XLC-Innovation 2021

Jakub Gál[1] Šimon Peter [2] Rastislav Gaži [3[Mentor]]

[1] SPŠSE Nitra, Slovakia, bukajlag@gmail.com
[2] Gymnázium Golianova, Slovakia
[3] XLC team, Slovakia
Homepage: xlc-team.info
Contact Email: bukajlag@gmail.com

Abstract. In this paper, we aim to present the solution of our Robot for RoboCup Junior Soccer Open 2021. This year we are bringing a few new major innovations such as using parallel EEPROM for computing angle of intersection with the line or our own architecture of a convolutional neural network that runs on Coral Edge TPU. We made some mechanical innovations as well, such as the usage of slidable cases for batteries, which allows us to concentrate 1,4Kg of weight on the bottom 40mm or a custom solenoid kicker, which works without the need of a voltage pump, increasing the safety. The whole robot is open source to help the community as much as possible. In case of any questions or suggestions please don't hesitate to contact us at the email address bukajlag@gmail.com.

1 Introduction

1.1 Background

We are the team XLC Innovation, which is a subteam of a bigger organization of the team XLC. Our team consists of two members Jakub Gál and Šimon Peter and a mentor Ing. Rastislav Gaži.

Jakub Gál is the captain of our team and is in charge of all Software and Electronics. He participated in Robocup Junior 2020 virtual poster session in the Open category where he won the best poster award in his category, in RoboCup Junior 2019 in Sydney in the Light Weight category he won a prize for the best poster & presentation and 2nd place in superteam games, as well as 14th place in individual games. He was also at RoboCup 2017 Nagoya in the Light Weight category where he won 2nd place in superteam games. He also won several prizes at a national level in Soccer Open, Soccer Light Weight, as well as rescue.

Šimon Peter is in charge of Mechanical Engineering and 3D printing. He was also a participant of RoboCup 2017 in Nagoya in the Light Weight category where he won 2nd place in superteam games and he also won several prizes in RoboCup Junior at a national level.

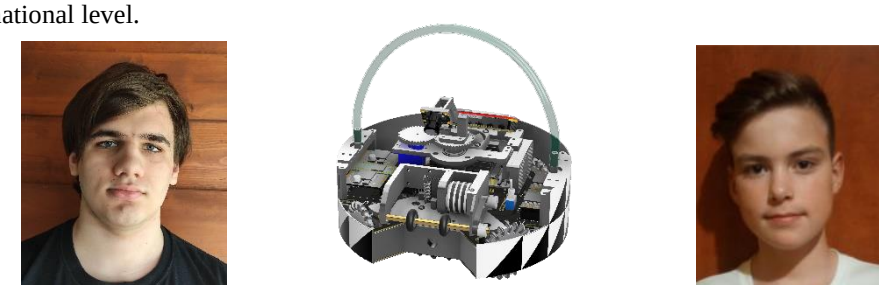

Fig. 1 Members of XLC-Innovation team consisting of Jakub Gál(left) and Šimon Peter(right) and this year robot(middle)

1.2 Highlights

As our team name suggests we think that innovation is the most crucial aspect when creating robots for the RoboCup Junior competition so, as every year, we decided to bring something new for this year's competition as well. Last year we used a convolutional neural network for detecting objects on the playfield, which was implemented on Coral Edge TPU for the best power and performance ratio, for the first time. This year we used CNN as well but instead of tiny YOLO V3, we created an optimized architecture just for RoboCup Junior Soccer Open. We also used Jetson Nano instead of Raspberry Pi 4 for even better performance. Another big innovation is the usage of parallel EEPROM for data processing instead of the usage of multiple microprocessors to drastically increase the processing speed of line intersection. Other mentionable innovations can be, for example, the usage of ARM Cortex-M7 on Teensy 4.1 as the main microprocessor that is overclocked to 1.008 GHz and using a brushless motor for the mechanism for ball manipulation.

2 Robots and Results

2.1 Hardware

Methods and bill of materials

We used multiple pieces of software for creating the robot's hardware. For creating PCBs we used mainly Eagle but also Allium Designer. Then for creating a 3D model of the robot we used Fusion 360 where we also created 3D models of PCBs to be sure everything would fit. For PCB manufacturing we used JLCPCB and for manufacturing the majority of mechanical parts we used FMD 3D printing. We used PLA and PET-G depending on the use case. We also used CNC for manufacturing our omniwheels.

Table 1. Bill of Materials

Name of part	Count	Cost for all
Jetson Nano	1	65$
Coral Edge TPU	1	60$
DC motors	4	24$
XA2212 motor	1	10$
Teensy 4.1	1	28$
VNH5019	4	25$
IMX 219 Camera	1	10$
Omniwheels	4	40$
PCBs	3	35$
Other	-	40$

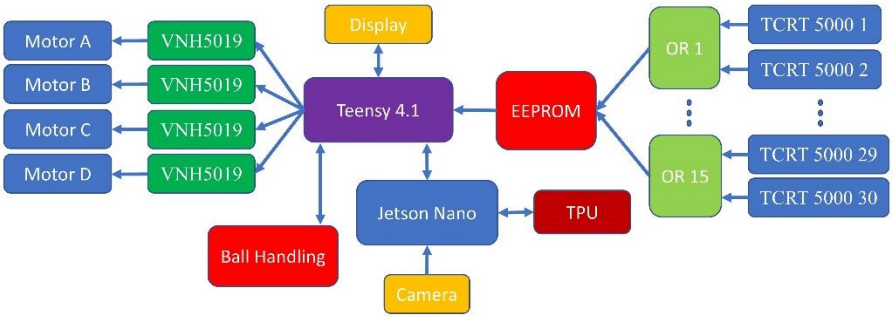

Fig. 2 Connection diagram of our robot

Movement system

For robot movement, we use four DC motors connected to 560rpm ZHENG25RP gearboxes that are held in place using parts, created from aluminium for better heat dissipation, that connect them to the mainboard. All four motor shafts are connected to custom omniwheels created using CNC. They are controlled by VNH5019 ICs that are placed on modules on the top side of the mainboard. They are very durable and in case of failure can be changed very quickly. They also have current sensing, which we use to detect whether the motor is stuck to prevent damage. Speed of the robot can be easily upgraded using, for example, Maxon RE Motor 339150 with Maxon 5.8:1 Gearhead 166157 or JoinMax motors but they are much more expensive and they wouldn't help innovate technology so we rather spent the money on prototyping.

Ball handling

The ball handling mechanism consists of two main parts which are the dribbler and the kicker. Kicker consists of a custom solenoid coil that we made to work most effectively at around 15V. This allowed it to be safer and we no longer needed a step-up converter like the one we used in our last year's robot design. For solenoid winding, we used 10.08m of 0.71mm copper wire. For powering the solenoid we used two 18000uF 25V CDE capacitors connected in parallel that are charged directly from the battery and then discharged through MOSFET to the solenoid to kick the ball.

For ball manipulation, we used a dribbler. The main change from the previous year is that we used a brushless motor instead of a brushed DC for better efficiency and speed. As the driver, we used 30A BLlheli-s ESC. We also made structural changes to make it more reliable as well as stronger. For motion transfer from the motor shaft, we used a GT2 belt to dampen the initial impact when catching the ball. To create better contact with the ball we also used springs. The dribbler’s initial angle and pressure can be changed using screws for adaptation to different types of playfield surfaces.

Power

As an energy source for our robot, we used four 18650 Li-Ion batteries connected in series. They are placed on top of the bottom PCB in two slidable cases that allow access. This placement also allows them to be just 15mm on top of the surface of the playfield, which helps with the stability of the whole robot. For voltage change to 5V, we used three separate LM2596S buck converters that can each output 3A. We used three of them to power different parts of our robot to make damage in case of failure as small as possible. Also for 3.3V, we used linear LM1117 for better voltage stability. Another safety precaution is the usage of 3 different fuses based on the type of load for the best switching speed possible.

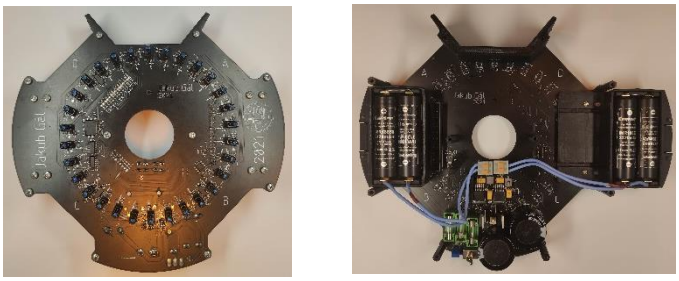

Fig. 3 Bottom PCB of our robot with power system, line detection system and kicker system

Line intersection detection

For detecting intersection with the line we used a circle of 30 pairs of phototransistors with IR LED(TCRT5000). They are connected to TC75S56FUTE85LF comparators, which compare voltage from phototransistors with an analogue reference set by a trimer. Outputs of each two comparators are then connected to OR gates MC74AC32. Outputs of OR gates are then connected directly to a parallel EEPROM AT28BV256 address output. We programmed EEPROM in a way that each address refers to an angle of intersection with the line scaled to eight-bit values without zero, which is reserved for situations when the robot doesn't see a line. Then these eights bits are converted to 3.3V logic that is compatible with MIMXRT1062DVJ6A (Teensy 4.1), using TXB0108PWR. We also used another OR gate to create an interrupt signal when at least one of the areas of pins is at a high logic level. This signal is also converted to 3.3V logic using BSS138 and is used as an interrupt to speed up reaction time by Teensy. By using this method, the time needed to detect line intersection angle is reduced to less than 3 microseconds (phototransistor maximum reaction time is 2 microseconds, but EEPROM maximum read time is 200ns). When creating this system we firstly wanted to use an AS29CF160B-55TIN NOR Flash memory, which would be a little bit faster and would allow for five more address inputs, however, we were not able to program it correctly(we used our own device with an ATmega 2560 for programming, because we don't have a flash memory writer). We also wanted to use an FPGA but this approach turned out to be better and faster.

Human interface

We also innovated our approach to interface with the robot on the playfield. In the past, we used a 1602 alphanumeric LCD display together with push buttons but we decided to change it for 2.4 TFT Color Screen LCD with push buttons for additional inputs. This allows for a more intuitive input together with more possibilities.

Navigation

Primarily for getting orientation data we decided to this year use MPU6050 that is placed directly in the axis of rotation of the robot to achieve the best results. We found out that the accuracy of the gyroscope is good enough but if needed it can be increased by using a BNO055 or any other I2C device.

Camera

We used an IMX219 wide-angle camera that is connected to Jetson Nano through a CSI-2 bus. This camera is placed on an adjustable arm that can rotate. Our first plan was to use a slipring to connect the cables from the camera to Jetson Nano but the connection turned out to be too loud and the impedance difference too big to get it working correctly, so we used an FPC cable. Many sliprings are made to be able to work with very low noise but they are much more expensive (the one we own should work too, but as it turned out it does not). For the running of the convolutional neural network, we used Coral Edge TPU connected through USB3 to the Jetson Nano. Then we used a UART to communicate between the Jetson Nano and the Teensy 4.1.

2.2 Software

Methods

For the programming of our robots, we used Python to create and run the neural network for object detection and C++ to program teensy and EEPROM. As a text editor, we used mainly Visual Studio Code and for Teensy programming, we used Arduino IDE. We also used Git and GitHub for the development.

Movement

Our robot can move in any direction needed. This is achieved by vector movement. We created a hashtable, which contains the ratio of motors' speeds that is needed for each direction. When the robot has moved this ratio is then scaled to the needed speed and regulated by a PID regulator. For PID regulator tuning we used an experimental approach. Firstly, we increased parameter P until the robot oscillated. Then we increased parameter D to get the robot more stable. We repeated this until it was possible to get the robot stable. Then we increased parameter I until the steady error was

eliminated. We also tried to use the Ziegler-Nichols method but it did not have as good results as the experimental approach.

Strategy

There are two different roles that the robots have- one is an attacker and one is a defender. When the robot is in attack mode it is always turned towards the opponent's goal and is trying to catch the ball. When it catches the ball it then turns towards a free spot in the goal and kicks the ball. On the other hand, when the robot is in defence mode he is always facing the opponent' side and is trying to stay behind the attacker. When the ball is closer to the defender than to the attacker the roles change. When the robot doesn't see his teammate it is automatically in attack mode. Whenever the robot detects the line it goes in the exact opposite direction from it.

Object detection

For the detection of objects on the field, we used to use recognition by their colour, but this approach needed calibration before every match and was highly influenced by the change of illumination as well as surrounding colours. This is why we decided to use a convolutional neural network which is much more robust and doesn't need calibration before matches. We used google ColabFor training this network because we don't own GPUs with enough power. Last year we used tiny YOLO V3, however, this year we decided to create our own optimized architecture. We started with YOLO V4 and then scaled it down and changed optimalization functions to RELU as well as other parameters to make it more optimal for our dataset. We also created a dataset of 2000 images that contains the ball, the goals and our robot. This dataset is divided into training and testing and for additional testing we also used a video. We augmented this dataset using random noise, brightness change and data loss for better generalization of the model as well as for preventing overfitting. After training, we also used quantification of weights to int8 for an even better ratio between performance and speed.

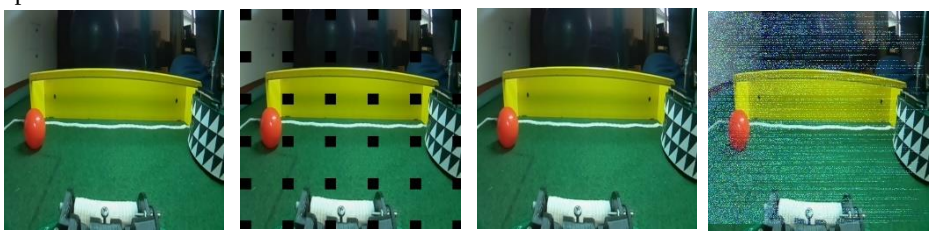

Fig. 4 Examples of argumentation used in our dataset

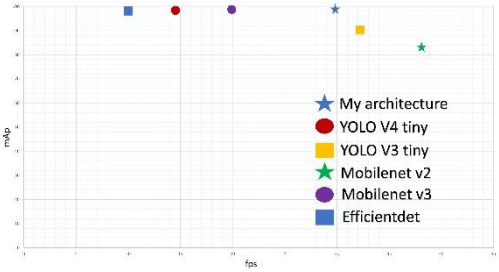

Fig. 5 Different neural networks architectures comparation

3 Conclusions and Future Work

3.1 Results

While making this year's robots we had to make many iterations of different designs. While making the dribbler we tested different shapes to achieve the best holding strength possible. We firstly tested designs with lines for cantering but it turned out to be inefficient. We also tried an angled type, which turned out better but the best one was designed with two separable wheels. We also tested different materials but decided to use silicone with glycerol as it turned out to be the best for this specific purpose.

For hardware that would be used to run the neural network for object detection, we firstly used Raspberry PI 4 but it turned out too slow so we used the USB Coral Edge TPU accelerator, which is very power efficient while maintaining the power of 4TOPS. Additionally, we decided to switch Raspberry Pi 4 for Jetson Nano because of its GPU, which helped while developing the robot as well as for acceleration of functions that cannot be executed on TPU.

When creating the object detection algorithm we firstly used an approach based on recognition of different colours of objects, however, this approach significantly depends on correct and stable light conditions, as well as on the fact that the surrounding environment doesn't contain similar colours to the detected object. Therefore, we tried different existing architectures of neural networks for object detection and used the best one, which at that time turned out to be tiny YOLO V3. However, we still saw space for improvement so we decided to create our own architecture based on YOLO V4, which turned out to be the most satisfactory one.

In some parts of our robot, we built on our previous experience, for example, when creating robot strategies that we then tested and chose the best of them.

3.2 Summary

While creating this robot we learned a lot of new information and gained many new experiences ranging from new finding out how hard using a NOR Flash Memory can be or finding where to buy glycerol and silicone in Slovakia to gaining new skills such as better knowledge of TensorFlow and NumPy and better soldering skills. We also learned to manage long-distance work by using GitHub, Zoom and other programs because of the Covid-19 pandemic.

3.3 Future

We have many plans that we want to implement in the future. One which is already in the last stages of development is a small camera that contains Coral Edge TPU along with a Compute module 4 and two replaceable IMX219 sensors. This camera also offers three USB-C connectors together with 1x HDMI,1x micro HDMI, 1x USB-A and a 40 pin GPIO that contains IC2, SPI, UART and more.

Another project we want to try is using Reinforcement Learning for creating a robotic strategy. We already tried it in a simulator but a hand-coded solution is always better. We see great potential in using reinforcement learning.

We also wanted to try using FPAA or dpASP with a neural network for object detection connected directly to a camera sensor. This approach would eliminate the need to convert the signal from analogue to digital as well as the need for all the connection blocks that slow down the performance of the whole system. However, there are not many FPAAs or dpASP on the market now. We found very few of these chips but none that would fulfil our requirements as they are not used very often. This may push us to create our own chip for this purpose in the future or to make it using discrete components, which would bring many other problems. However, we think that this idea is worthy of at least being tested.

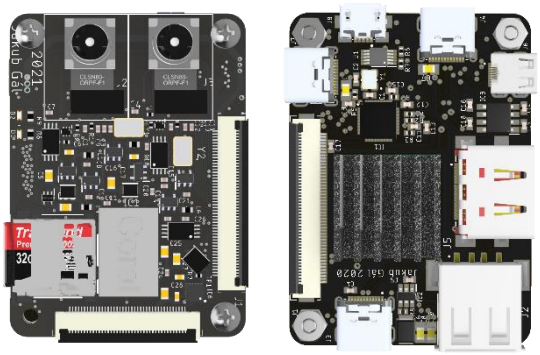

Fig. 6 Front and bottom views on our new camera design

4 Reference

1. 3D models of our robot, https://grabcad.com/jakub.gal-2/models
2. XLC team dataset for Robocup Junior Soccer Open, https://www.kaggle.com/jakubgal/robocup-junior-open-xlc
3. Source code of our robot, https://github.com/xlcteam
4. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, YOLOv4: Optimal Speed and Accuracy of Object Detection, https://arxiv.org/abs/2004.10934
5. Joseph Redmon, Ali Farhadi, YOLOv3: An Incremental Improvement, https://arxiv.org/abs/1804.02767
6. Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, https://arxiv.org/pdf/1801.04381.pdf
7.Webside of TensorFlow, https://www.tensorflow.org/
8.Webside of Coral Edge TPU, https://coral.ai/