SRC Extended Team Description Paper for RoboCup 2026

Fanqin Zeng[1,2], Xuanting Liu[1], Yu Zhang[1], Zhuoyan Chen[1], Yixuan Wang[1], Yucheng Gu[1], Shenghan Gao[1], Muhan Lin[1], Jianing Li[1], and Xiaoxiao Zhu[1*]

Student Innovation Center, Shanghai Jiao Tong University, P.R.China
[1]`sjtu_src2020@163.com`
[2]`zfq530754711@163.com`

Abstract. This paper introduces the technical improvements which the SRC Team made for RoboCup 2026. In the software part, we reconstruct the decision architecture from Lua to a Python-C++ framework and propose a data-driven ball dynamics model to replace ideal physical assumptions. In the electronics part, we redesign the power board using Flyback topology to ensure safety, migrate the embedded firmware to a FreeRTOS system, and adopt a modular circuit design for better maintainability. In the mechanics section, we optimize the split-type dribbler mechanism and enhance the overall assembly reliability.

1 Introduction

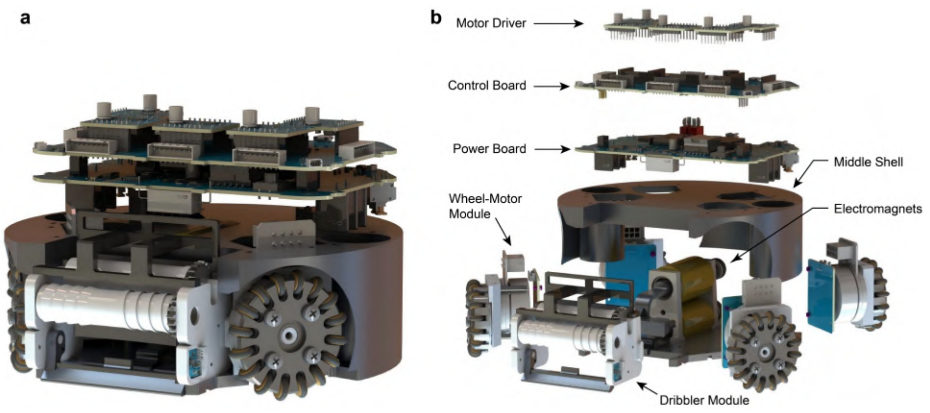

Fig. 1: SRC 2026 Robot. (a)Rendering image of the robot. (b)Explosion view of the robot.

We are the SRC Team from Shanghai Jiao Tong University, participating in the RoboCup Small Size League (SSL) since 2017. For the 2026 season, we have implemented comprehensive system upgrades to address architectural limitations and hardware reliability.

In software, we transitioned from a Lua-based framework to a Python-C++ architecture to better support modern AI workflows, alongside a new data-driven ball dynamics model. In electronics, we redesigned the power board using a Flyback topology for safety and migrated the embedded firmware to a FreeRTOS system. Mechanically, we optimized the split-type dribbler and adopted a modular design to enhance assembly reliability and maintenance efficiency. This paper details these technical improvements.

2 Software

2.1 Python-C++ Based Architecture

The overall software control system of our team is based on the Rocos system [1] and consists of three parts: Client (providing vision information and referee messages), Core (responsible for making decisions) and grSim [2] (official project, providing a virtual simulation environment). The architecture refers to the framework of Core application and our following discussion mainly focuses on that.

The Core part in Rocos and our older software version is Lua-C++ based architecture which used Lua as the language to organize the top layer and connecting C++ functions of low layer via tolua++ [3] tool. Lua is a lightweight language that can respond and adapt quickly to the situation on the field. It does not require recompilation and is suitable for being used as the top-level language for controlling certain parameters in the competition. C++ is responsible for encapsulating functions with large computational loads, which can enhance the real-time performance of the system. However, with the advancement of technology and the continuous evolution of techniques, the drawbacks of using Lua as the upper-level language have become increasingly prominent.

1. As Lua is a light weight language, all the top Lua modules are assembled dynamically which means the editor cannot provide the static examination, which makes it hard to debug and errors always occur randomly and unexpectedly during runtime.
2. All the Lua-C++ bindings rely heavily on tolua++ tool, which hasn't been maintained over fourteen years. A lot of C++'s new features are not supported and the binding process is tedious.
3. The Lua language cannot support newest AI technologies which is mainly written in Python.

Conversely, Python, as one of the most commonly used language, has the complete IDE support which supports static checking and many tool-tips, the largest active community and development ecosystem, convenient and easy-to-read grammar and state-of-the-art AI technology packages. Python is also developed as a "glue language" which means it's highly suitable for cross-code-language application scenarios. In modern robot operating systems, it's common to use Python as the top-level language for upper decision making and it can reach a speed of up to 60-70 Hz (also the usual frequency of vision data in ssl games) which satisfies the requirements for robot planing. For Python and C++ bindings, there are several binding tools like Boost, SWIG, Cython, nanobind and Pybind11 which provides various choices and mature solutions. Based on the above information, we have decided to reconstruct Core's Lua-C++ based architecture to a Python-C++ based architecture. Considering the learning-curve, performance, community activity, maintenance status and the tool-tips supports (i.e. `.pyi` files, Python Interface Stub File), finally we chose Pybind11 as the binding technology.

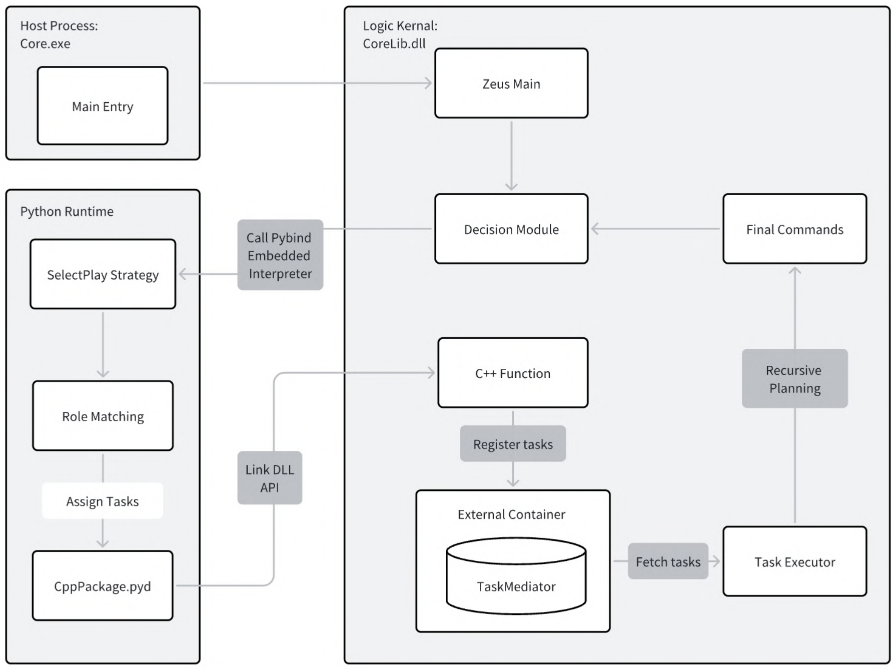

Fig. 2: Overview of Python-C++ Based Architecture.

The overview of software architecture is illustrated in 2. In essence, we utilizes Pybind11 to embed the Python interpreter which enables the C++ layer to call Python decision module and generate the corresponding `.pyd` files which enables the Python layer to invokes the C++ functions. The dll helps to share functions and variables between Python and C++. The `.pyi` stub files are generated via the pybind11-stubgen tool [4]. As for the binding code needed to generate `.pyd` files, we utilizes the litgen tool [5] to complete automatically which is usually more efficient.

The original Core part is divided into three parts as illustrated in 2. It's important to mention that in Rocos system, `Core.exe` is directly called by `Client.exe`. To ensure compatibility with the Lua-C++ version, we have retained `Core.exe` as the object that is called by `Client.exe`, but have separated the main body of the original `Core.exe` into `CoreLib.dll`. We are also considering using the Python module as the entrance of Core in the future. All the projects are organized via CMake.

The overall decision making process works like this: when the `main` function of `Core.exe` is called, it invokes the `init` function in Zeus Main module. Then entering the decision making loop. The decision module firstly init an embedded Python interpreter and calls the SelectPlay module to decide which competition strategy script to use. Then the strategy scripts plan different tasks which will later be assigned to concrete vehicles using munkres matching algorithm (i.e. the role matching module in 2). Each specific task will be stored in the TaskMediator, and then retrieved and specifically invoked recursively until sending the final commands.

What's more, we can now debug the whole system. Debugging in the C++ layer is achieved by generating `.pdb` files, and in the Python layer is implemented by using the debugpy package and attaching to it during run time.

The running efficiency has also been taken into consideration. We conducted performance tests on our programs, and the results indicate that the control frequencies of both the Lua-C++ and Python-C++ architectures are approximately 60 Hz, with no significant difference between them.

2.2 Data-Driven Ball Dynamics Modeling

In previous iterations, our physical modeling of the ball relied on corrections to classical rigid body motion equations. In certain scenarios, the model only accounted for translational motion, treated as constant linear deceleration. In others, rotational effects were incorporated by using a velocity threshold to divide the movement into two distinct stages of constant linear deceleration. Regarding the relationship between shooting intensity, ball velocity, and displacement, no analytical physical formulas were derived; instead, hard-coded constants were selected based on empirical experimental data.

However, recent testing and adjustments in electronic control have shown that these simplified modeling methods are no longer adequate. The primary limitation is that such approaches rely on overly idealized assumptions and physical laws that do not fully capture real-world dynamics. Comprehensive experiments have demonstrated that ball movement on standard competition pitches follows more complex patterns.

Consequently, in our latest modeling version, we have transitioned from theoretical physical derivation to a "data-driven" methodology. Specifically, we utilize standardized robots to conduct multiple field tests on actual pitches.

Data is collected via the vision module and subsequently analyzed and fitted. This process yields a series of formulas and combinatorial laws that reflect actual operational patterns rather than idealized physical laws.

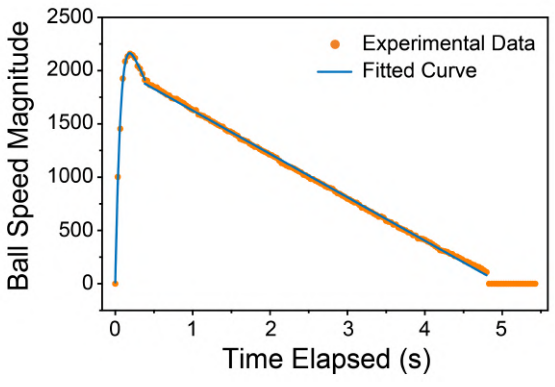

Fig. 3: An Example of Fitting.

In detail, we sampled shooting intensity values at uniform gradients, recording time, velocity, displacement, and other essential metrics from the moment of ball release to its final stationary state. Analysis in Origin(2025b ver.) revealed that the velocity-time curve is a two-stage piecewise function[Fig. 3]: the first stage corresponds to the instantaneous acceleration during the shot, while the second stage represents the free deceleration of the ball. The first stage exhibits a complex pattern; since the time interval is extremely short and precision requirements for this phase are relatively low, we employ polynomial fitting, selecting the optimal degree for each dataset rather than a fixed one. The second stage shows clear linearity and is modeled as a linear function.

For unsampled intensity inputs, experiments indicate a linear correlation between shooting intensity and displacement over a fixed duration. Therefore, we utilize linear regression based on the three nearest sampling points to calculate the required values for any given intensity.

For the internal calculations of interfaces that solve for shooting intensity based on specified target positions, terminal velocity, or travel time, we leverage the monotonicity of velocity, displacement, and duration with respect to intensity. We utilize the bisection method to solve for intensity values within the required precision range, while accounting for boundary conditions (such as insufficient intensity) to return appropriate reference values. For interfaces that provide outputs based on a given shooting intensity and prediction time, we directly employ the established models and the strategy for handling unsampled values to calculate and return reasonable results for the system's real-time decision-making.

2.3 YOLOv8

To robustly detect the ball and robots from monocular images, we tried to adopt Ultralytics YOLOv8 as the detector. At present, we have only tried applying it to the recognition of static images. In the future, we plan to integrate it into our Python–C++ decision pipeline.

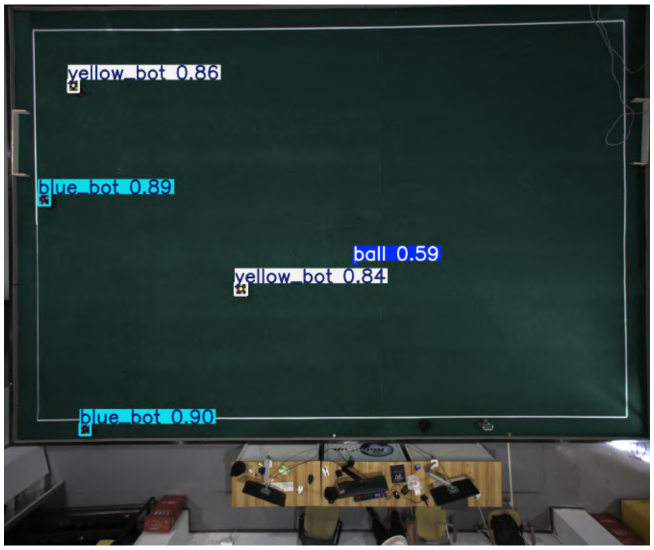

Fig. 4: The Detection Result of Yolov8 Model

Dataset and labeling. We curated a dataset from pictures taken by our camera, which is also used in the match, then split it into train/val/test sets conforming to the YOLOv8 format. Three classes were annotated: ball, blue_bot, and yellow_bot. The dataset was managed in Roboflow and exported with a standard data configuration file.

Training. We trained the lightweight YOLOv8n model using the Ultralytics API with input size 640, batch size 16, 100 epochs, and CPU device. The training configuration and logs are recorded in the runs/detect/train directory. The final validation metrics reached precision 0.986, recall 0.9077, mAP50 0.9548, and mAP50–95 0.7137. The best weights are stored.

Inference and integration. For deployment, we use the Ultralytics runtime to obtain bounding boxes in image coordinates, then convert the center of each detection to field coordinates via a homography estimated from four manually selected field corners. This provides (X, Y) positions for the ball and each robot that are consumed by the higher-level planner.

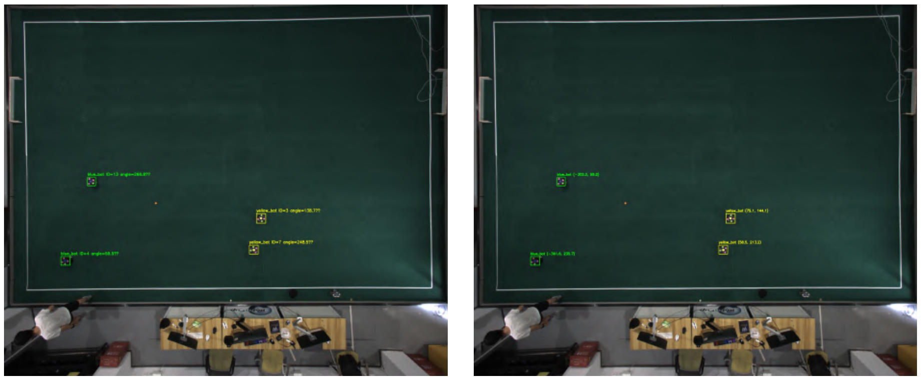
(a) Detection and ID visualization **(b)** World coordinates visualization

Fig. 5: Robot ID and world coordinate visualizations.

Robot ID and orientation. Based on color markers on the robot top (yellow/blue), we compute the sequence of four blocks around the robot, recover its heading using the largest angular gap heuristic, and map the normalized color sequence to a discrete robot ID via a lookup table. This augments detection with robot identity and orientation for downstream tasks.

3 Electronics

The development of SRC's electronic system began in 2022. Through four years of iterative refinement, we have come to recognize that outstanding engineering is not defined by the adoption of the most advanced techniques or technologies, but rather by achieving robust and human-centric functionality through the simplest possible design within strictly constrained hardware resources. Over the past year, we have critically examined the limitations of the existing electronic control system and held extensive discussions on robotic design with the TIGERs Mannheim [7] and ZJUNlict [8]. Based on these efforts, we have implemented systematic optimizations and enhancements to the electronic control system for our 2026 robot. These improvements are shown in table 1, among which the most important ones are redesigning the power board, migrating the embedded software platform, and developing a more intuitive modular electronic control architecture.

Table 1: SRC Electronic Improvements

Improvements	v2025	v2026
Power Board	Boost (UC3843AP)	Flyback (LT3751EFE)
Kicking Voltage	220V	180V
Encoder	Grating (HEDS-9731)	Magnetic (AS5047P)
Infrared Sensor	Single LED	PCB Circuit
Firmware	Simple Embedded Code	FreeRTOS Project
Battery	18650 Battery Pack 7.4V*2	Grepow Battery 2S*2
Manufacture	Individual Board	Panelized Board

3.1 Flyback-Based Power Board

Even though the competition rules did not impose many restrictions on the specific hardware structure of the robots, in the small group competitions, the high ball speed made it necessary for almost all teams to adopt the kicking scheme using high-voltage driven electromagnets. Therefore, a safe, stable and precise power source has long been a goal for us. Since the establishment of our team, our robots have been using the boost topology structure for voltage boosting. However, even after 8 generations of PCB layout optimization iterations, our power board based on boost voltage conversion still frequently exploded and caused a great deal of insecurity and uncertainty for the team.

In the continuous conduction mode (CCM), under the condition of ignoring the device voltage drop, the calculation formula for the boost voltage boost is as shown in Equation (1).

$$V_{out} = V_{in} \cdot \frac{1}{1 - D} \tag{1}$$

where D represents the duty cycle of the PWM signal.

It can be seen that in the 16V - 200V voltage boosting scenario of the SSL, the theoretical duty cycle has approached 92%, which is already close to the modulation upper limit of most power management chips. Furthermore, when the voltage boosting ratio is greater than 10, the parasitic parameters (inductance/capacitance ESR, diode drop, switch on-resistance) have a significant ”absorbing” effect on the gain, resulting in a sharp decline in efficiency. In practice, it is difficult to achieve an efficiency of over 85% [9].

Meanwhile, we noticed that the Kicking Device of the TIGERs Mannheim employs Flyback boost technology. This solution alleviates the pressure on the duty cycle by incorporating a transformer. Furthermore, by comparing the topological structures of the two (as shown in Figure 6b), it can be observed that the flyback scheme can perfectly achieve the isolation between the low-voltage

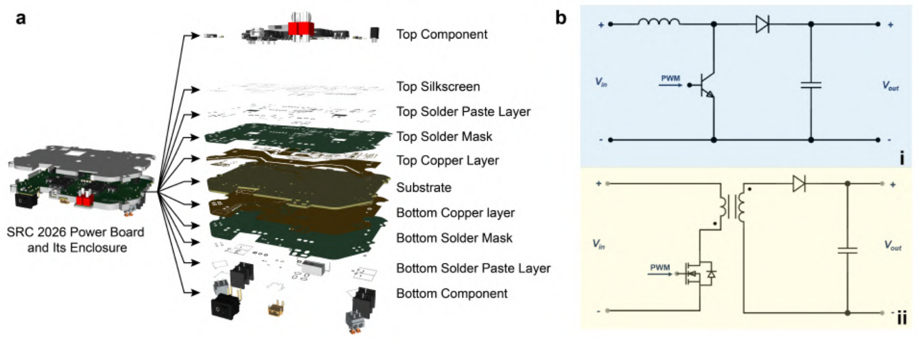

Fig. 6: SRC 2026 Power Board. (a)Power board and its laminated structure renderings. (b)Topological structures of (i)boost circuit and (ii)flyback circuit.

side and the high-voltage side, while boost scheme mixes all the voltage level together, which provides a very reliable guarantee for safety and stability.

Therefore, in the past year, we redesigned the new version of the power board. The new version not only adopts a robust flyback design, but also eliminates the redundant circuits from the previous version. Moreover, it has undergone engineering optimization in details such as PCB layout, copper plating, and wiring. The PCB design and laminate structure are shown in 6a.

After actual testing, it was found that the power board of the new version experienced a significant reduction in unexpected damage incidents during a one-year period of operation, and there were no reports of explosions, spontaneous combustion, or other accidents. Even in the intense competition of the 2025 RoboCup China Open, the damage to the power plates only occurred in the form of fuse melting or MOS transistor breakdown, which are all common and easily repairable issues. After the voltage reduction in early 2026 (with the target voltage adjusted to 180V), there are currently no reports of power board damage.

3.2 FreeRTOS-Based Embedded Firmware

For an extended period, the STM32F407VET6 has served as our primary microcontroller. Following detailed discussions with teams such as ZJUNlict and TIGERs Mannheim, we explored the possibility of migrating to a more advanced controller. However, higher-performance platforms like the STM32H7 or Raspberry Pi are typically adopted to meet the computational demands of executing complex, high-level behaviors directly on the embedded system. In our architecture, high-level action planning is performed entirely at the software level, while the low-level system is solely responsible for executing simple sub-actions. In this context, the STM32F4 already provides sufficient computational resources, and upgrading the microcontroller would introduce unnecessary development overhead.

In earlier versions of our embedded firmware, performance bottlenecks arose primarily from program architecture limitations. Prior to 2025, the embedded system operated on a simple main loop supplemented by two interrupts (Figure 7a). Extensive use of blocking delays resulted in significant CPU idle time, wasting computational resources that could otherwise have been utilized for concurrent tasks. Furthermore, our embedded development had long relied on standard peripheral libraries. While functionally stable, these libraries exhibit limited portability and complicate the integration of new features or external devices.

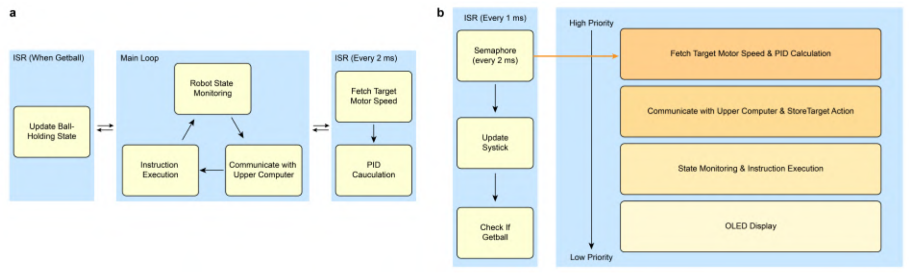

Fig. 7: Flowchart of SRC Embedded Firmware. (a) Simple embedded program. (b) Firmware based on FreeRTOS.

Consequently, we undertook a comprehensive redesign of the embedded firmware using FreeRTOS, a widely adopted real-time operating system, in combination with HAL libraries. The architecture of the new firmware is illustrated in Figure 7b, and the source code has been open-sourced on GitHub (https://github.com/sjtu-src/SRC_Firmware). The system is structured around four threads with distinct priorities:

1. **Motor Task** – Triggered by a 2 ms timer interrupt, this thread performs real-time PID control for the four wheel motors.
2. **Communication Task** – Handles communication with the host computer, decoding incoming instructions and storing kick, dribble, and other motion commands in global variables.
3. **Robot Task** – Executes the actual behaviors based on the instructions stored in global variables.
4. **Display Task** – Utilizes idle CPU cycles to show robot status information on an OLED display.

The adoption of the new embedded firmware has yielded a noticeable improvement in the robot's motion execution, characterized by enhanced smoothness and responsiveness. Furthermore, the codebase now offers significantly better readability and maintainability, while its modular architecture provides an easily extensible interface for the future development and integration of FOC-based motor drivers.

3.3 Design Optimization for Maintainability and Cost Efficiency

Excellent engineering design must not only achieve strong performance but also give full consideration to system usability, maintainability, and manufacturing cost. Years of experience in robot maintenance have shown that even the most robust systems may sustain damage during intense competitive matches. In the past, we integrated most of the vehicle's functions onto two PCB boards. In the event of a circuit failure, we could only perform temporary soldering to replace components or swap out entire boards, often resulting in one localized fault leading to the disposal of the whole board. This posed significant challenges for daily maintenance.

In practice, components prone to damage during robot operation are typically concentrated in high-power sections such as kicking mechanisms and motor drivers. In contrast, peripheral circuits for control and sensing operate on low voltages such as 5 V or 3.3 V and are rarely subject to failure. Therefore, it is entirely feasible to separate these vulnerable parts from the main board by designing them as smaller, standalone PCB modules connected to the motherboard via pin headers (shown in Figure 8a). This modular approach allows for direct removal and replacement in case of damage, offering considerable benefits in practical applications.

Table 2: PCB Designs of SRC 2026 Robot

Board	**Function**	**PCB**
Motor Driver	Actuate the motor to rotate as the instruction	Figure 2b
Encoder	Acquire the motor rotational speed	Figure 2c
Infrared Sensor	Detect whether the robot is holding the ball	Figure 2d
Control Board	Main Controller	Figure 2e
Power Board	Provide power and then boost the voltage	Figure 2f

Following the optimization of the electronic control system architecture, the hardware component of our robot's electronic system now consists of five main circuit boards, as detailed in Table 2. Moreover, to reduce manufacturing costs, panelization has been adopted as an effective approach. Taking into consideration production volume and quotations from the manufacturer (Shenzhen JLC Technology Group Co., Ltd.), different panelization layouts were designed for the respective PCBs, as illustrated in Figures 8b–8f. This strategy resulted in significant cost savings. For instance, producing five individual motor driver boards

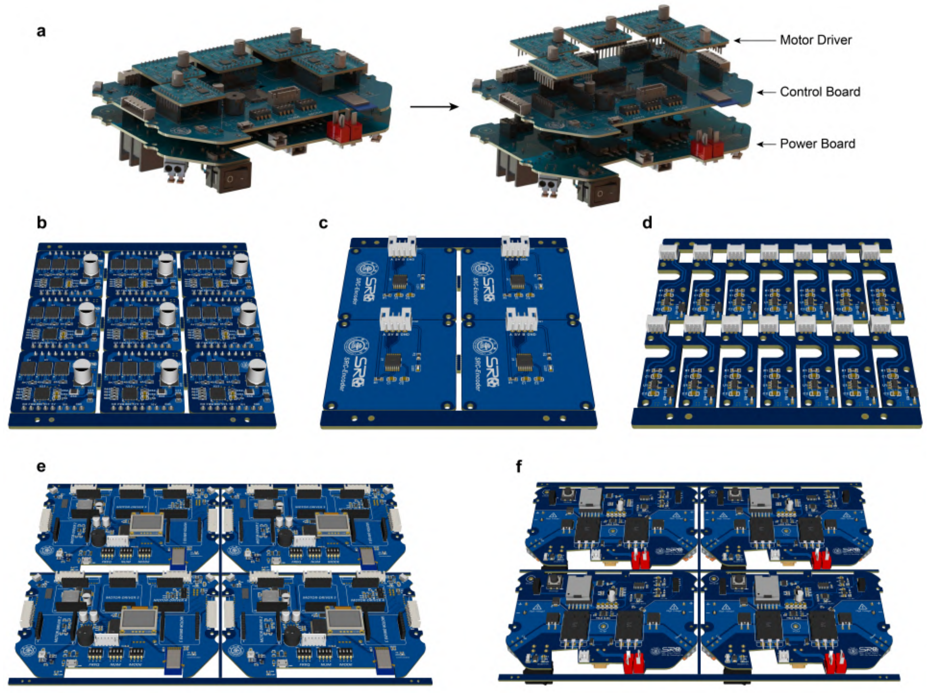

Fig. 8: Electronic Control System Architecture of SRC 2026 Robot. (a)Assembly View of the Modular Design. And panelization design of the (b)motor driver, (c)encoder, (d)Infrared Sensor, (e)control board, (f)power board.

with SMT assembly would cost approximately 500 RMB, whereas panelized production—yielding 45 units across five panels (each containing nine boards)—reduced the total cost to around 1,800 RMB. Further increases in production volume are expected to drive down the per-unit cost even more.

4 Mechanics

We have come to realize that, with the limited resources of a single year, it is difficult to simultaneously pursue aggressive performance iteration and significant improvements in stability. After careful consideration, our mechanical efforts this year have therefore focused primarily on performance enhancements in selected modules and on improving the overall system stability.

4.1 Dribbler Improvement

The dribbling capability serves as the foundation for our robot to execute passing, shooting, and other technical maneuvers. To maximize the dribbling performance while adapting to the newly designed kicking and chip-shot mechanisms, we have optimized the robot's mouth structure—i.e., the dribbling structure—by transforming it from an integrated design into the current upper-and-lower split design. When the robot approaches the ball, the ball will be drawn in and lift up the upper half of the dribbler, as shown in Figure 9.

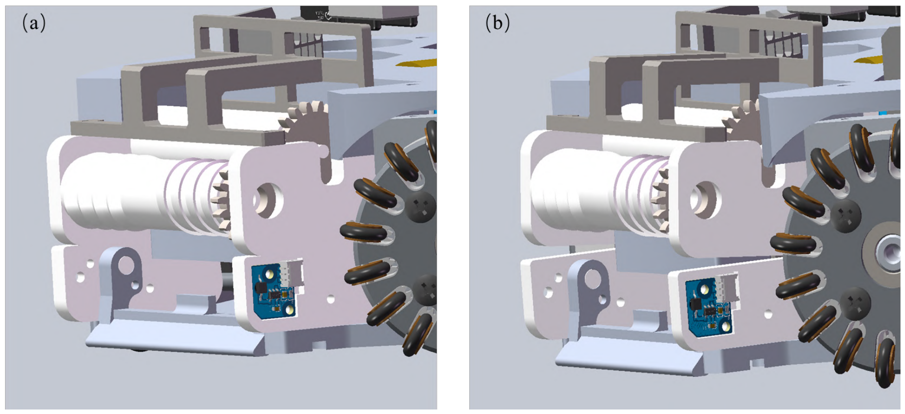

Fig. 9: SRC 2026 Dribbler. (a)Initial State, (b)Dribbling State

The new split-type dribbling structure consists of larger dribbling rollers, lower side plates and upper side plates on both sides, as well as upper fixing components for mounting and shock absorption. Compared with the previous version, this optimized structure significantly reduces the difficulty of assembly, disassembly, and maintenance. Meanwhile, we have tentatively determined the dimensions of each component through experiments to achieve the desired dribbling effect.

It should also be noted that we have optimized the parameters of our dribbling rollers (e.g., hardness, diameter, etc.) through experiments to achieve the desired dribbling effect.

4.2 Assembly Reliability

Over the past year, we have been continuously working to make the assembly of our robot more convenient and structurally robust, with the goal of minimizing damage during matches and ensuring that, even in the event of failure, disassembly and maintenance can be carried out efficiently. Based on our observations, most of the components that were previously prone to damage stemmed from unreasonable design choices and inappropriate material selection. Below, we elaborate on the improvements made in a modular manner.

For the kicking mechanism, the initial structure used to fix the electromagnet consisted of two rectangular plates fastened with screws. This design lacked sufficient resistance to vibration and durability under long-term use. This year, we extended four supporting legs from these two rectangular plates, increasing the contact area with the base plate along a vertical dimension, thereby significantly improving structural stability. As shown in Figure 10.

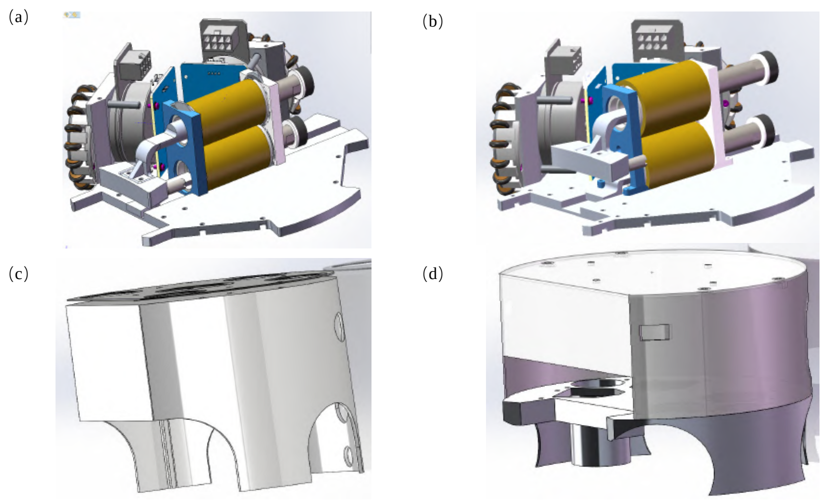

Fig. 10: SRC 2026 Kick Board & Shell. (a)(c)Before (b)(d)After

Regarding the dribbler device, in addition to the structural improvements mentioned above, we eliminated components that previously had to be fabricated using plastic 3D printing due to design constraints. This change greatly reduced the damage rate. Furthermore, the infrared sensor solution of the dribbler was upgraded: instead of tying LED emitters directly to the side plates, the LEDs are now soldered onto a printed circuit board, which is then firmly mounted onto the side plates. This method of fixation is clearly more robust and reliable. As shown in Figure 11.

As for the chassis shell, our previous design adopted a fully integrated body, which offered the advantage of easy installation but suffered from severe stress concentration at certain locations, making it prone to damage. This year, for the first time, we divided the chassis shell into two separate parts. The upper shell, which protects the circuit boards and supports the color marker board, is separated from the lower shell that covers the main body. With this design, the upper shell only needs to form a complete circular structure without regions of excessive stress, while the lower shell, no longer constrained to be integrated with the upper shell, also eliminates previously vulnerable stress points.

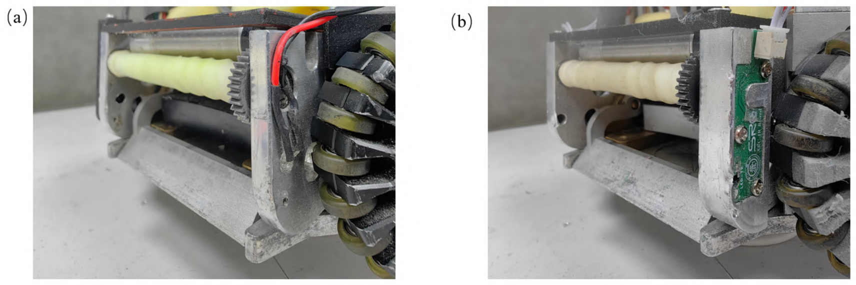

Fig. 11: SRC 2026 Infrared Sensor. (a)Before (b)After

5 Conclusion

In the previous sections, we introduced our development efforts regarding software, electronics, and mechanics. In the software section, we presented the reconstructed Python-C++ architecture and the data-driven ball dynamics model. In the electronics section, we highlighted the safer Flyback-based power board, the FreeRTOS embedded firmware, and the modular circuit design. As for mechanics, we improved the split-type dribbler and optimized the assembly reliability. We are confident that our robots will achieve excellent performance at RoboCup 2026 in South Korea.

References

1. Robocup SSL China. Rocos - RoboCup Open Source small size league framework. [Source code]. https://github.com/Robocup-ssl-China/rocos
2. Mohammad Mahdi Rahimi and Jan Segre and Valiallah Monajjemi and A. Koochakzadeh and Sepehr MohaimenianPour and Nicolai Ommer and Avatar, Kazunori Kimura and Jeremy Feltracco and Kenta Sato and Atousa Ahsani, GR-SIM [Source code]. https://github.com/RoboCup-SSL/grSim/
3. Waldemar Celes, tolua++. [Source code]. https://github.com/LuaDist/toluapp
4. Sergei Izmailov. Pybind11-stubgen tool. [Source code]. https://github.com/sizmailov/pybind11-stubgen
5. Pascal Thomet. litgen - Literate Generator. [Source code]. https://github.com/pthom/litgen
6. Y. Wang, F. Yang, X. Liu, Y. Wang, Y. Gu, S. Wang, S. Gao, H. Zhang and X. Zhu, SRC Extended Team Description Paper for Robocup 2025, 2025.
7. A. Ryll and S. Jut, TIGERs Mannheim Extended Team Description Paper for Robocup 2020, 2020.
8. Z. Huang, L. Chen, J. Li, Y. Wang, Z. Chen, L. Wen, J. Gu, P. Hu and R. Xiong, ZJUNlict Extended Team Description Paper for Robocup 2019, 2019.
9. Y. P. Siwakoti, A. Mostaan, A. Abdelhakim, P. Davari, M. N. Soltani, N. H. Khan, L. Li, and F. Blaabjerg, "High voltage gain quasi-SEPIC DC-DC converter," IEEE J. Emerg. Sel. Topics Power Electron., 2019, doi:10.1109/JESTPE.2018.2859425 .