# ZJUNlict Extended Team Description Paper

**Small Size League of Robocup 2023**

Zheyuan Huang, Chenrui Han, Ning Shen, Jialei Yang, Jiazheng Yu, Anke Zhao, Zhike Chen, Haozhe Du, Licheng Wen, Yunkai Wang, Dashun Guo, and Rong Xiong

State Key Lab. of Industrial Control Technology
Zhejiang University
Zheda Road No.38, Hangzhou
Zhejiang Province, P.R.China
rxiong@iipc.zju.edu.cn

**Abstract.** This paper mainly describes the work of the ZJUNlict team in the past three years, including hardware and software. In the hardware part, we did some exploratory work and redesigned the v2023 robot. In the software part, we have made improvements based on the 2019 strategy, including a new passing candidate points algorithm, a new breakthrough skill, and a parameter optimization module that improves the intelligence of our system. At the end of this article, we provide a brief summary of the work that has guided the development of the Small Size League competition in China in recent years during the epidemic.

## 1 Introduction

ZJUNlict is a team mainly composed of undergraduates and we have been participating in the competition since 2004. This paper presents our related work since 2019.

In the hardware part, we made some valuable attempts, and selected the compatible and stable parts to complete the design of the v2023 robot. Part of our circuit design refers to the open source of the team TIGERs [2].

In the software part, our previous work, especially the strategy part, proved to be effective in the 2019 competition. So we optimized the architecture on this basis, and expanded some modules. We have rewritten the pass point calculation module, and the performance is greatly improved. In the strategy selection, a single robot breakthrough with the ball is added to solve the lag occurred when the ball fails to be passed in the previous game. Finally, in order to reduce the difficulty of debugging a large number of parameters in the entire strategy framework, we added a parameter optimization module so that parameters can be self-tuned based on online feedback.

ZJUNlict team has not participated in international competitions since the Sydney competition in 2019, but we have not stopped working. In the past three years, we have organized and participated in 6 provincial and national competitions, including an online competition, with more than 20 participating college teams. We introduce this part of the work in detail in Section 4.

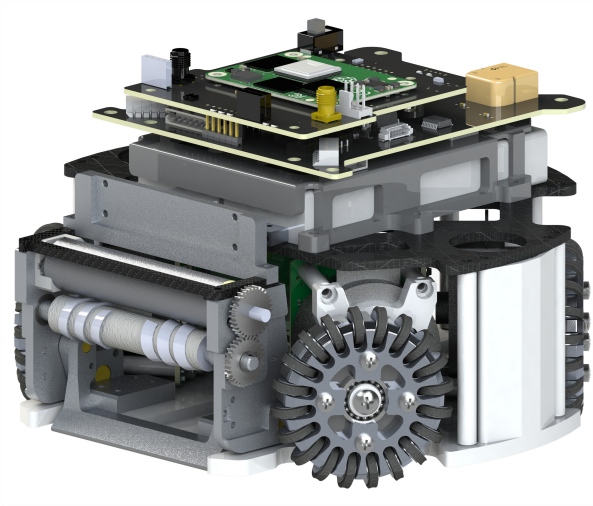


Fig. 1: CAD rendering of v2023 robot without cover

## 2 Hardware

The robot has undergone significant changes since 2019, aimed at improving performance and fixing initial issues. The mechanical design aims for a lower center of gravity and improved interaction, while in the circuit part, we switched to using a Raspberry Pi as the computing module and redesigned the power board and the motherboard. Figure 1 shows the rendering of v2023, with a comparison to v2019 in Table 1.

The Figure 2 depicts an exploded view of the robot's modules. The important improvements are explained below.

Part 2 is the gear box. We focus on improving the gear box for better robot movement by reducing the wheel's moment of inertia and friction in the gear box. The weight has been reduced and the bearings changed from steel to ceramics.

Part 3 is the battery box. We update the battery by replacing the original 4s1p lithium battery with a 2s2p battery, which is placed under the motherboard to lower the center of gravity. The robot's height without the hat is around 120mm. Despite the changes, the height of the cover remains unchanged to maintain space for the WIFI antenna.

Part 4 is the pattern card. During the game, the robot needs to know its own number to complete the command communication. Our old solution is to place the knobs on the robot motherboard, and the knob numbers need to correspond to the color code numbers when replacing the hat. Such an interaction is cumbersome and error-prone. In the new version, we put the circuit board carrying the knob in the pattern card, and use magnetic pins to communicate with the

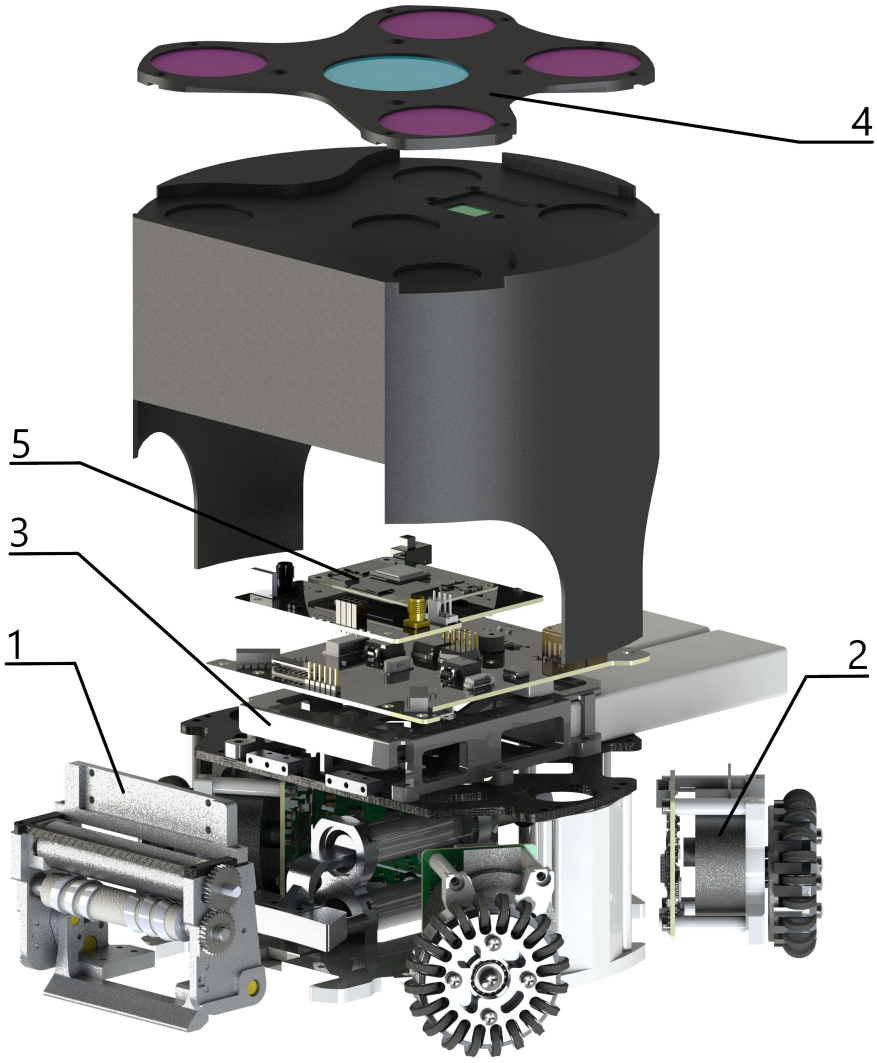



Fig. 2: Exploded view of major robot. ① Dribbler. ② Motion Mechanism. ③ Battery Box. ④ Pattern Card. ⑤ Computing Module

Table 1: Robot Specifications

| **Version** | **v2019** | **v2023** |
|---|---|---|
| Dimension | $\Phi 179 \times 149mm$ | |
| Wheel diameter | $\Phi 56mm$ | |
| Max ball coverage | 18.7% | 19.3% |
| Driving motor | Maxon EC-45 flat 50W | WZD-EM4317 |
| Driving gear material | 2Cr13 | POM |
| Motor driver microcontroller | N/A | ESP32-Pico-D4 |
| Encoder | E4T-X, 4000ppr | AS5600, 4096ppr |
| Dribbling motor | Maxon EC-16 30W | WZD-ECS1656 70W |
| Dribbling stall torque | 32.04mNm | 78mNm |
| Kicker charge | 4400uF@200V | |
| Wireless IC | 2 x nRF24L01+ | Intel AX210(WIFI6) |
| Power supply | Li-Po,14.8V 4s1p,2200mAh | Li-Po,14.8V 2s2p,2200mAh |
| Compute module | STM32H743 | Raspberry Pi CM 4 |

motherboard. We only need to modify the knob once during production and it can be used continuously.

Part 5 is the main computing module, which we've replaced the original STM32 chip with RPi CM4 [1]. This change brings several benefits, such as improved communication with WIFI6 network cards using PCIE, a higher main frequency, and a simpler debugging environment. However, the CM4 also posed the issue of insufficient GPIO, so we redesigned the circuit architecture, as shown in Figure 3, to resolve this.

## 3 Software

### 3.1 Dynamic Passing Points Searching (DPPS) v2

We introduced our passing strategy module DPPS v1 [1] in 2019, which achieved good results in the game. This year, we mainly optimize the running time of the algorithm and the quality of the solution.

In DPPS v1, candidate passing points were sampled with the ball as the center, uniformly along the passing line angle and speed. However, this sampling method resulted in an overall distribution of the candidate passing points that was not reasonable, a large number of candidate points were sampled in areas without our robots, which were all invalid candidate points due to the lack of receiving vehicles close enough. The safety check of these points caused waste

[1] https://datasheets.raspberrypi.com/cm4/cm4-product-brief.pdf

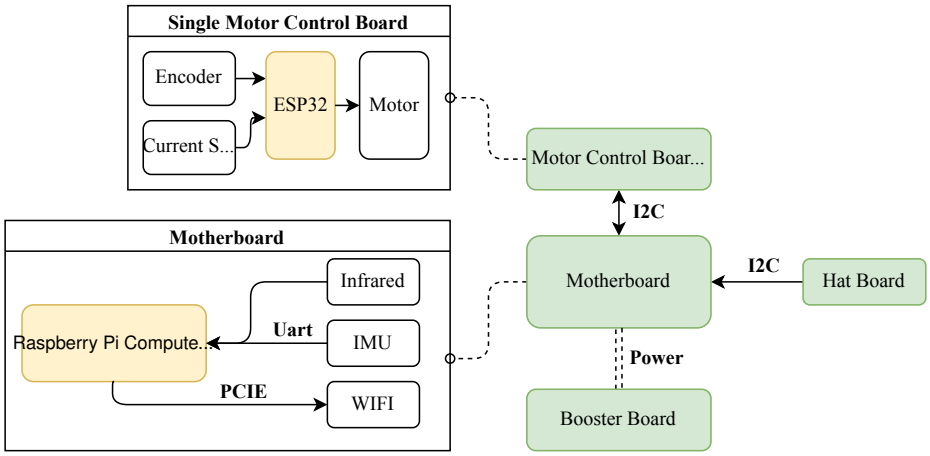



Fig. 3: Circuit Architecture

of computational resources. On the other hand, the sampling density was low near our robots that were far from the ball, which may miss potential feasible passing points. In fact, feasible passing points that passed the safety check were all distributed in the vicinity of candidate receiving robots.

In DPPS v2, based on the above considerations, improvements of sampling method for candidate passing points are shown in Table 2 and Figure 4.

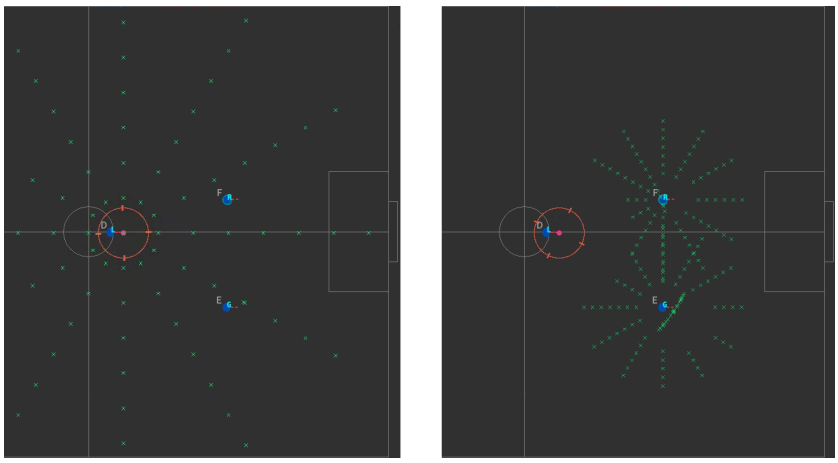


Fig. 4: DPPS v1 & v2 comparison

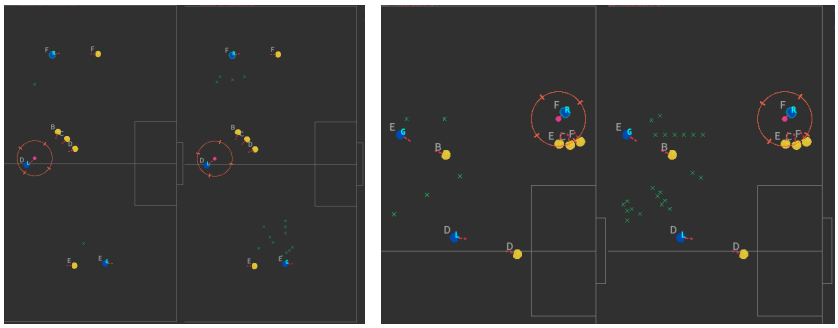
*Real-time Calculation.* In addition to the above improvements, since the interception prediction searches for different enemy robots are independent of each

Table 2: Changes of DDPS v2 over DDPS.

| Details | DPPS | DPPS v2 |
|---|---|---|
| **Sampling center** | ball | each candidate receiver |
| **Enumerate items 1** | passing direction | angle relative to receiver |
| **Enumerate items 2** | passing velocity | distance to receiver |
| **Candidate receiver** | all teammates(up to 16) | corresponding receiver(1) |

other, we can further perform parallel acceleration, which significantly reduces the running time of each thread.The execution time of DPPS v2 for one frame was optimized from 30ms to less than 10ms. Under the vision frame rate of 73FPS, the real-time search for passing points was achieved.

*Solution Performance.* A comparison of typical scenarios is shown in the Figure 5. In these scenarios, when the total number of sample points was comparable, DPPS v2 obtained significantly more feasible points than DPPS v1. Compared with DPPS v1, the new feasible points help robots to achieve smoother pass and receive combinations, and even to score goals.



(a) Scenario 1 (b) Scenario 2

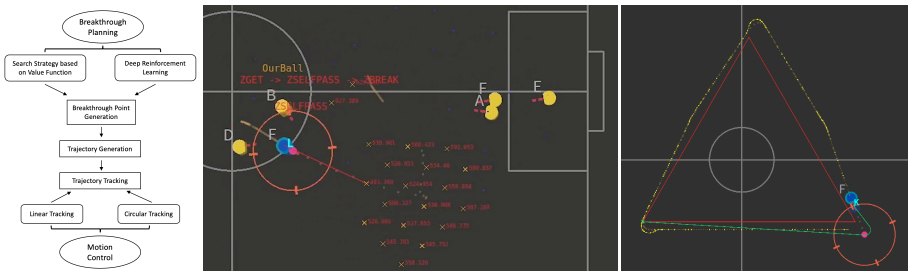
Fig. 5: DPPS Performance (left v1 and right v2 for each scenario)

### 3.2 Breakthrough with Self Pass

Dribbling is an effective offensive tool for advancing the ball in soccer. The dribbling mechanism allows our robot to control the ball in a narrow range, but the friction between the ball and ground increases during dribbling, limiting the robot's speed and direction. Additionally, the rules restrict the robot's movement

to no more than 1 meter while dribbling the ball. Due to these factors, ball-sucking for long distances is challenging.

To overcome these limitations and enable the robot to hold and advance the ball like a human player, we have developed a skill called "Self Pass". This skill tracks the ball's trajectory after being kicked, allowing for continuous dribbling. The skill is based on a technology framework with a breakthrough point planning upper layer, which generates points of attack through value-based search strategies or deep reinforcement learning. The motion control module at the underlying level tracks both linear and circular trajectories. Figure 6(a) illustrates the technology framework for the Self Pass skill.




(a) Framework (b) Feasible Points and the Best (c) Tracking Result

Fig. 6: Skill Self Pass

### 3.3 Planning

In this module, two methods were tested: the value-based strategy and the deep reinforcement learning-based strategy.

*Value-based Strategy.* In each round of planning, we conduct security check to evaluate the possible pass strategies. We first obtain all candidate pass points. To choose the best point, features are extracted including distance to the final target, angle with the opponent, refraction angle, angle between robot orientation and candidate point. A score of each candidate point is calculated with the weighted sum of the features. The computation is accelerated by GPU parallel computing, reducing the time to within 10ms. The results are shown on visualization software. In Figure 6(b), unsafe points are purple dots, safe points are yellow crosses, and the best Self Pass line is the red line connecting the robot and a yellow cross.

*RL-based strategy.* We also tried DDPG [5] and PPO [6] algorithms to generate the breakthrough point using deep reinforcement learning. We input the final target, field boundary, opponent position and velocity into the nerual network

model after normalizing them. The model outputs the best Self Pass coordinates. The reward function includes bonuses for getting closer to the final target and penalties for ball interception by an opponent. After 100,000 rounds of training, the agent learns to handle opponent challenges.

*Comparison* The reinforcement learning approach effectively increases ball handling time under pressure, but is not reliable in complex, changing environments like multiple opponent pressing and may not converge to the final target. The value-based strategy may require more computation but is more interpretable, allowing for human adjustments to breakthrough time and direction. It leads to better performance in simulation and real-world testing. Thus, the value-based strategy is used in actual competitions.

### 3.4 Motion Control

After generating the Self Pass point in breakthrough planning, the robot rotates to the target angle and kicks the ball. Dubins curve [4] is used to generate linear and circular trajectories for the robot to intercept the ball based on ball position and velocity. For motion control, bang-bang control calculates the velocity for linear trajectory tracking, and PID control calculates the position error compensation. During ball-holding, the robot turns around the ball instead of its center to maintain control. Lagrangian dynamics constraints are applied to distribute linear and angular acceleration reasonably within the robot's limits.

Figure 6(c) demonstrates the results of triangle trajectory tracking using the Self Pass module. The reference path is an equilateral triangle with a length of 3.5 meters, represented by the red lines in the figure. The yellow trajectory indicates the actual movement of the robot. The green cross marks the next target, while the green line represents the robot's path planning during the ball chasing process, which is generated by the Dubins Curve algorithm.

The motion control module can make the robot's running speed reach 2.8m/s while dribbling and it can still maintain a linear speed of 1.5m/s when turning.

### 3.5 Self-tuning Parameter Module (STPM)

As our strategy module evolves, the number of parameters continues to increase, such as the weight of evaluation functions and the thresholds in decisions. The more factors we consider to improve the system's intelligence, the harder it becomes to find the perfect set of parameters that works for every situation. We can only make adjustments based on our instincts and experience and test them out through matches to see if we have actually improved overall performance. This manual process is time-consuming and inefficient, making it far from a perfect solution. Hence, the idea of a self-tuning parameter module was introduced.

The module is mainly used to adjust complex parameters, particularly those of vague or abstract meanings. For example, the weights of factors in evaluation functions do not directly reflect certain behaviors on the court but affect

decision-making in different ways under different circumstances. Simply adjusting a certain factor does not always result in expected outcome as factors are interrelated. In such situations, the self-tuning parameter module will be utilized.

The module is designed to work both offline and online. When working offline, the module optimizes parameters in our strategy modules. With the environments and reward function set for the module, it will automatically run the environment, evaluate the results, and adjust the parameters in a repeat manner. Ultimately, it outputs the best set of parameters for the set environments. While working online, the module will select the best set of parameters from a library constructed offline. It monitors the game and determine if a certain part of our strategy is not performing as expected or if the opponent is using a certain type of strategy. Then, it chooses the best set from the library that should improve performance against that particular opponent.

**Module Structure.** Our system aims to run a specified environment repeatedly while collecting data. After each run, it should evaluate the results and generate a new set of parameters. To achieve this, the system is divided into three components: Core Strategy Maker(CORE), Environment Controller (ENV) and Optimizer (OPT). The communicating structure is shown in Figure 7

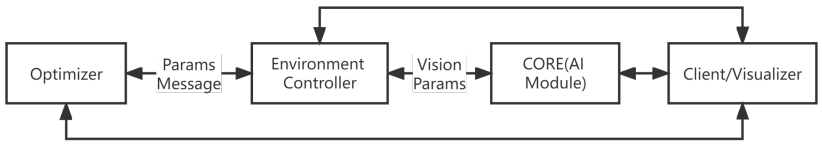



Fig. 7: STPM Structure

The Environment Controller is the core component of STPM. It automatically loads the specified environment(s), runs them, judges the end of a run, evaluates the score, resets the environment(s), and facilitates communication between other components. See pseudo-code in Algorithm 1.

CORE is mostly the original program, but now has the ability to modify parameters when it receives a message from the ENV. In certain scenarios, it takes an end-to-end approach instead of relying on manually extracted features.

The Optimizer's role is straightforward: it uses algorithms to generate a new set of parameters based on the results from the ENV and previous stored results (if required). By changing the algorithm, the strategy for finding the best results can be altered (e.g. optimizing near the original value or exploring new areas), allowing for adaptability to different situations. It replaces the manual task of adjusting parameters based on experience. See pseudo-code in Algorithm 2.

**Demostration.** We have selected a straightforward offline scenario for easier understanding with visualization. The yellow team consists of two backs and a goalkeeper, tasked with defending. The blue team has two attackers, where the

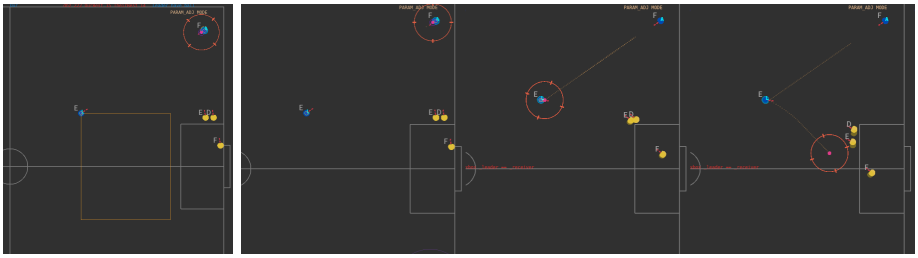
**Algorithm 1** Environment Controller Main

```
EnvList ← environments
ParamLib ← ∅
while TRUE do
    vision ← new vision
    for all Env ∈ EnvList do
        Env.update(vision)
        if Env.finished() then
            Env.reset()
            Score ← score evaluated during running
            Msg ← current set of params and corresponding score
            sendToOPT(Msg)
        end if
        if getMessageFromOPT() then
            Msg ← new msg
            Params ← decodeMsg(Msg)
            sendParamToCORE(Params)
            ParamLib.update(Params, Env)
        end if
    end for
end while
```

**Algorithm 2** Optimizer Main

```
ParamInfo ← ∅
while TRUE do
    if getMsgFromENV() then
        Msg ← new msg
        Param, Score ← decodeMsg(Msg)
        NewParam ← generateNewParam(Msg, ParamInfo)
        sendMsgToENV(NewParam)
        ParamInfo.store(Param, Score)
    end if
end while
```

assist robot (marked with 'A') will pass the ball to the shooter robot (marked with 'L') who will attempt to score. The position of the assist robot is fixed while the shooter robot's position (represented by the x and y values) is tuned within the limitations of the rectangle shown in Figure 8(a).



(a) Initial State (b) Process Schematic

Fig. 8: STPM demo. Adjust the position of receiver in a pass-and-shoot scenario.

In this simple scenario, the score is evaluated based on the shot quality. If the ball is blocked by the opponents' defenders and fails to enter the penalty area, the score is 0. Otherwise, the score is calculated using Equation1:

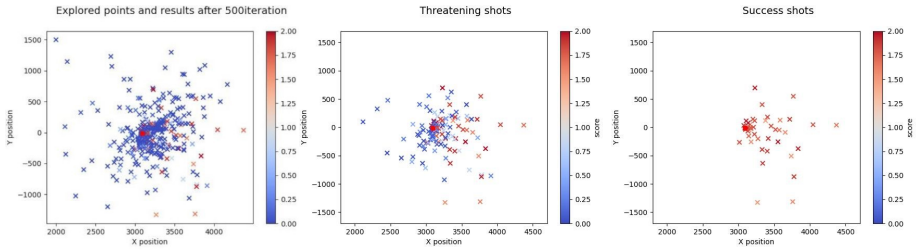
$$Score = (Dist2Baseline) * (Dist2SymmetricLine) + ScoreBonus \quad (1)$$

Distances are normalized and the score bonus is a constant value of 1.0, which is only added when a goal is scored.

ENV determines the end of a run if the ball is stopped by defenders or the goalkeeper, goes out of bounds, or goes into the goal. It then sends the score to OPT, which generates a new set of parameters using a Simulated Annealing algorithm [7] and sends them back. ENV stores the result and sends the new parameters to the CORE, which updates the parameters in the program. Meanwhile, ENV controls the simulation program to reset the ball and robots to their initial positions. Once the reset is complete, the entire process starts again with the new set of parameters.

The results of a 500 iteration test run are depicted in Figure 9. Each mark represents a set of parameters, and the color is determined by the score. The best-scoring point in each stage is marked with a red circle. As evident from the figure, the algorithm gradually discovered the best-scoring points. However, due to various uncertainties in the process, such as variations in passing, shot timing, defending reactions, etc. Even near the best-scoring point, many attempts still failed. To make the results clearer, we removed the results of score 0 and non-goals separately, as shown in Figure 9(b). It can be seen that the threatening shots that entered the penalty area mainly originated from the central part, where the angle for shooting at both sides of the goal is large. And the successful

shots obviously started closer to the goal, giving the defenders less time to react. However, if the shooter gets too close to the goal, the angle between the ball's speed and the line to the goal will be too large for a one-touch shot, thus reducing the opportunity to score.




(a) 500 iteration results (b) Effective results (c) Success results

Fig. 9: Results for One-pass-shoot Demostration

In general, the results match common sense and the capabilities of our skill sets, demonstrating the effectiveness of our module. Of course, real in-game situations are far more complex, requiring more sophisticated algorithms and judging conditions. Nevertheless, the overall process remains the same.

## 4 SSL in China

### 4.1 Summary of works

In the last three years, we worked hard to maintain the atmosphere of the game and lower the preparation threshold for new teams in China. We summarize our main work into the following two parts:

**Rules and official software.** We translated the rules into Chinese and maintain on github with the annual international competition updates. Due to the complexity of utilizing software such as autorefs, we also produced videos to explain the rules and software operations to ensure that new teams can easily participate in the competition. In the technical challenge part, in order to guide the teams to keep up with the international level as soon as possible, we set up a special technical challenge to guide each team to explore and make great progress in positioning ball calculation, passing coordination and other aspects.

**Robot system.** We mainly open sourced the software and provide video tutorials. We open sourced our mAn client software in the 2019 International Competition. On this basis, we also open sourced the Lua-based strategy framework mentioned in previous TDPs [3]. We integrated it into the rocos[2] project, and explain compilation and strategy coding with video and text tutorials. At

[2] https://github.com/Robocup-ssl-China/rocos

present, the video of this project is in Chinese version, and the English version will be re-recorded in the future. If anyone is interested in this, please contact us.

### 4.2 Competitions

In the last three years, we have supported RoboCup ChinaOpen three times including an online competition and three provincial competitions, all using autorefs. At present, the size of the ChinaOpen competition venue is 12m×9m, with 8 robots. It is expected to hold an 11vs11 competition in 2024. The competition in Zhejiang Province, where our university is located, currently adopts a 9m×6m, 6vs6 competition.

These tree-style competitions makes it easier for teams to participate in international competitions. At present, many teams in China have already possessed relatively high strategy development capabilities and basic hardware R&D capabilities. In the ChinaOpen in November 2022, the ZJUNlict team won the third place in the competition, which also proves the effectiveness of our related work in guiding other teams. We look forward to more teams participating in future RoboCup competitions and bringing more vitality to our competition.

## References

1. Chen Z, Zhang H, Guo D, et al. Champion team paper: Dynamic passing-shooting algorithm of the RoboCup soccer SSL 2019 champion[C]//RoboCup 2019: Robot World Cup XXIII 23. Springer International Publishing, 2019: 479-490.
2. Ryll A, Ommer N, Geiger M. RoboCup 2021 SSL Champion TIGERs Mannheim-A Decade of Open-Source Robot Evolution[M]//RoboCup 2021: Robot World Cup XXIV. Cham: Springer International Publishing, 2022: 241-257.
3. Li C, Xiong R, Ren Z, et al. Zjunlict: Robocup 2014 small size league champion[C]//RoboCup 2014: Robot World Cup XVIII 18. Springer International Publishing, 2015: 47-59.
4. Reeds J, Shepp L. Optimal paths for a car that goes both forwards and backwards[J]. Pacific journal of mathematics, 1990, 145(2): 367-393.
5. Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.
6. Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.
7. Kirkpatrick S, Gelatt Jr C D, Vecchi M P. Optimization by simulated annealing[J]. science, 1983, 220(4598): 671-680.