ROBOCUPJUNIOR RESCUE LINE 2024

TEAM DESCRIPTION PAPER

OverEngineering^2

Tim Dumaschus and Marius Keiser

Contents

Abstract

Our robot is designed to compete in the RoboCup sub-league rescue line. Over the course of two and a half years, we have designed and improved a camera-based robot able to conquer all challenges within the Rescue-Line rules. To achieve this performance, we are currently utilizing two cameras for line following and victim detection, as well as the new Raspberry Pi 5 and a Google Coral USB Accelerator. Over the years, we have come up with innovative solutions, such as integrating multiple AI's into the vision system of the robot and designing the chassis as well as all the mechanisms in Fusion 360. Besides an AI for detecting any victims in the evacuation zone, another AI is used to solve the difficult task of detecting the reflective silver tape at the entrance of the evacuation zone, surpassing the accuracy of any image processing method we have tried in the past. In order to achieve a faster development process, we decided to use Python as our main programming language, with additional performance improvements. The software for our two Arduino Nano's controlling seven distance and two gyroscope sensors is written in Arduino, with external programs written in Shell script for ease of use. The accurate line following is based on OpenCV and NumpPy image processing. The use of AI combined with a reliable gripper arm and storage system for all victims enables us to complete the whole evacuation zone in about 1:40 minutes, resulting in a huge time advantage over most of the other teams.

1. Introduction

a. Marius Keiser

After being interested in robotics for over a year, I learned about the Robotics Club. Between me joining in 2019 and Tim joining in 2022, I gained a lot of experience but never qualified for the German Open. I came up with the idea of using an AI model to detect victims in the zone, since we had many issues previously. Now I am responsible for the logic inside the evacuation zone and the development of all our AI models, recently adding a classification model to identify the silver strip at the entrance of the evacuation zone.

b. Tim Dumaschus

In April 2022, the Robotics Club and the former team #RoboEvolution, consisting of Marius and then one more team member welcomed me as a new member. I proposed using a camera-based line following system to compete at RCJ Rescue Line and took the lead on the project. To design the chassis, gripper, and storage system, I taught myself how to use Fusion 360. As the designer of all the different parts, I was also responsible for their correct assembly. Having the robot at home and being the only member of our team with prior programming experience in Python, the main programming language we use, I was able to program and test a significant portion of our code.

2. Project Planning

a. Overall Project Plan

Objective: As we have participated in the RoboCup Junior Rescue Line League for the last two years with a completely customized, self-developed camera-based robot, our main objective is, of course, to surpass the successes (last year's 3rd place at the European Championship) as well as the robot's overall performance of the last years. Since we are competing in the Rescue Line sub-league, the robot must be designed according to these rules and be able to accomplish all tasks and challenges within this league:

Table 1: Overview of Requirements and Final Solutions

Requirement	Possible Solutions	Final Solution
Overcome bumpers, ramps and seesaws while still being able to follow the line precisely	Drone; 4x Mecanum Wheels; 4x Self-made Neoprene Wheels; Chains	Decided to use the self-made neoprene just like last year. These wheels have the advantage that bumpers are almost ignored and they still have grip on ramps even with a robot of ~3 kg. Additionally we have moved the motors slightly lower to increase ground clearance.

Follow the black line; detect green intersection marker and red goal tile	Light-Sensors; Camera	We chose a camera because we have two years of experience with it, and sensor-based methods are too limited. An additional factor is the ability to be able to more easily adapt in difficult situations.
Turn 180° at double green markers	Time based turning; Turning according to the gyroscope sensors	Turning according to the gyroscope sensors is more accurate and doesn't fail if the robot gets stuck, so it can continue turning to try and un-stuck itself.
Detect obstacles	Camera; Pressure switch; Infrared distance sensor; Ultrasonic distance sensor	After testing multiple methods, we determined that the most accurate one is infrared sensor based. We have 3 infrared sensors in the front of our robot, of which the outermost 2 are tilted inward, giving us the ability to tell how we stand in front of an obstacle. (new this year)
Being able to navigate and turn within the 25 cm x 25 cm cross section between pillars at tile corners under elevated tiles	Compromise on functionality for a smaller robot; Tightly packed robot with more functionality;	Since we already had a mostly functional robot design from last year, we added more sensors, smoothed corners, and shortened the protruding LED mount so it could turn in tight spaces. We also packed the internals tighter, so we reduced the width and length slightly. Finally, we have added a "stuck detection" to prevent getting stuck. (new this year)
Detect the silver reflective tape at the entrance of the evacuation zone	AI classification model; colored laser / bright LED's (detecting reflection pattern with camera); dedicated silver sensor	We settled on using an AI classification model to detect the silver tape because our initial approach (detecting the bright reflection of our LEDs) did not work. The dataset is self-created, and we add to it regularly as problems arise. The model in its current state has been very reliable. (new this year)
Detect victims / evacuation points	Pattern search; AI detection model; OpenCV's Hough Circles; Color Sensor	Since we always had issues with OpenCV's Hough Circles we chose an AI detection model. It works very well and we always add new data to our custom dataset if we find areas where the AI fails. (new this year)
Pick up victims	Box coming from the top using rubber bands; Gripper coming from the front; Suction system	The box-based system last year did not work well, so we chose a gripper that grabs the victims from the side. The gripper is attached to an arm that rotates down when we need to pick up a victim and rotates sideways to sort them. (new this year)
Sort/Deliver victims	Storage with multiple compartments; Delivering each victim individually	We decided to go with a storage system since we will be able to save a lot of time (less driving to evacuation points). Additionally, we will be able to pick up any victim in our path without moving it. Otherwise, living victims would have to be prioritized. (new this year)
Compliance with height restrictions of 25 cm	Compromising on features for a smaller robot; Compact gripper system	After designing the compartment system, we had some space in between the two compartments, so the new arm is designed in a way that makes it fit in that gap. Thanks to its compact design, we don't have to compromise on the feature set of the robot. (new this year)
Adding a handle	3D printed handle; Metal wire handle with electrical tape	Because a 3D-printed handle would not fit on the robot, we chose a simple handle out of some metal wire. This way, we have a flexible handle that can sit behind the display. We confirmed its validity with the organizers of the German Open. (new this year)

Staying within the time limit of 8 minutes	Fast driving; Making few mistakes; Speeding up the line following; Speeding up the victim rescue	Since we are using a Raspberry Pi 5, we can run with ~40 FPS. That allows us to speed up the robot significantly while line-following compared to last year. Because with this improvement we already reached the maximum speed of our motors, we mainly focused on speeding up the victim rescue. (new this year)

Project Schedule: After we decided to participate in RCJ for one last year, right after the European Championship in June 2023, we first re-watched and evaluated our old runs, collected problems with the current design of the robot we encountered during testing, and then adjusted some of our final solutions in Tab. 1 accordingly. With this new plan of what needs to be changed and improved, we were finally able to create a schedule that can be seen in Tab. 2. Unfortunately, the tasks are now only shared between Marius (M) and Tim (T), as our third team member is no longer participating this year.

Table 2: Overview of Project Schedule

Milestone	Deadline
Settling on all the components around which the robot will later be designed around in CAD (M&T)	End June
Victim detection AI tests and development -> at latest by the time the hardware of the robot is completely finished, a working prototype should be available for testing (M)	End October
Checkpoint: All components are selected based on last years experience and recommendations, Tim can now start design the robot around them in CAD (M&T)	End June
Redesign the base frame of our old robot (victim gripper/storage excluded for now) (T)	Mid August
Checkpoint: After consultation in the team as well as some revisions, the base frame is finished, we can now begin the design of the victim gripper and storage (M&T)	Mid August
Finalize the robot's design in CAD so that the printing/assembly is finished by early October for software development during the vacation (T)	Mid September
Checkpoint: The CAD design of the robot is now complete after discussing possible gripper shapes and sorting types; AI development has also made good progress (M&T)	Mid September
Print and assemble the robot before the vacation starts (T)	Mid October
Checkpoint: Finished all assembly by mid-October, but had to disassemble the entire robot again during the vacation due to reliability issues with stiff cables.(M&T)	Mid October
GUI development; communication tests/upgrades for more distance/gyro sensors (Arduino -> Pi); camera implementation; implementation of old line follower (M&T)	End October
Checkpoint: GUI development, Arduino -> Pi serial communication, and line follower implementation successful; AI needs performance and accuracy improvements (M&T)	End October
Test & rewrite parts of the program to work with the new Raspberry Pi 5 (this milestone came unexpectedly due to the spontaneous release of the Raspberry Pi 5) (M)	End December
Optimize performance and structure of the old program; re-implement improved obstacle avoidance; improve gap detection/avoidance; add new program parts like "stuck detection" (After extensively testing the Raspberry Pi 5 for reliability and rewriting parts of the program, the robot needs to be upgraded from Pi 4B to Pi 5) (T)	Early January
Improve victim AI performance and accuracy to integrate it into victim rescue (M)	Early January
Checkpoint: Victim AI improvements, upgrade from Pi 4B to Pi 5; all improvements/additions to the old software went well; silver detection, and therefore the entire evacuation zone, is a major problem right now; remaining time until first tournament should be spent on perfecting the victim rescue, but without a working silver detection the rest of the evacuation zone won't work either, so this is now a priority (especially the time for finding the exit is very tight now) (M&T)	Early January
Implement a new AI for silver detection (M)	End January
Perfection of line following and implementation of victim rescue (finding the exit if possible in time) (T)	End January

Checkpoint: Qualification tournament - managed to perfect victim rescue, silver detection and line following; only spend very little time on finding exit, however, successful in one run (M&T)	Mid February
Performance evaluation of the qualification tournament; improve some issues discovered with line following (M&T)	Mid April
Implement a new find exit strategy (T)	Mid April
Improve victim/silver detection by gathering more training footage (M)	Mid April
Checkpoint: German Open - successfully implemented find exit strategy; improved victim/silver detection; made line following more reliable (M&T)	Mid April
Performance evaluation of the German Opens; post-competition optimizations; add code and strategy's for some possible edge cases at the Word Championship (e.g. obstacle/intersection on ramps, ...) (M&T)	Mid July

In general, we were able to stay relatively close to the schedule we had set up beforehand as a team. However, especially due to the spontaneous release of the Raspberry Pi 5 and problems with the silver detection, we had some deviations from our beforehand planning, which meant that we were not quite finished with the robot for the qualification tournament in Hanover and had to invest considerably more time than planned right before the competition. We made up for these delays in the schedule before the German Open, even though we had little time due to our final high school exams.
When setting up this schedule, we had already divided it up in advance so that as much work as possible could be done at the same time (e.g., Marius developed the AI detection while Tim worked on the CAD design). As you can probably recognize, we tried to schedule most of the time-consuming work during the vacations, as we only had limited time for robot development during school time. We prioritized the tasks right from the start and completed them accordingly. For example, Tim's first task was to complete the robot's hardware as quickly as possible so that we could use it to test and develop new software. Perfecting line following had priority at first because multiplying a few line points by rescuing victims is pointless. Similarly, the decision was made to prioritize rescuing victims over finding the exit of the evacuation zone.
Besides the constant exchange via Discord, we met at least once a week in person at the robotics club to compare the current development status and discuss design ideas.

b. Integration Plan

Each component we integrated into the robot underwent two phases. The first phase is the testing phase, where we create a simple testing setup that allows us to test the functionalities we need from that component. If the component satisfies our expectations, it can continue to the second phase. Then we program in the functionality into the main program, install the component onto the robot, and validate everything again. How each component satisfies the requirements of this competition can be found in Tab. 1. The connections & interactions between the different components are illustrated in Fig. 1.

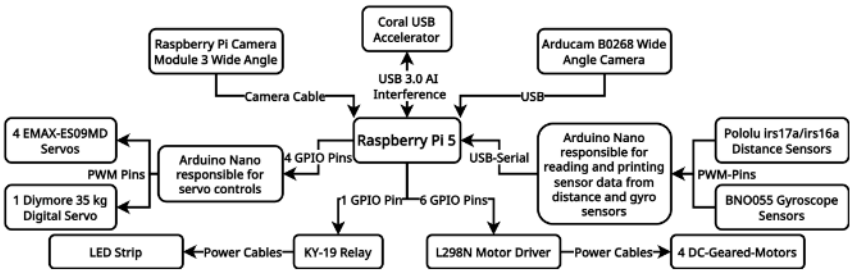

Figure 1: Integration Plan

3. Hardware

From the beginning, we decided to design and 3D print the entire structure of the robot. This not only allows us to customize the chassis to the components we use, making it more space-efficient, but it also gives us the ability to quickly make improvements to the chassis as soon as problems arise.

Based on the requirements of our new design, we began deciding on a list of components (which can be found at the References c. of this file) and drew a hardware diagram (Fig. 2) around them, which shows their connections and how they work together. Tim then continued with the CAD design of the robot, making sure that the power train, drive units, connections for switches, and digital voltmeters would fit inside the robot, because we rarely need to reach them. The upper plate of the robot mainly contains microcontrollers and sensors (the communication between them is already illustrated in Fig. 1), as well as the victim storage system and the servo to which the victim gripper arm is attached (the decision for this setup was already explained in Tab. 1).

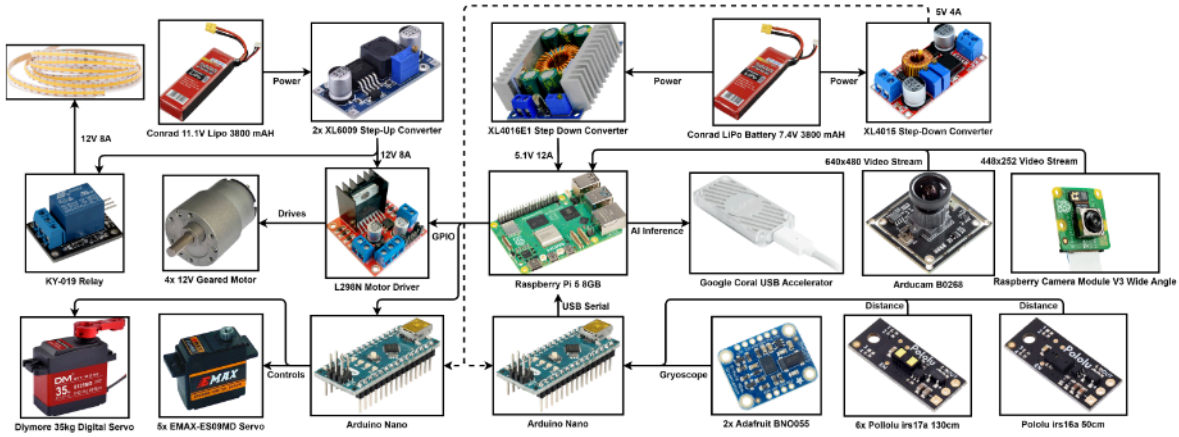

Figure 2: Hardware diagram of all installed components

a. Mechanical Design and Manufacturing

Chassis: Over the past two years, we have been designing and optimizing the chassis of our robot. Autodesk Fusion 360 was used to realize our design ideas and take advantage of the freedom that 3D printing offers. We have been using eSUN PLA+ as our main filament since last year and have had reliable experiences with it so far. This toughness improves PLA, and combined with various densities of infill, it provides the necessary strength for navigating a ~3 kg robot around the course.
Based on a number of components, including our powerful DC geared motors and large neoprene wheels that ensure good grip and safe overrun of speed bumps (see requirements in Tab. 1), the robot has already reached a considerable size. This time, however, we made sure that the robot had high, rounded corners because we often got stuck on sharp edges before (especially at pillars under elevated tiles, as stated in Tab. 1) as well as more ground clearance by lowering the motors to reduce the risk of getting stuck on speed bumpers. Despite, or perhaps because of, its size, we have once again decided to place a large touch screen above the victim storage system. This enhances the debugging process, adds aesthetic value, and has a minimal impact on the robot's overall size at this point. To provide an image free of shadows for our camera, LED-strips are installed relatively high and straight above the line. See the full CAD chassis design in Fig. 3.

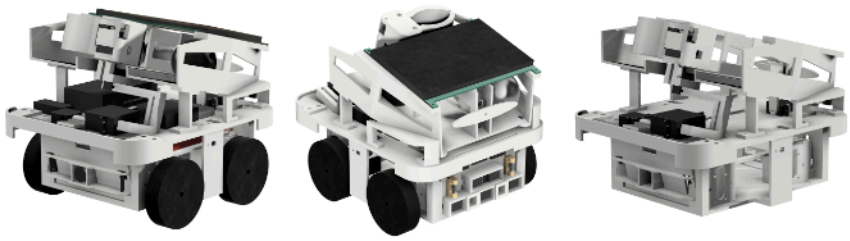

Figure 3: Robot Chassis design in CAD

Rescue Mechanism: As already identified in Tab. 1 our goal was to design a storage system that would sort the victims so they could all be collected at once, which greatly reduces driving times and enables us to pick up any victims on our path without possibly moving them to a position hard to reach. We also decided for a gripping mechanism that is attached to an arm to reliably pick up and lift the victims this year (as the approach of a lowering down box closed up with rubber bands onto the victims only performed poorly last year).
As shown in the renders (Fig. 4), the gripper opens using a small internal servo, which leaves enough space to

accommodate a distance-measuring IR sensor that confirms the successful pickup of victims. Furthermore, by adding two extra servos, the entire gripper mechanism can be lifted up and rotated on an additional axis. This allows us to sort the collected victims into different storage areas on the robot. The blades of the gripper are fitted with small edges that reach underneath the victims, enclosing and holding them in place during lifting. Initially, this small edge was only intended for the top, but after some victim gripping tests (where we also turned the gripper upside down), this solution proved to be very reliable.
Originally designed to fit a rescue kit as well, our storage system uses threestorage areas and two independently controlled barriers. This system allows us to pick up the victims in any order while always being able to deposit the living victims first. The robot sorts the living victims to the left and the dead victims to the right (see Fig. 4). When the main barrier is opened, the living victims roll out first. After that, the barrier will close again, and the barrier in front of the dead victims will open, so they will roll up to the main barrier, which can now be opened again to deposit the dead victims.

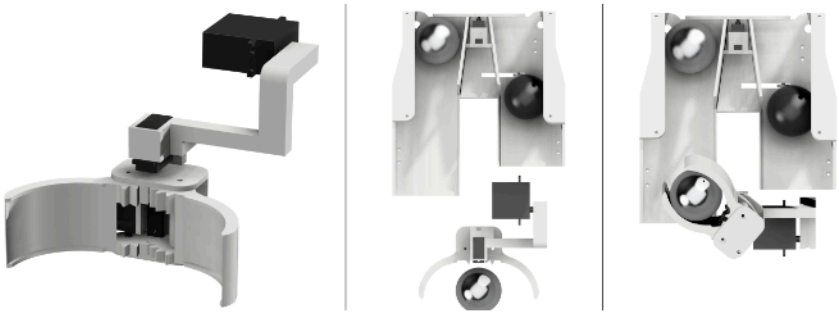

Figure 4: Victim Rescue Gripper Arm and Storage System

b. Electronic Design and Manufacturing

We use two Lipos with different voltages on our robot. The 7.4 V of one Lipo is reduced to 5 V using an XL4016E1 step-down converter for the Raspberry Pi 5 only, as well as an XL4015 step-down converter for the Arduino's, servos, and sensors. The 11.1 V of the other Lipo is increased to 12.1 V by two parallel XL6009 step-up converters to power the DC motors as well as the LEDs.
We currently utilize the Pololu irs16a and irs17a distance sensors (once with a maximum range of 50 cm and once 130 cm), because they have proven to be very reliable and precise in distance measurements and have rarely broken down (further details in Tab. 1). As they were also very reliable last year, we are using the BNO055 gyroscope sensors again, however, this year we are using two at the same time to compensate for potential sensor shift that sometimes occurred after a few rotations in the past. All these sensors are read out by a separate Arduino Nano, the sensor data is then sent via USB serial to our main controller, the Raspberry Pi 5. The Pi, to which our two cameras are connected, also controls the 4 DC motors with a direct connection via 6 GPIO pins to the L298N motor driver (both motors on one side are connected together to one output). The revoART LED strips can be switched on and off by Pi via the KY-019 relay (switched on for following the line and switched off for victim rescue). Finally, a second Arduino Nano is used, with which the Pi communicates via 4 GPIO pins to control the EMAX-ES09MD servos and the Diymore 35 kg digital servo, and thus the gripper arm and the victim storage system. These servos in particular were chosen because they all have durable metal gears and have been proven very reliable in past projects. (All the links to the components can be found in the References c., a connection diagram is provided in Fig. 2.)
There are a lot of cables inside and on top of the robot, which have been organized in the best possible way. However, custom PCBs could be used in the future to simplify repairs.

Power Issues With Raspberry Pi 5: While we were still using just two parallel step-down converters to power the Pi, Arduino's, sensors, and servos and then switched to the new Raspberry Pi 5, the robot began crashing randomly during testing. After some investigation, we identified the power supply as the problem. During some test runs, we started to monitor the current power consumption of the Pi using the command

```
vcgencmd pmic_read_adc.
```

When we saw from this data (Fig. 5) that the Pi requires currents of up to 7.5 A in some cases, the resulting voltage drops to under 5 V were only logical. Our step-down converters were simply not able to supply the

Raspberry Pi, which requires up to 7.5 A, as well as the Arduino's sensors and servos with a sufficient current and a stable voltage of 5 V. The first thing we did, as shown in (Fig. 5), was to increase the voltage of the step-down converters to 5.1 V to reduce potential drops. This solution has only helped to make the crashes less frequent, but not to prevent them completely. Finally, we were able to solve the problem by disassembling the entire robot again and installing a 12 A XL4016E1 step-down converter just for the Raspberry Pi (the voltage was again set to 5.1 V). This particular step-down converter is a somewhat unusual solution because it requires active cooling during operation, which we achieved by adding a fan as well as a large cutout in one of the side plates of the robot (see Fig. 5).

Figure 5: Voltages before and after step-down adjustments | Active cooling for 12 A step-down converter

4. Software

a. General software architecture

The main program relies on Python. Libraries like OpenCV and NumPy significantly contribute to the functionality of our code base, handling most tasks. Moreover, we use Numba, a just-in-time compiler, to address Python's speed limitations. We use the Ultralytics library for both running the silver classification model and the victim detection model. Our two Arduino Nano's are programmed through the Arduino programming language. The Arduino responsible for reading our various sensors communicates the data over a serial connection to the Raspberry Pi 5. The second Arduino Nano is responsible for controlling the five servos we use. The Raspberry Pi communicates through four digital pins with the Arduino, where a binary code represents a servo state.
The program is divided into 4 processes, excluding the main process, which handles the custom GUI. We have one process for each camera, where one is responsible for everything line-following-related and the other for anything inside the evacuation zone. The third process is used to read all the sensor data sent via USB serial. The last process is the controller, which determines what actions the robot should take based on the images and sensor data evaluated by the other processes. Below in Fig. 6 is a simplified diagram of the full program.

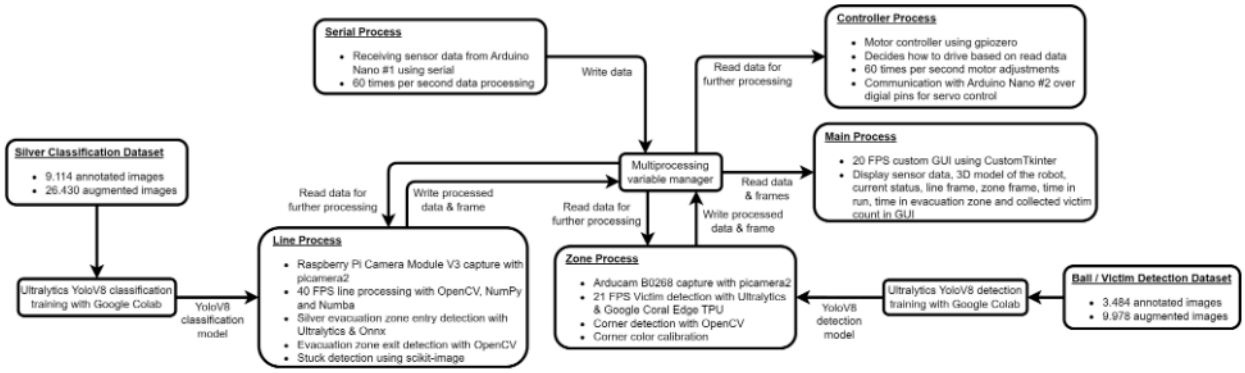

Figure 6: General software flow diagram

Line following: The line process, in combination with the controller process, handles line following, obstacle detection, gap detection, silver strip detection at the evacuation zone entrance, as well as evacuation zone exit detection. The line process runs at about 35-40 FPS, so we are able to drive at relatively high speeds. In the figure below (Fig. 7) is a simplified flow diagram of the line following process:

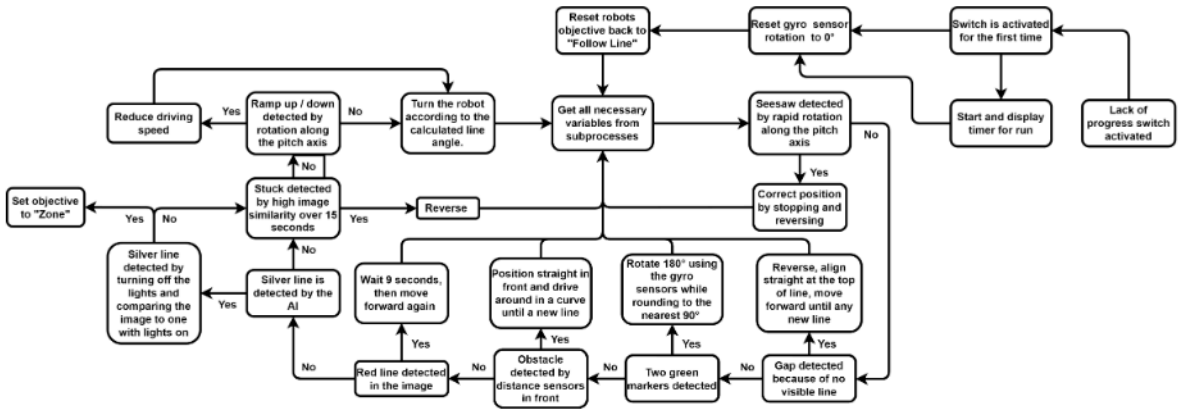

Figure 7: Line following flow diagram

The image processing of the line following will be explained in more detail in Section b.

Evacuation zone: This part begins executing as soon as the line follower has detected the entrance of the evacuation zone. The process operates from the bottom camera and continuously searches for victims until it has collected them all. We decided on the strategy of collecting all victims first before delivering them to the corners because we think it is more time-efficient than other methods, even with the downside of having to release all victims in the case of a LOP. The full evacuation zone strategy can be read from Fig. 8.

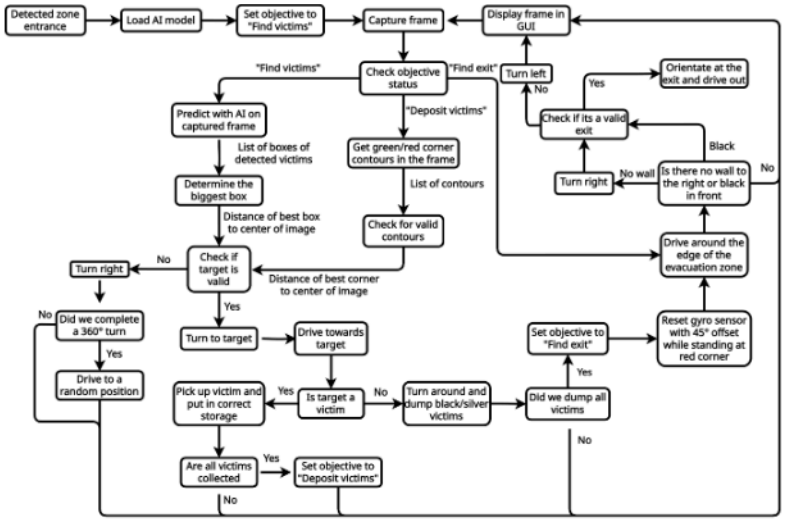

Figure 8: Evacuation zone flow diagram

A video of the process from entering until delivering all victims can be found here. More details on why we chose an AI model for victim detection can be found in Section b.

b. Innovative solutions

Line detection algorithm: The robot follows the line using a wide-angle camera mounted about 10 cm above the ground, looking directly down. Through the use of bright LED's we eliminate any shadows that may be interpreted as the line. The robot follows the line by initially searching for seven points in the image. In addition to identifying the lowest point on the line, both for the full image and a cropped version, the highest, leftmost, and rightmost points on the line are identified. The robot typically follows the line by centering the highest point of the cropped image. If this point is not at the top edge of the cropped frame, either the left or right point is selected for following, depending on their distance to the edge of the frame, to prevent overrunning the line. The point that the robot is currently following is highlighted in red on our GUI (see Fig. 9).
At intersections, this becomes more challenging, especially when standing at an angle, as the correct line may no longer be the only one at the upper edge of the frame (see 4th image in Fig. 9). This is why we calculate a theoretical yellow point by connecting and extending the lowest and highest points of the cropped image. In the case of an intersection, the point at the top of the frame closest to the yellow point is selected to follow. If a

green marker is detected, the point on the corresponding side is chosen (see 3rd image in Fig. 9). The detection of these is based on checking four areas around each side of any green marker in the frame for the black line.

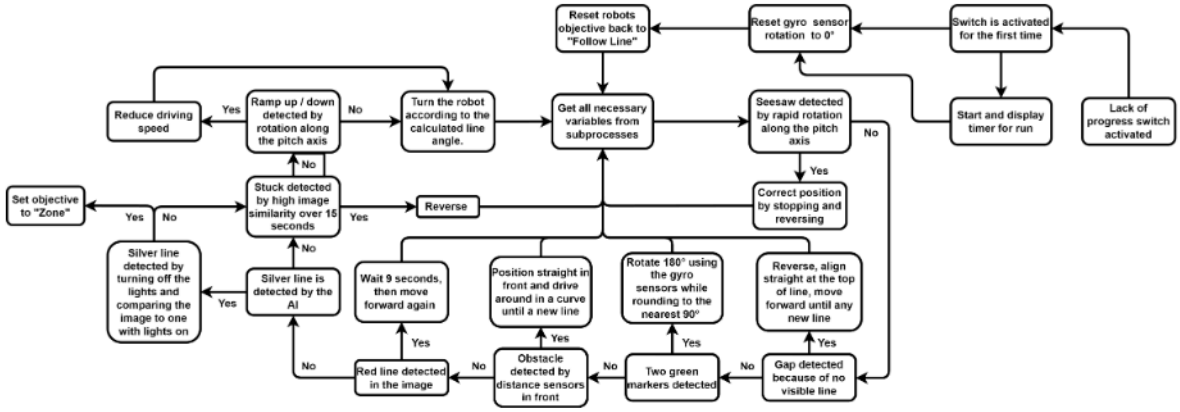

Figure 9: Image processing while line following

Using AI for victim detection: From the start of the German Open 2023 up until the European Championship 2023, we used the OpenCV function Hough Circles. We had many issues with this method because it would detect circle-shaped objects in the background as victims, and the robot would randomly miss victims. Additionally, it had many issues with being unable to detect silver victims due to their reflective nature.
After the 2023 season, we chose to try out an AI model. After some searching, we settled on the YoloV8 architecture by Ultralytics because of its simplicity. After annotating over 3.484 images with 9.568 boxes and many different training runs on Google Colab, we got a close-to-flawless AI model that can detect both silver and black victims separately (see Fig. 10). With the addition of a Google Coral USB Accelerator, we are able to run inference at ~21 FPS, making it possible to complete the entire evacuation zone in under 2 minutes.

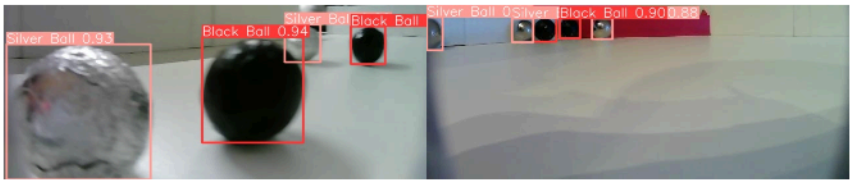

Figure 10: AI predictions on 4 images

Solving evacuation zone entrance detection with AI: We are unable to use the same method as last year, where we detected the silver line through the image suddenly becoming very bright because of our LED array, due to design decisions. We tried different solutions for detecting the silver strip through OpenCV image processing, such as detection in the HSV color space with a median blur. None worked because the silver strip looked too similar to black electrical tape on our camera. This is why we chose to use an AI model here as well. We chose the same YoloV8 architecture but this time as a classification model, which executes faster. After capturing over 9.114 images at different competitions and under many different conditions, as well as adding more images of specific conditions when problems arise, we now have a silver detection model that, so far, has never failed to detect the entrance of an evacuation zone in scored runs.

Advanced debugging with a custom GUI: The big screen we have installed on the robot has so far been used for the past 2 years to display a simple OpenCV canvas with some sensor values drawn on it, including the camera image. This solution worked but was neither pretty nor convenient. That's why we chose to design a fully custom GUI using CustomTkinter. The GUI displays a lot of different information, such as the streams of both cameras, the values of all our sensors, a timer for the length of a run and evacuation zone, a 3D model accurately showing the robot's rotation, and much more, which can be seen in Fig. 11. The new GUI is very convenient when debugging issues, because we don't need to print values to the terminal and can instead directly read them from our GUI. Additionally, we run screen recording software while we have official scoring runs, so we can exactly see when something went wrong. The screen recording gets included with the external recording in the YouTube videos we upload here.

Figure 11: Custom GUI while driving line

5. Performance Evaluation

Thanks to the progress made in development this year, we have succeeded in significantly improving the performance of our robot compared to last year. Back then, we often had time problems running the course, which is why we focused our development efforts this year on improving the speed of the robot while also improving its reliability in overcoming competition challenges.

If we were in the process of solving a particular problem or optimizing an approach, we initially focused exclusively on this part of the course and only tested this section again and again until the goal was achieved. We have written numerous scripts exclusively for debugging basic functions such as serial communication, GPIO control, motor tests, and camera control in this process. To shorten the waiting times for a normal program start, we have added a debug mode for problems with image processing, where only this runs without other program parts. For more difficult problems, we have often logged program states, sensor data, and variables via print to the console and used this data to find bugs in our code. Thanks to our GUI (further details in Section b.), which also displays all relevant data (see Fig. 11), we can now use screen recordings, which runs on almost all runs to identify the cause of failures retrospectively, e.g., if they only occur rarely, which allows to quickly reproduce such situations and find ways to fix them. We always test the robot on a large course in the school after a fix was implemented to make sure that no new problems arise.

With the two AIs, finding and solving problems often had the same structure. If, for example, we have noticed that victims or the silver stripe were not recognized well under certain conditions (such as lighting conditions or backgrounds) we took several hundred pictures of these situations, including at many different competitions, added them to our existing data set, retrained the AI and then tested it again in this situation. This often solved the problem. See Fig. 12 for an example of the application of this technique, where on the left the black ball in the shadow is detected with low confidence, and a silver ball is detected even though there is none. The image on the right shows the newly trained AI, with new data of the situation that improved both problems.

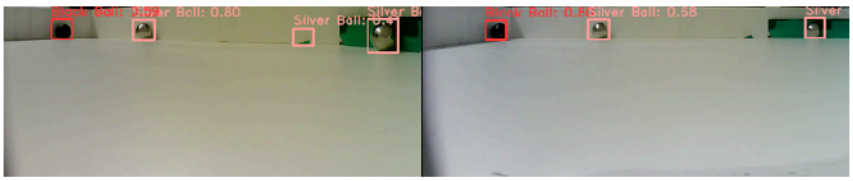

Figure 12: Improving the AI's performance by adding training footage

6. Conclusion

After 3 years of developing, redesigning, and improving the robot, we can safely assume that we have reached most of our goals, especially this year. From the start of this season until now, we have put much more thought into every aspect of the robot, from design to functionality. Through the experience gained over the past 2 years, we were able to test and build a robot that is now able to compete on a worldwide level. We found and used different ways for easier collaboration between the team members, which resulted in faster development. Last year's robot had many small issues that plagued its performance. This year, we learned from all these mistakes, and we think the robot is now much more reliable. We have eliminated nearly all hardware issues by building a more robust robot. The software has also matured a lot with the new structure, which is not only easier to work with but also faster, utilizing multiprocessing to split up work on the entire CPU.

We look forward to competing in the World Championship to not only test the robot against more difficult courses, but also exchange knowledge and ideas with other teams from all over the globe.

Appendix

If you want to learn more about our robot, feel free to visit these sites:

- Our YouTube channel
- Our Github oranisation (Code will be available once we finish all our competitions)

References

a. Software tools & Platforms

- Arduino
- Github
- Google Colab
- Fusion 360
- Python
- Raspberry Pi OS
- Roboflow

b. Libraries

- CustomTkinter
- gpiozero
- NumPy
- Numba
- OpenCV-Python
- scikit-image
- Ultralytics

c. Hardware

- 12 V DC Geared Motor
- Arducam B0268 16MP Wide Angle
- Arduino Nano
- 11.1 V/7.4 V Conrad Lipo Batteries
- Diymore 35 kg Digital Servo
- EMAX-ES09MD Servo
- Coral USB Accelerator
- KY-019 Relay
- L298N Motor Driver

- Pololu irs16a/irs17a 50 cm/130 cm Infrared Sensor
- Raspberry Pi 5
- Raspberry Pi Camera Module V3
- revoART, 12V COB LED strip
- XL4015 Step-Down Converter
- XL4016E1 Step-Down Converter
- XL6009 Step-Up Converter