TheRockys

Fabio Manz, Niklas Moxter, Fabian Ritzmann, Emilia Rontas,
G. Schaf, F. Gaul, and H. Baumann

IGS Rockenhausen, Muehlackerweg 25, 67806 Rockenhausen, Germany
heiko.baumann@igs-rockenhausen.bildung-rp.de
https://www.robotik-ag.eu

Abstract. This paper presents the progress of *TheRockys*, the RoboCup Junior Open 2 vs 2 team from Rockenhausen located at IGS Rockenhausen, Germany. It showcases the accomplishments of our team as well as the improvements of our robots regarding software, hardware and overall concept. We developed our robots with the intent of them being able to see, get and kick the ball from any point on the field while simultaneously knowing the positions of themselves, the goals and other robots. The robots have evolved into complex machines which require equally complex software engineering in object detection, orientation and driving maneuvers. To create a fully self-contained network, the software is becoming increasingly intelligent and now the programs need multiple interfaces to communicate with each other. For future competitions, we've already started working on robots suitable for the new regulations which will have smaller and optimized PCBs, vertical shooting kickers and a mirror with less distortion.

1 Introduction

The Robotics-Club at the IGS Rockenhausen exists since 2001. In 2006 members of our club firstly participated in RoboCup competitions. This was the beginning of a successful time which headed to the winning of the world championship in 2012 in Mexico City. Our team "TheRockys" (Fig. 1) started out in 2016 with two slightly modified LEGO NXT robots in the Lightweight League. The following year, we entered the Soccer Open 2 vs 2 league with our first completely self-designed robots, which we upgraded in 2018 and qualified for the German Championship. 2019 was a quite successful year for our team. Besides the 3rd place on the German Championship in Magdeburg we got the 7th place in the European Championship in Hanover. As known, all competitions were cancelled in 2020 due to the pandemic, so we developed completely new robots in the meantime which we are now using for the virtual World Championship in 2021.

Our four team members all have important roles that help improve the robot continuously. Fabio writes the software for the robots and designs the parts that need to be 3D printed or CNC milled in 'Fusion360' [1], Fabian and Emilia build, maintain and repair the hardware of the robots. Emilia specializes in soldering the tiny components to the PCBs and Niklas develops the PCBs in 'Target

Fig. 1. Our team for RoboCup 2021

3001!'[2] as well as creates the concept of the robots. The current year was a great year for us because we won the German qualification and can now call us the German Champion. Also, our entire robot concept has undergone significant changes, which we will explain in the following documentation. We are very pleased to be part of this year's worldwide competition and are excited to learn and improve by meeting many teams from all over the world.

2 Robots and Results

2.1 Hardware

Camera System: The greatest breakthrough is our camera system. Compared to the limited field of view (FOV) of the Pixy2 [4] we used before, the new Arducam [5] is positioned in the middle of the robot and points up towards a hyperbolic mirror which allows us to see the entire playing field. Since the camera must have a certain amount of distance to the mirror to work accordingly, all other components had to be located underneath the CNC milled camera holder to guarantee a large FOV, which resulted in the currently low body height of our robot. Another aspect of our camera system is the extremely low latency between seeing a ball and reacting to it accordingly. Additionally, we've got a bigger number of frames per second while keeping an extremely high resolution.

Lidar: Another major change is the use of the YDLIDAR G6 [8] instead of three 'Time of Flight' (ToF) sensors. Because the Lidar is a 360° scanner, it allows the robot to determine it's own position and that of other robots, as well as the goals precisely. This opens up many possibilities regarding tactical driving like passing the ball or avoiding opposing robots, as it is way more accurate and less error-prone than the ToFs. Those revealed to be very fragile and lost one of their axis quickly if a robot was to block the path. They also had inaccurate triangulation since they were controlled by our weaker microcontroller. The Lidar is screwed on to the upper PCB right underneath the camera.

Processors: Both of these core components are controlled by our newly added image processor Jetson Xavier NX [6]. All other processes like driving behavior, tactics, actuators and sensors are controlled by the Robodyn ATmega2560 Pro [7]. This board has the same processor as the Arduino Mega 2560 we used before, but on a smaller breakout board, which can – when needed – communicate with the Jetson.

Bottom PCB: The redesigned PCB (Fig. 2) contains the motor drivers, power supply and line sensors and is controlled by the ATmega2560 Pro on the main PCB. A 3D printed battery pack that easily connects to the bottom PCB is an integrated part of the chassis as it is retractable and works as a sliding system that supplies all our electronic components with the necessary power. This power is distributed by the collaboration of the bottom PCB and another PCB to create a safe and user-friendly construction. Furthermore, the 31 line sensors are aligned in a cross form to make the calculations of the line program easier and allow the robot to reach to the outermost parts of the field without being out of bounds. Another reason why we have so many sensors is to have a built in redundancy for defective sensors.

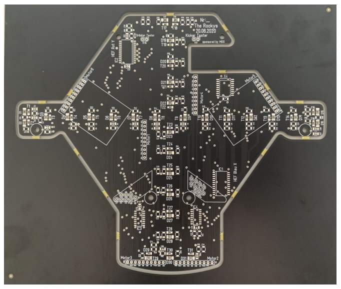


Fig. 2. Bottom PCB

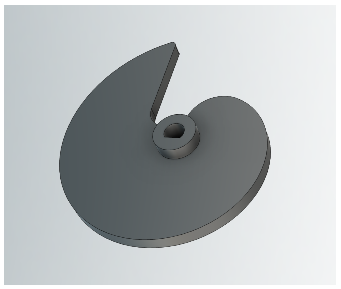
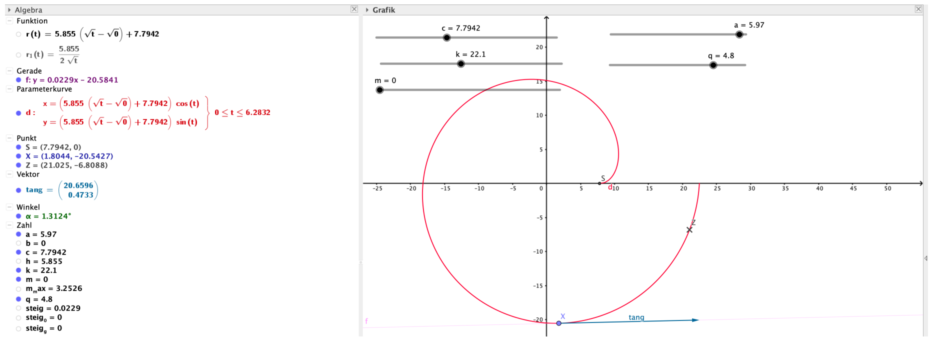


Fig. 3. Kicker-spiral

Kicker: Our mechanical kicker consists of several parts and has been sized down to the minimum to match the 20 cm diameter of the new robot. The heart of it is our root function calculated spiral that tenses the springs that are connected to the plunger and lets it kick after one full rotation. The spiral is driven by a

Pololu gear motor [9] over two bevel gears, of which one is fixed perpendicular on to a rod that is held in place by a 3D printed part and two bearings. We are especially proud of our kicker spiral (Fig. 3) because it helps to balance the motor load and therefore its current. This is achieved by a special root function, which is visualized in Fig. 4 with GeoGebra [3]. The function parameters also determine the stroke of our plunger. This spiral is so important because we are using a space and weight efficient motor, which only has a limited amount of torque. The spiral function is then fed into a custom Fusion360 Add-In, so it can be 3D printed.


Algebra

Funktion

$r(t) = 5.855\left(\sqrt{t} - \sqrt{0}\right) + 7.7942$

$r_1(t) = \frac{5.855}{2\sqrt{t}}$

Gerade

f: $y = 0.0229x - 20.5841$

Parameterkurve

$$d: \left.\begin{array}{l} x = \left(5.855\left(\sqrt{t} - \sqrt{0}\right) + 7.7942\right)\cos(t) \\ y = \left(5.855\left(\sqrt{t} - \sqrt{0}\right) + 7.7942\right)\sin(t) \end{array}\right\} 0 \le t \le 6.2832$$

Punkt

S = (7.7942, 0)

X = (1.8044, −20.5427)

Z = (21.025, −6.8088)

Vektor

$tang = \begin{pmatrix} 20.6596 \\ 0.4733 \end{pmatrix}$

Winkel

$\alpha = 1.3124°$

Zahl

a = 5.97

b = 0

c = 7.7942

h = 5.855

k = 22.1

m = 0

$m_m ax = 3.2526$

q = 4.8

steig = 0.0229

$steig_0 = 0$

$steig_g = 0$

Fig. 4. Design of the spiral-root-function

Dribbler: We have developed a more compact and stronger dribbler as its bracket is now milled out of aluminum and the roller is driven by a 12:5 gear transmission. Contrary to that, our previous robot used a 3D printed case which would melt because of the heat of the motor as well as timing belts that wore out quickly. The dribbler is pushed down and cushioned by a sponge, that is kept in place by a 3D printed part, to allow a firm grip on the ball. Furthermore, the grip is strengthened by a ramp which draws the ball to the middle of the dribbler.

Omniwheels: Carrying on, by designing the entire Omniwheels ourselves, it was possible to adjust them to our needs and test out different versions. Now they offer optimized properties like a good grip and running smoothly. Moreover, they are easy to manufacture despite their complex shape thanks to 3D printing.

Miscellaneous: At last we also have many not as noticeable but still important changes we made to the robot that include: A completely revamped chassis, solid aluminum motor brackets and a lot of small electronic components like levelshifters, connectors, multiplexers, Bluetooth modules and countless more.

2.2 Bill of Materials

- wide angle Arducam with custom 1.4 inch lens
- hyperbolic mirror (custom-made, aluminum)
- NVIDIA Jetson Xavier NX
- ATmega2560 Pro
- YDLIDAR G6
- HM-10 (Bluetooth module)
- BNO085
- VNH5019 Motor Driver Carrier
- Tattu 1050/1550mAh 4S 75C 14,8V Lipo-Akku
- 20.4:1/4.4:1 Metal Gearmotor 25Dx50L mm HP 12V Pololu gear motor
- 391:1 Metal Gearmotor 20Dx46L mm 12V CB Pololu gear motor
- 3 self designed PCBs
- Omniwheels (custom-made)

2.3 Software

Camera: Our camera software combines a very accurate positioning of ball and goal with a low latency and high refresh rate. All the operations on images are done through a combination of C++ and OpenCV [11] to get a reproducible and fast analysis of the surroundings. The structure of operations is relatively simple (Fig. 5) and starts with getting a frame from the camera. For the Jetson processor, there are multiple options to do this, but our testing resulted in GStreamer [**?**] being the fastest and most reliable software to get the raw camera frame into a OpenCV frame buffer. GStreamer is used multiple times in our camera software because it is really flexible. We decided to modify the existing NVIDIA Linux kernel to suit our needs better. This resulted in our *TheRockys Kernel* which can be called realtime, has a fast boot time, starts all of our own necessary services and most importantly adds Full HD support at up to 120FPS. The first analysis step is to convert the Mat from the BGR to the HSV color space. Our testing yielded that this color space is perfect for our needs because the camera should not be perceptible to lighting conditions. We simply discard the value channel, as it only provides information about how bright the color is. By doing it this way, we save a third of our GPU compute power. Next we need to threshold the entire image with a range around the calibrated ball and goal color. This operation is parallelized and runs on the GPU using the NVIDIA CUDA technology. Then we find the contours out of the mask image. They are then ranked by factors like size, distance to the last object, color accuracy, shape, distance to the bot and more. This delivers us a position of the object in pixels, which is calculated via the mirror formulas to get the distance in centimeters from the bot. All the information like ball and goal positions and size will be transmitted to the Lidar process via a custom IPC library. This library uses shared memory between the two programs for instant transfer of the data, and GNU semaphores. To supervise the operations of the program, we have a custom calibration program. One extra feature is that the program can save the videos to our SD-Card in realtime which makes analyzing the footage and debugging a lot easier.

Lidar: The Lidar program is following a different approach when it comes to using libraries. We firstly tested a ROS with Hector-SLAM, but this solution required a lot of resources, was not accurate, had a lot of overhead and was not adaptable to special functions like detecting where other robots are positioned. Because of this, we have decided to build the whole architecture from the ground up, giving us the most flexibility. The only library we are using is the open source YDLidar library [10] to communicate with our scanner, therefore we do not need to handle the low-level communication and deserialization of all packets. The program consists mainly of a lot of mathematical operations on arrays. To start with, we get a distance array back from the SDK and our objective is to determine the position of our robot. For this we firstly need to do some filtering on the data and then turn all polar points around the center, so we do not need to worry about rotation in the whole project. Out of that pixel array we calculate all the straight lines which are then ranked to get the lines which are corresponding to the walls of the playing field. Next we find the positions of the goals. With these details in mind, we can determine where the robot is standing on the field. As a last step we calculate the position of the other bots on the field. For this we look at all the remaining points which are not part of any wall or goal and find the blobs. This information is then meshed with the camera information and send to our ATmega. The calibration approach is primarily the same as for the camera system, which is also the reason why they share most of their code.

ATmega: Our microcontroller is coordinating all software subsystems. It is the master, which is getting all the data and is calculating the next moves out of the gathered information. This central piece is connected to all other sensors and actuators, which are arranged in a star configuration around the MCU. It is handling tactics, which we have in two varieties. Firstly we have all kinds of predefined tactics which have fixed targets such as getting the ball as fast as possible or scoring a goal as fast as possible. Then the microcontroller is computing a complex algorithm which is deciding with the aid of our current speed, the line sensors, the current angle and the intention of the rest of the program what it's doing next. There are a lot of different scenarios, among which the algorithm distinguishes. These are selected based on the above named features. For example, if the robots is hitting the line really fast, we need to break instantly. But if the robot is not going as fast, we allow him to drive over the line-up to a fixed point. This point is also depending on his speed and angle. This behavior allows the robot to catch the ball outside of the line, but on the other hand never get out of bounds. The processor is also using PID tuned controllers for various tasks like driving to the ball, kicking exactly in the goal or driving to an exact position on the field. Our team is very proud to have a function named 'driveToPos(x)' because this shows how easy it is to program even complex tactics and strategies. The existence of this function shows very well how our abstraction works, as everybody is able to understand and use the function. But under the hood, there is a lot going on, as we discussed in the Lidar section and this section. In addition to the calculation of all the driving behaviors, the microcontroller is connecting to all the sensors and is controlling all the actuators. [Fig 6] Together with all this hardware, we store their calibration in the EEPROM because every robot needs a slightly different calibration. Another big thing that is handled by our microcontroller is the Bluetooth communication. This step is essential, as the robots coordinate with each other. The exchange of absolute positions of the ball, the goals and the positions of all robots are significant for a well coordinated teamwork.

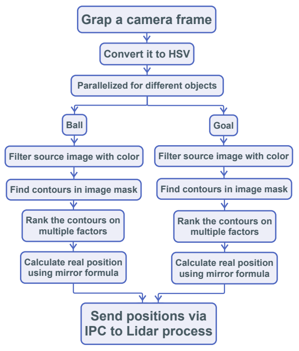

Grap a camera frame
Convert it to HSV
Parallelized for different objects
Ball
Goal
Filter source image with color
Filter source image with color
Find contours in image mask
Find contours in image mask
Rank the contours on multiple factors
Rank the contours on multiple factors
Calculate real position using mirror formula
Calculate real position using mirror formula
Send positions via IPC to Lidar process

Fig. 5. Simplified Camera Flowchart

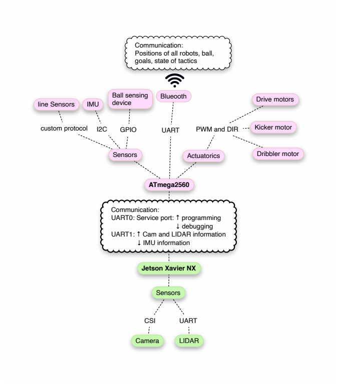


Fig. 6. Internal Communication

3 Conclusion and Future Work

3.1 Results

One crucial experiment we did, was a theoretical approach that specified our hyperbolic mirror. We calculated the necessary values the mirror needed for the robot to see the entire playing field depending on which camera we used and determined the decisive factors such as camera height, mirror height and curvature of the mirror. Following that, our camera was chosen while taking the maximum field of view allowed in the rules and the diameter of the mirror into account. For programming we try to use the most common software architectures because they're well known and have good support. The previously used Raspberry Pi has in comparison to the Jetson plainly not enough performance but on the other hand good community assistance. Furthermore we're really trying to nail down low latency, f.ex. we prefer a bare bones C approach over the Arduino IDE, which is adding a lot of overhead to basic features, and try to use embedded programming languages where we have exact control over the instructions the machine is doing. For spectators, it might look like a small improvement to change from the BNO055 to the BNO085 IMU but the previous one caused many problems which were based on internal design flaws we couldn't fix. We became aware of the problems when we drove a lot with the robot and then further investigated them by putting the robot under extreme conditions. Lastly, we generally tried to make our hardware user-friendly and out of small but easy to produce components. Our specialization in additive manufacturing makes the quick testing of new ideas possible.

3.2 Conclusions and future work

Over the span of a year we've learned to design and construct more complex and well-thought-out PCBs, circuit diagrams and overall mechanical parts which, when put together, could be assembled and repaired faster than ever before. We have also built our own camera system to fit our requirements as well as using a Lidar on a robot for the first time. Thus we comprehended how to become experts in powerful ARM CPUs and GPU-architectures. Another thing we mastered was the communication of two processors and a stable custom Bluetooth communication. One last thing we gained experience in was partnering and publicly exchanging information with big companies such as NVIDIA or YDLIDAR. We're already very satisfied with what we have achieved so far but we're going to refine the robots even more. Our plans currently contain the optimization of our custom-made mirrors, as we've already started extensive testing on how to ameliorate them. Our previous attempts of experiments may have neglected that the distortion of the resulting picture would be improvable so we want to create a new mirror while taking this into account. Our team is already preparing for the TC proposed changes to the new rules. This includes routing the next PCBs to accomodate the smaller sized robots and adjusting the dribbler-kicker-ensemble to work with the new golf ball. This enhancement allows us to kick vertically and miniaturize the construction. Another constant improvement we're always working on is to drive even faster while still handling the ball without getting out of bounds. To comply with that acceleration we need to enforce our hardware to make it even more solid and stable.

References

1. Autodesk, https://www.autodesk.com. Last accessed 19 Jun 2021
2. Target3001!, https://ibfriedrich.com. Last accessed 19 Jun 2021
3. Geogebra, https://www.geogebra.org. Last accessed 19 Jun 2021
4. PixyCam, https://pixycam.com/pixy2. Last accessed 19 Jun 2021
5. Arducam, https://www.arducam.com. Last accessed 19 Jun 2021
6. Nvidia, https://www.nvidia.com. Last accessed 19 Jun 2021
7. Robodyn, https://robotdyn.com. Last accessed 19 Jun 2021
8. YDLidar, https://www.ydlidar.com. Last accessed 19 Jun 2021
9. Pololu, https://www.pololu.com/product/3484. Last accessed 19 Jun 2021
10. YDLIDAR SDK, https://github.com/YDLIDAR/sdk. Last accessed 19 Jun 2021
11. Open CV, https://opencv.org. Last accessed 19 Jun 2021
12. GStreamer, https://gstreamer.freedesktop.org. Last accessed 19 Jun 2021