ROBOCUPJUNIOR RESCUE MAZE 2026

TEAM DESCRIPTION PAPER

Skarlax Robotics

Abstract

Skarlax Robotics is a three-member team competing in the RoboCupJunior Rescue Maze League. Our robot is a compact, modular platform built primarily from FDM 3D-printed components and three custom PCBs. Its drivetrain consists of four self-poured silicone wheels driven by Pololu Micro Metal Gearmotors, mounted on a freely tilting front axis that allows the robot to navigate bumpers and debris reliably. The main computing is handled by a Teensy 4.1 microcontroller running C++, supported by an ESP32-C3-Supermini for mapping in MicroPython, and two OpenMV AE3 cameras for victim detection. Distance sensing relies on eight Adafruit VL53L4CD time-of-flight sensors plus one long-range VL53L4CX at the front. Rescue kits are deployed via a centrally positioned MG90 servo mechanism for accurate placement near victims. The mapping algorithm uses Dijkstra's pathfinding and is simulated across a test suite of 680 test worlds, while victim detection combines a trained neural network for letter victims and custom circle-detection logic for cognitive victims. What sets our robot apart is its clean separation of concerns across hardware and software layers, its extensive use of automated testing including GitHub CI integration, and an iterative development process that produced three distinct robot versions within a single season, each directly addressing failures identified in competition.

1. Introduction

We had some time problems with this TDP, as we originally didn't know the deadline and only saw it a couple of days ago. Nevertheless, we still tried to provide as much information as possible in this document and keep our past text passages as up to date as possible. If anything is unclear and you want to know more, please just contact us, we'd be happy to provide you with more information.

a. Team

Lars

Lars (18y/o) is the oldest member of the team and has been participating in the RoboCup Junior for 7 years now. He formerly participated in the Maze Junior League, in which he and his former team qualified for multiple German Open and once the European championship. In the past couple of years, he built maze robots for the normal Maze League and already competed in the 2024 European Championship. This year, he is primarily responsible for the robot's hardware, including PCB and 3D Design. He also writes the software for some hardware level applications and for the Cameras.

Quentin

Quentin (15 y/o) is the newest member of the team. He formerly competed in a different Maze Entry Team. He joined our team this year and is now responsible for the high-level code of the robot, including our driving algorithms and the mapping.

Sören

Sören is the third member of our team. Since he is at a student exchange this year, he is not at the competition.

2. Project Planning

a. Overall Project Plan

Our objective

At this point, we want to distinguish between personal goals of our members and the team goals:

Personal goals

For us, RoboCup is not only about the days at the different competitions each year. It's also about learning new things, working together in a great team and personal improvement.

We make sure that everybody works on the things he's actually interested in and can learn something new every day. As a team, we meet twice a week at our school's robotics club, each Tuesday and Friday. This allows us to meet on a regular basis and exchange the current state of the project. In the weeks before the competition, we often also meet at home to continue working apart from the school's robotics club.

Team goals in the competition

As a team, we try to give our best to be successful in the competition. It's always nice to meet/exchange ideas with other teams at the German national competition and sometimes even further international events.

Our project plan

For our everyday workflow and the team planning, we're using GitHub Issues and a GitHub Project. For the basic structure of our workflow, we're using Milestones on GitHub to sort our issues and work more structured. For this year's competition, our plan looked like this:

Milestone	**Deadline**
Robot V1 hardware ready	15.12.2025
V1 software ready	20.01.2026
Local qualifying tournament 31.01.2026-01.02.2026	
Robot V1.1 hardware improvements done	19.02.2026
Working version for the German Open	11.03.2026
German Open 11.03.-14.04.2026	
Hardware improvements for the EM	04.05.2026
Working version for the European Championship	29.05.2026
European Championship 03.06.-06.06.2026	

By adding these deadlines to our GitHub Repository, we managed to have an overview of the progress, as GitHub automatically calculates the progress. Just because they're above each other in the table doesn't mean we didn't work on them simultaneously, as hardware and software are mostly developed separate from each other by different team members. Hardware deadlines are always earlier as not everything can be tested with its own testing software, and the software needs to be tested on the finished hardware.

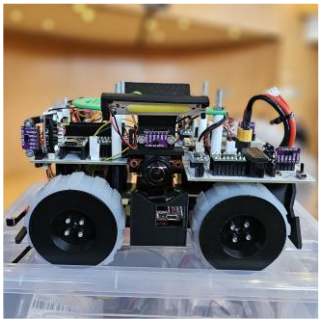
To work efficiently and keep track of what we must do (next), we're using the Issues in our Repo. This allows us to simply create a new issue in case something needs to be implemented/improved/fixed, which can then be digitally assigned to a team member who then automatically sees his tasks in the Project. We can also add deadlines this way, which allows us to keep efficiency high.

We're now using GitHub Issues as we had some struggles in the past years, which has now definitely improved, but we're sometimes still missing deadlines as school and other private things get in the way.

b. Integration Plan

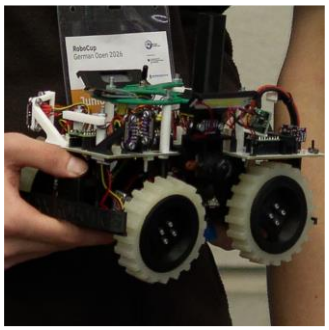
We knew that we wanted to use as many components as possible from the previous year, as we already knew how to work with them and about their reliability. We also went for some new components, including the Teensy, the motors and in the newest version of the robot even the distance sensors and cameras.
For this year's competition, we built our robot and improved it continuously. In the following, we're giving a short introduction to the (major) different versions of the robot. This paper will then primarily cover V2, the latest version, as this is the one we're using for the European championship and since it's the newest version, it also includes fixes for the problems with the older versions.

Version 1



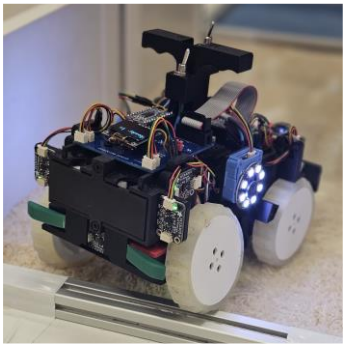
The very first version of our robot was built for the local qualifying tournament (back then we thought we would use this for all upcoming tournaments of the season). It also had the first version of this year's wheels, which we quickly found out were only big enough for 1.5cm bumpers, which we then tried to fix by software. It also had the original OpenMV H7's we used the last two years, which were fast enough and reliable. The design was less modular, as it was primarily structured into the lower section with the tilting axis and two base plates, on which the PCB was located. The PCB was the biggest part of the robot and therefore also marked the outer boundaries of the robot.

Version 1.1



For the German Open, we made several improvements to our original design, such as bigger wheels and a new camera placement. We also added a new push button to the whole width of the front responsible for detecting obstacles and other objects. We had some problems with the ejection mechanism, which we (once again) tried to fix by improving the software.

Version 2



After the problems with the ejection mechanism, we decided that we would need to move it closer to the robot's center to assure proper rescue kit placement. We therefore had to restructure the robot's platform, as this would not really work with the idea of a single full-size PCB. We therefore switched to a more modular design by adding a third base plate by reducing the size of the existing ones and moving the ejection mechanism with a better design to the robot's center. We wanted to make as little change in total as possible, which is why we kept most of the components. Only the cameras have been updated due to their smaller size now, which allows them to be installed more intuitively. We also added encoders to our motors, which we want to use for better driving accuracy. The full breakdown of this design and the electronics is below.

3. Hardware

Our robot is primarily built out of FDM 3D Print components and three PCBs. We designed the robot in a modular way, allowing it to change components fast if needed. Our three PCBs host most of the hardware, with only the cameras, sensors and motors being mounted apart from them. For the movement, we choose 4 self-poured silicone wheels each being mounted to its own motor. Our ejection mechanism uses its own servo. For

electronics, we choose a Teensy with an ESP as secondary computing unit. The Teensy is connected to all the other components, while the ESP is only connected to the Teensy via UART. This makes the Teensy the main decision-making machine while the ESP is only responsible for the Mapping.

a. Mechanical Design and Manufacturing

Main Structure of the robot

As previously explained, we printed most of our robots' components with an FDM Printer. This allows us to modify and improve the robot at a high speed without having to wait long time for the manufacturing procedure. The Robot is separated into three greater areas:

- Front Area with battery, sensors, Front-PCB and the tilting front axis
- Middle Area with the rescue kit ejection mechanism, the handle and the cameras
- Rear area with mounting space for the Power-PCB and Main-PCB as well as the rear axis and some more sensors

In the following, we are going to explain the robot's mechanical design by taking a closer look at the three separate areas. We also added some pictures to the appendix to clarify the descriptions. Before doing so, the following table will give an overview over the components in the robot, allowing us to shorten the following text by just referring to "the motors" instead of the full names.

Name	Module	Position/Count/Further information
Motors	Pololu Micro Metal Gearmotors 1:150 12V HPCB with Encoders	Two are mounted to the back base plate while the other two are located at the tilting axis.
Ejection Motor	(No-Name) MG90 Servo Motor	We use an MG90 (SG90 with metal gears)
Push Buttons	---	Simple push buttons for obstacle detection
Main-PCB	---	The PCB mounted at the back, on top of the Power-PCB.
Power-PCB	---	The PCB mounted at the back of the robot.
Front-PCB	---	The PCB mounted at the front of the robot.

Front Area

The front area of the robot consists of four main 3D Printed Parts as well as one PCB (see at 3b).

The tilting axis is mounted between two surrounding components and consists only of the two front motors as well as the bar that's allows the whole axis to tilt. The axis can tilt freely within about 12 degrees in both directions from its normal horizontal position. This allows the robot to move more easily over bumpers and debris, while keeping the overall axis design as simple as possible. We also thought about installing springs to improve the robots' driving dynamics, but since the tilting axis worked well enough, we decided to keep it as simple as possible.

The two surrounding base plates are being screwed together (we use a lot of heat inserts) after installing the tilting axis. This allows us to build a modular, but strong foundation for the robot. The base plate to the middle contains the mounting point for the color sensors, while the luminance sensors for the black-tile-detection are mounted to the front assuring early detection of black tiles to stop the robot in time. We also added two push buttons to the front of the robot, in case the distance sensors do not see obstacles or walls (even though we had no previous problems with transparent walls).

On top of these three components, our battery cage is mounted. The cage contains the robot's battery and protects it from damage. The front PCB as well as the four front Distance Sensors are also mounted to the battery cage.

Middle area / Ejection Mechanism

The middle area only consists of two components, the base of the ejection mechanism and the tower. These components are both mounted to the connecting components of the back and front area.

The base of the ejection mechanism contains the MG90, an angled slide for the rescue kits and space for the gyro, which is mounted at the center of the robot.

The tower contains space for 8 rescue kits and two mounts for the power and LoP-Switch. We also added an ergonomic design to make the robot easier liftable and assure a good grip (we actually dropped our previous robot once, which was quite a shock).

On both sides of the tower, the cameras are mounted, including an RGB-LED-Ring that assures proper lighting for the cameras.

Rear area

The rear area consists of another three FDM Components and two PCBs.
The base plate is the mounting point for both the rear motors as well as the two PCBs. On both sides, there are side panels mounted, which have the distance sensors mounted on the corners and protect the PCBs from damage.

Testing & problems occurred

We also had some problems with the hardware, which we tried to solve.
At the national competition, we had some problems with bumpers on the ramps inside the danger zone. Due to this very high incline, the robot got stuck. We solved this problem by adding a chamfer to both the front and the rear of the robot.
We also had a problem with the rescue kit deployment. Even though it was very robust and worked fine, the position of the deployed rescue kits was not always close enough to the victim. Therefore, we moved the rescue kit deployment mechanism closer to the middle of the robot, assuring that rescue kits will now always be perfectly delivered.

b. Electronic Design and Manufacturing

The robot's electronics are split onto three separate PCBs, which accommodate all but the sensors and the cameras. Before we explain the positioning of the components, this is the full list of the components in the robot:

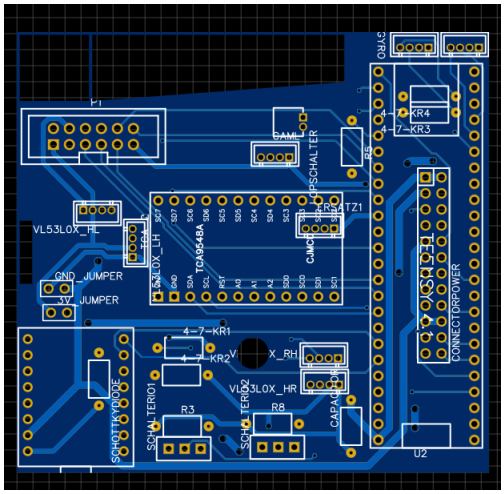
Name	Module	Position/Count/Further information
Main Controller	Teensy 4.1	We have chosen this microcontroller for its very fast chip (600Mhz) and the variety of ports (8 UART, 3 I2C and 35 PWM).
Secondary Controller	ESP32-C3-Supermini	This really small microcontroller is still fast and reliable and runs Micropython. It is connected to the Teensy via UART and handles the mapping.
Main Distance Sensors	Adafruit VL53L4CD	We are using eight of these very fast distance sensors. They are specifically designed for this low-distance application with fast reading rates. We mounted two of them on each corner looking into the two directions. Therefore, we're having two sensors on each side to detect walls and align with the walls.
Long-Distance Sensor	Adafruit VL53L4CX	There is one of these sensors mounted at the front of the robot. The sensor is designed for ranging up to 6 meters, allowing us to even get a signal when standing on ramps or other long "hallways" in the maze.
Cameras	OpenMV AE3	We upgraded from the older OpenMV H7, due to the smaller size of the new cameras while still being in the same performance league. This allowed us to build the robot even smaller and mount the cameras at the height of the ejection mechanism. They run MicroPython on their 400Mhz Chip, which also includes an NPU for faster NN execution.
Luminance sensors	Pololu QTRX-MD-01A	They are installed to detect black tiles by using the reflectance. We're still having some trouble with false detections when the robot is standing on bumpers, therefore we may change back to our previous sensors.
Color Sensor	(No-Name) TCS34725	We're using this sensor due to his small formfactor (and price) and his good performance.
Battery	SLS XTRON 1300mAh 3S1P LiPo	This battery is a good compromise of capacity and size. 1,3Ah are enough for testing and the runs, but its still small enough to fit between the wheels of the tilting axis.

The three PCBs all serve a different usecase. We choose to separate them like this:

- Main PCB: This hosts the Teensy and the ESP as well as the connectors to the other two PCBs and the rear sensors.
- Power PCB: The power PCB has the battery connector, the motor drivers and their connectors and the voltage regulators.
- Front PCB: To clean up the cable mess, we added a third PCB at the front of the robot that connects to the main PCB and serves as a splitter for the front sensors.

We designed the PCBs using EasyEDA (Pro) and ordered them from a Chinese manufacturer (JLCPCB), since we do not have the equipment needed for printing them at home. For a better overview now a detailed description of each PCB:

Main PCB



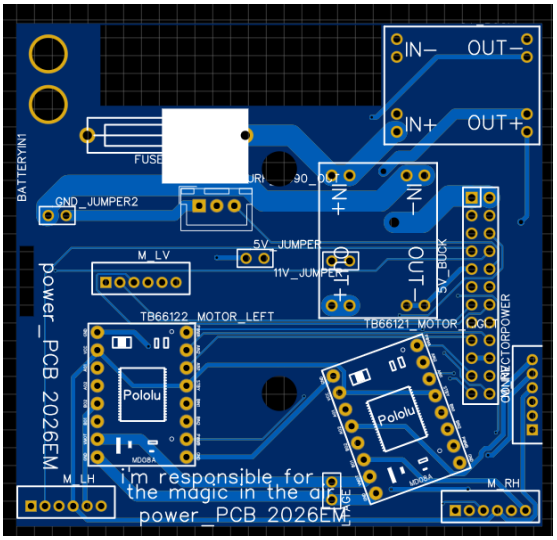
The main PCB is approximately 70x70mm and is home for the following components:

- Teensy 4.1
- ESP32-C3-Supermini
- TCA9548A I2C Multiplexer
- 24-pin Connector to Power PCB
- 12-pin Connector to Front PCB
- JST-Connectors for rear distance sensors, gyro, cameras and switches

Since these are all components with low power consumption, we chose smaller track widths for this PCB to allow a denser placement of the components. After problems with the previous PCB, we split the sensor up on three different I2C-Busses (on for Main, one for Front and the gyro has its own) and added Pull-Ups on each bus, assuring a stable data connection.

A diode at the ESP makes sure we can flash it while the robot is running without any risks.

Power PCB



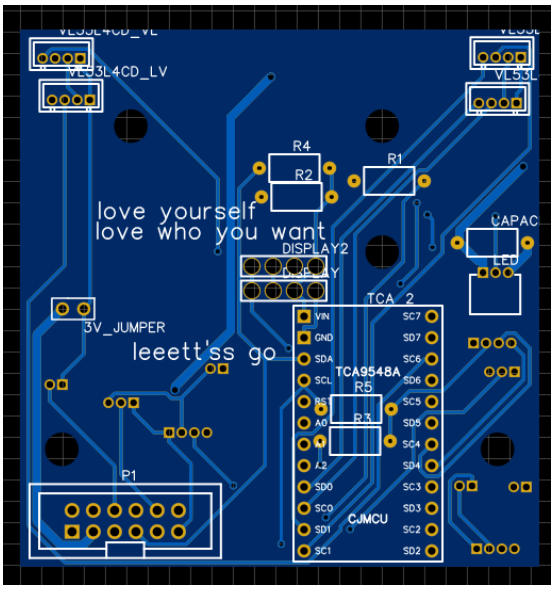
The power PCB is located directly under the main PCB and therefore also has a size of 70x70mm. It hosts the following components:

- XT60 battery connector
- Two voltage regulators (one for 5V, one for 3.3V)
- Two TB6612FNG motor drivers
- A fuse to save the robot from short circuits
- Connectors for the motors as well as a voltage display and the SG90

For all the high-power consumption components, we designed this separate PCB with bigger track widths for up to 5A. We choose an external power switch in the cable from the battery, as this removes the need for a connector on the PCB, which would have

been a problem due to the limited space. The 24-pin connectors is connected to the Main PCB using some self-designed spacing headers and run the power as well as all the motor data and PWM to the Teensy.

Front PCB



The front PCB is approximately 65x65mm. It features the following connectors:

- TCA9548A I2C Multiplexer
- 12-pin Connector to the Main PCB
- Connectors for distance sensors, LEDs and front color/luminance sensors

This PCB was designed to reduce the number of cables running from the back of the robot to the front and to add some space for a second i2c multiplexer. Besides the i2c connectors for distance and color sensors, we also added analog inputs for the luminance sensors and power outputs for LEDs.

PCB Testing / Development Process

We developed these PCBs with the experience (and the problems) of the previous PCBs we built in mind. We knew that we had to improve the power lines on the PCB to allow higher currents for the motors and the LEDs. We also knew we had to do something to separate the logic hardware, which is quite sensitive, from the motor drivers and the LEDs. To do so, we designed separate PCBs, as this would also allow us to use the space more efficiently.

To make sure that the PCBs would actually fit before ordering them and shipping them all the way from China, we exported them as 3D Models and printed them with our 3D Printer. This allowed us to check for tolerance and accuracy of the design. After we fixed some minor things, we ordered the PCBs at JLCPCB.

4. Software

a. General software architecture

Our Code is split apart onto Teensy, ESP and the Cameras. The Code on the Teensy runs in C++ based on PlatformIO, while both ESP and Cameras are programmed in Micropython.

Code on the Teensy

The Teensy runs the software for all our sensors and communicates with OpenMVs and the ESP32, from which it gets the command of what to do next, e.g. driving forward or turning left. The ESP executes the algorithm and sends the Teensy commands for driving.

The serial communication between the Teensy and the ESP as well as the Cameras use our own protocols to achieve steady and robust communication. The protocol has 13 different commands and uses a check digit. It works with an eight-bit number sent to the Teensy by the ESP. The first four-bits represent the command to be executed, and the last four-bits represent the checksum. After the protocol the payload follows. The payload is trimmed by spaces. It is also used for calculating the checksum. First the payload is made to one number by leaving out the spaces. Afterwards the digit is reversed ($\mathrm{di} = \mathrm{i} - \mathrm{th}$ digit from the right (0-indexed)). Then the digit is looped through and if the single digit is even it is added to the total, but if else the number is multiplied by two and if the product is greater than 10, 9 will be subtracted and then the sum is added to the total. At last, the number is trimmed to four-bits to make an eight-bit number with the command number. I came up with the idea as I was asked how to make communication secure.

$$S = \sum_{i=0}^{n-1} \begin{cases} di & if\ i \equiv 0\ (mod\ 2) \\ 2d_i - 9 \cdot \left\lfloor \frac{2d_i}{10} \right\rfloor & if\ i \equiv 1\ (mod\ 2) \end{cases}$$

Our robot is programmed in Platform IO/C++. We use common libraries for the sensors and build our own ones for the motors and the I2C Multiplexer. For debugging we use a display where we add our own display skin. With the display we have an overview of the current state of the robot. For calibration, we're using the Teensy internal EEPROM to store values (RGB and C).

For driving and turning, we use the gyro and the TOF sensors to make sure the robot turns/drives properly without accidentally getting stuck or driving into walls/obstacles.

For driving one Tile we use the TOF sensors to calculate how far we need to drive. Encoders are used to limit the driving range. The turning uses only the gyro.

The Teensy has mainly code for driving and the sensors, but apart from the commander there are simple functions for depth first algorithms prioritizing right/left. These functions are used after the main algorithm on the ESP is done.

We are currently working on simulating the robot for faster debugging and testing. Our project programs are split into separate folders for more structure: base (serial/display...), algorithm (command handler, back up algorithms), drive (driving functions), sensors (sensor codes). Mainly we have objective classes for the sensors.

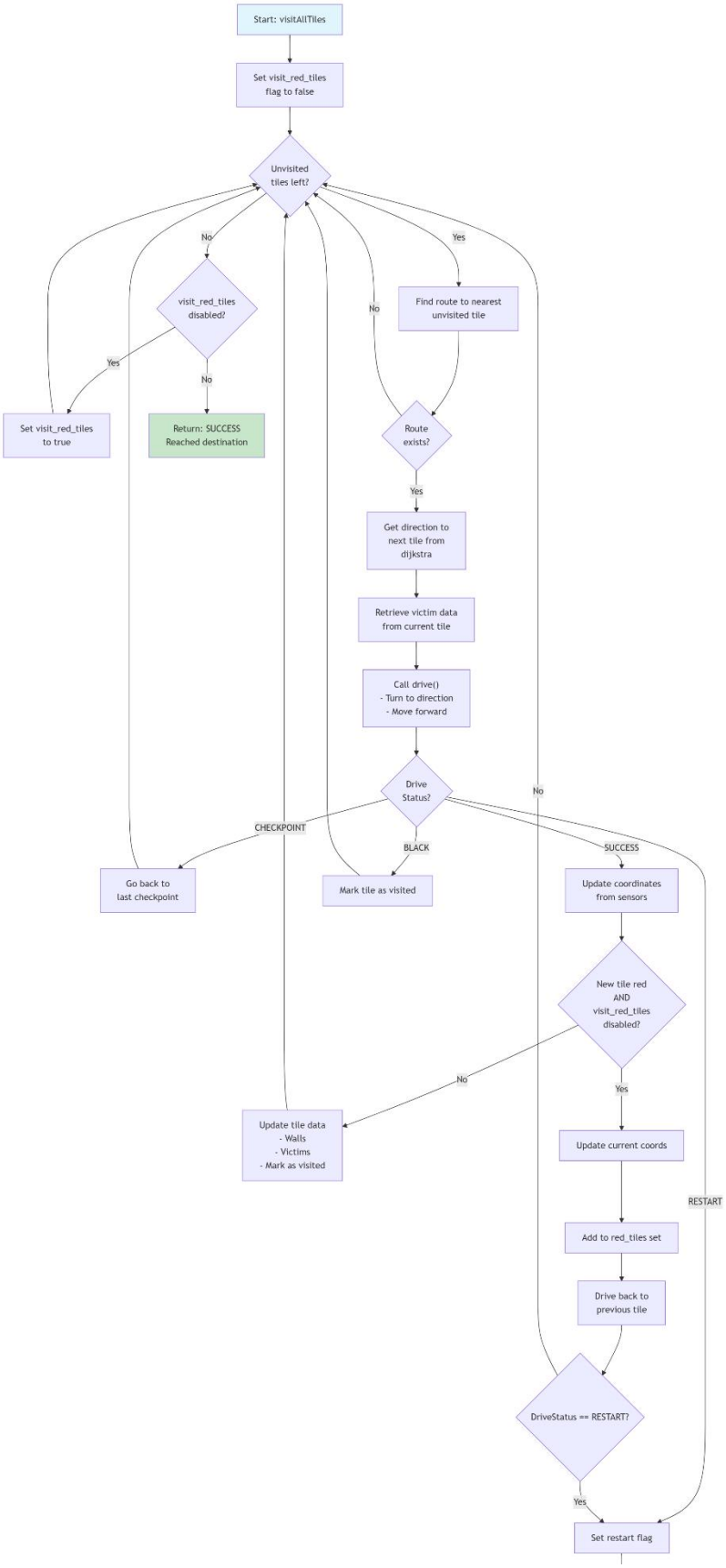


Figure 1: The robots exploring mechanism

Code on the ESP

The Mapping Algorithm is written in MicroPython and runs on the ESP microcontroller. It first explores all tiles outside of danger zones, then it continues with the ones inside those zones. In the end, the robot navigates back to its origin. It uses Dijkstra's path-finding algorithm to find the optimal route to the nearest tile which should be visited in the respective step. This reduces driving time by preferring routes with longer distances in order to avoid blue tiles. The algorithm is fully tested with a simulation featuring 680 different test worlds. In combination with logging of all important steps, it greatly simplifies debugging.

Code on the Cameras

We're using OpenMV AE3's with their default software environment. We only accept victims when they are (almost) centered in the frame as the lighting conditions are the best around the image center. For the cognitive victims, we added a feature detection that detects the circles. By using the integrated VLX-Distance Sensor, we can assume which size the biggest circle needs to be and therefore sort out everything else. We then scan pixels within the circle bounds to check whether they're colored or white. If everything is colored, we scan some pixels around the circle to look for the white background of the wall. If both of these scans are successful, we scan pixels within the inner circles and get the color by calculating the average. We then already calculate the value of the victim to check whether it's a real or a fake target. In case it's real, we send it to the Teensy, which then has to stop and eject rescue kits if needed.

For the letters, we trained a black and white neural network with ten thousands of images of the maze. If the NN matches at a given percentage (currently 95+), we accept the victim and send it to the Teensy.

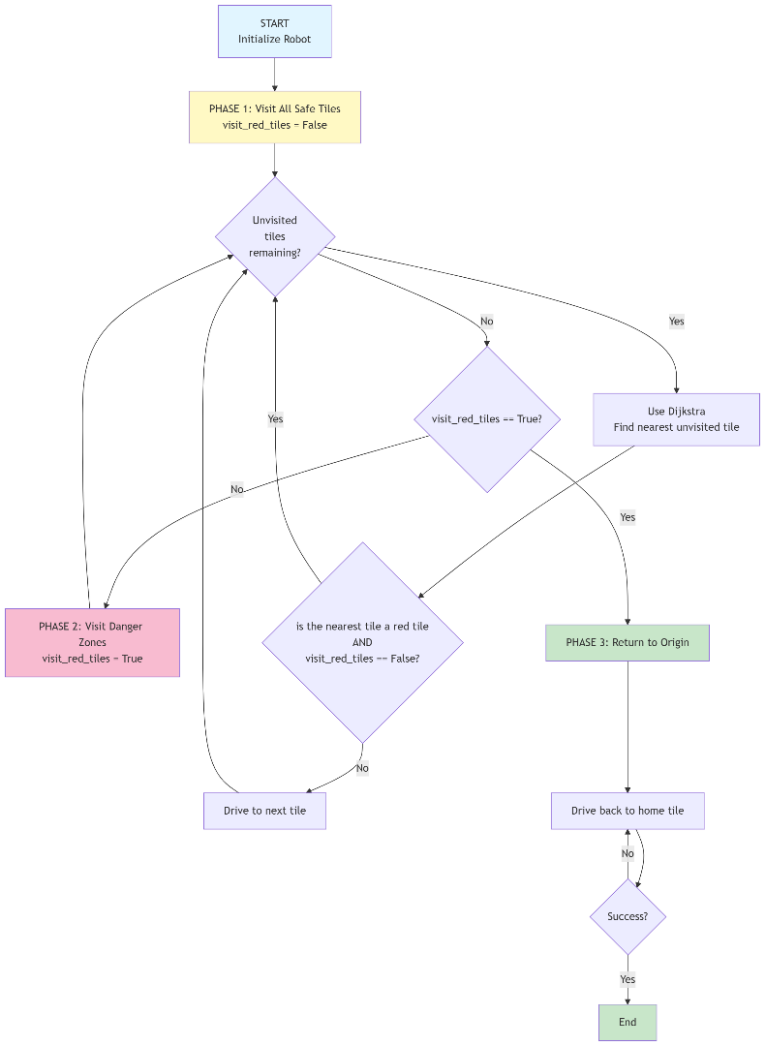


Figure 2: Overview over the mechanism of the ESP Code

We also added a check with the internal distance sensor so the camera only searches when there is a wall that could contain a victim. Our previous camera didn't have this feature and therefore we had a lot of false alarms sent to the Teensy, which cost us precious time we need for the PID correction.

Innovative solutions

To avoid as many problems as possible, we added test cases to our software this year. They run directly on the GitHub Repository and test the Teensy Code as well as the mapping. This allows us to make sure that there are as little problems as possible, especially with nasty problems like typos.

5. Performance evaluation

We tested our robot in the actual maze, and this is our robot's performance:

Test scenario/Challenge	Outcome/Results
Bumpers, Debris and Ramps	Thanks to the new wheels, we solved most problems with this. They offer enough grip for ramps and are big enough to drive over bumpers in danger zones. Only problem is when we hit a big bumper (2cm) at a steep angle. This still needs software side help to work.

Mapping	Mapping is working fine, as the robot can save victims to prevent double-detection and navigation is also working fine.
Ejecting rescue kits	This caused some problems as the space needed for the SG90 was used by an component on the PCB, but we fixed this.
Victims	Letter victims are working fine, the NN is doing an alright job. Only some problems with false detections, but we hope to fix this by adding more test images and maybe increasing the resolution in the future. Cognitive victims are still a problem, as the basic software implementation is ready, but it's hard to ensure thresholds are working fine in every environment. This needs further calibration.
Power consumption	No problems with the capacity of the battery so far.
Sturdiness	So far, no problems occurred with bumpers at high speed or driving into undetected obstacles.

6. Conclusion

Over the course of the 2025/26 season, we developed and refined a capable Rescue Maze robot through three hardware iterations and continuous software improvements. The modular three-PCB architecture proved its value by allowing targeted upgrades without full redesigns, and the centrally repositioned ejection mechanism resolved the rescue kit placement issues encountered at the German Open. On the software side, the Dijkstra-based mapping algorithm demonstrated reliable performance across hundreds of simulated test worlds, and the neural-network-based letter victim detection operates at an acceptable accuracy. Key remaining challenges are the cognitive victim detection, which still requires environment-specific calibration, and occasional false positives in the luminance sensors on bumpers. Looking ahead to the European Championship, we look forward to keep improving our robot so it meets our expectations. The season has been a strong learning experience in hardware-software co-design, structured project management, and iterative engineering.

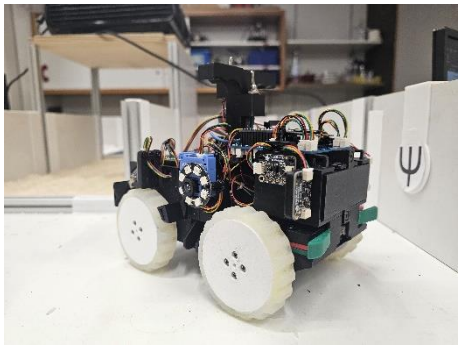
References

For software development, we used the following tools:

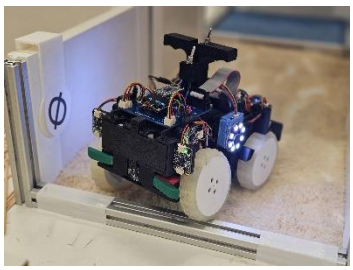
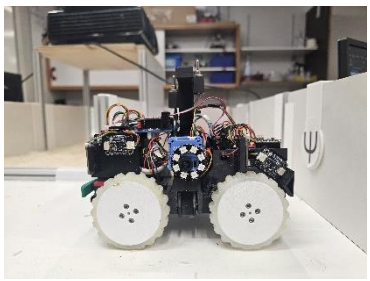
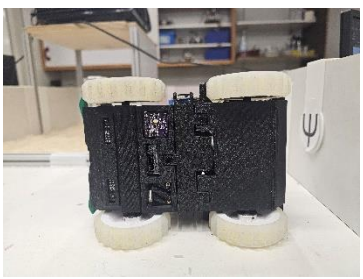
- CLion and PyCharm by JetBrains
- OpenMV IDE
- GitHub (for our Git and issues)

For hardware and electronics design, we used the following tools:

- Autodesk Fusion (Student License)
- EasyEDA Std & Pro (they are both free)
- Bambu Studio (only for 3D printing)



Images

From left to right: Robot eagle perspective, side of the robot, bottom of the robot and front/corner of the robot.