luhbots Soccer
Team Description for RoboCup 2023

Larissa Seegemann, Fabrice Zeug, Patrick Ebbighausen, Lukas Waldhoff, Max Westermann, Sebastian Knackstedt, Max Känner, Tim Füchsel, Robert Hart, and Tobias Pahl

Institute of Automatic Control
Gottfried Wilhelm Leibniz University Hannover
Appelstr. 11, 30167 Hanover, Germany
soccer@luhbots.de
https://luhbots-hannover.de

Abstract. luhbots has placed third in the RoboCup 2022 Small Size League Division B. Able to establish ourselves in this division as a first-time contender, we present this year's Team Description Paper (TDP). The focus lies mainly on the software setup exclusively with C++ instead of ROS and our novel strategy structure including artificial intelligence (AI) components.

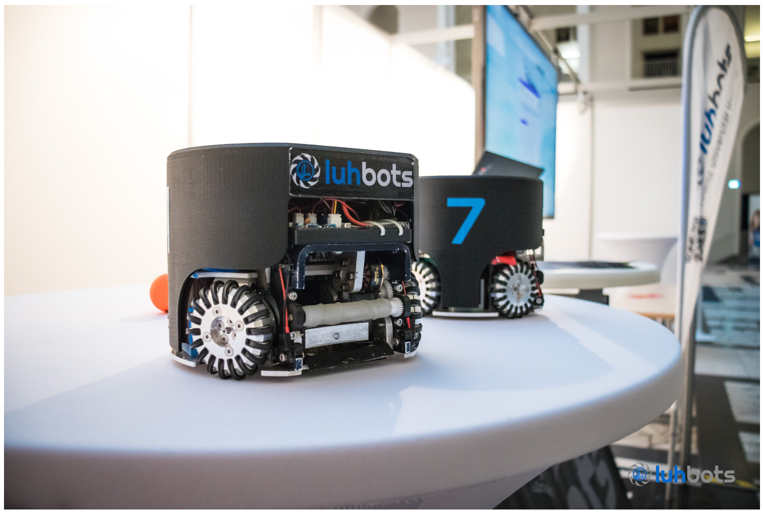

Fig. 1: Two robots of our first generation

1 Introduction

Having opened its Small Size League (SSL) branch in 2019, the luhbots Soccer team grew in recent years to an astounding amount of 34 active members. For the first time ever we are able to offer six different fields of activity - mechanics, electronics, firmware, software, strategy and organisation. Our team is comprised of students stemming from a variety of degree programs, such as, not only engineering, but also economics, physics, etc.. With the help of the new as well as the long-standing members, we look forward to the RoboCup 2023 in Bordeaux where we can present our recent innovations.

We, the student robotics team of the Leibniz University Hannover, were able to put robot generation 1 (see Fig. 1) to the test at the RoboCup 2022 in Thailand. The robots proved to be highly reliable in their hardware and electronics which are presented in detail in last year's paper [1]. As we owe our success to the functional setup, we decided to make minor improvements to the current fleet of robots for the upcoming RoboCup 2023, instead of building an entirely new generation.
Comparably, the software is undergoing the largest change, as we switch from the Robot Operating System (ROS) to a solely C++ based approach. Our strategy is also being revised. The non-adaptive decision making process is being partially replaced by AI components.

The section 2 describes the new dribbler design offering a more exact and space-saving structure with better torque transmission. The upgraded mainboard design is discussed in section 3 for which the firmware is described in section 4. The reworked basestation is also discussed in the firmware section.
The section 5 outlines our entirely revised software development process and repository setup. The reworked strategy is described in detail in section 6.

2 Mechanics

The robot's structure has changed slightly compared to the previous year. The base plate was adjusted to obtain more space inside the robot's body. Furthermore, the dribbler system offers significant new features which are presented in this chapter. The base plate changes are not covered in this paper, as they do not affect the robot's performance.

2.1 Dribbler

Previously the focus lied on two main principles, simple production and an inexpensive final product. The V1 dribbler body is built from a single piece of bent aluminium and a drone motor drives the roller (see Fig. 2a). The drone motor is the Black Bird V2 Freestyle Motor from T-Motor [2] and it's details can be derived from Table 1.
The disadvantage of the V1 dribbler design is that it presents an unwieldy structure. This is due to its single-sheet setup and the large diameter of the drone motor. The prior brings with it, due to the bending process, a low production accuracy. Additionally, the injection-moulded polyacetal gears in combination with the drone motor cannot transmit sufficient torque onto the ball, because the gear's material leads to poor tolerances. Furthermore, the ball centering with the roller's shape only engages for accommodating conditions where the relative speed between ball and robot is slow and linear. The dampening with the help of a thin layer of soft foam behind the dribbler frame leads to vibrations and bouncing of the ball out of the dribbler. This causes problems for the reliability of the light barrier readings. Therefore we conclude that the ball is not satisfyingly manipulable with the old dribbler drive.

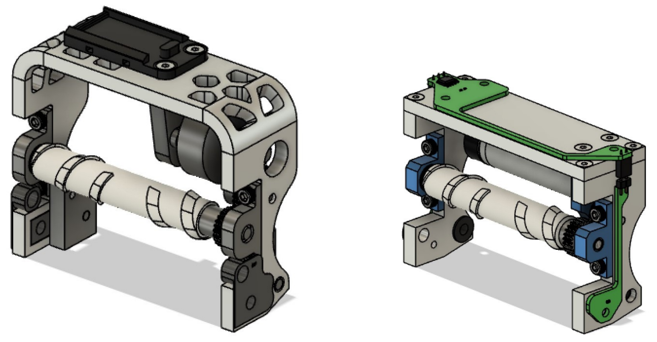
(a) V1 (RoboCup 2022) (b) V2 (RoboCup 2023)

Fig. 2: Dribbler versions

With the aim to improve the points mentioned above, an overhaul of the entire dribbler is required. The revised dribbler V2 will be integrated into the robots for the upcoming tournament. First of all, a new drive motor is chosen, namely the ECX SPEED 16 from maxon [3] whose configuration can be derived from Table 1. The ECX SPEED motor can spin significantly faster than the Black Bird drone motor and the integrated Hall sensors enable precise speed control.
The plastic gears are replaced with full metal gears, two of which are made out of steel and one out of brass placed in between. The metal material offers higher workability, smaller tolerance and improved handling of torque. The gear ratio is adapted to the new motor increasing the torque while limiting the rotation speed to around 20000 rpm (see. Table 1).
In order to increase manufacturing accuracy, the dribbler body V2 consists of three separate elements which are screwed together (see Fig. 2b). Smaller tolerances are achieved by milling flat the functional surfaces of the laser-cut parts. Other teams such as the TIGERs Mannheim in [4] or ER-Force in [5] show their implemented dribblers consisting of separate parts as well. This inspired us to do the same.
The self built roller bearing remains the same, as it allows to quickly replace the roller. It should be considered, however, to make them out of aluminium for greater stability and durability, instead of plastic. The bearing can be reviewed in detail in [1].
Figure 2 as well as Table 1 show the current dribbler model V2 with all the adjustments mentioned compared to last year's version V1.
Further improvements will be made to the roller shape to achieve a more reliable centering and to the dampening system to get a smoother ball manipulation. These are, however, not discussed in this paper.

Table 1: Dribbler comparison V1 from RoboCup 2022 and V2 for RoboCup 2023

	V1	V2
Body setup	single piece	three parts
Motor model	T-Motor Black Bird V2 Freestyle Motor 1950KV	ECXSP16L BL KL A STD 24V
Gear material	plastic	steel and brass metals
Gear ratio	$\frac{24}{19}$	$\frac{26}{15}$
Roller speed	≈ 10000 rpm	≈ 20000 rpm

3 Electronics

The electronics department aims to overhaul the entire current electronic structure for the RoboCup 2024. However, for this year's RoboCup 2023 minor improvements should suffice, as the overall setup proved to operate reliably during last year's tournament.

3.1 Mainboard

The mainboard is one of the major revised electric components. The previous version, presented in detail in [1], was not usable for the tournament. The microcontroller was not reliable enough, because it sporadically sent faulty bytes over SPI leading to the ESCs receiving incorrect commands. This was possibly due to faulty hardware components. The worst case scenario lead to a destruction of the motordrivers when certain safety checks were disabled. Instead a provisional setup consisting of an ESP32, a breakout board, regular buck converters and a NRF24L01 module was used. The NRF24L01 module had issues communicating in the RoboCup 2022. For this reason a new mainboard is designed for 2023. The new design features two RP2040 micro controllers. One is used for controlling the motors and the kicker. The other RP2040 controls the dribbler, the power switch and the communication with the server. An IMU is placed on the mainboard so that the robot is able determine its position more accurately.

The connection to the server is cause for most issues regarding the provisional mainboard. The server is able to send packets to the robots with a significant information loss, but the robots are unable to send status information back. Based on this a new RF transceiver is chosen. The SX1280 allows to optimize a larger variety of settings. Additionally the SKY66112 amplifier is used to boost the signal. The TIGERs Mannheim [6] used the same transceiver and amplifier combination in Bangkok [4] and their problems regarding any connection issues where mitigated.

3.2 Basestation

The basestation is adapted to use the new RF transceiver and amplifier. TIGERs Mannheim connect their basestation over ethernet [6] which leads to faster response times. We adopt this technique in the redesigned basestation. This allows the basestation to receive referee commands and vision data directly with which it can respond quicker. The basestation is a self built Raspberry Pi HAT and can be used independently without a server.

3.3 Dribbler

As the Dribbler hardware is redesigned, the dribbler electronics follow suit. The main difference is the integration with the hardware. The ball detection light barrier is built from three pcbs which are connected at their respective edges and mounted onto the dribbler body, described in section 2. This reduces the amount of cabling that needs to be done which is less error-prone.

3.4 Kicker

The kicker design used in Bangkok proved successful. It allows reliable control of the kicking strength. However, sometimes the mechanism that retracts the plunger, which consists of elastic bands, fails. To increase reliability, the elastic bands are replaced with weak springs for the RoboCup 2023 in Bordeaux France. The kick strength is controlled via the pulse width at the IGBT gate, instead of adjusting the capacitor voltage, as described in last year's TDP [1].

In the medium term, we plan to further increase the safety of our kicker by lowering the voltage below 120VDC.

4 Firmware

With the change of the microcontroller the firmware is rewritten entirely. In foresight of possible future microcontroller upgrades, the firmware is coded in Rust [7] which would allow the code to be updated accordingly, instead of completely rewritten. The firmware uses the embedded-hal crate [8] to allow for reuse of code across multiple architectures.

4.1 Mainboard

As outlined in section 3 the new mainboard contains two microcontrollers. This brings with it challenges concerning the interaction between them. A UART connection with hardware flow control was decided upon. Two methods of communication over the UART are implemented. One is connection based and the other connection-less. Both use the same type of packet which consists of a start byte, the data, a 16-bit CRC, and a stop byte. To ensure the data or CRC does not contain start or stop bytes, they are cobs encoded [9]. The connection based communication uses an acknowledgement and timeout mechanism with one packet in flight. For the dribbler motor we switched from a timing based protocol to a digital protocol called Dshot. Dshot allows a better control of the motor. Additionally the protocol enables the ESC to send telemetry, such as the current motor speed, back to the microcontroller [10]. Also, with this the mainboard can set a specific dribbler speed instead of setting just the torque.

4.2 Basestation

For the basestation several new features are planned. Alternatively to the last version it communicates via Ethernet, instead of USB.This change is inspired by Mannheim TIGERs' open source firmware design which is reviewable in [11]. This makes communication easier, because we do not need to use the serial-over-usb function anymore. The Ethernet connection also allows to listen to ssl-vision packets and send the vision information directly to the robots. More information about this decision can be found in section 5.2. On the basestation side this works by listing to both our own control packets and ssl-vision packets and merges them together. The basestation also contains RGB-Leds to allow fast visual indication of robot status. This should help to indicate problem sources more easily, for example a loss of connection can now be seen by simply looking at it.

5 Software

The previous software setup appears to be the major drawback for our team during the RoboCup 2022 in Bangkok. The design with ROS causes a large overhead in general and is cumbersome to code for our desired application. The strategy development with python leads to a software crash during game states which are not anticipated in the code. Also the software repository grew nearly unsupervised over the last three years which resulted in disarray. For these reasons the software for the RoboCup 2023 is completely overhauled. No code is transferred from the previous version.
Our experience from the past years allows us to have sufficient understanding of what our software should be able to do and how it should be structured. It is now developed with a modular structure in mind. This makes the code cleaner and generally easier for new members to comprehend. Additionally each commit in the team is signed off by another team member.

5.1 Replacement of ROS Modules

The ROS based approach to our code is rejected for the new software structure. This is due to the system not being sufficiently suitable for our application. ROS' node based approach makes it difficult to debug code, as nodes run independently from each other. Also the main application for ROS is running on the robot it controls. In our setup we need a software which can run without having ROS' system dependencies installed.
Removing ROS from our software resulted in a complete overhaul of our software, since it relied heavily on some of the ROS modules. New concepts for these models need to be implemented and are presented below.

The Visualization was previously implemented with a ROS 3D visualization tool called RViz [12] to be able to have 3D visualizations of the robots and game data. RViz offers a way to have a GUI representation for displaying internal states of our software and the ability to take control over execution and parametrization of different tasks. This is mandatory for testing and adjusting robot behaviour without the need to write code. Moving away from ROS, the new solution needs to offer said capabilities.
The new visualization is called Luhviz and is also written in C++, so that it is better integrated into the rest of our software. The Dear ImGui GUI [13] library is used for creating the basic window and interaction widgets like buttons. Moreover a render view is provided which loads OpenGL code to achieve a 3D renderer inside the GUI. While RViz could only be extended with the use of plugins, writing our own visualization allows for a finer control and a more fitted solution for our own custom software. In the beginning it was tedious to recreate certain parts of RViz, but now it is much easier to include new features and adapt the appearance and practicality to our team's desires. An extract of our visualization can be seen in Figure 3.

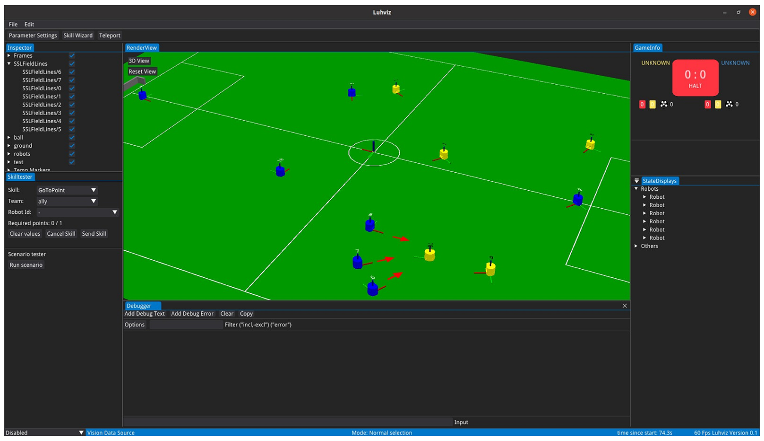

Fig. 3: The new Luhviz GUI

To allow other modules to display objects in Luhviz, an interface is developed called MarkerService which works similar to the markers in ROS. There are different types that can be created by other modules and displayed via the service. The marker types can be either 3d or 2d. The 3d markers are being used for structural objects like the robots, cubes, spheres and indication arrows. The 2d markers on the other hand can be used to display advanced state information for example LineStrip, Heatmap and Text making them especially useful for debugging and analysing. The MarkerService converts these markers so they can be displayed with OpenGL in Luhviz. This way markers form the basis for

displaying and simulating the gameplay as well as providing many analysis and debugging possibilities for the remaining software modules.
Luhviz can also send commands to other modules, e.g. if the user wants a robot to perform a certain skill, there is the so-called data proxy. This interface connects Luhviz to the simulation, the LocalPlanner and all other relevant modules, allowing the user to change behaviours and settings. By separating Luhviz from the rest of the software and having crosstalk enabled through the marker service and data proxy, the visualization is clearly differentiated from the rest of the software. This means it can each be easily replaced if necessary. It also makes the use of automatic testing for GUI somewhat possible.

The Configuration allows loading, editing and accessing of configuration parameters during runtime without using ROS. As a replacement format for the configuration files which were formerly written in YAML, TOML was chosen. That is because it has a clear syntax and convenient grouping of parameters. When it comes to implementing the module the first task is to parse the TOML files. That is achieved with the toml++ library for C++ [14] which completes the extensive parsing and manual editing of TOML files. Next an architecture for the configuration module is needed. It has to be as robust as possible so it was decided that the parameters should be available as members of a struct in C++. This means compile-time-errors are raised if an incorrect parameter name is used in the code. The challenge is to make the parameters iterable and thread-safe, but at the same time statically typed. The parameters can then be dynamically shown in the visualization and their values loaded from a configuration file, without writing extra code for each parameter. This is achievable by representing each parameter as a class providing methods for thread-safe data access and singular configurations. The representation as class provides methods for iterating over its members as well as loading and saving to and from TOML-files. Combining this with a class which stores all the configuration classes it is possible to dynamically iterate over all configurations including their parameters while also being able to access them statically.

5.2 Robot Control

The robot control is newly implemented in the new framework. Instead of the ROS transform framework [15] our self developed transform tool is used. It is inspired by the ROS transform framework, but improves it in terms of multi-threading, handling of velocities and extrapolating. A circular buffer is used to store poses and velocities. New data is always written in the element next to the current latest element. When the writing process is complete the pointer to the latest element is updated accordingly. Reading can be done simultaneously and thus the use of mutexes is avoidable, as data cannot be changed after it is pushed to the buffer.
The new tool has the capability to store velocity information. During our participation in the RoboCup 2022, the velocities were calculated by numerically

deriving the position for the two latest positions. Even though the positions were filtered by a Kalman filter [16], which was adapted to compensate for the vision delay, heavy noises in the velocity data were detectable. Thus, we completely changed our strategy to determine the position and velocity of our robots. The position of the robots is now calculated on the robots themselves. For that the base station sends the measurements from the shared vision system to the robots. The firmware on the robots then uses this information in combination with data from the motor controller and the newly implemented IMU to calculate its own position. This position is then sent to the central server to be used in the robot control and strategy modules. It is noteworthy that this robot position calculation strategy is heavily inspired by the implementation of TIGERs Mannheim, as presented in [17].

We also reworked and -implemented the robot control algorithms in the new framework. The robot control is now based on the circular fields approach, initially proposed in [18]. It originates from the artificial potential fields approach by Khatib [19] which describes a reactive strategy to control robots. While the Khatib's method is often used as local control algorithm in combination with global planning unit, the circular fields method from Singh et al. is capable of globally controlling a robot to the desired position. The approach is further refined in [20] and [21]. A key part of the circular fields is the determination of rotation vectors which define field's rotational direction an through this the bypass direction for an obstacle. Becker et al. proposes to use a multi agent framework to evaluate every possible combination of bypass directions inside a simulation and derive the best outcome based on a cost function. We already implemented this approach for the RoboCup 2022 and used it successfully. However, a big disadvantage is the computational cost required to evaluate all combinations of rotation vectors. Especially when controlling multiple robots at once, the computational load of the multi agent method cannot be provided by an ordinary laptop's processor an thus a powerful desktop computer needs to run the planner. To avoid the use of a separate desktop computer during the tournaments, the new approach bases on the works of Haddadin et al. in [20]. Here, the rotation vector is defined using the robot to target vector. Obstacles that are on the robot's left side, when it is moving towards the target, are avoided to the right and vice versa. With that, the robot always tries to stay on the direct line from robot to target. Additionally to the approach proposed in [20], all control units cooperate. To avoid collisions with our own robots, both control units use the same rotation vector if two robots are close to each other. This avoids that both robots try to bypass the other in the same direction in the global coordinate system. Different to all previously referenced approaches, we still use our feature framework to assign high level tasks which can be reviewed in more detail in [1].

The feature framework was also improved, compared to the last years version and is renamed to "skill/task framework". Skills are now actions which every robot is capable of executing, for example obtaining the ball or moving to a point. These skills are defined in C++ using a builder class. This makes the coding of new skills relatively easy for software development members who have only basic C++ skills. At the same time the framework provides powerful features to customize the behaviour of the robot through custom callbacks and overwriting of constants. Skills are composed with so called components that can be combined individually. The components define how commands sent to the robot are calculated. A task describes the specific execution of a skill. ”Robot 3 should obtain the ball” or ”robot 2 should mark enemy 1” are examples for tasks.

5.3 Continuous Integration

The continuous integration approach aids our software development. Besides the code itself, the development process also underwent changes. We grew up to 15 members in the software department since last year and need a workflow which can handle that amount of people. To achieve this, we mainly use GitLab with all its features [22]. Our new development process is presented below.
When a feature is requested or a bug is found an issue is created. The issue can then be picked up by a software team member who feels comfortable resolving it. A new feature branch from this issue is created. After the required changes are made, this feature branch contains exclusively all the changes for the given issue. This make it easy to review the code when the corresponding merge request is created from the feature branch into the main branch. Our continuous integration server verifies the overall code quality with the respective changes. The verification includes if the software builds, the tests are successful and the code does not contain any bugs. The bugs are found with statics-analyzers. If the examination is successful the code an be merged into the main branch. With every merge into the main branch, extra build steps are executed, for example the generation of an up-to-date version of the software documentation. With this improved development process we can easily manage many people working on the same codebase, while always having a functioning version of the software.

5.4 Python Bindings

Python Bindings are introduced to simplify and improve strategy development, which is explained in more detail in section 6. In addition to the replacement of ROS modules, another change in our new software is the ability to use it as a Python module. These bindings are generated via pybind11 [23] automatically with every merge to the main branch. Since the strategy development requires extensive prototyping including trial-and-error testing, it is inconvenient to develop the strategy in C++. The Python bindings work by loading a python script and running the module. The module loads the C++-Core of the software. The Python script replaces some parts, for example the task assignment module. This works via callbacks which are called by the software when a new calculation is necessary. Now a new strategy can be tested more easily without having to recompile the project. This also makes it possible to have multiple people developing new tactics and strategies simultaneously for the robots without having to learn the entire complexity of C++.

6 Strategy

Our previously implemented strategy showed drawbacks during last year's RoboCup 2022 competition. for example the threshold to shoot goals was too low, the passing skill execution was poor and generally the team played too passively. We assume this is due to our non-adaptive decision making process.
Because the luhbots Soccer branch has grown, we are able to introduce the strategy department as an independent field. The accompanied improved strategy is presented in this section.
The task of the software strategy is to analyze the current state of the game and assign tasks to the robots based on this analysis. These can include, for example defending a particular opponent or playing a pass to a teammate. This assignment of tasks is done in several steps. The information flow used in this process is shown in Fig. 4.

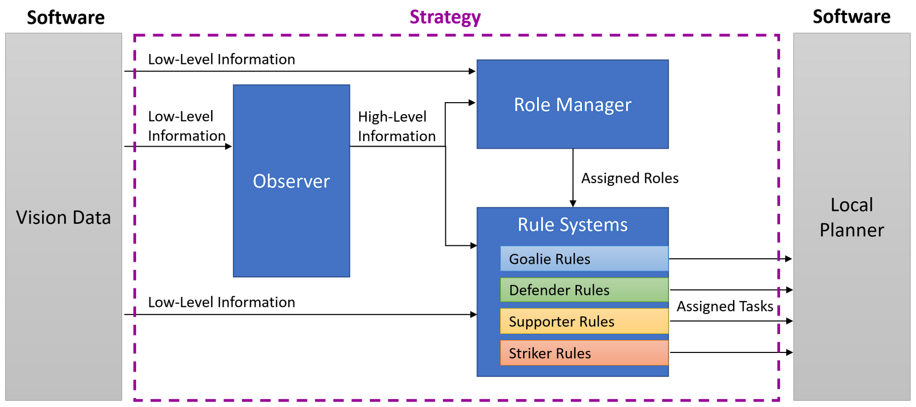

Fig. 4: Information Flow and Components of the Strategy Module

6.1 Observer

The observer has the task of converting low-level information from the vision into high-level information such as goal probabilities or danger levels of the opponents. Previously, this conversion was done by heuristics. First, features are defined which can be used to abstractly describe the current state of the playing field. When choosing features, it is important that they describe all essential properties of the current state completely and accurately. Relative distances between the robots, the ball and the goal can be used, as well as any lines of sight,

proximity to the shooting path, etc.. From a weighted combination of these features, scores can be determined that describe high-level information on which the task assignment can be based in the following steps.
The experiences of the last years have shown that this approach is in principle suitable to describe the state of the playing field, but the result is subject to a human bias and the optimization of the weightings is also extremely time-consuming. Therefore, the goal is to automate this process in the future using machine learning methods. Similarly to the previous implementation, the features are defined first. Subsequently, the training data is recorded in a specially created training scenario or a game simulation. For example, a goal kick can be simulated and features such as the distance of the ball from the goal or the distance of opponents from the line of fire are recorded. It is also stored whether or not a goal was scored under these conditions. This information is then used to train a model with random forests, as described in [24], which are mainly used for this purpose. However, as [25] shows, convolutional networks could also prove suitable. We hope to achieve better results, with the analysis of the playing field, reduce the human bias and manual optimization effort of heuristics.

6.2 Role Manager

A major problem in the past was to find a balance between offensive and defensive behaviour. Since all robots are built the same way, each robot can in principle perform both offensive and defensive tasks. With the previous strategy these tasks are assigned by a central rule system. This system is able to perform a reasonable task assignment in every possible situation. However, this task is extremely complex due to the continuous state space and the large number of robots. This results in a confusing code, which makes adaptations and extensions much more difficult. Therefore, our role management system is now split into two steps. The first step is to determine how many attackers, defenders, etc. are needed. This is done by the Role Manager which assigns a suitable role to each robot. The second step is to evaluate for each robot which task should be performed, based on the role assigned to it. This is described in section 6.3.
The high-level information of the observer can be used to decide how many robots are needed for each role. For example, the average threat level of the enemies determines how many defenders are needed at a given moment. On the other hand, if it is evaluated that our team has had possession of the ball for a long time and an offensive is very likely to result in a goal, more robots can be assigned to an offensive role supporting the current attack. At the same time, however, a set amount of defenders should exist at all times to prevent counter-attacks. The Role Manger can guarantee this by specifying a minimum number of defenders which cannot be undercut. This way, more offensive and defensive systems can be designed by increasing or decreasing these limits accordingly. In the future, the use of reinforcement learning to further improve role assignment will also be explored as shown in [26].

6.3 Task Assignment

In the past, game decisions were made by a central rule system as described in [27]. Each rule contains a priority, a condition for execution and an action on execution. The conditions of the rules are checked in descending priority. If the rule's condition is met, the associated action is executed, otherwise the rule is skipped and the next rule is evaluated.
It has been shown that such a system is in principle well capable of making game decisions. However, if a central system is used to control all robots this quickly reaches a high level of complexity which makes manual adjustments and extensions to the system difficult. Therefore, we replace the central system by several smaller systems. Each of these smaller systems only controls robots of a certain role, instead of the team as a whole. This allows the development process to be parallelized regarding the separate systems. Changes to the offensive behavior no longer also affect the defensive. We hope to simplify the development and optimization of the rule systems and as a result improve our overall gameplay.

7 Acknowledgements

Special thanks to our supporting institutes, the Institute of Automatic Control (irt) and Institute of Mechatronic Systems (imes). We appreciate the financial and educational support. Also we want to mention all RoboCup 2022 SSL teams' fair play. They were incredibly forthcoming and gave us advantageous insight into their research, especially TIGERs Mannheim. Finally, grateful for being part of the Leibniz University Hannover, we also want to thank our university for supporting several different student associations which allow us to exchange interdisciplinary ideas.

References

1. SEEGEMANN, Larissa, 2022. luhbots Soccer Team Description for RoboCup 2022
2. T-MOTOR. https://uav-en.tmotor.com/. Last accessed 24 Jan 2023
3. maxon, https://www.maxongroup.us. Last accessed 23 Jan 2023
4. OMMER, Nicolai; RYLL, Andre; GEIGER, Mark, 2022. TIGERs Mannheim (Team Interacting and Game Evolving Robots) Extended Team Description for RoboCup 2022
5. BERGMANN, Paul; ENGELHARDT, Theresa; HEINEKEN, Tobias; HOPF, Valentin; SCHMID, Michel; SCHMIDT, Mike; SCHOFER, Felix; SCHUH, Kristin; STADTLER, Michael, 2022. ER-Force 2022 Extended Team Description Paper
6. TIGERs Mannheim, https://www.tigers-mannheim.de/. Last accessed 21 Jan 2023
7. Rust, https://www.rust-lang.org/. Last accessed 27 Jan 2023
8. embedded-hal github, https://github.com/rust-embedded/embedded-hal. Last accessed 23 Jan 2023
9. CHESHIRE, Stuart; BAKER, Mary, 1999. Consistent overhead byte stuffing. In: IEEE/ACM Transactions on networking, 7. Jg., Nr. 2, S. 159-172.
10. DSHOT - the missing Handbook, https://brushlesswhoop.com/dshot-and-bidirectional-dshot/. Last accessed 23 Jan 2023

11. Firmware of Mannheim TIGERs, https://github.com/TIGERs-Mannheim/Firmware. Last accessed 27 Jan 2023
12. ROS rviz pacakge, https://wiki.ros.org/rviz. Last accessed 23 Jan 2023
13. Dear ImGUI, https://github.com/ocornut/imgui. Last accessed 23 Jan 2023
14. toml++ library, https://marzer.github.io/tomlplusplus/. Last accessed 28 Jan 2023
15. ROS tf package, https://wiki.ros.org/tf. Last accessed 21 Jan 2023
16. KALMAN, Rudolph Emil, 1960. A new approach to linear filtering and prediction problems. In: Journal of Basic Engineering
17. ABBENSETH, J.; and OMMER, N., 2015. Position Control of an Omnidirectional Mobile Robot.
18. SINGH, Leena; STEPHANOU, Harry E.; and WEN, John T., 1996. Real-time robot mo- tion control with circulatory fields. In: Proceedings of the 1996 IEEE Inter- national Conference on Robotics and Automation, Minneapolis, Minnesota, USA
19. KHATIB, Oussama, 1985. Real-time obstacle avoidance for manipulators and mobile robots. In: Proceedings of the 1985 IEEE International Conference on Robotics and Automation, St. Louis, Missouri, USA
20. HADDADIN, S.; BELDER, R.; ALBU-SCHÄFFER, A., 2011. Dynamic Motion Planning for Robots in Partially Unknown Environment. In: Proceedings of the 18th World Congress The International Federation of Automatic Control, Milano, Italy
21. BECKER, Marvin; LILGE, Torsten; MÜLLER, Matthias A. and HADDADIN, Sami, 2021. Circular Fields and Predictive Multi-Agents for Online Global Trajectory Planning. In: IEEE Robotics and Automation Letters 6
22. Gitlab Docs, https://docs.gitlab.com. Last accessed 23 Jan 2023
23. pybind11, https://github.com/pybind/pybind11. Last accessed 23 Jan 2023
24. BREIMAN, L.. 2001. Random Forests, In: Machine Learning 45, 5–32. https://doi.org/=10.1023/A:1010933404324
25. POMAS, T.; NAKASHIMA, T., 2018. Evaluation of Situations in RoboCup 2D Simulations Using Soccer Field Images. In: RoboCup 2018: Robot World Cup XXII.
26. HU, C; XU M. and HWANG K-S., 2020. An adaptive cooperation with reinforcement learning for robot soccer games. In: International Journal of Advanced Robotic Systems. https://doi.org/=10.1177/1729881420921324
27. LIU, H.; GEGOV, A. and STAHL, F. T., 2014. Categorization and Construction of Rule Based Systems. In: Communications in Computer and Information Science. https://doi.org/=10.1007/978-3-319-11071-4_18