ThunderVolt Description Paper 2022

Lucas Haug[1], Lucas Tonini Rosenberg Schneider[1], Gabriel Cosme Barbosa[1], Arthur Pedroso Porto Belli[1], Felipe Gomes de Melo D'Elia[1], Jonas Gomes de Morais[1], Leonardo Isao Komura[1], Maria Eduarda Dall'Orto de Araujo[1], Pedro de Azeredo Nogueira[1], Pedro Henrique Machado Almeida[1], Ricardo Tamay Honda[1], Sergio Magalhães Contente[1], Thalles Carneiro da Silva[1] and Thays Fonseca Romano[1]

***Abstract*— ThunderVolt is the IEEE Very Small Size Soccer team of ThundeRatz, the robotics group from the Polytechnic School of the University of São Paulo. The team was debuted in 2020 and, since then, it has evolved a lot, specially in the simulated environment. Last year, for example, we obtained second place in the 3x3 simulated category from the Brazilian Robotics Competition (CBR) and now we are currently in third place at the Brazilian Simulated Robot Soccer Championship. With that in mind, the last few months have been focused on developing the hardware and firmware of the robots and also the vision system for the upcoming competitions. In this paper, we discuss more deeply the work that has been done in all different areas of the project and the plans we have for future development.**

I. INTRODUCTION

The *IEEE Very Small Size Soccer (VSSS)* competitions have grown a lot in the national robotics scenario, mainly because robot soccer is a challenging and stimulating problem that requires a complete engineering solution. That is why some members of ThundeRatz, the robotics team from the Polytechnic School of the University of São Paulo, started developing the ThunderVolt project in 2018.

The solution designed for ThunderVolt can be divided into five different modules: the mechanics, responsible for the physical specification of the robot; the electronics, which consists in the hardware used on each of the team's robots and on the RF transmitter; the firmware, that comprises the C code embedded on the devices described above; the game strategy, where the decision-making and the navigation systems are implemented; and the interfaces, responsible for the communication between the game strategy module and the outside world (or the simulation), which includes, for example, the vision system, the serial connection with the RF transmitter and the UDP sockets used in the simulation. This paper describes in more details each of theses modules and how they were developed.

II. MECHANICS

The main change in our mechanical project was the chosen material in the 3D Fused Deposition Modelling (FDM) process. Some problems were found in the FDM tests with acrylonitrile butadiene styrene (ABS), the previous selected material, mainly due to the difficulty to maintain the fusing temperature of the plastic in the available 3D printer. So, it was decided to use Polyethylene terephthalate glycol (PETG), a plastic with similar properties to ABS but with lower fusion temperature. An rendering of the CAD project is shown in Fig. 1

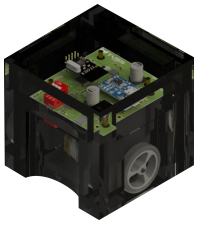

Fig. 1. Vision reference points

III. ELECTRONICS

For its complete operation, the project uses two types of embedded devices: the robots and a transmitter dongle. The transmitter is connected to a USB port of the computer running the game strategy and is responsible for receiving the speed commands for each robot from the connected computer and transmitting them to the robots using the Nordic Semiconductor's radio frequency module nRF24L01. The robots then receive the speed commands using the same type of radio module and control their motors accordingly.

Since last year, this behavior has not changed, but some of the hardware of the project had to be revised and modified. The main points were described in the following section.

A. nRFDongle

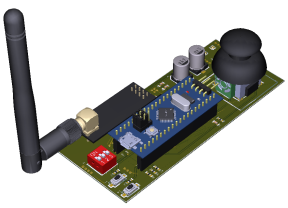

Fig. 2. 3D model of the PCB ("nRFDongle") from Thundervolt

Following last year's development, the team implemented the new iteration of our dongle board. This board is connected to the computer running the strategies via USB. The

[1]All authors are university graduate students at the Polytechnic School of the University of São Paulo and members of the ThundeRatz Robotics Team. contato@thunderatz.org

commercially known as "BluePill" board was chosen for this application, since it has a STM32F103C8T6 microcontroller onboard and not much computing power is needed for encoding of messages and transmitting them, it was also conveniently chosen in the way that it receives 5V from the voltage regulator and supplies back the 3V3 needed by some of the electronic components on the board.

The communications between the robots and the dongle board are made using the Nordic Semiconductor's radio frequency module nRF24L01, also attached to the board. This module is responsible for sending the speed commands encoded by the Bluepill to the robots.

This PCB also features other components that are used to test individual robots, such as a three way DIP Switch to select a robot (up to 7 different robots can be selected this way), a joystick to pilot the selected robot and 2 buttons that perform other specific functions.

B. TPM VSS

The VSS board, now renamed as TPM-VSS (TPM means Thunder Power Management, which ThundeRatz uses to identify electronic boards that power robots), is undergoing a redesign, however, the old PCB boards have been welded and tested, and succeeded satisfactorily in all tests performed by the team.

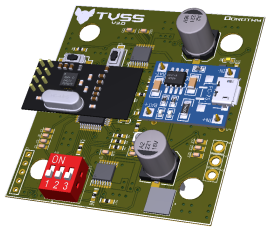

Fig. 3. 3D model of the Old PCB ("VSS") from Thundervolt

The reasons the PCB boards are in the process of a new project are: the difficulty of finding the components in the market, as some of its integrated circuit components are out of stock, a better optimization in the organization and improvement in electronics by redesign of routing and vias.

In that vision, it is important to highlight the two of its main components which had to be revised and besides that, a important component that was already present in the previous board project, but was not used before.

Motor Encoders: To control the velocity and the positioning of a robot, knowing the rotation of the motor is fundamental. Therefore, it was chosen, since last year project, the Magnetic Encoder Pair Kit for Micro Metal Gearmotors, by Pololu. This components are fitted to the motors and connected to the board by cables.

H-Bridge Motor Driver: In the old project, the driver that it was supposed to be used was the DRV8874 by Texas Instrument, but, because of its lack in the market, the DRV8871, by the same manufacturer, was utilized meanwhile, a driver that was used before in the old boards. So, to replace it, the DRV8231ADSGR, by Texas Instrument, was chosen. Its functioning is very similar to the old one, but with the addition of a pin for sensing electric current from the motor, in order to facilitate the monitoring of the performance of the robots.

Boost voltage regulator: The change of the boost voltage regulator, in the redesign, will be the replacement of the current component, TPS61089RNRR, by the TPS61088QRHLTQ1, both from Texas Instrument. This change was needed, just because the old one is missing in the market, and the new one performs the exact same function as the previous one.

It is noteworthy that most components from the project could be replaced considering the stock instability of electronic components in the present moment.

IV. FIRMWARE

The firmware of both embedded devices was initially based on the STM32ProjectTemplate [7], an open source template created by ThundeRatz to simplify the firmware development for the STM32 family of microcontrollers [18] using the STM32CubeMX software [10]. Besides that, ThundeRatz wrote the STM32Guide [20], a public guide to help programming the STMicroelectronics' microcontrollers.

The next two sections describe the main changes that where made in the firmware since the last version.

A. Communication

Instead of using UART, the new version of the nRFDongle board's firmware will communicate using the USB protocol. Thus, the Cyclic Redundancy Check (CRC) algorithm that was used in the old versions has become obsolete, since the USB protocol already implements CRC on every package sent.

Another characteristic that have been changed since the last version of the firmware is the message that is sent. Now, the USB sends the real velocity of each wheel in rad/s and not the PWM's duty cycle.

B. Control

With the addition of the incremental encoders in the robot's project, it was necessary to develop the part of the firmware that would handle this device, and also the closed control loop for the wheel's speed.

Firstly, the readings from the encoder were made using timers with Encoders Mode [19], a feature from the STM32 micro-controllers that configures two channels from the same timer to work together in order to read quadrature signals. After that, the timer value is read in a regular interval and the difference in the number of counts is used to estimate the wheel's speed.

Now, the encoders can be used to collect data about the motor system. One important test that was made consisted in applying different input duty cycles and comparing to the output speed in rad/s, which can be seen in Figure 4.

This acquired data was used to build a cost function for the parameters of a first order LTI system (linear time-invariant system). This cost function was then minimized using the

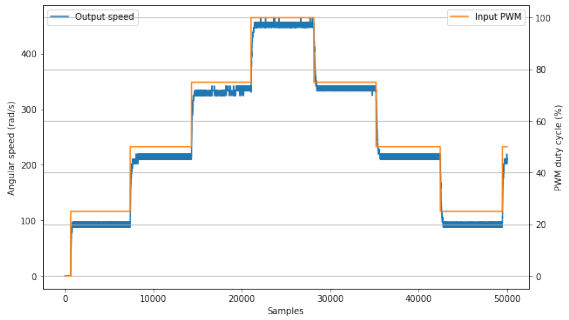

Fig. 4. Motor input duty cycle and output speed in time

Nelder-Mead method and the obtained system was compared to the real values using the same input, as shown in Figure 5.

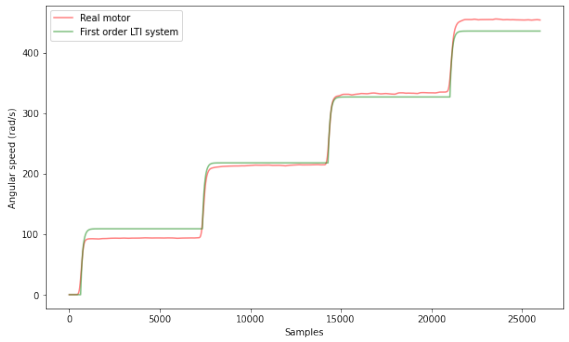

Fig. 5. Comparison between real motor and obtained linear system

Lastly, the LTI system was used with an Linear Quadratic Regulator (LQR) to estimate the optimal PID constants to control the system. These constants were used with an Anti-windup PID Controller described in [2], which was implemented in the robot's firmware and would take the speeds sent by the nRFDongle as setpoint and the actual speeds calculated with the encoders as feedback.

V. GAME STRATEGY

The game strategy as described in this section comprehends all the necessary features in order to decide which task each robot should execute and how to execute it.

The definition of a game strategy for multiple robots in a dynamic environment is a complex task, composed of several parts. Thus, in order to enhance the modularity and organization of the project, the team divided the software in many different packages using the *Robotic Operating System (ROS)* [5] framework, which provides many tools and libraries that are useful for developing robotic applications.

Of these packages that encompass the strategy system, the ones that were researched and improved in the last year are the packages that concern the decision-making system and the navigation methods.

The decision-making structure was better formulated by developing a model for the organization. The entities in this model, the coach, responsible for the top-level decisions relative to the game state, and the robots' behaviors, which describes what the robot should do based on the role it was assigned by the coach, where also refactored and improved in order to achieve a better performance in the game.

As for the navigation methods, which are necessary to provide a way for the robots to follow the decisions made, the field research that has been improved concerns the local planners, that is, better ways to generate a velocity command for a robot based on a global plan or on others input variables.

The next sections will discuss each of these parts in more details.

A. Organization Model

In order to better distribute the responsibilities of each entity in the system and improve the structure, maintenance and performance of the project, a organization structure was develop for the decision-making system. To enhance the formulation of this organization, a model was develop for it using the MOISE [1] specification, which can be seen in the Figure 6.

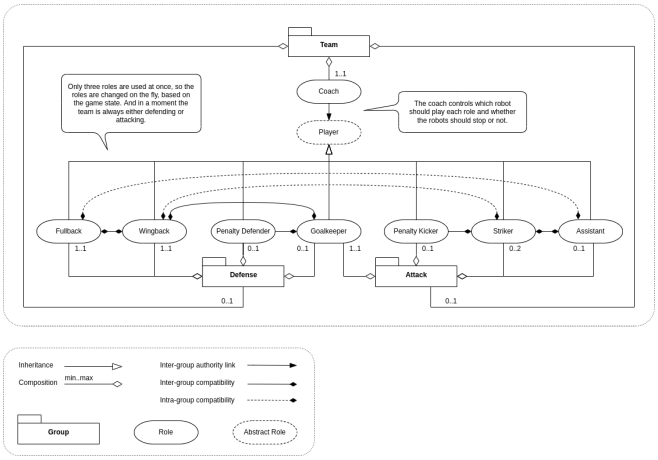

Fig. 6. ThunderVolt Organization Model

In this organization, seven roles where defined, in order to have a more granular control over the kind of actions that each robots can play. Some roles are compatible with each other, thus allowing a robot playing a specific role to play another one that is compatible with the current role. Not all roles are played at once, so there is a central entity, the coach, which is responsible for controlling the robots and defining the role that each robot should play.

B. Coach

As described before, in order to choose which robot is going to have each behavior (eg. goalkeeper, defender and striker), there is an entity in the organization, the coach, that defines the robots' roles according to the positions of the robots and the ball. The coach is a program that is always analyzing the positions of the players to see if it is necessary to change the behaviors for a better performance. Additionally, the coach is also responsible to chose the

position of each robot when a foul occurs, deciding, for example, how to kick a penalty or how to deal with a goal kick.

1) Structure: Previously it was develop a finite state machine to implement the coach's core feature, which is to chose the roles of the robots, however, this control architecture proved to be non-scalable, difficult to maintain and to improve. For this reason, the coach structure was refactored using a different control architecture, the behavior trees [3], which are more modular, flexible and scalable, besides already being used to define the behaviors of the robots.

That way, the behavior tree developed for the coach receives the message from the VSSReferee [14] and, depending on the message, defines initial states for the team, defining if the team is going to attack or defend, for example, and setting initial roles for the robots. Then, while the game is running, the flow of the tree constantly deal with the changes in the game to set the best roles for each robot at each moment. The part of the tree that handles role switching have two main branches, one for when the team is attacking and another for when the team is defending, inside of each branch the tree deals with the changes inside each of theses states, like changing the robot with the striker role with the robot with the assistant role in the attack state when needed.

2) Wave Evaluator: Thinking about improving the role changes performed during the game, an evaluation function was created, so that for each possible position of the ball on the field, the function gives an evaluation to each player, according to the possibility that that player has to catch the ball or to score the goal. This function uses a set of Gaussian distributions as follows (where m, k and l are constants:

$$f(x,y) = e^{-\frac{y^2}{2m^2}} \cdot (e^{-\frac{x^2}{2k^2}} \cdot H(x) + e^{-\frac{x^2}{2l^2}} \cdot H(-x))$$

Generating the format shown in the Figure 7.

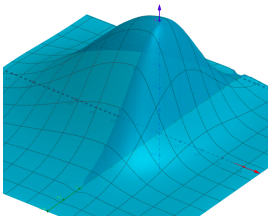

Fig. 7. Wave Evaluation 3D Function

C. Behaviors

Last year, our team created 3 types of behavior trees: the defender, the striker and the goalkeeper. Since then, we have implemented new behaviors for more specific situations, those being: fullback, wingback, assistant, penalty kicker and penalty defender.

The assistant, with the striker, composes the two behaviors of the attack situation. The assistant has the function to wait for a good opportunity to kick towards the goal, while the striker is who effectively fights for the ball with the foes and tries to score.

Referring to the defense situation, the fullback stays at the penalty area vertical line, waiting for the opportunity to move the ball away, while the wingback, like the striker, fights for the ball with the foes and tries to avoid the foes' score.

The last two behaviours are used in the penalty situation, the penalty kicker tries to score and the penalty defender tries to prevent it.

D. Local Planning

The local planning task must be flexible enough to provide a few different ways to control a robot. For tasks like following a path provided by the global planner, a Dynamic Window Approach Controller (DWA) was implemented. This consists in sampling all possibilities from the control space of the robot (linear and angular speeds) and simulating the robot's trajectory for each one. After that, some metrics (e.g.: proximity and alignment to the target) are used to evaluate the trajectories, choosing the one with the highest score.

However, generate and evaluate all trajectories surpasses the maximum time for the player to execute an action. Some changes were tested to optimize the time consumed (such as reducing the number of trajectories and using the parallel processing library OpenMP [15]). Although this decreased the time consumption, it was still over the maximum time.

Therefore, the local planning is currently only using vector fields, while other options are being analyzed, such as a PID controller follower [11].

VI. VISION

The vision section of the project is the one responsible for translating the real world positions and velocities of the ball and robots in the field into digital information which can then be stored, processed and used internally for the decision-making process of the team.

This is achieved by configuring a monocular camera set up with a top-down view of the field to publish its raw image data to a ROS topic, which is then processed by the vision algorithm, using CUDA and OpenCV, in order to accurately correlate each field entity (robots and ball) with its image, by detecting the distinguishing colors on each of these objects (as well as the static field marks) and tracking them throughout the field.

A. Technics

To correctly detect the robots, ball, and field marks, we need to identify these elements in the image captured by the camera. Some technics are used to accomplish this task.

First, a blur is applied to the image to reduce its noise and unnecessary details. OpenCV has several blur types (e.g.: gaussian blur, bilateral filtering, etc), which can be applied for each pixel, considering a certain number of near pixels (kernel). Another important task is to detect robots, ball and field marks colors. To achieve this, we determine a certain interval for each of these colors, using HSV (Hue, Saturation, Value) scale. Since RGB varies according to lightness, HSV shows to be a more reliable scale to calibrate these intervals,

with hue representing the desired color interval, and saturation and value being adjusted according to the environment. Therefore, the OpenCV receives the image in BGR format (equivalent to RGB), and then convert it to the HSV scale.

Finally, with color intervals determined, we can detect the elements we want using blob detection. Moreover, dilate and erode operations are applied to the image to omit small detections or join near blobs.

B. Algorithm

The vision algorithm is responsible for continuously identifying and processing game information received by the camera.

In order to do this, it first fixes a frame and finds the three field markings that are used for dimension reference:

Fig. 8. Vision reference points

Following that, the algorithm adjusts certain aspects of the frame. It adds filters and corrects the size and rotation of the image in a change of basis, from the basis defined by the camera itself to a basis at the center of the field.

When all this is set, the robots and ball are located using their known colors, and their information are calculated in the center of the field basis. In this process, velocities and angles are calculated using stored information from previous frames.

The information acquired is published in a ROS topic and the loop repeats itself throughout the game.

C. Camera Calibration

Before any high level image processing can be done by the vision algorithm, we must first treat the raw image collected from the monocular camera used by the team in order to remove any distortions caused by the lens and, therefore, get the consistent and accurate image data which will be fed into the vision algorithm.

In order to achieve this, we calibrate the camera using the Image Pipeline [8] stack, as well as the following ROS packages:

- Camera Calibration Parser [6]
- Camera Info Manager [4]
- Launch Testing Ament Cmake [13]

With these in hand, and with the camera publishing to a ROS topic, we can begin the calibration process by using a checkerboard pattern printed on a sheet of paper, and following procedures described in the official ROS camera calibration tutorial [16].

First, we start the camera calibration node, passing the parameters size (horizontal squares x vertical squares) and square (length of the sides of an individual square) of the checkerboard used, and then move the checkerboard to different positions and angles in the camera's field of view, until sufficient data has been collected to finalize the calibration.

Finally, we save the configuration generated by this process in a compressed file, which can then be unzipped and used by the team.

VII. INTERFACES

With the objective of modularizing the control code, a repository which contains different packages that perform the interface between thundervolt and different usage scenarios was created. This makes it simple, for example, to switch between using the code with the physical robots and using it with a simulation environment.

A. Teleoperation

In order to validate the functioning of the physical robots, it was necessary to implement a new package in the project, the Teleoperation. Developed with Python language and using the Pygame library [9], this new package is intended to receive commands from a keyboard or joystick to control the movement of robots. This implementation is extremely important to test structural features of the project, such as the motors control loop.

B. Serial Communication

The serial communication between the nRFDongle and the computer running the strategies will be made with USB, as explained in Section III. In order to achieve this goal, it was necessary to create a new package that opens the serial port and encodes the messages from a ROS topics to send them through a serial wire.

Initially, it was implemented a new class that describes the object in C++ that opens and configure that serial port in Linux correctly. The user configures the information about the name of the port (default is /dev/ttyACM0) and the baud rate (default is 9600) which are going to be used in all methods implemented in the class.

After setting up the port successfully, the main node creates a ROS subscriber that reads the messages published by the decision making-system about their current velocity and converts them to a protocol used in the serial communication established by the team. Then, these messages are sent to the nRFDongle by USB.

C. UDP Adapters

The adapters were initially conceived to help other teams to use TraveSim [12], without necessarily having a ROS compatible project. However, the structure was reused to the develop an interface between our code running in an ROS environment and the FIRASim and VSSReferee, used in the simulated matches. Basically, the adapters translates ROS messages to protobuf messages and vice-versa, doing the conversion required by certain communications contexts.

The team uses it in four communication fronts; thus, four adapters were created: Vision Adapter, Replacer Adapter, Command Adapter and Referee Adapter.

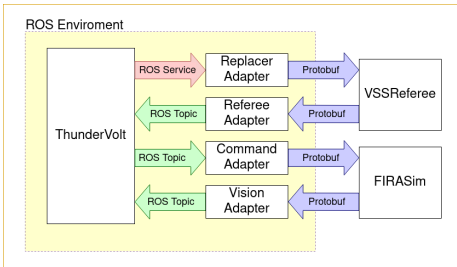

Fig. 9. Adapters Architecture

The Vision and Referee adapters both receive a protobuf message through UDP multicast protocol, translate it to a ROS message and publish to ThunderVolt through a ROS Topic. The difference is that the Referee Adapter receives its messages from the VSSReferee and the Vision Adapter receives its messages from FIRASim.

The Replacer and Command adapters both receive a ROS message from ThunderVolt and translate it to a protobuf message to be passed on. They have 3 major differences - the fist is the protocol communication method: Command Adapter uses UDP unicast protocol while Replacer Adapter receives its messages through UDP multicast protocol. The second resides in the fact that the Replacer Adapter is communicating to ThunderVolt by a ROS Service, not a ROS Topic as the Command Adapter. And the third is in their destination: the Replacer Adapter sends the protobuf message to the VSSReferee, while the Command Adapter sends it to FIRASim.

ACKNOWLEDGMENT

The ThunderVolt team would like to thank the ThundeRatz Robotics Team, Amigos da Poli Patrimonial Fund and Polytechnic School of the University of São Paulo for all their support. In addition to giving recognition to our sponsors STMicroelectronics, Altium, SolidWorks and Circuibras.

REFERENCES

[1] Jomi Fred Hübner, Jaime Simão Sichman, and Olivier Boissier. “A Model for the Structural, Functional, and Deontic Specification of Organizations in Multiagent Systems”. In: *Advances in Artificial Intelligence*. Ed. by Guilherme Bittencourt and Geber L. Ramalho. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 118–128. ISBN: 978-3-540-36127-5.

[2] J. R. B. A. Monteiro et al. “Anti-windup method for fuzzy PD+I, PI and PID controllers applied in brushless DC motor speed control”. In: *2013 Brazilian Power Electronics Conference*. 2013, pp. 865–871. DOI: 10.1109/COBEP.2013.6785216.

[3] Michele Colledanchise. “Behavior Trees in Robotics”. MA thesis. SE-100 44 Stockholm, Sweden: School of Computer Science and Communication - KTH, 2017. ISBN: 978-91-7729-283-8.

[4] Jack O’Quin. *Camera Info Manager*. Version 1.12.0. Apr. 3, 2020. URL: http://wiki.ros.org/camera_info_manager?distro=noetic.

[5] Open Robotics. *Robotic Operating System*. Version ROS Noetic Ninjemys. May 23, 2020. URL: https://www.ros.org.

[6] Patrick Mihelich. *Camera Calibration Parser*. Version 1.12.0. Apr. 3, 2020. URL: http://wiki.ros.org/camera_calibration_parsers.

[7] ThundeRatz. *STM32ProjectTemplate*. Oct. 9, 2020. URL: https://github.com/ThundeRatz/STM32ProjectTemplate.

[8] Patrick Mihelich, James Bowman. *Image Pipeline*. Version 1.16.0. Nov. 12, 2021. URL: http://wiki.ros.org/image_pipeline.

[9] Pete Shinners. *Pygame Library*. Version 2.2.12. Dec. 27, 2021. URL: https://www.pygame.org/news.

[10] STMicroelectronics. *STM32CubeMX*. Version 6.3.0. July 8, 2021. URL: https://www.st.com/en/development-tools/stm32cubemx.html.

[11] Nobleo Technology. *tracking_pid*. Feb. 2, 2021. URL: https://github.com/nobleo/tracking_pid.

[12] ThundeRatz. *TraveSim Adapters*. Version 1.0. July 5, 2021. URL: https://github.com/ThundeRatz/travesim_adapters.

[13] Pete Baughman, William Woodall. *Launch Testing Ament CMake*. Version 1.0.2. May 10, 2022. URL: https://index.ros.org/p/launch_testing_ament_cmake/.

[14] VSSSLeague. *VSS Referee*. Mar. 12, 2022. URL: https://github.com/VSSSLeague/VSSReferee.

[15] OpenMP Architecture Review Board. *OpenMP API*. URL: https://www.openmp.org/.

[16] ROS Navigation. *ROS Camera Calibration Tutorial*. URL: https://navigation.ros.org/tutorials/docs/camera_calibration.html#tutorial-steps.

[17] Pololu Robotics and Electronics. *Micro Metal Gearmotors*. URL: https://www.pololu.com/category/60/micro-metal-gearmotors.

[18] STMicroelectronics. *STM32 32-bit Arm Cortex MCUs*. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[19] STMicroelectronics. *STM32 cross-series timer overview*. URL: https://www.st.com/content/ccc/resource/technical/document/application_note/54/0f/67/eb/47/34/45/40/DM00042534.pdf/files/DM00042534.pdf/jcr:content/translations/en.DM00042534.pdf.

[20] ThundeRatz. *STM32Guide*. URL: https://github.com/ThundeRatz/STM32Guide.