ZJUNlict Extended Team Description Paper

Small Size League of Robocup 2026

Hanchuan Liu, Haoxuan Guo, Peiyin Feng, Qi Jin, Xilun Luo, Hao Chen, Yuyan Tang, Hengzhi Zhang, Xingwen Zheng, 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 presents the ZJUNlict teams work over the past year, covering both hardware and software advancements. In the hardware domain,We comprehensively improved the robots dribbling and motion performance. On the software side, we integrated a gradient-based trajectory optimizer to ensure smooth, continuous-curvature paths in complex navigation. Experimental results show that these optimizations significantly reduce traversal time and enhance the robot's agility in high-dynamic game scenarios.

1 Introduction

Since 2004, ZJUNlict has been actively participating in competitions. In 2023, after the pandemic, we resumed participation in offline world competitions and have continued to do so ever since. Over the past year, we have made significant improvements in both hardware and software to enhance overall robot performance.

In the hardware section, we focus on the principle and design of the active dribbling system and the resulting improvement in field adaptability. We also replaced the drive motors with FOC wheel motors to reconstruct the motion system, and introduced an optical-flow-based ranging system to further improve the robots motion performance.

In the software section, we present the dynamic modeling, simulation, and control strategies for our asymmetric four-wheeled omnidirectional robots. We detail the integration of a new trajectory optimization method as an enhancement over the traditional RRT-based planner, providing experimental evidence to validate its superior efficiency. Finally, we describe our restructured hierarchical strategy architecture and the overall system workflow.

2 Hardware

2.1 Active Dribbling

Historically, we have placed great emphasis on dribbling and treated it as a core component of our tactical system. During RoboCup 2025 and Japan Open 2024, we clearly realized that our dribbling approach is significantly affected by ground friction. To address this issue, we introduced the concept of Active Dribbling. In our TDP 2025, we mentioned the concept of a one-dimensional massspring-damping system, abstracting the dribbling mechanism accordingly[1]. As shown in Fig. 1, m represents the ball, and M represents the mouth. The parameters k_1 and c_1 denote the elastic contact characteristics between the dribbling tube and the ball, while k_2 and c_2 represent the elastic buffering characteristics between the body and the mouth.

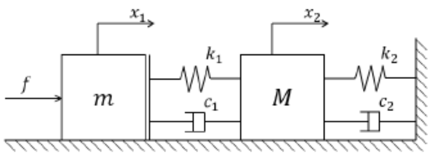


Fig. 1. Simplified dribble system model[1]

For a considerable period of time,The functions of k_2 and c_2 are entirely realized by the sponge material. However, both the elastic coefficient and damping coefficient of the sponge are difficult to quantify. Therefore, we attempted to introduce an FOC motor to control the rotation of the mouth, enabling it to quantitatively simulate a one-dimensional massspringdamping system.

This approach brings two main advantages. First, during ball reception, the motor can emulate a spring, allowing the ball to enter the mouth more easily instead of bouncing out. Second, during dribbling, the motor simulates damping by applying a downward force F_N to the ball after it enters the mouth. This normal force directly increases the friction force f between the ball and the ground. Moreover, the incremental friction force f can be adjusted to accommodate fields with different friction coefficients.

$$f = \mu(F_N \cdot \sin\alpha + mg) - F_N \cos\alpha \tag{1}$$

$$\Delta f = F_N(\mu \sin\alpha - \cos\alpha) \tag{2}$$

Given that α is sufficiently large, Δf remains positive most of the time. Thereby enhancing our adaptability to smooth surfaces, as shown in Fig. 2.

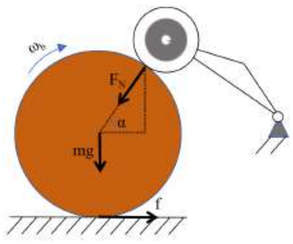


Fig. 2. Force Analysis of the Dribbling Structure

The actual rotation angle of the ball-capturing mouth is very small, which imposes high precision requirements. Therefore, we adopted a gear transmission mechanism to control the rotation of the mouth. In order to accommodate an additional motor and gear set within the limited space of the mouth, we designed the side plate of the mouth as part of the transmission gear, as shown in Fig. 3.

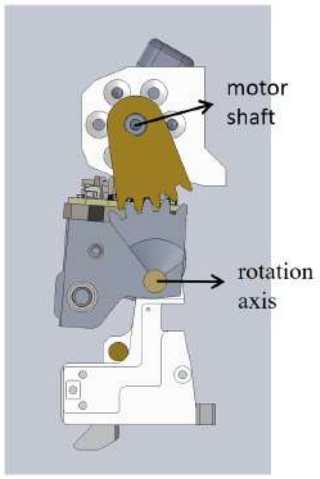


Fig. 3. Left Side View of the Dribbling Structure

2.2 Color Recognition

The current version of the robot relies on either using encoders or accessing the Raspberry Pi system to adjust the vehicle number, which greatly limits our flexibility during competitions. Therefore, we developed a color recognition system based on the STM32F405 microcontroller and the TCS34727FN color sensor to accurately identify the vehicle number by recognizing the color of the car's color marker. This enables quick identification of the vehicle number before the competition.

The core design requirement is that the color recognition accuracy must be high enough to closely match human visual perception and effectively suppress infrared light interference caused by other components.

To address the issue of insufficient differentiation between similar colors, we repeatedly adjusted the RGB threshold values, added saturation detection, and optimized the sensitivity for red color recognition. In order to make the color recognition less sensitive to lighting conditions, we designed an adaptive threshold adjustment mechanism, including simple white balance calibration and anomaly detection to filter out outliers.

Additionally, an LED auxiliary lighting system was designed for low-light environments, with the lighting angle optimized based on the actual installation position. Table 1 shows the experimental data for the color recognition system.

Table 1. RGB data collection

Expression	Value	Location	Refresh
clear	3263	20000098	5 Hz
red	1712	2000009A	5 Hz
green	1093	2000009C	5 Hz
blue	720	2000009E	5 Hz
f	–	08003CB8	2 Hz
detectedColor	COLOR_YELLOW	200000A0	5 Hz

2.3 FOC Motor Controller Hardware

The wheel-drive system of the ZJUNlict robot is upgraded from a six-step commutation speed controller to a sensored Field Oriented Control (FOC) drive with a torque/position hybrid interface. The goal of this hardware redesign is to provide accurately controllable phase currents and rotor position feedback, so that high-level dynamics controllers can rely on a well-defined torque actuation channel instead of a purely kinematic speed source.

The controller is built around a 24V DC bus and a DC-link capacitor that stabilizes the supply for a three-phase MOSFET inverter bridge. Phase currents are measured by shunt-based current sensors on each phase, routed to three ADC units on the STM32G4 microcontroller. A rotor position sensing module, supporting both encoder and Hall-based configurations, provides electrical angle information required by the FOC algorithm. All power-stage switching is driven by a high-resolution timer (HRTIM), which also triggers synchronized injected ADC conversions twice per control period so that current samples, rotor position and DC bus voltage are acquired in a deterministic manner.

On top of this measurement and actuation backbone, the MCU executes a Clarke transform to obtain α–β currents, a Park transform to obtain d–q components, and inner PI regulators for i_d and i_q. The inverse Park transform generates three-phase voltage references, which are converted into high-resolution PWM compare values by a Space Vector PWM (SVPWM) modulator running on the HRTIM peripheral. The hardware supports additional functions required by later control layers, including bus-voltage sensing for field weakening and torque limiting, as well as a galvanically-isolated CAN interface that serves as the primary command and telemetry channel.

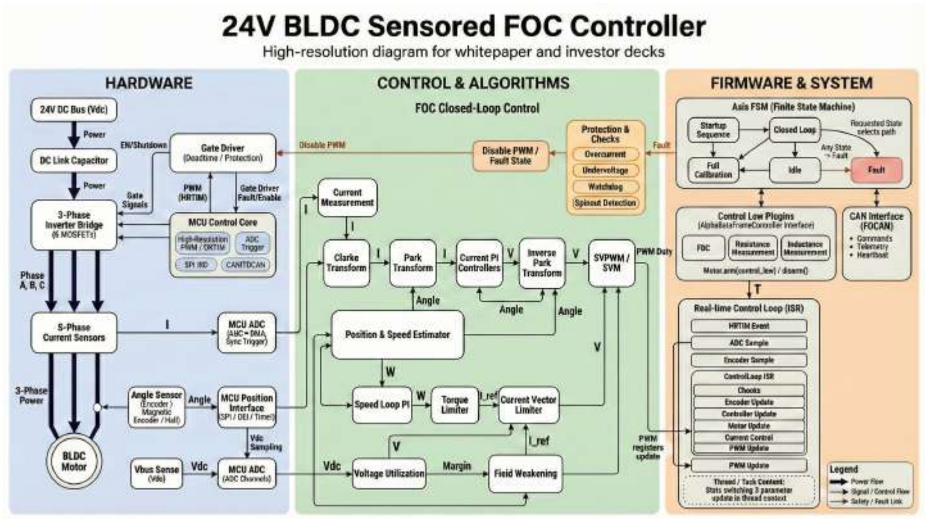


Fig. 4. System-level architecture of the 24V sensored BLDC FOC controller, including power hardware, measurement and actuation paths, and the real-time control core.

2.4 Optical-Flow Odometry Hardware

The odometry subsystem is designed to provide on-board planar localization and velocity feedback for the robot on an STM32 platform. In the previous system, velocity feedback primarily relied on the overhead vision pipeline. Besides spatial accuracy degradation in regions with strong lens distortion, the control bandwidth is fundamentally limited by the end-to-end transmission latency of the vision feedback. In highly dynamic motions, this latency can lead to an unstable or overly conservative velocity closed loop. By enabling the robot to measure its own velocity and incremental position in real time, the system gains more embodied perception and can reduce sensitivity to both feedback delay and global-vision speed inaccuracies, thereby improving motion performance in high-speed, high-dynamics game scenarios.

The hardware stack is built around an STM32F4 series microcontroller running FreeRTOS, and integrates a single optical-flow sensor with an IMU. The optical-flow module provides accumulated displacement increments in the body frame, (dx_{raw}, dy_{raw}), while the IMU provides yaw angle ψ, yaw rate ω_z, and body-frame linear accelerations (a_x, a_y). Due to mechanical constraints, the optical-flow sensor cannot be mounted at the instantaneous center of rotation. This fixed installation offset, parameterized by (L_x, L_y) with respect to the rotation center, introduces an apparent translational displacement during pure rotation. This rotation-induced displacement (lever-arm effect) must be decoupled in software; otherwise, the raw optical-flow increments will be systematically biased whenever the robot turns in place.

Based on the above sensing configuration, the subsystem applies a deterministic preprocessing and fusion pipeline. First, the yaw increment $\Delta\psi$ is computed from successive IMU yaw estimates, and the rotation-induced displacement is predicted by a rigid-body kinematic model,

$$\begin{bmatrix} dx_{\text{rot}} \\ dy_{\text{rot}} \end{bmatrix} = \begin{bmatrix} \cos(\Delta\psi) - 1 & -\sin(\Delta\psi) \\ \sin(\Delta\psi) & \cos(\Delta\psi) - 1 \end{bmatrix} \begin{bmatrix} L_x \\ L_y \end{bmatrix}. \tag{3}$$

The true translational increment is then obtained by subtracting this term from the raw optical-flow measurement. Second, to suppress high-frequency jitter in the optical-flow-derived velocity, a first-order IIR filter is applied with a fixed coefficient $\alpha = 0.15$. Third, an extended Kalman filter (EKF) is used in a planar position–velocity state formulation, where IMU accelerations drive the prediction step and the preprocessed optical-flow velocity updates the measurement step. Finally, the estimated body-frame increments are projected into the world frame using the yaw angle, and the global pose is initialized by the overhead vision system at robot placement.

3 Software

3.1 Dynamics Simulation and Control Software

RoboCup SSL robots are constrained by the mechanical space requirements of ball-kicking and ball-lifting mechanisms, which leads to an asymmetric four-wheel omni-wheel layout. In this configuration, the thrust center of the wheel group is significantly misaligned with the robot's center of mass, and lateral motion inevitably introduces an unintended yaw torque (Parasitic Yaw Moment). To study and compensate this coupling safely and repeatably, we build an engineering Software-in-the-Loop (SIL) simulation platform based on ROS2 Humble and Gazebo Fortress. The simulator is launched through `ros_gz_sim` with a unified `.launch.py` entry. Torque control is implemented via `gz_ros2_control` using the `ForwardCommandController` in effort mode together with the standard `JointStateBroadcaster`. A dual-layer description is adopted to preserve dynamics fidelity: on the ROS2 side, the `robot_state_publisher` reads the URDF to obtain the joint structure and TF tree, while on the Gazebo side, the spawner loads an SDF model so that physics parameters such as friction coefficients are not lost. IMU and odometry plugins are added and bridged back to ROS2 via `ros_gz_bridge` for observation and algorithm development. To improve contact stability for roller omni-wheels, wheel collision geometry is modeled as spheres instead of cylinders to ensure single-point contact; anisotropic friction is approximated by differential coefficients μ_1 and μ_2, and only wheel–ground collision pairs are retained to reduce computational load. The resulting simulator reproduces key real-world behaviors: under longitudinal torque steps the robot exhibits drift that can be mitigated by ramped commands, and under lateral open-loop torque the asymmetric chassis produces a circular trajectory.

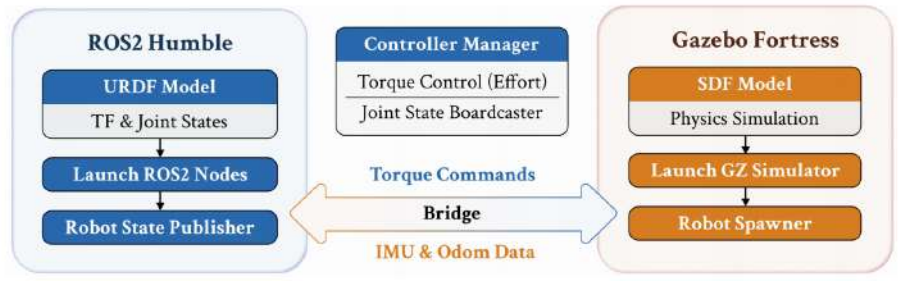


Fig. 5. Robot simulation and control architecture based on ROS2 Humble and Gazebo Fortress, showing the dual-layer URDF/SDF description and the torque-control bridge.

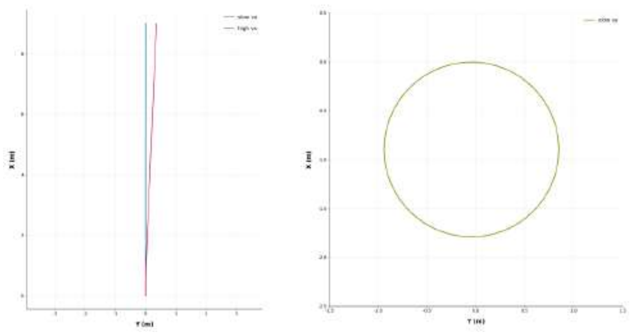


Fig. 6. Motion trajectories of the simulated robot under different torque commands: longitudinal motion (left) and lateral open-loop motion (right), highlighting the circular path induced by the asymmetric layout.

On top of this simulator, the control algorithm targets the execution-layer mapping from commanded chassis velocity to individual wheel torques under the asymmetric structure. A closed-loop trapezoidal motion-profile planner generates smooth acceleration commands with an interface reserved for jerk constraints, which can be upgraded to an S-curve speed planner. In each control cycle, the desired acceleration and its saturated reference are computed as

$$a_{\text{req}} = \frac{v_{\text{target}} - v_{\text{current}}}{\Delta t}, \tag{4}$$

$$a_{\text{ref}} = \text{clip}(a_{\text{req}}, -a_{\max}, a_{\max})\,. \tag{5}$$

and the internal reference velocity is updated according to

$$v_{\text{current}} \leftarrow v_{\text{current}} + a_{\text{actual}}\,\Delta t. \tag{6}$$

The desired inertial force is given by $F_{\text{inertial}} = Ma_{\text{ref}}$. A simplified ground-robot resistance model is then applied to obtain the applied force

$$F_{\text{applied}} = Ma_{\text{ref}} + F_{\text{resist}}. \tag{7}$$

$$F_{\text{resist}} = F_{\text{viscous}} + F_{\text{coulomb}}. \tag{8}$$

$$F_{\text{viscous}} = Dv. \tag{9}$$

$$F_{\text{coulomb}} = C\,\text{sign}(v). \tag{10}$$

where $D \in R^{3\times3}$ is a positive-definite viscous damping matrix and $C \in R^3$ is the Coulomb friction coefficient vector. To cope with unknown and time-varying friction, an online gradient-descent adaptation updates (D, C) using the dynamics error

$$D \leftarrow D + \eta\,(e \otimes v)\,\Delta t. \tag{11}$$

$$C \leftarrow C + \eta\,e \odot \text{sgn}(v)\,\Delta t. \tag{12}$$

$$e = (F_{\text{applied}} - F_{\text{resist}}) - Ma_{\text{real}}. \tag{13}$$

where $\otimes$ denotes the outer product and $\odot$ denotes element-wise multiplication. Finally, wheel force (and torque) commands are obtained via a center-of-mass (COM)-centered pseudo-inverse mapping. The torque distribution matrix is constructed with respect to the calibrated physical COM,

$$B = \begin{bmatrix} \cos\gamma_1 & \cos\gamma_2 & \cos\gamma_3 & \cos\gamma_4 \\ \sin\gamma_1 & \sin\gamma_2 & \sin\gamma_3 & \sin\gamma_4 \\ L_1 & L_2 & L_3 & L_4 \end{bmatrix}_{3\times4}. \tag{14}$$

$$L_i = (x_i - \Delta x_{\text{com}})\sin\gamma_i - y_i\cos\gamma_i. \tag{15}$$

and the Moore–Penrose pseudo-inverse $B^\dagger = (B^TB)^{-1}B^T$ is used to distribute the chassis-level force to wheel-level forces such that the resultant driving force acts at the COM, thereby suppressing the parasitic moment and improving decoupling of longitudinal, lateral, and rotational dynamics.

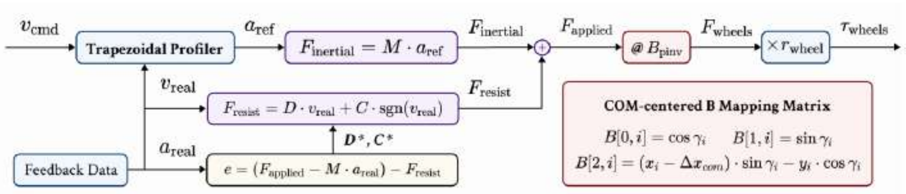

v_{cmd}
Trapezoidal Profiler
a_{ref}
$F_{\text{inertial}} = M \cdot a_{\text{ref}}$
F_{inertial}
F_{applied}
$@\,B_{\text{pinv}}$
F_{wheels}
$\times r_{\text{wheel}}$
τ_{wheels}
v_{real}
$F_{\text{resist}} = D \cdot v_{\text{real}} + C \cdot \text{sgn}(v_{\text{real}})$
F_{resist}
D^*, C^*
a_{real}
Feedback Data
$e = (F_{\text{applied}} - M \cdot a_{\text{real}}) - F_{\text{resist}}$
COM-centered B Mapping Matrix
$B[0,i] = \cos\gamma_i$ $B[1,i] = \sin\gamma_i$
$B[2,i] = (x_i - \Delta x_{com}) \cdot \sin\gamma_i - y_i \cdot \cos\gamma_i$

Fig. 7. Dynamic control flowchart of the COM-centered four-wheeled omnidirectional robot, illustrating the trapezoidal profiler, resistance model, online parameter adaptation, and COM-centered torque distribution.

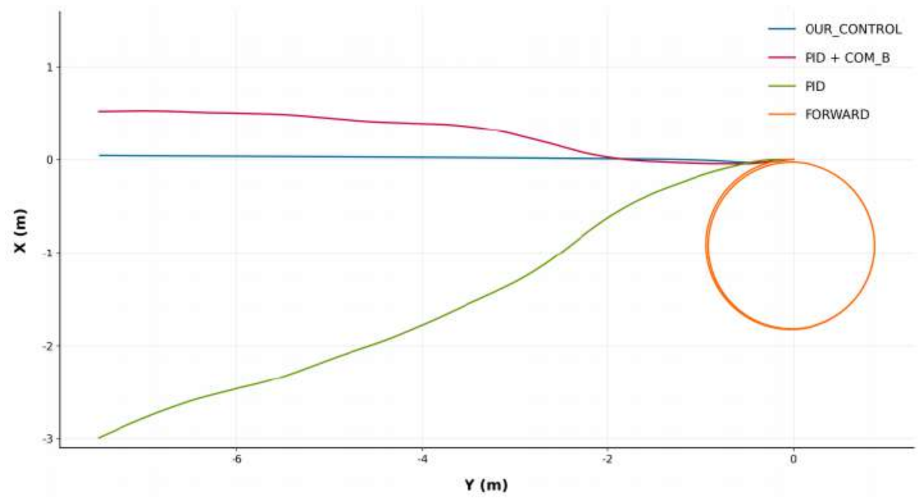


Fig. 8. Trajectory comparison under lateral motion between the proposed dynamics controller, PID with COM compensation, pure PID, and open-loop torque control.

3.2 New Path Planning Method: EGO-Planner

We have been using RRT planner as our path planning algorithm since 2012[2]. In recent years, the RRT algorithm has been proven to be a highly efficient and viable path-planning solution for RoboCup SSL. It enables robots to navigate complex obstacles environments effectively, laying a solid foundation for high-performance motion control. Our approach builds upon the CMDragons framework[3] with several key enhancements:

Firstly, we implement a bidirectional search strategy that grows two RRT trees simultaneously from both the start and goal configurations. To accelerate convergence, the expansion process does not rely solely on uniform random sampling; instead, it incorporates a goal-biased sampling mechanism that selects the target direction with a specific probability.

Secondly, once the two trees are connected, a path-smoothing and pruning heuristic is applied. Starting from the initial position, the algorithm iteratively identifies the furthest reachable node that maintains a collision-free line-of-sight. This node is then added to the path, and the search continues from that point until the goal is reached, effectively minimizing redundant waypoints and streamlining the final trajectory.Algorithm 1 shows the pseudo-code of this process.

Algorithm 1 Improved Bidirectional RRT with Path Smoothing

```
**Input:** $q_{start}, q_{goal}, P_{bias}$, Obstacles
**Output:** Optimized Path $S_{final}$
$T_{start}.init(q_{start})$; $T_{goal}.init(q_{goal})$
**loop**
    $q_{rand} \leftarrow$ RandomSample($P_{bias}$)        ▷ Probabilistically choose $q_{goal}$ or random
    $T_{start}$.Expand($q_{rand}$)
    **if** $T_{start}$ and $T_{goal}$ connected **then**
        **break**
    **end if**
    Swap($T_{start}, T_{goal}$)                              ▷ Bidirectional growth
**end loop**
$S_{raw} \leftarrow$ ReconstructPath($T_{start}, T_{goal}$)
**Step 2: Path Smoothing (Greedy Pruning)**
$S_{final}.add(q_{start})$
$curr\_node \leftarrow q_{start}$
**while** $curr\_node \neq q_{goal}$ **do**
    Find furthest node $n \in S_{raw}$ such that IsCollisionFree($curr\_node, n$)
    $S_{final}.add(n)$
    $curr\_node \leftarrow n$
**end while**
**return** $S_{final}$
```

The raw path generated by the RRT algorithm typically consists of a series of line segments, with motion executed via trapezoidal velocity profiles between waypoints. However, this approach often lacks smoothness when encountering sharp turns-such as navigating around the corners of the defense area. Such abrupt changes in direction lead to suboptimal trajectories and increased travel time, potentially resulting in defensive failures. To address this, we have incorporated insights from the EGO-Planner framework[4], commonly utilized in the UAV (unmanned aerial vehicle) field, to optimize our trajectory generation.

EGO-Planner is a gradient-based local planning method that generates collision-free and dynamically feasible trajectories without the need for a pre-defined ESDF (Euclidean Signed Distance Field). The algorithm formulates trajectory generation as a non-linear optimization problem. By representing the trajectory using B-splines, it optimizes a composite objective function that balances smoothness, dynamic feasibility (velocity and acceleration limits), and collision avoidance. Unlike traditional methods, it employs an iterative approach to push the trajectory away from obstacles by generating environmental force constraints only when collisions are detected.

The implementation process of our deployed EGO-Planner is as follows:

First, a grid map containing obstacle information is generated based on the shared vision data, accounting for both static and dynamic obstacles. Subsequently, a simplified global trajectory is generated and checked for collisions to undergo iterative optimization. Since the trajectory in EGO-Planner is represented by B-splines, the optimization process essentially focuses on the adjustment of B-spline control points.

During this process, a KD-tree is employed to identify the control points requiring optimization. An objective function is formulated by incorporating three penalty terms: smoothness, collision avoidance, and dynamic feasibility. Upon completion of the optimization, the system evaluates the validity of the trajectory; if the planning fails, an A* algorithm is triggered for emergency path searching. Once a valid trajectory is obtained, a PID feedback controller is utilized for trajectory tracking to generate motion control commands. The workflow of EGO-Planner is illustrated in Fig. 9.

Experiment During matches, robots frequently need to navigate around the corners of the penalty area. To evaluate the performance in such scenarios, we designed an experimental setup specifically focusing on corner navigation to compare the efficiency of our EGO-Planner against the previously implemented RRT-based method.

Path Planning Comparison. As illustrated in Table 2, the EGO-Planner significantly optimizes path planning through trajectory smoothing. Compared to the baseline RRT algorithm, EGO-Planner generates more direct and continuous trajectories, effectively avoiding the inefficiencies and detours associated with the piecewise linear planning characteristic of RRT.

Execution Time Analysis. To further quantify the performance, we conducted a comparative analysis of motion execution time under various conditions.

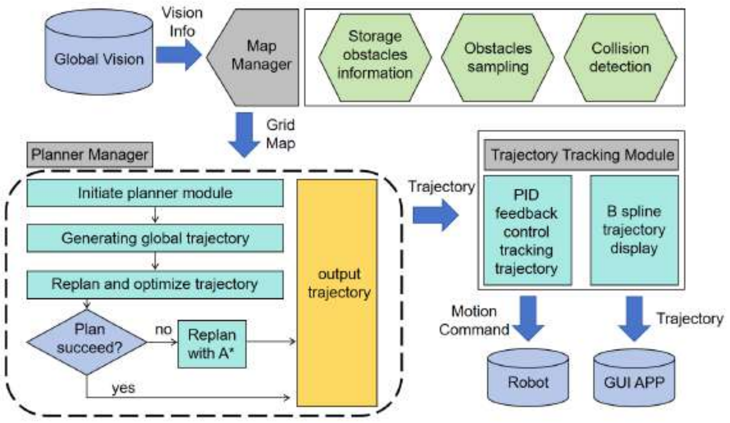
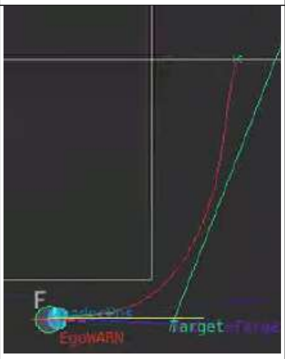
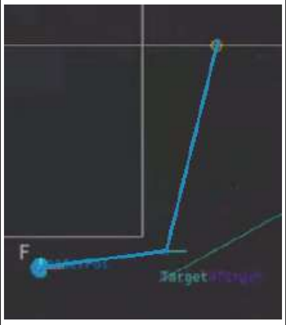
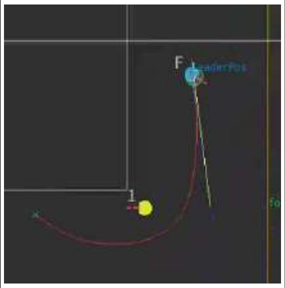
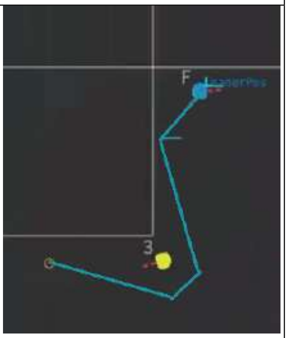
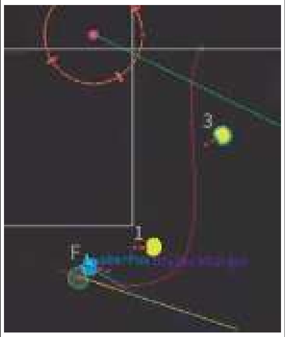
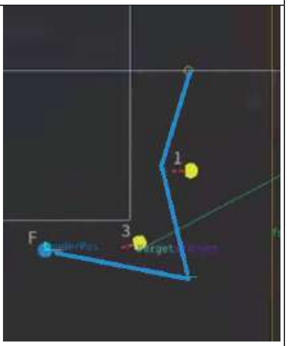


Fig. 9. The workflow of EGO-Planner

(a) Performance in Obstacle-Free Scenarios: As shown in Table. 3, the performance of both algorithms is comparable in the absence of obstacles. During testing, we observed that when EGO-Planner utilizes an intermediate waypoint as a temporary goal for corner navigation, it undergoes an acceleration-deceleration cycle where the velocity approaches zero at the waypoint before re-accelerating. This segmented planning behavior prevents EGO-Planner from demonstrating its full performance potential in simple scenarios.

Table 2. Path Comparison between EGO-Planner and RRT

Scenario	**EGO-Planner**	**RRT (Baseline)**
No Obstacles		
One Obstacle		
Two Obstacles		

(b) Performance in Scenarios with Obstacles: In complex environments, EGO-Planner exhibits a clear advantage. With one obstacle present, EGO-Planner reduced traversal time by **59.9%** and **44.2%** in two respective directions. With two obstacles, the time reduction further reached **64.8%** and **39.1%**, demonstrating superior obstacle avoidance and efficiency.

Table 3. Traversal Time Comparison in Obstacle-Free Scenarios (pos1 ↔ pos2)

No.	RRT (p1-p2)	EGO (p1-p2)	RRT (p2-p1)	EGO (p2-p1)
1	3.6328	2.6479	2.3191	2.9135
2	2.8135	2.6951	2.4604	3.1193
3	2.4338	2.9893	2.3794	2.6099
4	2.5149	1.7876	2.5155	2.5121
5	2.4181	2.6775	2.1361	2.4892
6	2.3573	2.2370	2.3304	2.7023
7	2.4024	1.9739	2.4522	2.6016
8	2.4480	2.6328	2.3293	2.4464
9	2.7710	1.6600	2.7804	2.4420
10	2.5978	2.6826	2.4102	1.5182
11	2.4601	1.8465	2.3888	1.7790
12	2.4597	3.2463	2.3788	2.7091
13	2.2460	1.9782	2.3409	2.2973
14	2.6019	2.9186	2.6019	2.4958
Mean	**2.5827**	**2.4267**	**2.4160**	**2.4740**

Table 4. Comparison of Traversal Time with One Obstacle

No.	RRT (p1-p2)	EGO (p1-p2)	RRT (p2-p1)	EGO (p2-p1)
1	3.9490	1.9408	2.9734	2.0321
2	3.5931	1.4248	2.7890	1.9688
3	5.9353	1.7324	3.3563	1.9576
4	5.6549	1.6679	3.0517	2.1968
5	3.2617	1.7642	2.8425	1.5101
6	2.8367	1.2670	4.2216	1.4508
7	3.6348	1.8342	3.5030	2.1778
8	4.1743	1.3366	3.5061	1.6808
9	3.8397	1.4180	4.0860	1.2633
10	2.8450	1.5403	2.6086	2.1480
Mean	**3.9724**	**1.5926**	**3.2938**	**1.8386**

In conclusion, the experimental results demonstrate that EGO-Planner provides superior motion planning performance during corner navigation within complex environments, successfully addressing the limitations of our previous RRT implementation.

3.3 Software Framework

The strategy module has undergone minimal changes this year, as our priority shifted toward motion control systems. Adjustments were primarily focused on rectifying specific issues identified during RoboCup 2025 to improve the overall robustness of our competitive play.

Table 5. Comparison of Traversal Time with Two Obstacles

No.	RRT (p1-p2)	EGO (p1-p2)	RRT (p2-p1)	EGO (p2-p1)
1	3.3039	1.6753	3.2064	2.5428
2	4.4182	1.9328	2.7307	1.7598
3	3.9411	1.5373	3.2469	1.5084
4	3.8349	1.4486	3.4715	2.2309
5	3.7712	1.3331	3.5273	2.0067
6	3.7693	1.4342	3.5363	2.1268
7	3.4180	1.2637	2.7336	1.9033
8	5.3485	1.2731	3.6475	1.8343
9	6.6890	1.5237	2.7199	2.0135
10	3.4365	1.3542	3.5667	1.7832
Mean	**4.1931**	**1.4776**	**3.2387**	**1.9710**

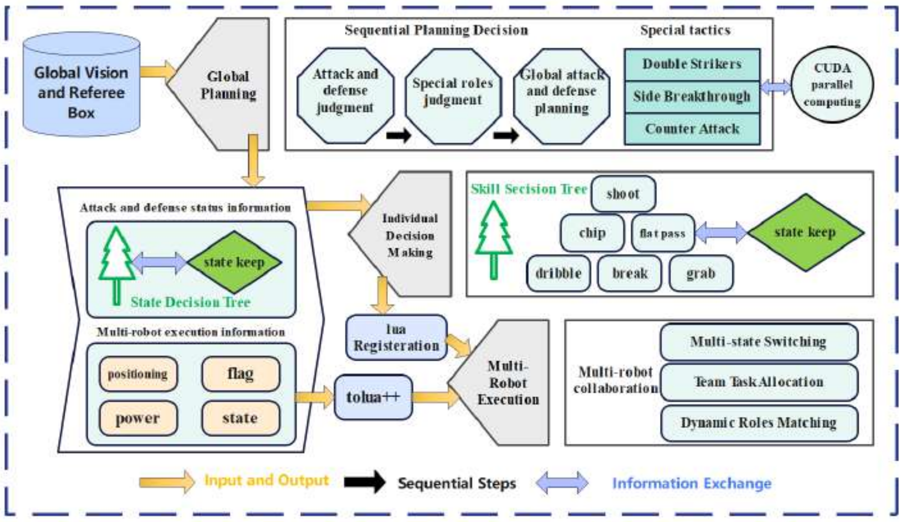


Fig. 10. Strategy Module[1]

The software architecture of our system follows a three-layer pipeline: pre-calculation, task allocation, and command generation. Fig. 10 shows the overall workflow.

Pre-calculation Layer. To address the challenges of computational redundancy within our complex decision system, we streamlined the calculation workflow to ensure efficiency and accuracy. By utilizing CUDA for parallel acceleration, we can rapidly compute optimal tactical points, such as:

– **Tactical Points:** Passing and shooting targets.
– **State Estimation:** Global match state and ball-handler behavior patterns (flat/chip pass, shooting, or dribbling).

Task Allocation Layer. Our strategy is implemented in a Lua-based environment. The system utilizes the pre-calculated information to define task groups, which are then assigned to robots via the *Hungarian algorithm* to achieve optimal global task distribution.

Execution Layer. In the final stage, each robot processes its assigned task to generate real-time motion control commands, ensuring responsive and coordinated team behavior on the field.

References

1. Wu, Z., Wang, L., Yang, Z., Yang, S., Wang, L., Fu, H., Cai, Y., Xiong, R.: ZJUNlict Extended Team Description Paper 2025. arXiv preprint arXiv:2511.02315 (2025). https://arxiv.org/abs/2511.02315
2. Wu, Y., Yin, P., Zhao, Y., Mao, Y., Wang, Q., Xiong, R.: ZJUNlict Extended TDP for RoboCup 2012 (2012)
3. Zickler, S., Bruce, J., Biswas, J., Licitra, M.,Veloso, M.: CMDragons 2009 Extended Team Description[J], 2009.
4. Zhou, X., Wang, Z., Ye, H., Xu, C., Gao, F.: EGO-Planner: An ESDF-Free Gradient-Based Local Planner for Quadrotors. IEEE Robotics and Automation Letters **6**(2), 478–485 (2021). https://doi.org/10.1109/LRA.2020.3047728