ROBOCUPJUNIOR RESCUE 2021
TEAM DESCRIPTION PAPER

πthon πraten

Introduction

Team

Hello everyone,

we are Lukas Krämer and Jan-Niklas Freund from Wipperfürth in Germany. The Team πthon πraten (πthon πrates in English) was founded in summer 2017 and has participated in RoboCup every single year since then. In our team, Lukas is responsible in our team for programming the robot and testing new ideas, while Jan comes up with new approaches to improve the performance of our robot. In the following we are going to summarize the approaches of our robot and the development in both its performance and our personal skills we acquired ourselves.

In 2018 we used a LEGO Mindstorms robot and two color sensors for linefollowing, but our focus has always been on saving the victims because we considered that part the most fascinating one. That is why we already thought a lot about the rescue area back then and spent hours over hours developing a gripping mechanism with a collection system for the balls.

In 2019, we moved on to something more advanced and started getting into the world of microcontrollers, custom sensors and better motors. We started building our first self-made robot made from an aluminium chassis and did not use any LEGO parts despite the chains. Additionally, we started to develop our own mounts for sensors and printed them with a 3D printer.

A year later, we realized that it takes a lot of time to have to re-engineer the whole chassis to only make a small change on the robot, so we took a step back and used LEGO again to build our robot in order to remain flexible. Nevertheless, we continued to use commercial sensors and an Arduino because it allowed us to use significantly more sensors than with Mindstorms. 2020 was also the first year that we used a Raspberry Pi and a camera, at least for line detection and intersections. The performance of our robot was good enough to qualify for the RoboCup GermanOpens which unfortunately could not take place due to the pandemic.

During lockdown, we used our free time to model a new robot in a CAD-program which we then 3d printed afterwards. Furthermore, we improved the program from he last year and added the ability to detect the victims and the rescue kit as well.

Project Planning

Overall Project Plan

As we have already mentioned, it was always our intention to be comparatively successful in the rescue area since this could be a real life scenario, there are extremely many approaches to it and from year to year new ideas keep coming up to improve the technique and technology from before. It has always been our intention to take part in the GermanOpen at one point.

That is probably why it has always been our goal (which seemed impossible in the beginning to actively search for the balls with a camera or some sort of sensor instead of driving through the evacuation zone aimlessly and hoping to get one.

Today we can proudly say that actively recognizing victims is our personal milestone, because we have worked on this for a long time and this ability makes our robot more or less unique. Although the general evaluation of the camera image in 2020 to follow the line, was also a challenge, there are significantly more tutorials, documentations, etc. that have made this work easier compared to the victim search. More information on the exact implementation of the camera evaluation will follow in this document under the heading "Software".

Over the years, however, the exact analysis of the task and the set of rules also became more and more important, as new rules were added and we had to adapt our robot or the software accordingly, in order to be successful in the competition.

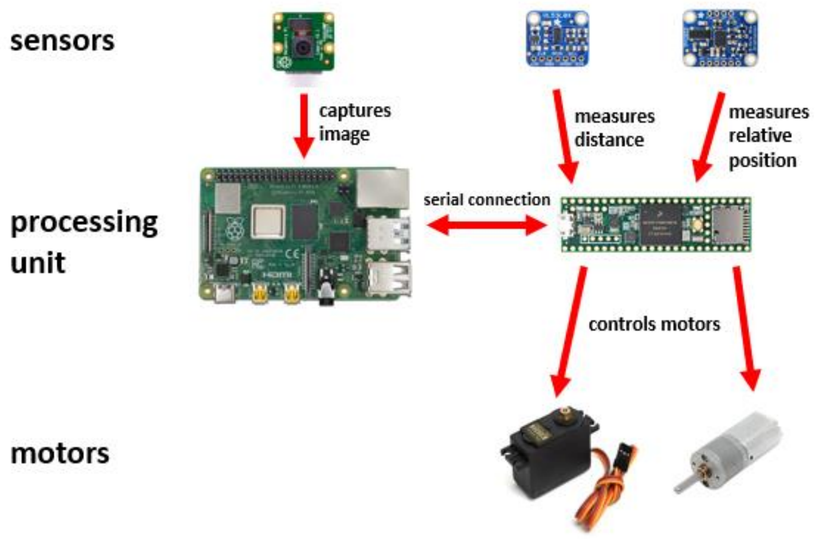
For example the rescue kit was something that challenged us in the beginning since we had no gripping mechanism that could lift both a cube and a ball, which more or less forced us to think about a new solution and to create a more advanced design. In addition to that, we had to implement a new function in our program to search for blue color during linefollowing.

Integration Plan

Now let's move on to the general technical implementation of our current robot. For a more detailed explanation of the related fields, have a look at the "Software" or "Hardware" section, since we will not go into detail here.

On the hardware side, our robot consists of a battery for power supply, a Raspberry Pi for camera evaluation and a Teensy microcontroller for controlling the motors and reading out sensors. For the driving mechanism, we use two ordinary dc-motors that were made compatible with LEGO chains using a milled aluminum adapter. In addition, two servo motors are used to move the arm up and down to pick up the rescue kit and the balls. Moreover, there is a laser sensor that can measure the distance to objects and is used to recognize the obstacle during linefollowing and the walls in the rescue area.
Last but not least, apart from the camera, there is actually only a gyro sensor worth mentioning, which helps, for example, to turn by exactly 90 degrees at a green dot or to orientate the robot in the rescue area.

Schematic of how our robot works:



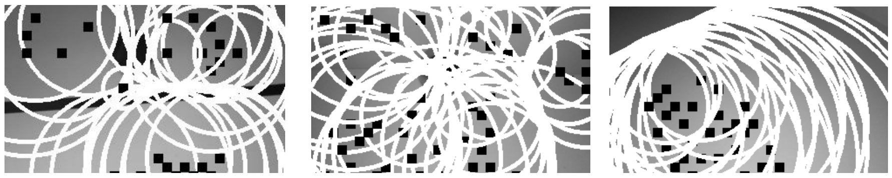
Testing

When it comes to “normal” performance testing of our robot, we have to admit that your strategies are quite usual. Of course, we try to optimize the time spend for testing a new feature as much as we can by just testing the “new” things that could go wrong. When we for example implement a new way of finding the exit of the evacuation zone, we comment out all the unnecessary code beforehand (like following the line, detecting silver, searching for victims and evacuate them to the black corner).

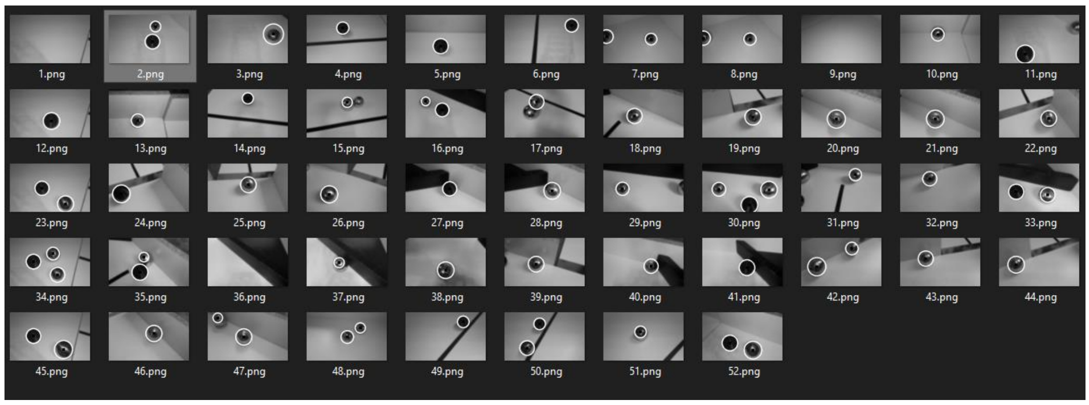
But one solution about finding the most efficient way to search for the balls saved us a lot of time:

The function that actually detects the balls takes several parameters to specify and optimize the search for our specific use case. The individual parameters range from the minimum/maximum radius of the circle to highly complex aspects which the function needs to work properly. Therefore, also the accuracy for our use case (detect silver/black balls with a diameter of around 5cm) fluctuates from no detected victims at all, to an image full of big and small detected balls even if there isn’t a single ball (see the pictures below). After a few hours of experimenting with different values, we came up with a better idea. We took around 60 pictures from different angles, under different lighting situations and with a varying amount of balls ranging from zero to the maximum of three possible ones. Afterwards, we manually counted how many balls all the pictures included together and wrote a small python script that tries (out) all different parameter combinations with our test footage. It keeps track of how many balls were actually found per single run and compares it with our preset reference. After 30 minutes of tense waiting, we finally got a parameter combination that seemed to work best for our scenario.

Here are some pictures for illustration:



Catastrophic recognition of the victims



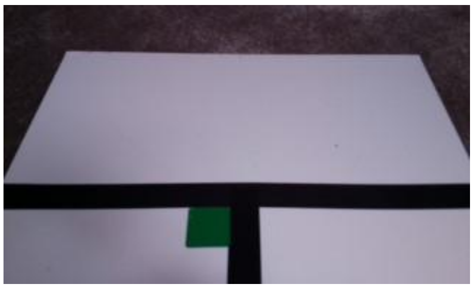
Perfect result using the parameters our script suggested

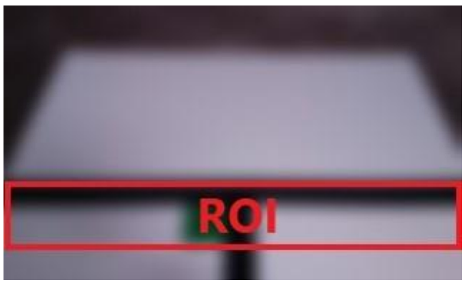
Software

General software architecture

As you already know, we do not only use a Raspberry Pi for the camera evaluation, but also a Teensy 3.6 microcontroller to control the motors, measure the absolute orientation of the robot and the distances to the obstacles. Therefore, we also need to write two separate programs, one in Python for the Raspberry Pi to capture the image from the camera, detect the line, intersections, victims etc. and one in C++ for the microcontroller in order to read out the sensors and control the motors. This may sound like a bottleneck, but we needed the Teensy in the past for controlling the motors and interfacing the sensors since the GPIO pins of the Raspberry Pi are a bit harder to use, some important libraries for our sensors are not supported and we didn’t want the Raspberry Pi to do lots of small tasks that we could outsource to another chip. In the near future however, we plan to design a new circuit board/shield for the Raspberry Pi in order to control everything from it directly and get rid of any sort of additional microcontroller and the (slow) serial communication between those two.

When it comes to the software implementation, you can say that most of the important things get handled by the Raspberry Pi and the Teensy acts like a slave that just controls everything. During linefollowing, the Raspberry Pi captures a frame from the camera in a low resolution and cuts out a region of interest (ROI) for evaluation. If we process the whole image which consists of over 61000 unique pixels (320p * 192p = 61440p or in full resolution 2592p * 1944p = 5038848p), the FPS (frames per seconds) would be extremely low and the robot would be forced to drive very slowly. In order to process the image even faster and to handle noise like small dust particles or narrow gaps, we blur the image before the actual evaluation using the “GaussianBlur” function from the OpenCV library.


Normal image



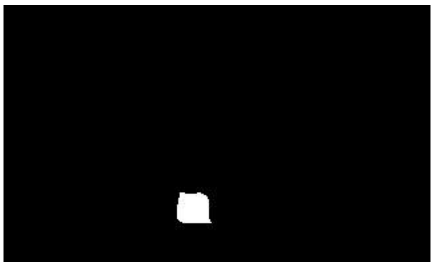
blurred image and ROI

In addition to blurring the image, we also convert it from BGR to HSV color space because it is easier to distinguish colors there:

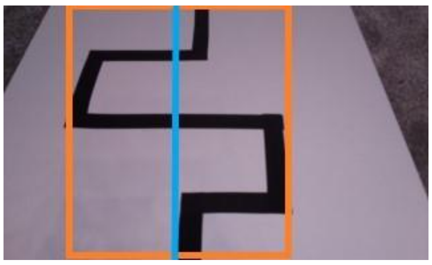
Image in HSV color space

But now let’s finally move on with explaining the actual processing.

First, the Python program finds all black (and green, red and blue) contours and converts them into a so-called binary image. In the binary image, everything is black except for the place where the contour is located. Here again a photo where the program converted all green contours to a binary image:



Afterwards, the program checks whether there are some green contours (-> intersection) blue contours (-> rescue kit), red contours (-> end of parkour) or black contours (-> line). Usually there are only black contours, so the program starts drawing a rectangle around them and divides it into two halves (please note that the Raspberry Pi doesn’t process the whole image, the image below is just to demonstrate the working principle):



The blue line that divides the rectangle is the more or less optimal way our robot has to drive, so we just wrote a function that calculates how far away the center of the robot is from the middle of the contour. This number ranging from –160 to +160 gets send to the Teensy that controls the right and left chain using two motors.

Furthermore, the Raspberry Pi sends the according commands to the Teensy when for example a green contours arises that is on the left side of the blue line (-> robot has to turn -90°).

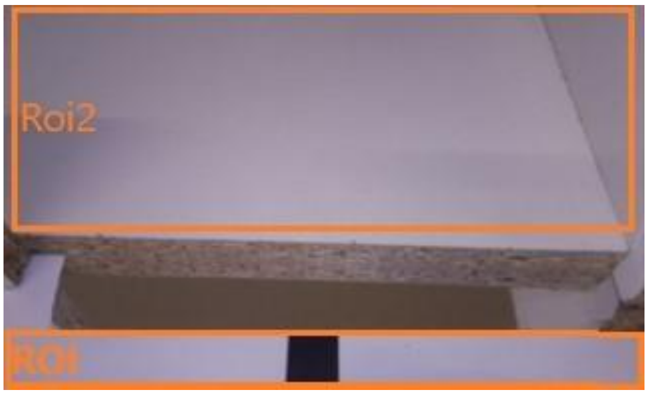
The same principle applies to all different sorts of contours despite the silver one to which I will write something in the next paragraph.

Innovative solutions

As I have already mentioned, detecting silver contours is nearly impossible since there are a lot of reflections from other objects like the robot itself or a spectator who wears a red pullover. That’s why we had some trouble detecting the entrance of the evacuation zone because most of the time the robot just went “crazy” as soon as the silver strip was in the ROI for processing. After experimenting with all kinds of ideas, like placing a neon red foil in front of a bright led, so the silver strip would reflect red and we could detect a red contour instead of a silver one, we finally decided to detect the rescue area without the silver strip. We thought about what distinguishes the evacuation zone from a normal 30x30 tile with a line on it and we found two unique features:

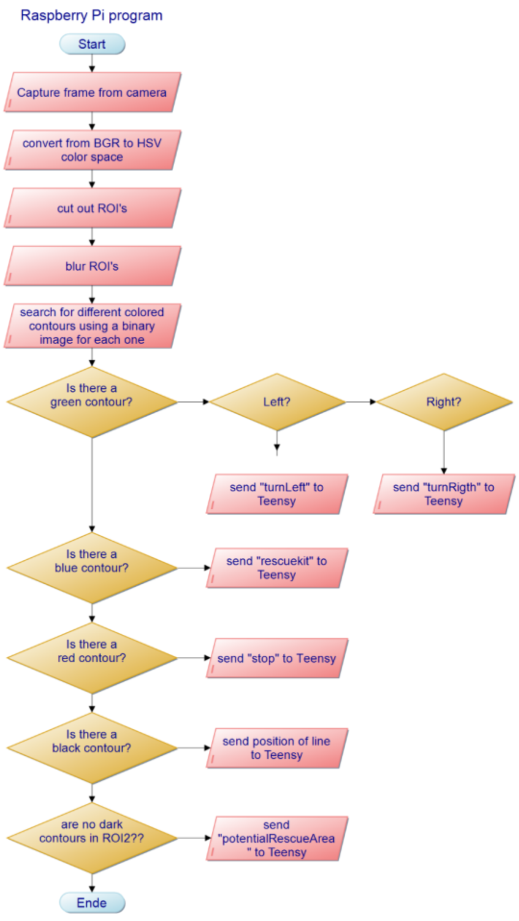
1. The front distance sensor should measure a distance between 90cm and 120cm
2. The camera can’t see a dark contour because the floor of the rescue area is white

So, we combined these pieces of information and used a second, comparatively big ROI, to search for dark contours before the ROI for the linefollowing would detect silver and lead to undesired behavior:



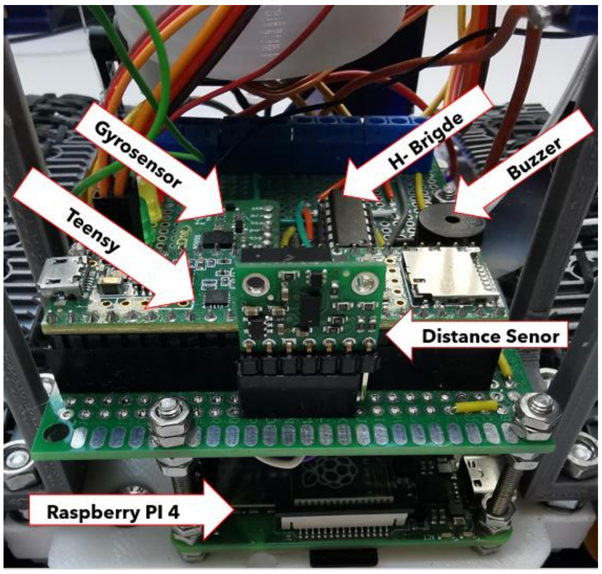
If the Raspberry Pi does not find dark contours in ROI2, he sends a command to the Teensy that which then checks if the distance is between 90cm and 120cm which would mean that there must be the rescue area. Of course, this procedure is not perfect because there are some passages without a black line in the parkours (like long gaps), but with the help of the distance sensor we rarely detect it erroneously and we could improve our technique by using a second distance sensor to one side or by driving 20cm forward when the robot thinks he “detected” silver and check again if there are no contours in darker color.

The program of the Raspberry Pi in a structure diagram:


Raspberry Pi program
Start
Capture frame from camera
convert from BGR to HSV color space
cut out ROI's
blur ROI's
search for different colored contours using a binary image for each one
Is there a green contour?
Left?
Right?
send "turnLeft" to Teensy
send "turnRigth" to Teensy
Is there a blue contour?
send "rescuekit" to Teensy
Is there a red contour?
send "stop" to Teensy
Is there a black contour?
send position of line to Teensy
are no dark contours in ROI2??
send "potentialRescueArea" to Teensy
Ende

Hardware

As you may imagine, the camera is the most important sensor on our robot, because it does not only detect the line, intersections etc., but also the victims in the evacuation zone. Theoretically, we could use any camera that features a USB interface, but we decided to use the normal Raspberry Pi camera V2 since there are lots of resources on the Internet and we can interface the camera without additional drivers. As we have already mentioned, the camera image gets processed by the Raspberry Pi, which then passes the relative line position to a Teensy 3.6 microcontroller through a serial connection. The Teensy however, acts like a slave that controls our motors (Pololu 20D) with the help of an L293d h-bridge in order to drive the robot.

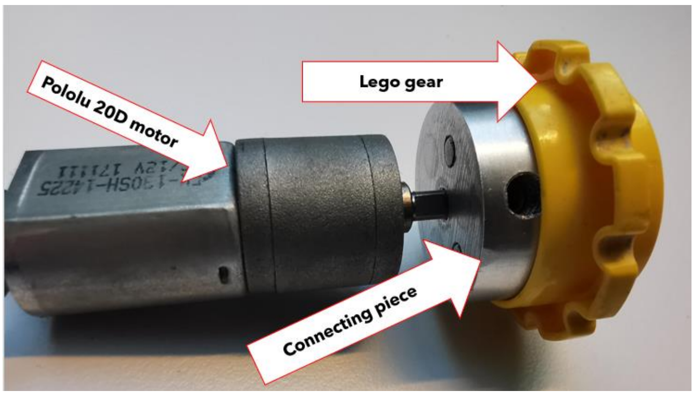


Our Robot has a 7,4V, 1200mah Li-ion battery underneath, which has been perfectly balanced, so the robot has an ideal center of gravity and can climb up the ramps without a problem.

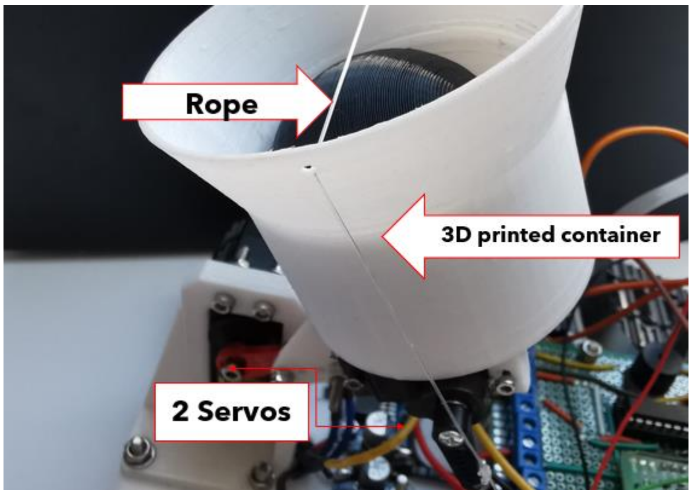
However, the different devices of the robot need different amounts of voltages, for example our motors need 12V, whereas the rest needs 5V which is why we use a boost-converter and a buck-converter to step the voltage up and down .

We also have two different servo motors, which are used for the gripper. One is to raise the gripping mechanism and the second one to loose and tighten the rope (see the picture below):

One of the things we made ourselves is a connector between our Pololu 20D motor and our Lego chains, which we will get to talk about later. Our problem was always that either the motors were not good enough or the chains had too little grip. It was not possible to have both at the same time, so we wondered if there existed an adapter for the motors which would allow us to screw Lego parts onto them, but there wasn't one, so we milled it ourselves from a piece of aluminum.



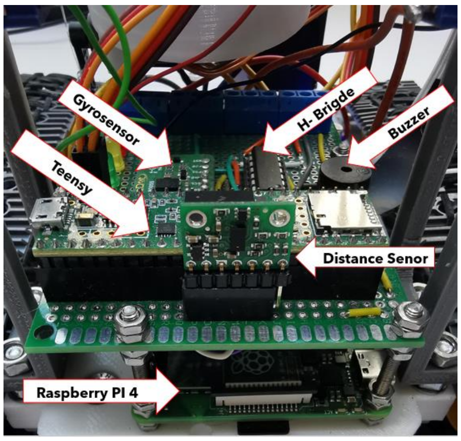
We also developed a gripper ourselves. There were some teams with a similar gripper that we were inspired by. But of course we still had to design it and print it with our 3D printer. The idea came from the fact that we used to have a huge gripper and a tank with which we always collected several balls at the same time. This was not good, because the balls have to be put into the corner in a certain order, first the living ones and then the dead ones. With the new gripping mechanism and the fact that we now search for bullets with the camera, it has become much easier.



Let's move on to the chains. We have been part of competition for four years and know a few teams by now. They all say that it is always a problem to have the right chains. One could of course use tyres, but we had bad experiences with those in our first year. Due to this we decided to use normal Lego chains and cover them with rubber bands. We tried a lot of things, like designing them ourselves and 3D printing them, but they had no grip. Then we tried rubber hoses and many other things, but finally we ended up with Lego chains covered with rubber bands as our best option.

Mechanical/ Electronic Design and Manufacturing

In the following we will explain the rough construction of our robot. Its construction is quite simple. Everything starts with the camera, which is installed in the front in a self-designed holder with a led strip. Then it continues with the Raspberry PI, which lies at the bottom of the base, and from there it goes to the Teensy, that lies slightly above the Raspberry. A distance and gyro sensor are also installed there. You can easily see in the picture below. Take a look underneath the robot, where only the battery is installed, because it is quite large and heavy. The robot gets the current through cables that lead to a buck and boost converter, which then passes on the correct voltage (12V or 5V) to the corresponding devices. Finally, only the rear part of the robot remains, where the rescue functions are located, that contain two servos and the gripper.



All of the important modules are listed and explained in “Hardware”, including everything about our rescue mechanism. Moreover all electronic designs are named there.

Performance evaluation

First of all, there are many special features about our robot, one of them is that everything about our robot is self-built. One example is the self-soldered circuit board, which actually took several attempts and hard work in Lukas’ basement. Or another self-produced part on our robot is the 3D printed camera mount, which is also self-designed. We created a 3D printer at school a few years ago, and Lukas now has one at home, too. We did this because we always had high standards and parts from LEGO or others did not always give us the quality we wanted. What we really like about our robot is that it is able to search for the victims by using a camera since last year. We especially like this because only a few teams can do this and it is really amazing to see how the robot searches for the bullets every time it turns around and every time it finds one, we are verry happy because it has worked one more time. The rescue area is one of the hardest aspects to program, because there are so many variables that you have to take into account, for example when new rules are released.

But to be honest there are some things we still have to improve. Among them there are three noticeable points, it could be that there are more and of course a robot is never perfect, but we will always give our best. One point is the slow speed in general, but especially when searching the victims. Unfortunately we do not have a solution for this yet, because it is difficult to find the right ratio of correct driving and fast driving. If you want to make the robot faster, there is always the danger that it does not manage the parkour or needs several attempts.

In addition, the robot drives partly jerky, that probably comes because we do not yet now all the values exactly, one would have to optimize them. But this needs several attempts, for which we had little time.

And one of the things we also noticed is that the robot often recognizes the obstacle as a black line. We suspect that it could be because the camera looks very far to the side and therefore recognizes the grey/black obstacle as lines. If we knew that for sure, we would fix it quickly.

All in all, we are quite satisfied with the performance of the robot and of ourselves and we would never have dreamed that we would ever get this far. But now we have made it and we will give everything to get even further.

Conclusion

To summarize this document, you can say that we, as most other teams, worked hard on our current robot and that we progressed from total “newbies” with LEGO Mindstorms to something more advanced. Furthermore, we can finally detect the victims with a camera, but we still need to improve our program, for example to distinguish between dead and living victims, drive a bit smoother and not that jerky and avoid processing pauses when searching the balls. Since we didn't have enough time to replace the Teensy this year, the next step after this competition will be to start designing a new board and then to only use a Raspberry Pi to get rid of the transmission between Raspberry Pi and Teensy and thereby eliminate the small pauses in the evacuation zone.

If you have got any further questions, please do not hesitate to ask for clarification via mail (robocup.evb@gmail.com).

Reference

- Our YouTube channel
- Our website (in German and not up-to-date
- OpenCV documentation
- Linefollower tutorial using OpenCV