Metropolia Ammattikorkeakoulu

2018

Zumo project

PROJECT REPORT

RUKHSORA NAZAROVA, MIKHAIL STEFANTSEV, MARIIA SAVELEVA

Table of Contents

1. Abstract

1.1.Summary of the project.

This report reviews details of working on the robots project, challenges that we were faced with. We decided to implement specific methods for each part of the project.

- In sumo wrestling part there were used reflectance sensor to stay inside the ring, accelerometer to detect hits and countdown timer to stop when the robot gets out from the ring.
- Line following task was conducted by using basic PD controller, that provided robot's fast and accurate pass of the route.
- In maze task, structure was implemented to calculate current position of the robot, regarding the whole area of the maze.

As a result, we successfully passed all 3 parts of the robots competition, however our robot was not first in line following competition, where we lose with 18ms difference with the winner.

2. Introduction

2.1.The goal and scope of the project.

The goal of the robot project is to maintain our knowledge about C programming language, working with external libraries in a case of the Zumo library. Additionally, to design and build a Zumo robot software that will allow it to pass all the challenges in the final assignment.

Another important task during development of the project was team collaboration, usage of special tools like version control system, task tracking and communication during the development process.

3. Methods and materials

3.1.GIT

Our team uses GIT as version control system.

Git is a version-control system for tracking changes in computer files and coordinating work on those files among multiple people. [1]

It allows multiple developers to access, edit and upload their changes to shared repository that is situated on remote server.
In our project, each week's assignment and each final project was made in a separate branch. The project branch hierarchy shown on Figure 1.

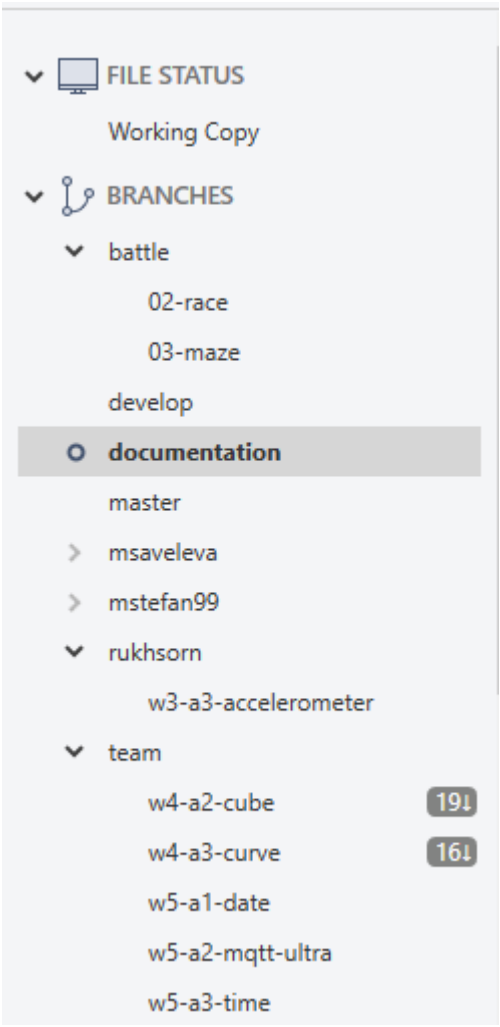


Figure 1

3.2. Bitbucket

Bitbucket is a web-based version control repository hosting service owned by Atlassian, for source code and development projects that use either Mercurial (since launch) or Git (since October 2011) revision control systems. Bitbucket offers both commercial plans and free accounts. [2]

The other popular solution named Github was also available, but the reason our team has chosen Bitbucket was the availability of free private repository feature.

3.3. Trello

Our team decided to use Trello as a task management instrument.

Trello is a web-based project management application originally made by Fog Creek Software in 2011, that was spun out to form the basis of a separate company in 2014 and later sold to Atlassian in January 2017. [3]

It allows to create boards and tasks, assign tasks for each team member, add links, comments, checklists, due dates and also has integration with Bitbucket.

For our project we decided to use Agile methodology similar to Scrum and created four boards:

- To Do
- Doing
- Testing
- Done

To Do board collects all the tasks that supposed to be done in near future.

Doing is a board that stores tasks currently in development. It is important to track the current state of project, so team can rearrange some tasks in progress.

Testing is a board for tasks that supposed to be tested.

Done is the list of tasks that are done and tested.

The board is shown on Figure 2.

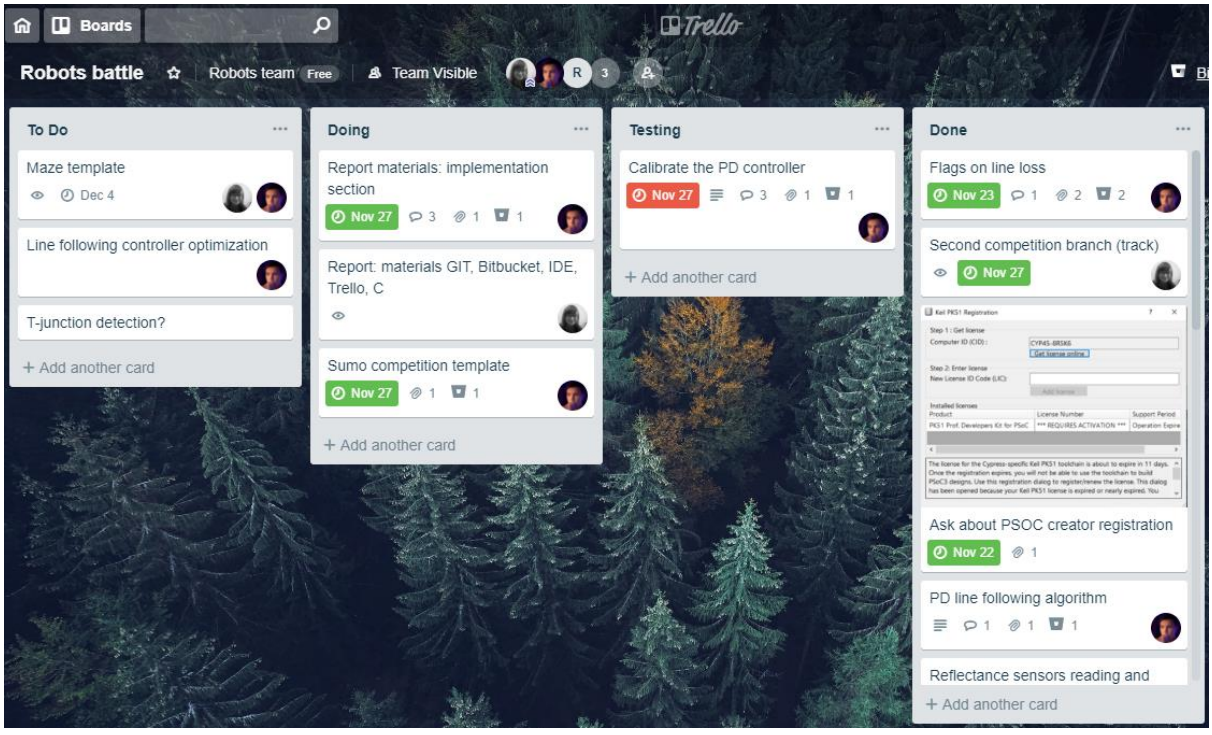


Figure 2

4. Theoretical background

4.1.Zumo Shield

According to Pololu's official site [4], the Zumo Shield consists of dual motor drivers, a buzzer that is able play sounds and music, a user push button that allows to switch on and off the robot and an accelerometer that measures acceleration in 3-dimensions, compass and gyroscope that tracks orientation.

The Shield is fastened to the chassis and merged with its battery terminals and motors. In our case, CY8CKIT059 is inserted into the Shield.

The Shield's features in more details:

- Integrated DRV8835 dual motor drivers are able to supply two high-power micro metal gearmotors with sufficient current.
- Piezo buzzer for playing mere sounds and music, the tones can be generated in the background without taking up a lot of processing power.
- LSM303D 3-axis accelerometer and 3-axis magnetometer used to detect impacts. The compass gets huge amount of interference from the motors, batteries, PCB, and its surroundings with proper calibration.
- L3GD20H 3-axis gyroscope that is generally used to track rotation. With this sensor and the LSM303D mentioned above, the shield effectively has a built-in MinIMU-9 v3 IMU module that can optionally be used to make an attitude and heading reference system (AHRS).
- Optional user pushbutton.
- 7.5 V boost regulator for powering the CY8CKIT059 from the Zumo's 4 AA batteries.
- General-purpose prototyping areas and an expansion area at the front for connecting additional sensors (it is easy to add a Zumo reflectance sensor array or up to five QTR sensors for edge detection or line following).

4.2.Zumo Chassis

Black ABS plastic is used as main material to make the chassis. Zumo chassis has also sockets for two micro metal gearmotors and a detachment for four AA batteries. The battery detachment terminals protrude through the chassis and are easily accessed from the top side. Also, chassis include acrylic black plate. This plate keeps the motors in place, additionally it can be used for mounting electronics, such as microcontroller, motor drivers and sensors.

The drive system consists of two black silicone tracks, one on each side, that are each supported by a freely spinning idler sprocket and a motor-driven drive sprocket.

The Zumo chassis uses two motors, one for each thread. [5]

4.3.PSOC CY8CKIT059

"ARM® Cortex®-M3 CPU in a single chip. Process sensor signals with the 24-bit hardware DFB coprocessor, offload traditional CPU tasks to the CPLD-based Universal Digital Blocks and increase system performance with the peripheral-to-peripheral DMA controller. Integrate high-precision custom 20-bit Analog Front Ends with the Programmable Analog Blocks including opamps, PGAs, filters, comparators, SAR and Delta-Sigma ADCs and the industry's best CapSense touch-sensing solution." [6]

4.4.Sensors

Sensors can allow for a reliable robot operation in changing conditions such as changes in battery charge level, nonlinear motor output curve and other changing external conditions. Usage of sensors allows the robot to correct its actions in case of any changes due to which pre-programmed algorithm may not work as desired. Zumo shield has a few onboard sensors which can be used in a user-written program with the help of the Zumo library as described below.

4.4.1. Ultrasonic sensor

Zumo robot can use an ultrasonic sensor to position itself relative to its surroundings. In this project the sensor will be used in the final task to detect maze walls. Zumo library provides the command to read ultrasonic sensor measurement in centimeters.

Zumo library includes a function called `Ultra_Start()` which starts task handling sending ultrasound signal and interrupt receiving it and calculating the distance knowing the time to take the signal to reflect from the object in front of the robot and hit the sensor.

4.4.2. Reflectance array

Zumo shield has a reflectance array consisting of 6 sensors located underneath the shield board behind the bulldozer blade on the front. These sensors are used in detecting a line for the robot to follow. Zumo library provides commands for reading both digital data from sensors using the threshold value and raw sensor readings. On a top design level Zumo library uses a set of timers synchronized with SR switches. On function `reflectance_start()` execution a separate task scanning sensor readings in the background is created and run every millisecond. The result of scanning can be written in the structure of type `sensors_`.

4.4.3. Accelerometer

The robot has a built-in accelerometer for detecting hits if the ultrasonic sensor did not detect the obstacle in advance which can be used in the sumo battle for detecting hits from sides and back where ultrasonic sensor cannot detect the approaching opponent robot. Zumo library provides function which allows to read the acceleration in x, y and z direction.

4.5.MQTT

"MQTT is a message-based light weight protocol for sending data from sensors to receivers. A simple MQTT setup has three parties: a broker, a publisher and a subscriber. Broker is the central hub that receives data that the publisher sends and forwards it to the subscriber. In a typical setup, the sensors (in our case the robots) produce data that they publish to the broker. Any number of subscribers can connect to the broker and request to receive the data. Zumo robot has a Wi-Fi module for wireless network connections. The module gives the robot TCP/IP connectivity that is used to send MQTT messages to a broker. Zumo library automates the setup of TCP/IP connection to a broker and provides a `printf`-like interface for sending MQTT messages. To enable TCP/IP and MQTT you need to edit `zumo_config.h`." [7]

5. Implementation

5.1. Batteries

First challenge we faced was related to the batteries in the robot. Since Zumo uses conventional nickel metal hydride batteries that should never be discharged below the certain threshold in order to stay functioning we had to implement the function which constantly checks the voltage of the batteries and notifies the user in case of need to charge the batteries. When battery voltage gets too low, the robot starts blinking the onboard LED in full power that can't be discarded in any way. It is also planned to implement the feature that locks the motors in case of low battery charge in order to prevent them from further discharging and subsequently damaging them. This required us to calculate the real voltage from the readings of the battery ADC.

The ADC connection diagram is shown on the Figure 3:

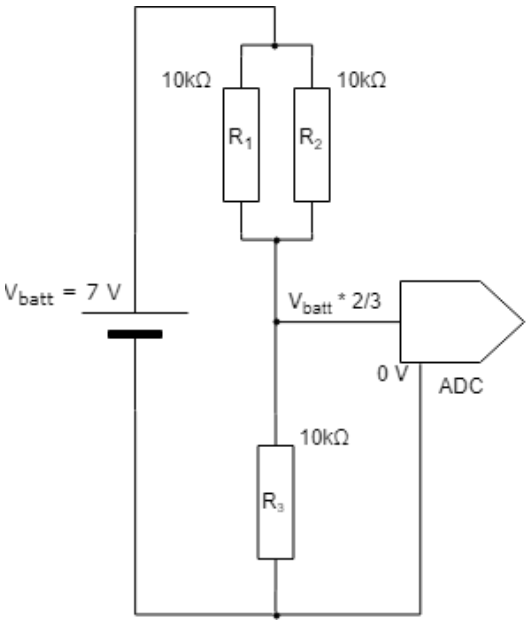


Figure 3

As seen from the diagram, the ADC input is connected to the output of the divider to lower the voltage as it may exceed the maximum allowed voltage of the ADC itself. That is why it is required to not only convert ADC output to volts, but also get the source voltage.

To convert the ADC output to the volts, the following conversion coefficient was used: $\frac{\mathrm{V_{ref}}}{2^{\mathrm{bit\_depth}}-1} = \frac{5}{4095}$, where V_{ref} is the reference voltage of the ADC (*5V as specified in the documentation)* and $\mathrm{bit\_depth}$ is number of bits used by ADC (*in our case,* $\mathrm{bit\_depth}$*=12*).

To get the source voltage the voltage conversion coefficient is needed. It is calculated like:

$$\mathrm{k} = \frac{\mathrm{R_{total_{12}}} + \mathrm{R_3}}{\mathrm{R_3}} = \frac{\frac{\mathrm{R_1} \cdot \mathrm{R_2}}{\mathrm{R_1} + \mathrm{R_2}} + \mathrm{R_3}}{\mathrm{R_3}} = \frac{\frac{10\Omega \cdot 10\Omega}{20\Omega} + 10\Omega}{10\Omega} = \frac{3}{2},$$

where $R_{total_{12}}$ is the equivalent resistance of R_1 and R_2.

So, the source voltage equals to:

$$\mathrm{ADC_{out}} \cdot \mathrm{k} \cdot \frac{V_{ref}}{2^{bit\_depth} - 1} = \mathrm{ADC_{out}} \cdot \frac{R_{total_{12}} + R_3}{R_3} \cdot \frac{V_{ref}}{2^{bit\_depth} - 1} = \mathrm{ADC_{out}} \cdot \frac{3}{2} \cdot \frac{5}{4095}$$
$$= \mathrm{ADC_{out}} \cdot \frac{15}{8190} \approx 1.831 \cdot 10^{-3},$$

where ADC_{out} is the output level of ADC (*an integer number between* 0 *and* $2^{bit\_depth} - 1$).

5.2.Sumo

For the sumo competition several sensors were used, including reflectance and accelerometer as well as the countdown timer. The algorithm functions as follows: the robot drives to the first line, waits for IR and then enters the circle and sets the timer for 5 seconds. When inside, the robot can detect the circle with its reflectance sensors and turn back to stay inside. Robot also uses the accelerometer to detect hits received from another robots. The algorithm is shown in the Figure 4. The timer in our robot is set in the beginning of the competition and is reset constantly while inside the circle and every time it is being hit. This allows the timer to run out only in the case when the robot is outside the circle and receives no hits. When the time is up, the robot sends logs and brings the motors to a stop.

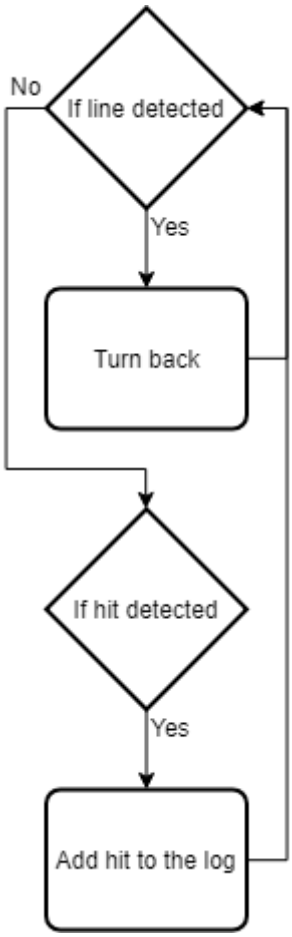


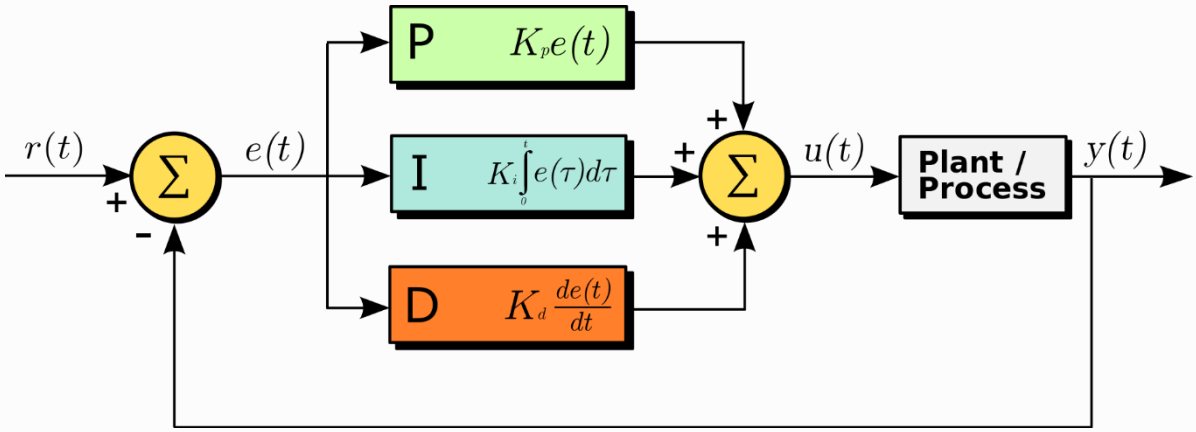
Figure 4

5.3. Line following.

Our robot uses a basic PD – controller with line loss control with that meaning the robot will continue steering in the direction the line was lost to get back on the line where PD controller steps back in and helps the robot to adjust its position after the line was lost.

The PD controller outputs a single value between reference values `-255` and `255` which is the maximum value of the 8-bit integer type (also known as `byte` or `uint8`) which matches the absolute maximum value of the motor speed with the sign accounting the turn direction of the robot. The output value is then used by motor control function receiving an integer variable and calculating the speed of both motors accounting for the direction of the turn (the sign of the value) and the maximum allowed speed of the robot (being passed as an argument).

"A proportional–integral–derivative controller (PID controller or three-term controller) is a control loop feedback mechanism widely used in industrial control systems and a variety of other applications requiring continuously modulated control. A PID controller continuously calculates an error value $e(t)$ as the difference between a desired setpoint (SP) and a measured process variable (PV) and applies a correction based on proportional, integral, and derivative terms (denoted P, I, and D respectively), hence the name.



On the figure: A block diagram of a PID controller in a feedback loop. r(t) is the desired process value or setpoint (SP), and y(t) is the measured process value (PV).

In practical terms it automatically applies accurate and responsive correction to a control function. An everyday example is the cruise control on a car, where external influences such as hills (gradients) would decrease speed. The PID algorithm restores from current speed to the desired speed in an optimal way, without delay or overshoot, by controlling the power output of the vehicle's engine.

The first theoretical analysis and practical application was in the field of automatic steering systems for ships, developed from the early 1920s onwards. It was then used for automatic process control in manufacturing industry, where it was widely implemented in pneumatic, and then electronic, controllers. Today there is universal use of the PID concept in applications requiring accurate and optimised automatic control." [8]

The algorithm for line following is shown on Figure 5.

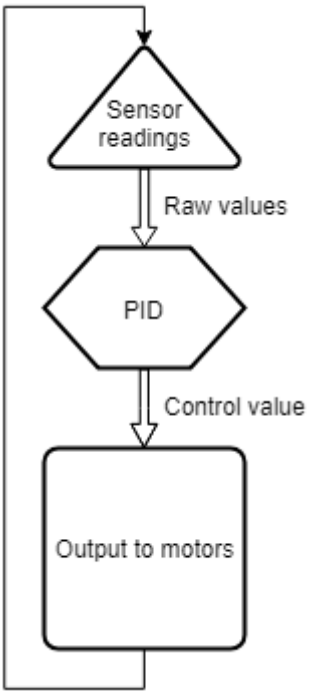


Figure 5

The PID controller outputs a single control value based on the readings from robot's sensors, which corresponds to an offset from the given setpoint. This value is then passed to the motor controller which then sets the speeds of both motors. This allows the robot for continuous adjustment of its movement based on the position relative to the line.

5.4. Maze

For position tracking a structure was implemented. It consists of x and y coordinates of types int and enumeration for direction (forward, left, right, backward). Current position of robot is constantly updated on each detected cross and after each turn.

When detected an obstacle, considering the position robot should choose either left or right turn. It turns left if x <= 0 and right otherwise. This turns priority check helps to keep robot as close to center as possible and avoid obstacles on maze's borders by turning to the right direction.

If robot faces the obstacle once, true value is saved to special variable, along with the previous direction. This information helps to avoid obstacles even if they situated on different parts of maze (left and right) and conflicts with our x <= 0 condition for turns.

When robot reaches the last full maze's line (y = 0), then it turns in the right direction (if needed) and moves to the finish line. The algorithm for that case is shown on Figure 6.

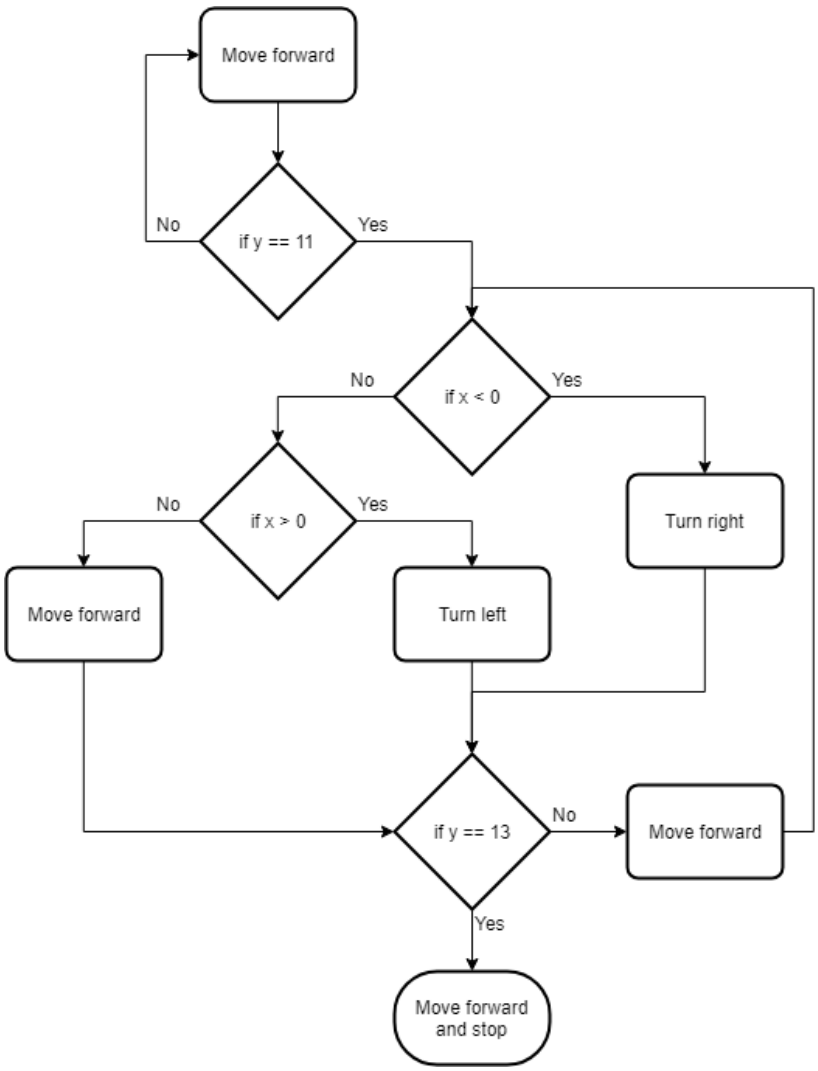

Move forward
if y == 11
No
Yes
if x < 0
No
Yes
Turn right
if x > 0
No
Yes
Move forward
Turn left
if y == 13
No
Move forward
Yes
Move forward and stop

Figure 6

6. Results

The robot functioned during the competition as expected without visible issues turning up in a second place in the line following, being outrun by another robot by as little as 18 milliseconds, as seen in the log extract below.

```
Zumo018/ready line
Zumo033/ready line

Zumo018/start 18375
Zumo018/stop 48840
Zumo018/time 30465

Zumo033/start 12163
<…>
Zumo033/stop 42646
Zumo033/time 30483
```

The robot was able to complete all the competitions as expected, including also maze and sumo. The only issue was encountered when the robot was shut off earlier than expected, thus resulting in sending an incorrect log as seen below, though it should be noted, that when operated correctly, the program does not encounter such problem and sends a correct and complete log.

```
Zumo033/ready sumo
Zumo033/start 18091
Zumo033/hit 18600 173
<…>
Zumo033/hit 40313 151
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
```

7. Discussion

7.1. Before the competition

Our team had some challenges concerning robot's motor that was broken when we programmed it in the second week of working on our project. When the robot was changed, we adjusted our programs to fit the new robot.

Next obstacle was that we could use robots and all equipment only in the classroom, hence we used to come earlier or stay after classes. Moreover, we decided to cooperate distantly if someone had health issues, by using Telegram chat.

In our team it was hard for newcomers to catch up with the robot, yet we helped each other.

In our case, MQTT had a great effect on our line-following function. Specifically, since MQTT runs synchronously with the main program, thus taking away vital processing time needed for main algorithms, such as line following and sumo, a solution had to be found to overcome this problem. Since sending logs in real time resulted in complete malfunction of those. To overcome this difficulty a new log caching system had to be designed. The system allows for caching all log entries while running an algorithm and outputting all of them only after the end of the competitions when the main had already finished working. Once the system was ready, another problem arose: the robot can store up to 59 log entries before the memory overflow. Because of that, a solution was designed to send the log immediately, clear the log and start over. This allows the program to store and send any amount of log entries without any issues.

On the other hand, it was not hard to delegate our work between team members. We communicated well and could share issue solutions, brainstormed together.

7.2. After the competition

After the competition our team started investigating the results and came up with a decision that the batteries of the robot were not giving the full power due to several factors.

First of all, the batteries were charged about 11 hours before the competition and were not recharged again before the start, which resulted in a slight self-discharge of the batteries.

Secondly, the batteries have been exposed to a cold condition before the competition which could also lead to a slight degradation of battery performance.

This could be confirmed by the fact that when the batteries were charged in between the competitions, neglecting the self-discharge effect and heating up at the same time, a higher power output was noticed from run to run, resulting in a faster track completion each time, as seen below.

```
Zumo033/ready line
Zumo033/start 18413
Zumo033/stop 50894
Zumo033/time 32481

Zumo033/ready line
```

```
Zumo033/start 17243
Zumo033/stop 47773
Zumo033/time 30530

Zumo033/ready line
Zumo033/start 12093
Zumo033/miss 13580
Zumo033/line 13682
Zumo033/miss 17316
Zumo033/line 17608
Zumo033/stop 42208
Zumo033/time 30115

Zumo033/ready line
Zumo033/start 12163
Zumo033/miss 17440
Zumo033/line 17709
Zumo033/stop 42646
Zumo033/time 30483
```

The whole log can be found in Appendix 1.

8. Conclusion

At the end of the course our team made all the assignments and three tasks: sumo battle, race and maze.

While working on this project, our team has extended knowledge of C language and skills of working with external libraries. We designed and implemented solutions for line following, which included PD controller, MQTT logs cache system and also position tracking system for Maze.

Completing the tasks also required a lot of teamwork and communication. Our team learned to use different tools and techniques for the purpose of coordinating the process, so every member always knew his or her responsibilities and tasks. This approach helped a lot to overcome difficulties our team faced while working on the tasks.

9. References

[1] Multiple Authors, "GIT," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Git. [Accessed 29 November 2018].

[2] Multiple Authors, "Bitbucket," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Bitbucket. [Accessed 27 November 2018].

[3] Multiple Authors, "Trello," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Trello. [Accessed 29 November 2018].

[4] Pololu, "Zumo Shield for Arduino, v1.2," 2018. [Online]. Available: https://www.pololu.com/product/2508 . [Accessed 29 November 2018].

[5] Pololu, "Zumo Chassis Kit," 2018. [Online]. Available: https://www.pololu.com/product/1418. [Accessed 29 November 2018].

[6] "CYPRESS," [Online]. Available: http://www.cypress.com/file/157971/download. [Accessed November 2018].

[7] M. Autors, "Get started with Zumo library," 2018.

[8] Multiple Authors, "PID controller," 22 November 2018. [Online]. Available: https://en.wikipedia.org/wiki/PID_controller. [Accessed 26 November 2018].

10. Appendix 1. Competition logs

An entire competition log can be found here, including the whole log for Zumo033 and extract of Zumo018 logs during the final round of the line following competition.

```
// Line following
Zumo033/ready line
Zumo033/start 18413
Zumo033/stop 50894
Zumo033/time 32481

Zumo033/ready line
Zumo033/start 17243
Zumo033/stop 47773
Zumo033/time 30530

Zumo033/ready line
Zumo033/start 12093
Zumo033/miss 13580
Zumo033/line 13682
Zumo033/miss 17316
Zumo033/line 17608
Zumo033/stop 42208
Zumo033/time 30115

Zumo018/ready line
Zumo018/start 18375
Zumo018/stop 48840
Zumo018/time 30465

Zumo033/ready line
Zumo033/start 12163
Zumo033/miss 17440
Zumo033/line 17709
Zumo033/stop 42646
Zumo033/time 30483

// Maze
Zumo033/ready maze
Zumo033/start 11838
Zumo033/position 0 1
Zumo033/position -1 1
Zumo033/position -2 1
Zumo033/position -2 2
Zumo033/position -2 3
Zumo033/position -2 4
Zumo033/position -2 5
Zumo033/position -2 6
Zumo033/position -1 6
Zumo033/position -1 7
Zumo033/position -1 8
Zumo033/position -1 9
Zumo033/position -1 10
Zumo033/position -1 11
Zumo033/position 0 11
Zumo033/position 0 12
Zumo033/position 0 13
Zumo033/stop 22506
Zumo033/time 10668
```

```
// Sumo
Zumo033/ready sumo
Zumo033/start 18091
Zumo033/hit 18600 173
Zumo033/hit 18719 188
Zumo033/hit 18734 213
Zumo033/hit 20231 88
Zumo033/hit 20242 240
Zumo033/hit 20321 141
Zumo033/hit 20346 191
Zumo033/hit 21874 259
Zumo033/hit 21889 230
Zumo033/hit 21973 107
Zumo033/hit 21993 186
Zumo033/hit 23467 241
Zumo033/hit 23586 194
Zumo033/hit 25109 252
Zumo033/hit 25223 201
Zumo033/hit 25866 86
Zumo033/hit 25916 320
Zumo033/hit 25931 206
Zumo033/hit 26307 233
Zumo033/hit 27167 265
Zumo033/hit 28484 247
Zumo033/hit 28583 202
Zumo033/hit 28602 173
Zumo033/hit 29310 310
Zumo033/hit 29344 71
Zumo033/hit 29364 259
Zumo033/hit 29384 189
Zumo033/hit 29428 241
Zumo033/hit 30098 79
Zumo033/hit 30111 248
Zumo033/hit 30225 203
Zumo033/hit 30997 173
Zumo033/hit 31358 263
Zumo033/hit 31482 189
Zumo033/hit 31537 156
Zumo033/hit 31616 298
Zumo033/hit 31635 82
Zumo033/hit 31670 110
Zumo033/hit 31690 285
Zumo033/hit 31705 92
Zumo033/hit 31714 94
Zumo033/hit 31744 97
Zumo033/hit 31789 99
Zumo033/hit 31804 117
Zumo033/hit 31823 299
Zumo033/hit 33705 256
Zumo033/hit 34471 199
Zumo033/hit 34832 309
Zumo033/hit 34851 171
Zumo033/hit 34866 69
Zumo033/hit 37339 78
Zumo033/hit 37345 248
Zumo033/hit 38087 237
Zumo033/hit 38221 186
Zumo033/hit 39051 97
Zumo033/hit 39061 261
Zumo033/hit 39364 336
Zumo033/hit 39452 171
Zumo033/hit 39482 196
Zumo033/hit 39492 348
Zumo033/hit 39507 305
Zumo033/hit 39591 162
Zumo033/hit 39601 335
Zumo033/hit 39615 145
Zumo033/hit 39626 342
Zumo033/hit 39705 325
Zumo033/hit 40106 163
Zumo033/hit 40140 344
Zumo033/hit 40150 155
Zumo033/hit 40160 340
Zumo033/hit 40175 148
Zumo033/hit 40185 355
Zumo033/hit 40199 191
Zumo033/hit 40274 350
Zumo033/hit 40288 210
Zumo033/hit 40313 151
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
Zumo033/hit (null)
```