitbucket: igus rebel, https://bitbucket.org/truphysics/igus_rebel/src/master/. Last accessed 2 Mar 2024
27. MoveIt: motion planning library, https://moveit.ros.org/. Last accessed 2 Mar 2024