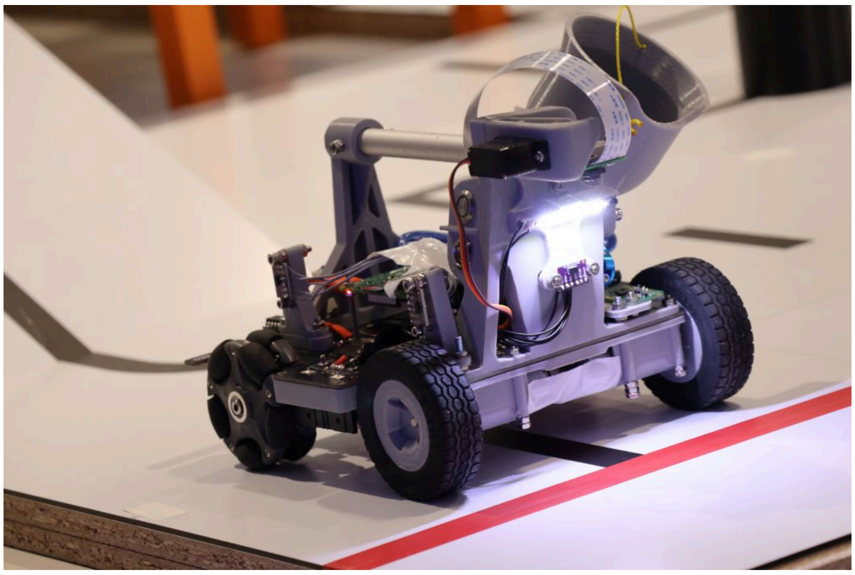
RoboCupJunior Rescue Line 2024

Team Description Paper

Team BitFlip



Abstract

Our robot competes in the Rescue Line sub-league and is the result of years of development. The two main capabilities that set the robot apart are the use of computer vision and AI, which allow it to perform each task in the discipline. The computer vision capabilities (line following, intersection detection and rescue-kit detection) have been in development since the 2019 season. Since 2022, we use AI to detect the victims and the entrance to the evacuation zone. The basic configuration of this year's robot consists of four driven wheels, a single tiltable camera mounted high up at the front of the robot and a cup for victim pick up in the front. The "brain" of the robot is a Raspberry Pi which interfaces with the rest of the hardware through a custom-made PCB. The robot has seven motors which are controlled by a separate microcontroller and five sensors, including the camera. All the self-made parts are modeled in CAD and 3D-printed. All software is written in C and C++ with additional tools in Python.

1. Introduction

Our team consists of Sven, Jan-Niklas and Lukas, each having a certain role within our team. Through years of collaboration, we found a workflow that works for us while everyone is able to bring in their ideas to the development of our robot. As we all have finished school now, we live all across the country which makes collaboration difficult. Nevertheless, we managed to efficiently divide the tasks between ourselves while making important decisions and some development and testing in video calls or in person.

While Jan-Niklas is mainly responsible for everything related to the organization and coordination of a successful RoboCup season, Lukas developed most of the electrical and mechanical parts of the robots and currently runs our YouTube channel as well as our website. From neural networks to computer vision, Sven has developed most of the software. Testing and assembly was mostly done by Lukas, as he has the best access to Tools and also owns a 3D-printer.

2. Project Planning

Since our team participated in some international RoboCup events over the past few years and this will be our last participation, our goals for this season were quite ambitious. The experience from previous seasons increased the development speed significantly, so we were able to get much more done in less time. We already knew what software we wanted to use (e.g. CAD software, PCB design software, common libraries) and were therefore able to start developing a completely new robot just days after the end of the season.

For us, the project planning phase begins with carefully reviewing all our competition runs. We identify every issue we can see in videos of the runs and also look at issues we identified during assembly and testing. Along with this exhaustive list of problems we also incorporate concepts for performance improvements into the new design. The main issues we identified were the accuracy of our neural network, the complex victim collection mechanism which we wanted to simplify and also the stability of our software and electrical systems, particularly the robot's power supply. We also had ideas for improving line following with a different controller.

With this we began tinkering with concepts and brainstorming which led to a first CAD design. At the same time we drew up the schematic for our main PCB. These designs were then iteratively improved and parts were tested until we had a complete design. On a smaller scale we use this process after each competition to improve for the next.

Despite all these new developments, we also wanted a very reliable robot, so we placed an emphasis on being able to reuse proven hard- and software from the previous years while also incoperating a modular design approach facilitating repairs.

Compared to previous seasons, we also laid out a rough development schedule to meet our goal of a "finished" robot (of course with room for software optimization) for the qualification tournament at the end of Feb 2024. According member labels are as follows: *Jan*, *Lukas*, *Sven*, *whole team*

Note: our actual schedule can be found in the engineering journal

Aug: *brainstorming phase*, *concept planning*
Sept: *first CAD design*, *design and order PCB*
Oct: *basic assembly done (chassis, PCB, motors)*, *PCB blow up test*
Nov: *first integration tests*, *start Arduino firmware development*, *communication test (Pi ↔ Arduino)*
Dec: *finish CAD design*, *assemble camera*, *test camera and basic linefollowing*, *in person meeting (Christmas holidays)*
Jan: *linefolling software done, start rescue development (victim pick up and corner detection)*, *add side distance sensors*
Feb: *finish rescue (mostly exit)*, *testing*
Mar: *performance evaluation of tournament, post-competition optimizations, write documentation (TDP, Engineering Journal, Poster, YouTube, website)*
April: *testing, German Opens*
May: *performance evaluation of tournament, post-competition optimizations, revise docs based on referee feedback*

The schedule was developed as a team and each step was intensively discussed as we tried to follow an iterative development process where each new component is tested and potentially revised. The last weeks before the competitions were mostly spent on testing our robot to ensure its reliability in constrast to former robots where major changes were often implemented the night before the competition.

We also put an emphasis on communication so we conducted elaborate video calls after major changes to both hardware or software to reflect and adjust if necessary.

3. Hardware

Given the current Rescue Line Rules, we compiled a list of the most important tasks our robot needs to accomplish as well as how those tasks could be solved. The possible solutions were then discussed as a team based on different criterias such as reliability, effectivness, cost, savety, ease of development and personal competences until we settled with an approach for the task (right column).

Requirement	**Possible solutions**	**Our solution**
Detect line	Light sensor Color sensor Photodiode Camera	Camera
Move robot	Drone two motors with chain two front motors + two back omni-wheels two front motors + two back motors with omni-wheels	4WD, two omni-wheels in back
Detect obstacle	Push button Camera HCSR04 ToF sensor (ultrasonic) VL53L0X ToF sensor (light) Lidar	VL53L0X
Detect victims (living/dead) Detect corners (red/green)	Camera Conductivity? Color?	Camera
Detect silver	Light sensor Conductivity? Camera	Camera
Pick up victim	Gripper Cup with rope (active: tightned by servo) Cup (passive: rubber rope)	Cup with rope
Start/Stop program	Push Button Distance? Detect hand on handle Detect hand in Camera frame	Push button

Based on this we drew a basic hardware diagram that fulfills all our requirements (Figure 1). The Raspberry Pi runs the main program and acts as the main controller, the only input it receives is from the start/stop button. It sends motor signals to the Motor Controller over USB which runs a second program, designed to be very simple with as little functionality as possible. It controls the four drive motors, the arm, rope and camera servos as well as our sensors.

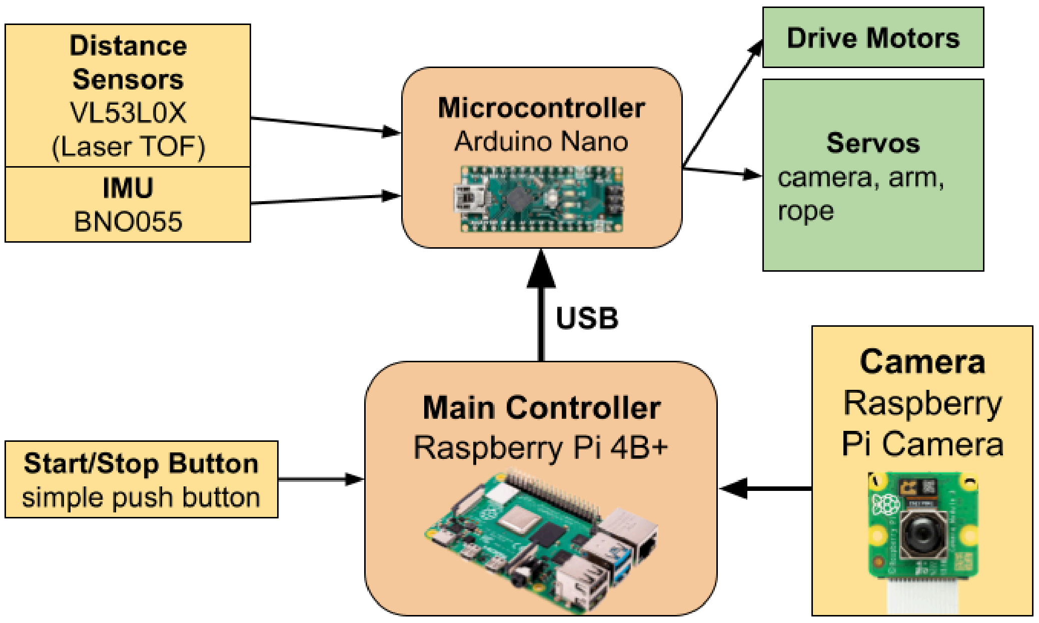


Figure 1: Basic Hardware diagram

3a. Mechanical Design and Manufacturing

The heart of the robot is the PCB which is also part of the structure. It is supported by a 3D-printed frame. Mounted on the bottom are four identical 3D-printed motor mounts, which have gone through several iterations since the first design in 2021 to solve problems such as deformation due to the robot's weight. All four wheels are powered by 12V DC motors. In the front there are LEGO tires, which have proven to have excellent traction, mounted on two 3D-printed rims which compared to last season now utilize two screws to mount to the motor shaft for redundancy. The back wheels are omni wheels that can slide sideways while still driving the robot forwards. This results in a center of rotation of the robot around the front axle, enabling the robot to follow the line more accurately. This is because the camera field-of-view only rotates around the line and does not move side-to-side when the robot follows the line.

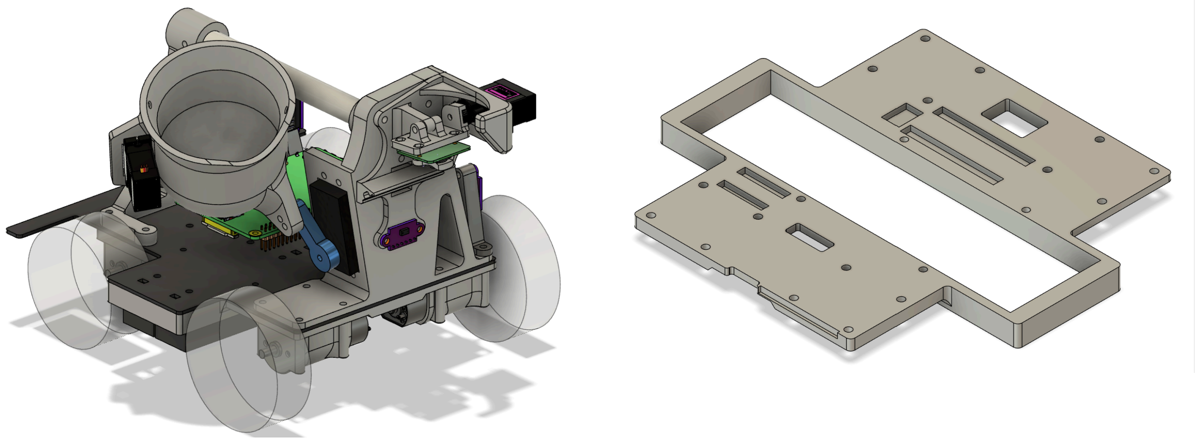


Figure 2: The robot assembly and chassis in Fusion 360

The main PCB has mounts for the Raspberry Pi and the Arduino Nano as well as other electrical components such as the H-Bridges, buttons and power supply. The mount for the four Li-Ion 18650 batteries is soldered to the bottom of the PCB and protrudes through a hole in the chassis. Together with the four motors this gives the robot an extremely low center of gravity.

The front of the robot has a mount for a powerful servo that moves the arm, which has a cup on the end that encloses the victim with a rope that is tightened by a small servo. This is an improvement to the claws from last year in robustness and enables the robot to more reliably pick up victims next to walls. The arm's plane of rotation is tilted 10° to allow the arm to clear the central camera.

For detecting the obstacle and the walls in the evacuation zone we use IR distance sensors. There are two sensors on the right side of the robot which allows us to follow a wall more accurately than with one sensor.

We use one camera which is tilted up and down by another small servo. Below the camera is an LED light to provide optimal lighting conditions for line following and to eliminate shadows in front of victims.

We removed the storage tank from last year. It only complicated repairs and did not provide us with any time savings in the evacuation zone. It also forced us to use an awkward position for the handle, which we could move to the center for this year's robot.

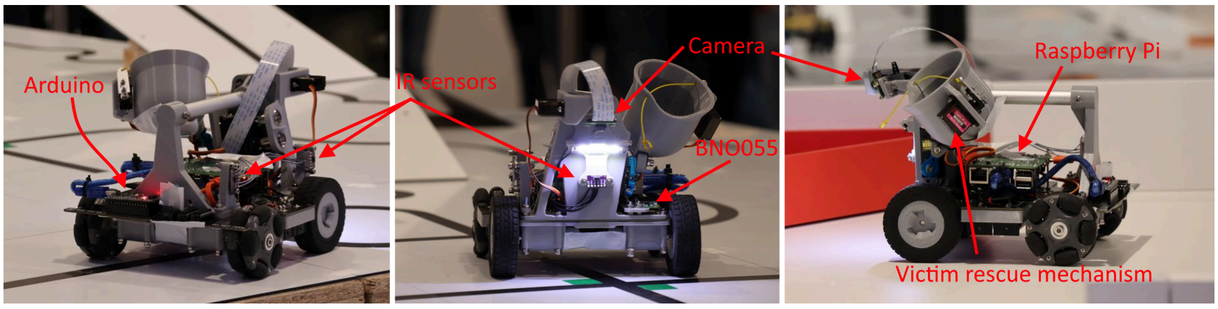


Figure 3: Overview of the robot

3b. Electronic Design and Manufacturing

Regarding the general hardware design of our robot (Figure 1), we use a Raspberry Pi 4B that is responsible for image processing and general program flow and could therefore be described as the brain of our robot. The robot also has an Arduino Nano in a Master-Worker configuration using USB that receives simple motor commands to control our actuators and can report sensor values back to the Pi. In previous years, we did not rely on a separate microcontroller for driving motors, but since the Raspberry Pi does not feature enough hardware PWM channels, we decided to add a separate microcontroller. The result was a large improvement compared to the software PWM we used for last year's robot. The motors can now be driven at much lower speeds and more accurately.

While we try to follow the simplistic "vision only" approach wherever possible due to more flexibility and independence from unreliable sensor measurements the robot still has three VL53L0X laser time-of-flight distance sensors to detect the obstacle and for easier navigation in the evacuation zone as well as a BNO055 IMU used for accurate turning.

Our custom PCB we ordered from a professional manufacturer is now powered by four 18650 Li-Ion cells connected in series. Their voltage is converted to 5V using a step-down module *(FEICHAO 8A UBEC)* to power the electronics and servos. In contrast to last year, we now supply our 12V motors with the batteries directly, therefore removing the need for unreliable step-up/step-down modules. As our batteries exceed the maximum rated voltage of our motors, we use a PWM feedback loop that monitors the battery voltage and adjusts the duty cycle accordingly, so our motors are never powered with more than 12V.

To control the drive motors, we replaced the commonly known L293D H-Bridges with the SN754410NE, as they don't get as warm during normal operation and should therefore increase longevity.

The development of the PCB started by noting the things we learned from the electronic design of the robots of the last three seasons. We started testing parts of the electronics on a breadboard. Then, we designed the PCB in EasyEDA and had our design critiqued by the other team members before actually ordering it on JLCPCB. After manufacturing, we began by testing the circuits with a multimeter and started installing components. Before mounting the Raspberry Pi, we tested the power supply, the motors and the motor controller.

We tested the electronic system with all the actuators and sensors before assembling the robot. During this phase, we also wrote and tested the parts of the software that directly interface with the hardware on a low level.

During the initial design phase we tried to facilitate hardware debugging which is why our PCB now features measuring pads for all important voltages as well as grounded mounting holes making it easier to attach a multimeter or oscilloscope.

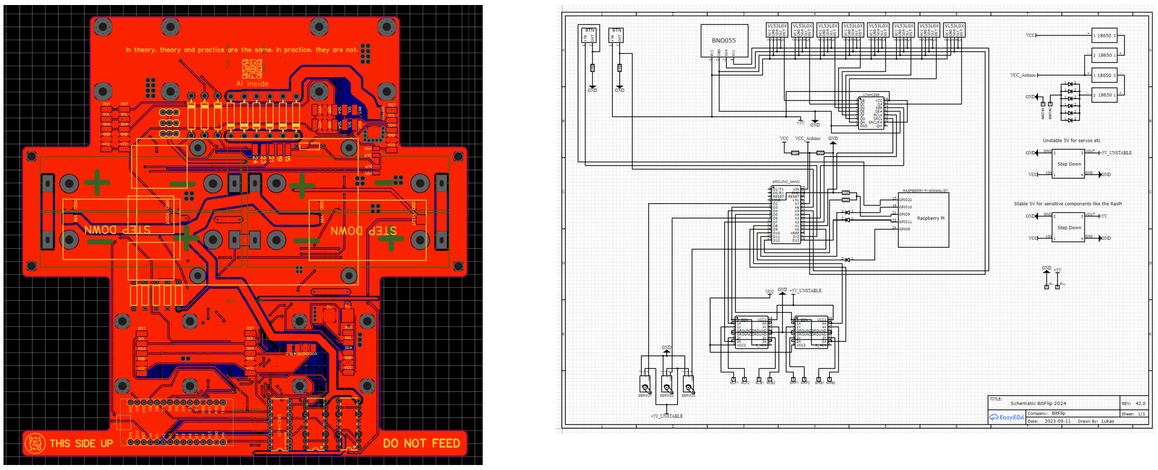


Figure 4: The main PCB design and schematic in EasyEDA

4. Software

4a. General software architecture

Below is a basic diagram of the robot's and auxiliary software. The main program is written in C. We use TensorFlow Lite to run inference on our Machine Learning models. The motor control program is written in Arduino C++. In addition to these two programs, we use Python on Google's Colab notebooks for dataset management and NN training.

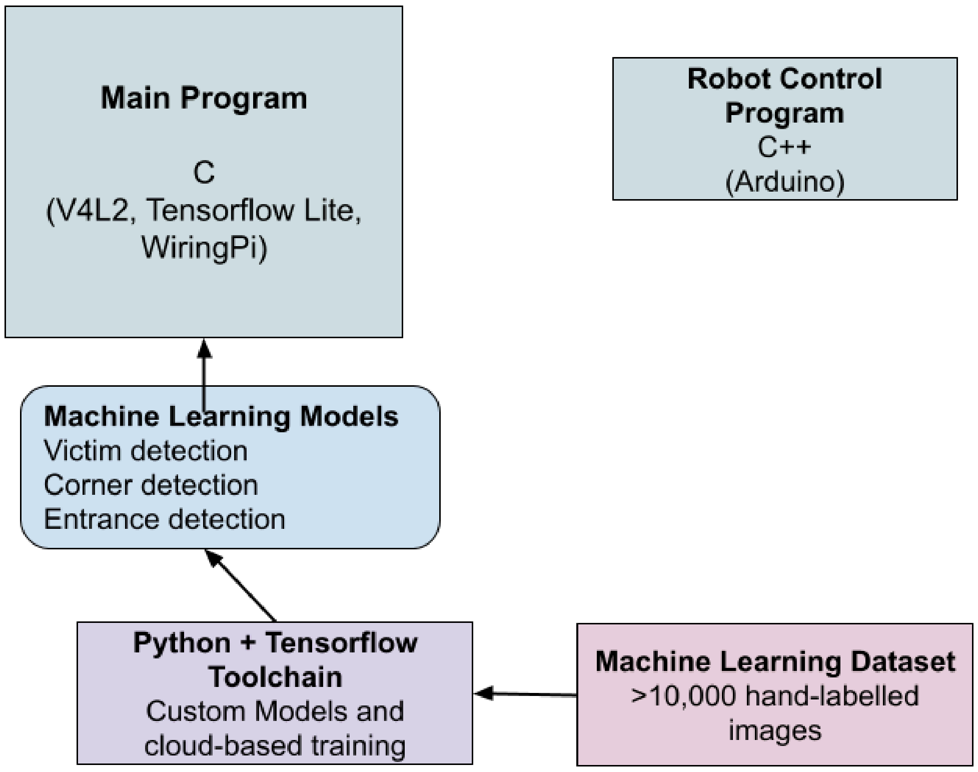


Figure 5: Basic software diagram

The program runs on three threads: The main thread handles button presses and starts and stops the robot. The second thread is essentially the main program and does computations and the main logic. The entrance detection NN runs in a separate thread to avoid performance penalties for line following.

Line

This part handles line following, intersections, obstacle avoidance, entrance detection. The main line method runs up to 120 times per second. The things it does are illustrated in the following diagram:

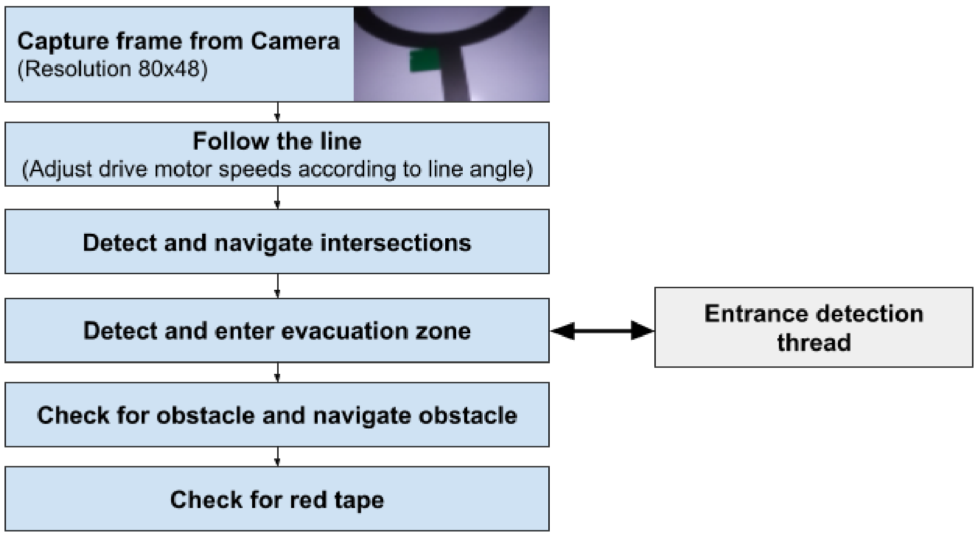

Capture frame from Camera
(Resolution 80x48)
Follow the line
(Adjust drive motor speeds according to line angle)
Detect and navigate intersections
Detect and enter evacuation zone
Entrance detection thread
Check for obstacle and navigate obstacle
Check for red tape

Figure 6: Flow diagram of the main line-following routine

The individual subroutines are explained in more detail under section 4b.

Rescue

This is the part of the program that takes over control of the robot once it has entered the evacuation zone, and until it has found the exit. The robot mostly operates from the center, where its camera can see all parts of the zone. The robot repeatedly turns to search for victims. When it has found one, it incrementally approaches the victim until it is close enough, picks it up and delivers it to the right corner. The robot repeats the process until all the victims are rescued. Then, the robot searches for the exit and returns to following the line.

4b. Innovative solutions

Wireless programming

Since the Arduino Nano is connected to the Raspberry Pi using a USB cable for communication anyways, we program it directly from the Raspberry Pi. This means we can completely program the robot wirelessly, which saves much time during rapid iteration.

Very fast and accurate line following using computer vision

The robot follows the line using a custom algorithm that operates on the captured frame: First, we apply a thresholding operation to separate the image into black and non-black. Now we iterate over each pixel of the image. For each black pixel, we determine the angle between a line from the bottom center of the image to the pixel and the vertical. The angles of the individual pixels are then combined with a weighted average to get an estimate of the angle of the line. Pixels that are very close to the bottom center or very far away get low weights. These weights are constant, so they are calculated once at the beginning of the program, and then stored in an image (Figure 7).

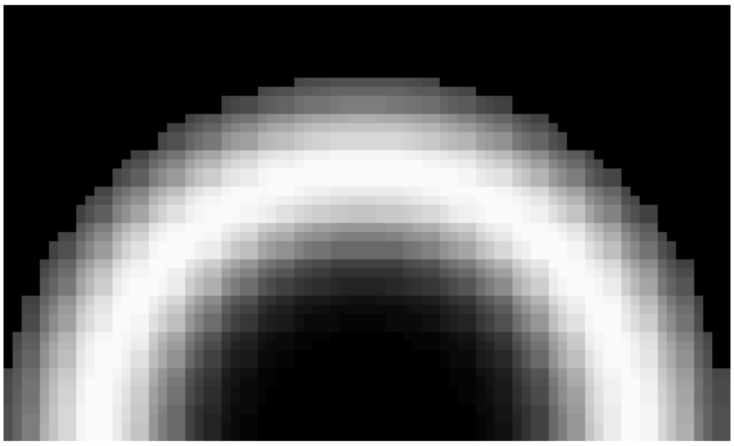


Figure 7: The distance-based pixel weight map (white is 1.0, black is 0.0)

Sometimes with complex tiles, a different part of the line comes into view which we do not want to follow. To minimize their influence the pixel angles are also weighed according to how close they are to the last line angle. To improve crossing T-shaped intersections the angles are also weighed according to how close they are to the center line.

This line angle is used as the error value for a proportional-derivative controller. The control variable is added to a base motor speed (80%) for the left motors and subtracted from the base motor speed for the right motors.

With proper tuning of the controller we have an extremely fast and accurate line follower (almost one tile per second on a straight line). The line following speed is actually limited not by the Pi's processing speed but by our slow front distance sensor, which would be unable to detect the obstacle before the robot crashed into it if the robot was driving much faster.

You can find simplified pseudocode for this algorithm in the appendix.

During line following, we constantly scan for green dots. Colors are detected by calculating the ratio of one component of color to the other two and comparing it to a threshold. We use a recursive algorithm to find groups of green pixels. If there is a green dot (or multiple), we cut out a part of the image around the green dot. Now we calculate the average position of the black pixels in that region of interest. The quadrant of this average pixel now determines whether to ignore this green dot and continue forward, or to turn right or left (Figure 8).

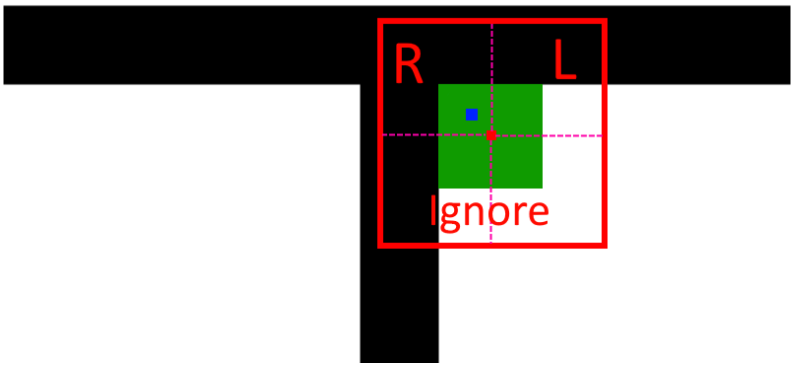


Figure 8: Determining the correct direction at an intersection. The average black pixel (blue) falls into the "turn right"-quadrant.

Victim detection neural net

We wanted to improve on the accuracy of the neural network from last season, so we decided to use a proper object detection model instead of our self-made model architecture. The MobileNetV2 architecture was suggested to us by a Team at the world championship, which we fine-tuned for our purposes with much less data than we had to use last year. The new model is also more accurate, especially when looking for victims that are far away.

In previous years, we used OpenCV's *HoughCircles* method to detect the victims. This method was very unreliable and rarely worked under different lighting conditions. At one point, the robot repeatedly detected debris which happened to be placed in a circular pattern. The neural network which we are using now works reliably under different lighting conditions and is resistant to noise.

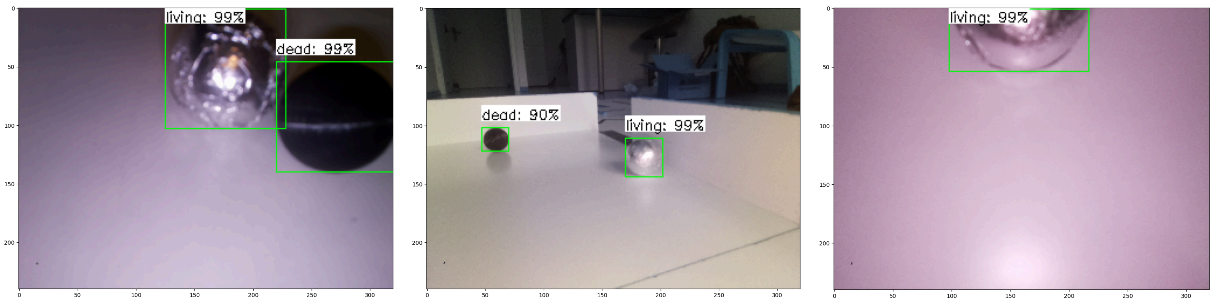


Figure 9: Victim neural net predictions

Detailed live debugging

When operating the robot, it is often hard to know which part of the program we are looking at and what the internal values of the robot are. For this season, we wanted to incorporate a detailed live debugging screen into the program. It has different modes for line following, obstacle avoidance, victim and corner finding and summarizes the most important variables and images.

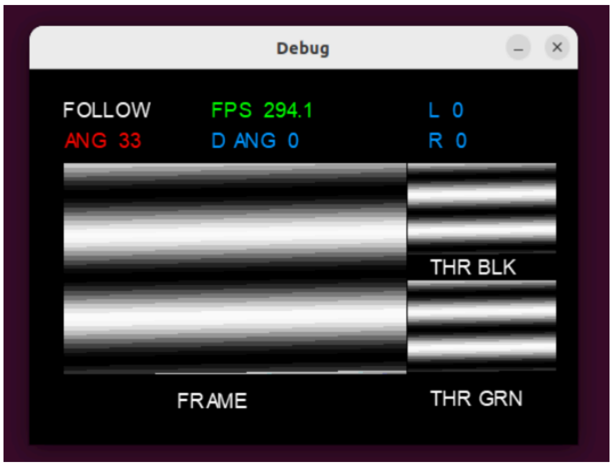


Figure 10: Debugging screen during simulated line following. Displayed are the color frame, black and green thresholded frame, motor speeds, FPS, computed line angle and line angle rate of change

5. Performance evaluation

We again were able to significantly improve the performance of this year's robot compared to our previous ones not only as a result of experience but also of even more intensive testing of both hardware and software. While most software subroutines were first tested in small test scripts before being implemented into our main program, we still conducted elaborate practice sessions for the robot after major changes to the program where we let it maneuver through the whole field (both line following and evacuation area) while monitoring and logging everything the robot was doing. Importantly, we always store essential frames on the SD-Card whenever the robot detects an intersection, the entrance to the evacuation zone, or a victim. This allows for post-evaluation and gives us further training data for the neural networks, in case of false positives.

As performance evaluation of neural networks in real-time can be challenging, we used a sufficient training/validation split and testing scripts to test the performance of newly trained neural networks before using them on the robot itself. To further test the resilience under competition conditions we often make use of debris or speed bumps placed all over the field to test the design. Experience tells us that most issues are only uncovered under the harshest test conditions. Debris in the evacuation zone in particular is one of the major reasons we are using a neural net instead of standard circle-detection algorithms in the first place.

6. Conclusion

After a year of developing this robot, we can say that we have improved in many areas. We reached many of the goals we set ourselves after last year’s season: We put more effort into the planning and conceptual design of our robot. Our development process was much more elaborate, and we planned, prototyped and tested more. We switched to mostly cloud-based services for better collaboration (Git, Fusion 360, Google Colab) and our communication became more frequent. For the first time we used a detailed Engineering Journal to document everything we did during development and testing.

The overall design of the robot is also much more mature and reliable. It is now easier to work on the robot’s hardware and iterate due to its more modular design. The software and electronics have also become much more robust due to a new software architecture and our new power supply.

We are looking forward to this year's RoboCup season to not only be able to test the performance of our robot during actual competition while also talking to other teams as well as discussing their mechanical and software solutions.

Appendix

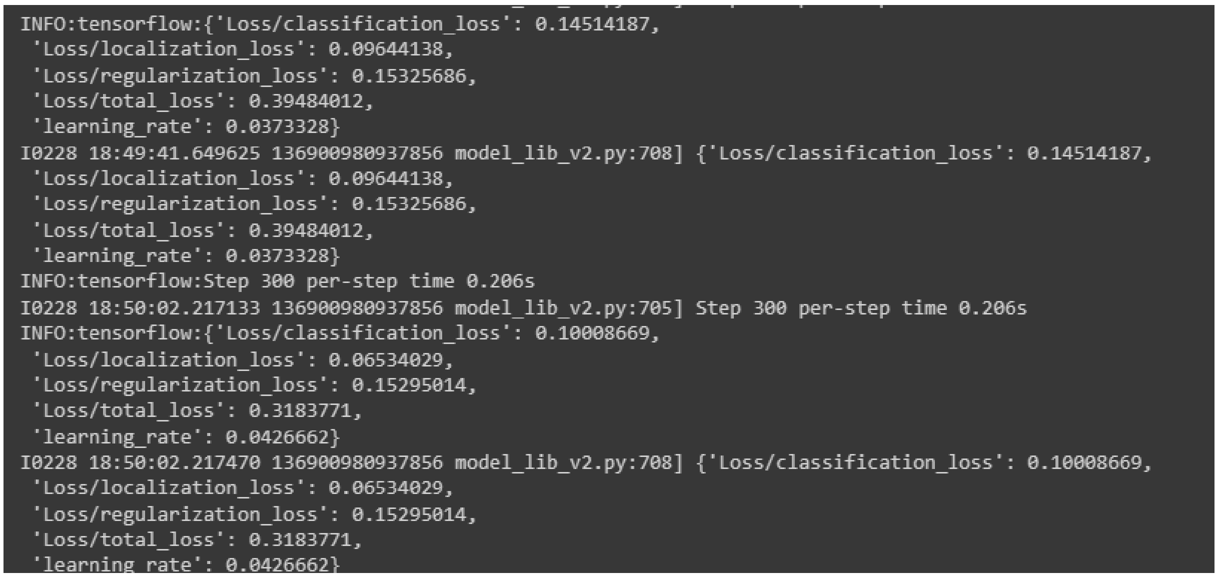
If you would like to learn more about our robot, feel free to visit:

- Our YouTube channel
- Our website
- Our GitHub repository

If you have suggestions, comments, or questions about our development you can also write us an email at: robocup.evb@gmail.com

Simplified pseudocode for following the line using a weighted sum:

```
// Difference weight based on difference to last line angle
function diff_weight(x):
  // x in radians
  x *= 2 / pi
  return 0.25 + 0.75 * exp(-16 * x^2)

// Distance weight based on distance to bottom center
function dist_weight(x):
  e = (3.25 * x - 2)
  f = exp(-e^2) - 0.1
  if f > 0: return f
  else: return 0

sum = 0
sum_weights = 0

bottom_center_x = image_width / 2

bottom_center_y = image_height

for every pixel x, y:
  if pixel is black:
    distance = sqrt((x - bottom center x)^2 + (y - bottom center y)^2)

    angle = atan2(y - bottom center y, x - bottom center x) + 90°

    weight = diff_weight(angle - last_angle) * dist_weight(distance)

    sum += weight * angle

    sum_weights += weight

line_angle = sum / sum_weights
line_angle_d = (angle - last_angle) / dt

u = Kp * line_angle + Kd * line_angle_d
motor_speed_left = base_motor_speed + u
motor_speed_right = base_motor_speed - u
```



```
INFO:tensorflow:{'Loss/classification_loss': 0.14514187,
 'Loss/localization_loss': 0.09644138,
 'Loss/regularization_loss': 0.15325686,
 'Loss/total_loss': 0.39484012,
 'learning_rate': 0.0373328}
I0228 18:49:41.649625 136900980937856 model_lib_v2.py:708] {'Loss/classification_loss': 0.14514187,
 'Loss/localization_loss': 0.09644138,
 'Loss/regularization_loss': 0.15325686,
 'Loss/total_loss': 0.39484012,
 'learning_rate': 0.0373328}
INFO:tensorflow:Step 300 per-step time 0.206s
I0228 18:50:02.217133 136900980937856 model_lib_v2.py:705] Step 300 per-step time 0.206s
INFO:tensorflow:{'Loss/classification_loss': 0.10008669,
 'Loss/localization_loss': 0.06534029,
 'Loss/regularization_loss': 0.15295014,
 'Loss/total_loss': 0.3183771,
 'learning_rate': 0.0426662}
I0228 18:50:02.217470 136900980937856 model_lib_v2.py:708] {'Loss/classification_loss': 0.10008669,
 'Loss/localization_loss': 0.06534029,
 'Loss/regularization_loss': 0.15295014,
 'Loss/total_loss': 0.3183771,
 'learning_rate': 0.0426662}
```

Figure 11: MobileNetV2 model during fine-tuning

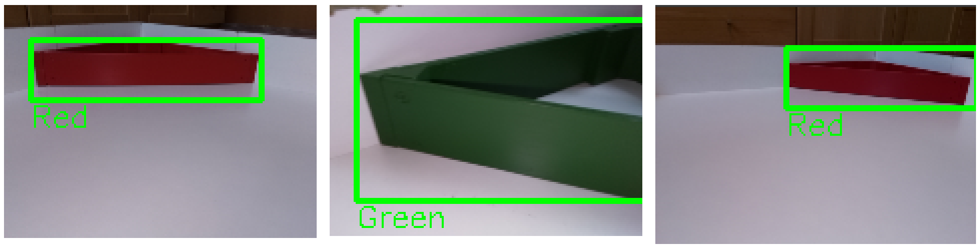


Figure 12: Corner neural net predictions

References

Software and platforms

Google Colab

GitHub

Fusion 360

Libraries

TensorFlow

WiringPi

Hardware

Arduino Nano

Raspberry Pi

BNO055 library

Pololu - 20D Metal Gearmotors

Anycubic i3 Mega S 3D printer