Zumo project

Project report

Rukhsora Nazarova, Mikhail Stefantsev, Mariia Saveleva

2018

# Robot projects

Table of Contents

[Robot projects 0](#_Toc531940348)

[1. Abstract 2](#_Toc531940349)

[1.1. Summary of the project. 2](#_Toc531940350)

[2. Introduction 3](#_Toc531940351)

[2.1. The goal and scope of the project. 3](#_Toc531940352)

[3. Methods and materials 4](#_Toc531940353)

[3.1. GIT 4](#_Toc531940354)

[3.2. Bitbucket 4](#_Toc531940355)

[3.3. Trello 5](#_Toc531940356)

[4. Theoretical background 7](#_Toc531940357)

[4.1. Zumo Shield 7](#_Toc531940358)

[4.2. Zumo Chassis 7](#_Toc531940359)

[4.3. PSOC CY8CKIT059 7](#_Toc531940360)

[4.4. Sensors 8](#_Toc531940361)

[4.4.1. Ultrasonic sensor 8](#_Toc531940362)

[4.4.2. Reflectance array 8](#_Toc531940363)

[4.4.3. Accelerometer 8](#_Toc531940364)

[4.5. MQTT 8](#_Toc531940365)

[5. Implementation 10](#_Toc531940366)

[5.1. Batteries 10](#_Toc531940367)

[5.2. Line following. 11](#_Toc531940368)

[6. Results 13](#_Toc531940369)

[6.1. 13](#_Toc531940370)

[7. Discussion 14](#_Toc531940371)

[7.1. 14](#_Toc531940372)

[8. Conclusion 15](#_Toc531940373)

[9. References 16](#_Toc531940374)

## Abstract

* 1. Summary of the project.

Robot project consists of three main tasks.

Line following – is held by using 6 reflective sensors of the Zumo robot. Idea of this task is to follow the black line with curves, which has the length of approximately 10 m.

Sumo wrestling is battle of three robots. Every robot tries to push other robots out of the ring. The last robot at the ring will be a winner.

Maze task’s requirement is by avoiding obstacles on the grid, robot should get to the end of it.

Tasks require MQTT and IR remote usage.

## Introduction

* 1. The goal and scope of the project.

The goal of the robot project is to teach/maintain our knowledge about C programming language, algorithms, 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.

## Methods and materials

/\* What methods and materials are we using. Creator, libs \*/

* 1. GIT

Our team use 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]

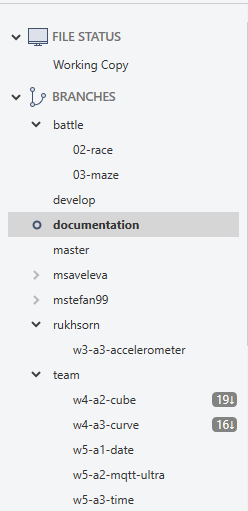
It allows multiple developers to access, edit and upload their changes to shared repository that is situated on remote server.  
In our project, each week’s assignment and each final project was made in a separate branch. The project branch hierarchy shown on Figure 1.  
  


Figure 1

* 1. 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 is also available, but the reason our team choose Bitbucket was because of free private repository feature.

* 1. 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.

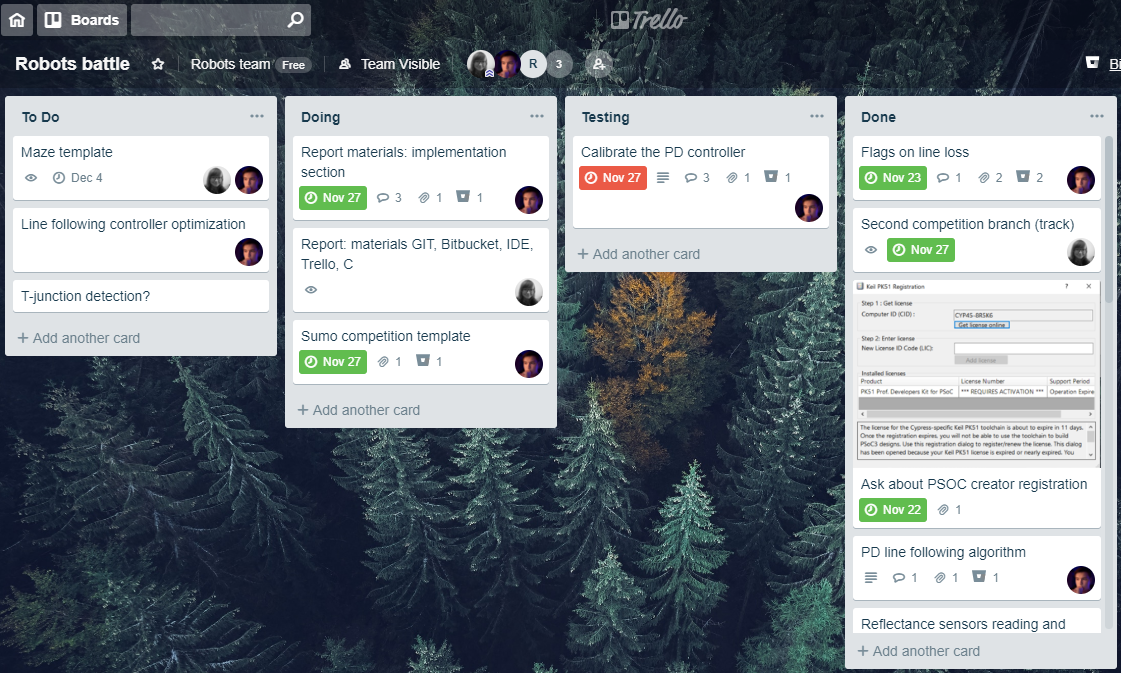


Figure 2

## Theoretical background

/\* Some theory about how sensor, robot works, how some algorithms work \*/

* 1. Zumo Shield

The Zumo Shield consists of dual motor drivers, a buzzer that is able play mere sounds and music, a user push button that allows to switch on and off the robot and an accelerometer that measures acceleration in 3 dimensional, compass and gyroscope that senses impacts and tracks orientation.

The Shield fastens to the chassis, merges to its battery terminals and motors. In our case, CY8CKIT059 inserts into the Shield.

The Shield’s more detailed features:

* Integrated DRV8835 dual motor drivers capable of delivering enough current for two high-power (HP) micro metal gearmotors.
* Piezo buzzer for playing simple sounds and music. The buzzer is controlled by one of the Arduino’s PWM outputs, so the tones can be generated in the background without taking up a lot of processing power.
* Integrated LSM303D 3-axis accelerometer and 3-axis magnetometer that can be used to detect impacts. The compass gets a lot of interference from the motors, batteries, PCB, and its surroundings with proper calibration, it can be used for rough orientation measuring in many environments.
* Integrated L3GD20H 3-axis gyroscope that can be 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) so that your robot can its orientation.
* Optional user pushbutton.
* 7.5 V boost regulator for powering the Arduino 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).
  1. Zumo Chassis

The chassis is made from black ABS plastic and has sockets for two micro metal gearmotors and a compartment for four AA batteries. The battery compartment terminals protrude through the chassis and can be accessed from the top side. A black acrylic plate is included with the chassis. This plate holds the motors in place and 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 tread.

* 1. PSOC CY8CKIT059

PSoC 5LP is integrated programmable SoC, combining high-precision and programmable analog and digital peripherals with an 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.

* 1. 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.

* + 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 took the signal to reflect from the object in front of the robot and hit the sensor.

* + 1. 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\_.

* + 1. 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 , and direction.

* 1. 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 WiFi 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.

## Implementation

* 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: , where is the reference voltage of the ADC (*5V as specified in the documentation)* and is number of bits used by ADC (*in our case, =12*).

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

where is the equivalent resistance of and .

So, the source voltage equals to:

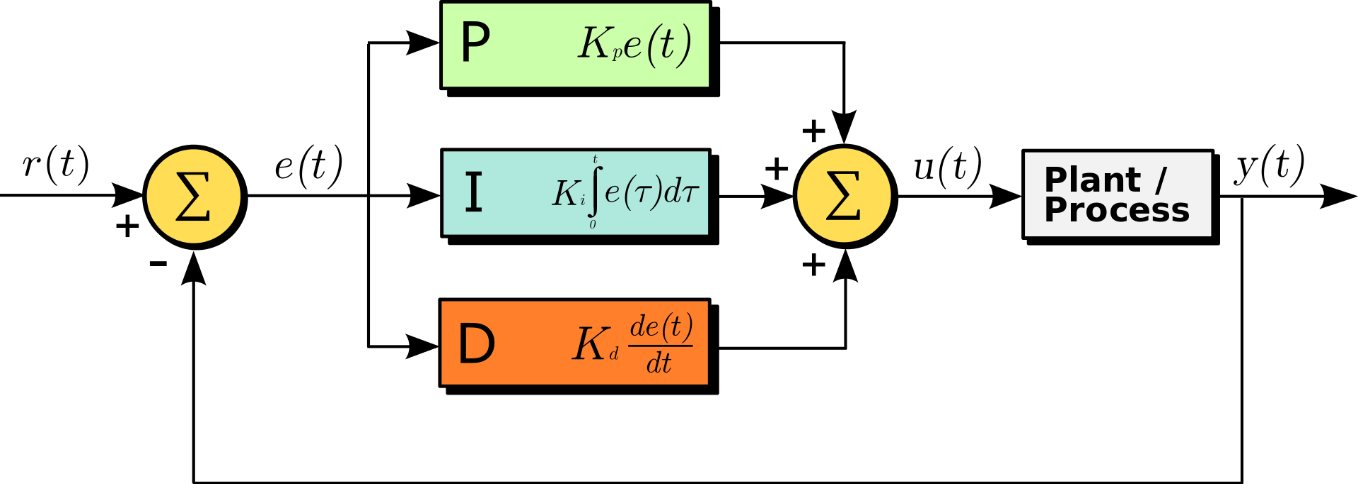
where is the output level of ADC (*an integer number between and* ).

* 1. 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 ) as the difference between a desired setpoint and a measured process variable 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.” [4]

The algorithm for line following is shown on Figure 4.

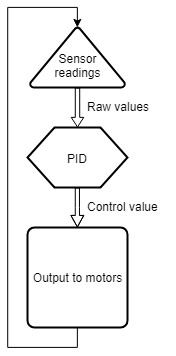


Figure 4

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.

* 1. Maze

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

When detect an obstacle, consider the position robot could 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 turn it to the right direction (if needed) and move to the finish line. The algorithm is shown on Figure 5.

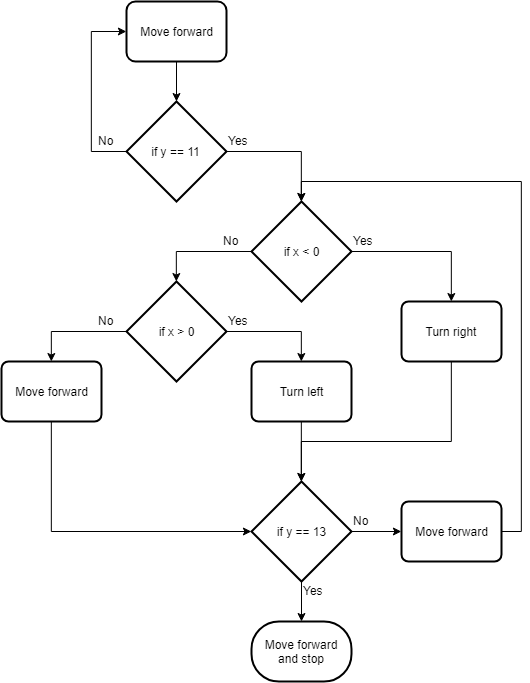


Figure 5

## Results

/\* won or not, how was the challenges \*/



## Discussion

/\* What was challenging? Did we have problems? How did we solve them? Why did robot fail, for example? \*/



## Conclusion

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

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

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

## 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] | Multiple Authors, "PID controller," 22 November 2018. [Online]. Available: https://en.wikipedia.org/wiki/PID\_controller. [Accessed 26 November 2018]. |
| [5] | Pololu, "Zumo Shield for Arduino, v1.2," 2018. [Online]. Available: https://www.pololu.com/product/2508 . [Accessed 29 November 2018]. |

CY8CKIT-059 UserGuide

Zumo library