The ER-Force RoboCup Logistics League Team 2018

Michael Scholz[1], Julian Sessner[1], Florian Eith[1], Maximilian Zwingel[1], Stefan Merbele[1], Lennarth Gruendel[1], Valentin Garbe[1], Sebastian Reitelshoefer[1] and Joerg Franke[1]

[1] Institute for Factory Automation and Production Systems (FAPS), Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany

Abstract. The ER-Force Logistics League team was founded and participated the first time at RoboCup Logistics League Worldcup in 2016. The paper gives an overview over the used hardware and software. The hardware contains the robots and their components like sensors and grippers. The software setup is based on open source software packages, mainly from the Robot Operating System, with additional self-programmed software packages for special tasks required in the RoboCup Logistics League.

1 Introduction

The ER-Force team is part of Erlangen Robotics, a student-run initiative founded in 2007 to participate at the RoboCup Soccer SmallSize League with several remarkable successes in the past years. In 2016 a second team was founded to participate at the RoboCup LogisticsLeague (RCLL). The founding was initiated by the Institute for Factory Automation and Production Systems (FAPS) of the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU). The FAPS Institute supports the team until now and provides the infrastructure for the playing field. The ER-Force RCLL team consists of doctoral, master and bachelor mechanical engineering and mechatronic students.

The following paper describes the basic gameplay and the overall concept of the RCLL, followed by a description of the hardware setup and the software structure and its components.

2 RoboCup Logistics League

The RoboCup is a robot competition for international students and scientists initiated in 1997. Initially the robots were built and programmed to play football. To stress the propagation of robotics in our everyday life, several other competitions were created during the last years. To represent industrial robot applications, the RobotCup Industrial with its two leagues Logistics and @Work was established. Whereas teams in the RoboCup@Work League have to handle several different work pieces to simulate industrial production scenarios, the RCLL's focus is on intra-logistic challenges and the robots have to handle standardized work pieces. The standardized systems used in

the RCLL are Robotino robots and Modular Production Stations (MPS) by Festo Didactic.
During gameplay two teams share one playing field. Each team is allowed to play with three robots. The MPSs are placed randomly on the field, whereas each team has its own set of MPSs. As a third component the Referee Box (refbox) provides each team the current information about the game phase and especially the orders of the products, which need to be produced by the team's robots. In the first phase of the gameplay, the exploration phase, the robots have to find the team's MPSs in the field and report their positions to the refbox. During the second phase, the production phase, the robots have to to produce the ordered work pieces using the MPSs. The work pieces are simple cylindrical parts, composed of a base element, ring elements and a cap element. The complexity of the products can be varied by using zero to three ring elements. During production phase the robots are used to transport the products to different MPSs. The actual production is done by the MPS. Each team has one Base Station (BS), which provides the base elements to the robots. The rings are placed onto the base element by the Ring Station (RS). The Cap Station (CS) puts the final cap on top of the product. Each team has two RS and two CS. Finally, the product needs to be transported to the Delivery Station (DS). Since 2017 a fifth station, the Storage Station (SS), has been introduced. The SS allows to store several products.

3 Hardware Setup

The ER-Force RCLL team has three Robotino v3 robots equipped with an additional computer, a gripper and several sensors. The Robotino robot's main components are the omnidirectional drive, which enables the robot to move, and the integrated computer. The additional laptop is used on the one hand to program the robot and on the other hand provides additional computing resources. To capture the environment several sensors are used. The Robotino is equipped with a ring of one-dimensional infrared sensors at its base and a Logitech HD Pro Webcam C920 mounted at the tower. For a better panoramic view and better exploration results two more cameras are mounted towards the rear sector. The main sensor for localization are the incremental encoders in the robots' drive system, which provide odometry information and a LIDAR-Sensor. In our setup we use the low-cost LIDAR-Sensor RPLidar A2 by Slamtec [1]. It has got a range between 8 and 16 m, depending on the scanning frequency, which is up to 4000 samples per second. The field of view is 360 degrees. The laser scan data is used for localization, obstacle detection and machine alignment described later. Finally the robots are equipped with self-built grippers, described in the following.

3.1 Gripper

To interact with products during the production phase a new gripping concept is designed. It improves the last years` design in terms of fail-safe-behavior and energy consumption. A rendering of the current gripper model is shown in Figure 1.

The current model consists of six parts, which are mostly additively manufactured with plastic filament. The main part of the gripper is the baseplate, pictured in light grey. The basic element (1) is shaped as a guide rail for products during loading and unloading. During movement of the Robotino the guide rails keeps the product safe in place without any additional energy needed. To load and unload the gripper the rail (2) and cantilever (3) are used. With the help of two stepper motors (4) it is possible to tilt the rail up and down and move the slide forwards and backwards.

To pick a product the rail is tilted up and the cantilever is moved forward. Being positioned over the product the rail is moved down into its original position and the cantilever is moved backwards. To unload the product this procedure is repeated in reverse.

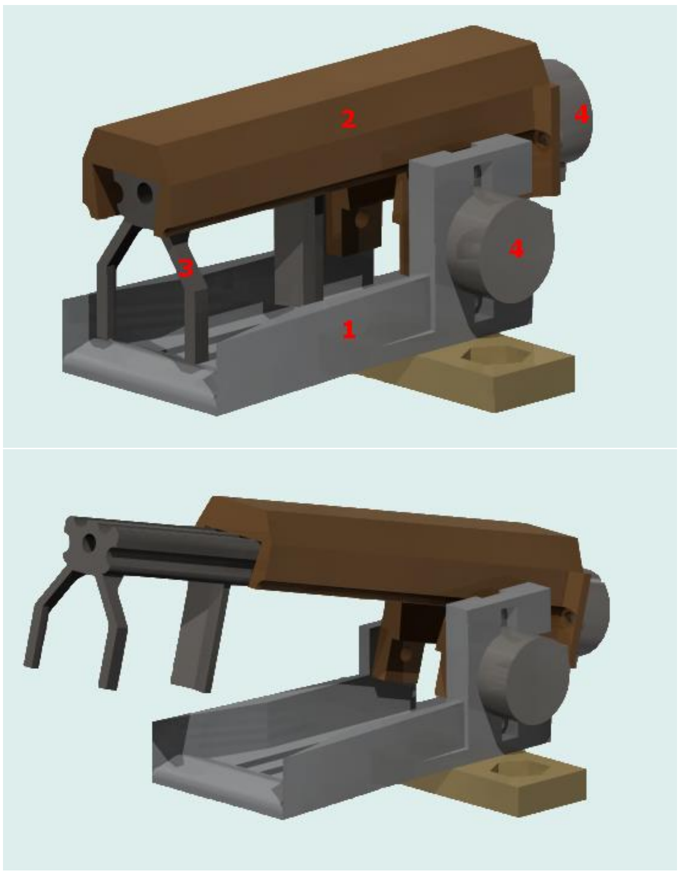

Fig. 1. Gripper Rendering: Basic element with guide rail (1), rail (2), cantilever (3), stepper motors (4)

The height of the gripping system can be controlled manually by two leadscrews, which connect the gripper system to the tower of the Robotino. To control the gripper, an Arduino Nano is used. It is programmed to communicate with the superior software and handle all loading and unloading movements. Further it provides visual feedback by controlling some LEDs, representing states of the gripper and the entire robot system.

4 Software Setup

4.1 Software Infrastructure

The entire software runs distributed on the Robotino itself, the additional laptop and an automated server on the game field side. The Robotinos and the laptops use Ubuntu 14.04. Our software is mainly based on the Robot Operating System (ROS), where the Robotinos and the Laptops use the distribution Indigo Igloo. For image processing we use OpenCV 2.8. The algorithms for the machine detection are implemented in MATLAB and exported as standalone ROS node. Most of the code developed by the ER-Force team is written in C++.

4.2 General Software Structure

According to the ROS philosophy [2] of separated, task specific packages our system contains various packages from the ROS Community as well as packages developed by the ER-Force Team itself. The packages can be divided into four categories, which will be described in the following. With increasing category, the focus on the Logistics League for a package increases and the reusability for other applications of the packages decreases. Most of the described nodes are running together parallel on the Robotino, the laptop and the remote server.

4.3 Category 1: Driver Level

The driver level is the lowest level in our software hierarchy. It's the software working directly with the hardware and has the highest grade of reusability for other projects. Because of that most of the packages were not developed by the ER-Force team but are general purpose packages publicized by the ROS Community. The following packages belong to the driver level:

- robotino_node package: General package to interact with the Robotino via ROS. It was developed by the Robotics Equipment Corporation GmbH [3] and is available at github [4].
- usb-cam package: General package to get the camera picture of an usb camera into the ROS world. It is available at github [5].
- rplidar-ros package: General package to get the laser scan from laser scanners developed by slamtec [1] into the ROS world, available at github [6].
- Gripper_ros package: A package developed by the ER-Force Team. The gripper is controlled by a combination of a microcontroller and a ROS node. The node enables the state machine to interact with the gripper and controls the display of status information of the robot system with built in LEDs of the gripper. To start an ac-

tion of the gripper a ROS service called. Depending on the parameters of the call, the gripper system has to open, close or send its status. This information is forwarded by a serial connection to an Arduino Nano, which controls the actual functions of the gripper.
The connection between the ROS node and the Arduino Nano is constantly checked with a ping signal. The current connection status is shown with LEDs on the gripper. For the movement the Arduino Nano controls the two motors with an ULN2003 stepper motor driver running with a 5V power bank.
In the case the sensors of the Arduino detect a faulty condition, either with the microswitches for motor movement or the photoresistor to detect a gripped part, an error code is sent to the ROS node. The node then forwards this error code to the state machine to trigger fallback behaviors.

4.4 Category 2: Processing Level

- navigation metapackage: General package to localize and move autonomous robots. It was developed by the ROS Community [7]. For the localization of the Robotino on the game field we use the gmapping subpackage during the exploration phase and the amcl subpackage during the production phase. For path planning tasks we use the move_base package.
- ar_track_alvar package: Package to detect the alvar Markers in the camera view topic published by the usb-cam package. It was developed by the ROS Community [8]. The published marker data is further processed in the Skill level.
- machine_side_detection: This package detects a machine side with the laser scanner. It was developed by the ER-Force team. The goal is to get the robot in the right position in front of the target machine to deliver or take a product. This is an alternative solution for the machine detection with the alvar markers, which identifies the markers clearly, but is error-prone when the Robotino is too close to the machine. For the localization of the machine with the laser scanner two nodes were developed. The first node scan_to_image processes the laser scan from a standardized ROS laser scan message (alike a point cloud) into a local map image. The second node machine_detection detects the machine side in the image with the Probabilistic Hough Line Transform function of OpenCV [9] and publishes the relative pose for the alignment procedure.
- multimaster_fkie: General package to synchronize data between separated ROS systems (e.g. robots). It was developed by the Fraunhofer FKIE [10]. It is used for sharing data between the different robots and the remote server, as well as observing the robots remotely from the game field border. The package also makes an external visualization of the game possible. The multimaster_fkie package is available at GitHub [11].
- machine_into_map: The machine_into_map is a package developed by the ER-Force Team for the Logistics League, which draws the recognized machines into the global obstacle map. It also provides a node to draw the empty obstacle map based on the game field parameters.

- joy: The joy package was developed by the ROS community to get the joystick inputs of a common game controller and publish them in a message [12]. This is used to move the Robotino directly, e.g. for initial positioning and travelling to the game field.
- modified_map_server: Modified map_server package from the navigation meta-package to update dynamic changes of the obstacle map (which is not provided in the original package).
- cmd_vel_fusion: The cmd_vel_fusion package is a package developed by the ER-Force Team for the Logistic League. It fusions different cmd_vel_* topics sent by different nodes with different priorizations into one resulting cmd_vel topic to move the robot.
- cpp_redis: The redis database is the key value store for the dynamic parameters. It is one major change since 2017, where the ROS Parameter Server turned out to be overcharged. Redis is available at [13], whereas the cpp_redis package, which gives access to the database via C++, is available at [14].

4.5 Category 3: Skill Level

- llerf2_lib package: The Logistics League ER-Force (LLERF) Library was refined and provides all constants, general functions, message types, static parameters and dynamic parameters related to the Logistics League.
- llerf_basic package: The llerf_basic package replaces the old robotino_service package. It takes the preprocessed data and offers different services the robot can execute. The different nodes in the package communicate with the processing level via topics, among themselves mainly via the Redis database and with the behavior level via services. They can be divided into seven groups, which will be explained in the following:

Group 1: General Purpose
The group 'General Purpose' contains nodes which are the services’ backbone. The most important node is the refbox_communicator node, consisting of an active and passive version. It receives the messages sent by the refbox and converts them into LLERF consistent data and vice versa. The gotopose, odom_move and unsafe_move nodes provide different services to move the robot in different ways. The movement can be controlled by the ROS move_base node or from the nodes directly. The align_alvar and align_machine nodes align the robot in front of a MPS machine using the alvar Markers as reference or the machine side, detected with the laserscanner. The align_machine node requires the specification of a so-called machine interface (conveyor input or output, shelf or slide). The mps_tf_broadcaster node broadcasts the static machine geometry as tf2-tree for the alignment nodes (tf is a ROS package to organize different coordinate systems). Furthermore, there is the robot_localization node to localize the robots pose within the map depending on seen alvar markers in combination with the known machine poses during the production phase.

Group 2: Synchronization
The group 'Synchronization' contains nodes for the exchange of information among the robots and the remote server.

Group 3: Exploration
The group 'Exploration' contains three nodes, the scan_alvar node, the database node and the machine_pose_analyser. The scan_alvar node subscribes the published topic from the ar_track_alvar package, filters the recognized markers, gets the corresponding machine poses and publishes these. The database node subscribes the published data from the scan_alvar node and writes the machines data to the parameter server. Additionally, it mirrors the machine to get the corresponding machine pose from the other side of the game field.
The machine poses are continuously collected from all robots and evaluated by the machine_pose_analyser node at the remote server regarding consistent zones and orientations. At the end of the exploration phase the machine zones and orientations with a high confidence are reported to the refbox.

Group 4: Production
The group 'Production' contains nodes for the production phase. The transport_generation node converts each order sent by the refbox into multiple products and then into transports. These transport orders are then distributed by the transport_distribution node to the robots on request. For a distribution several requirements are necessary, like the machine must not be blocked by another robot or the target conveyor is free. Currently the transports are distributed in a "first comes - first served" manner to the next requesting robot, where just the absolute restrictions concerning blocking are obeyed. Other ways of distribution are planned to be implemented in the future. The third node is the material_manager node, which observes the filling statuses of the base- and ring-color magazines and informs the replenisher (via the observation group) if one runs empty.

Group 5: Observation
The group 'Observation' contains four nodes, the field_visualizer node, the robot_visualizer node, the rviz_visualizer node and a logger node. The first node field_visualizer represents the current state of the game field basically in the exploration phase in a textual user interface. The robot_visualizer node provides also a textual user interface, which shows all information (basically about the production phase) of the entire system, like current products, robot status, machine status and transport statuses, etc. The node rviz_visualizer converts all collected data (machine position, robot position, machine signal, etc.) into 3D marker data, which can be displayed by the ROS 3D visualizer tool Rviz. The logger node records the outprints of all nodes for later analysis. All three nodes can be used on the robot itself or from an external observation computer beside the field (in combination with the multimaster_fkie package).

Group 6: Testing
The group 'Testing' provides some minor nodes to test specific services of the other groups. To mention is simulated_machine_state node, which replaces the physical machine interaction from the robots by triggering the different actions with the tools 'rcll-set-machine-state' and 'rcll-machine-add-base' provided by the refbox. Furthermore, there is the fake_robot node, which imitates the physical robot regarding the movement.

- markerless_machine_recognition package: Package to recognize the different machine types without markers for the “Markerless Machine Recognition and Production” challenge. To detect and classify the different MPS stations, a convolutional neural network for image classification is used. The AlexNet network architecture was chosen due to memory friendly performance. Pretrained on the ImageNet dataset, we retrained it within MATLAB and exported the code to interact with ROS. The retraining dataset consists of a mix of real-life images of the MPS stations, which we extracted from video files, and rendered images of CAD models. We used Blender to model the different machines, texturizing and rendering them. Currently we achieve 90% accuracy in recognition. To further improve prediction accuracy in the game, the robot performs a movement pattern, which gives different angled views on the MPS station and thus ensures higher certainty in classification.

4.6 Category 4: Behavior Level

The control algorithm of the exploration phase has been included to the llerf_basic package described before. Group 7 contains the state machines for exploration and production phase. Since 2017 we replaced the old state machine, based on the ROS package Smach written in python, by a new one written in C++ now containing also the handling of errors during the game.

5 Rulebook Requirement Confirmation

We confirm that our robot and monitoring systems will satisfy the requirements given in the RoboCup Logistics League Rulebook 2018.

References

[1] Slamtec. Available online at https://www.slamtec.com/en/, checked on 5/6/2018.

[2] ROS/Introduction - ROS Wiki. Available online at http://wiki.ros.org/ROS/Introduction, checked on 5/6/2018.

[3] REC - Robotics Equipment Corporation: Robotino®. REC · Robotics Equipment Corporation. Available online at http://servicerobotics.eu/robotino/, checked on 5/6/2018.

[4] robotino-ros-pkg package - github. Available online at https://github.com/raultron/robotino-ros-pkg, checked on 5/6/2018.

[5] usb_cam package - github. Available online at https://github.com/bosch-ros-pkg/usb_cam, checked on 5/6/2018.

[6] rplidar_ros - github. Available online at https://github.com/robopeak/rplidar_ros, checked on 5/6/2018.

[7] navigation package - ROS Wiki. Available online at http://wiki.ros.org/navigation, checked on 5/6/2018.

[8] ar_track_alvar package - ROS Wiki. Available online at http://wiki.ros.org/ar_track_alvar, checked on 5/6/2018.

[9] Hough Lines Probabilistic - OpenCV 2.4. Available online at http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#houghlinesp, checked on 5/6/2018.

[10] Fraunhofer FKIE. Available online at https://www.fkie.fraunhofer.de/, checked on 5/6/2018.

[11] fkie/multimaster_fkie. Available online at https://github.com/fkie/multimaster_fkie, checked on 5/6/2018.

[12] joy package - ROS Wiki. Available online at http://wiki.ros.org/joy, checked on 5/6/2018.

[13] Redis. Available online at www.redis.io, checked on 5/6/2018.

[14] cpp_redis - github. Available online at https://github.com/Cylix/cpp_redis, checked on 5/6/2018.