Delft Mercurians Team Description Paper RoboCup 2025

The Delft Mercurians

Delft University of Technology
Molengraaffsingel 29, 2629 JD Delft, Netherlands
contact@delftmercurians.nl
https://delftmercurians.nl/

Abstract. This paper goes over the progress the Delft Mercurians team has made in the past year in terms of robot design and development, to compete in RoboCup Small Sized League division B. The paper presents the hardware, embedded-electrical, and software aspects of the robot as well as the research done into smart strategy-making. The future plan for the team is also described.

1 Introduction

Delft Mercurians is a multidisciplinary RoboCup Small Size League team based in Delft, the Netherlands, which debuted in Robocup 2024 and now aims to participate in Robocup 2025 in division B of the Small Size League [1] [2]. The team was founded in September 2022 by members of the Robotics Students Association (RSA) and is made up of robotics enthusiasts and students of the Technical University of Delft. Currently, the team consists of thirty part time members divided into four departments: Hardware, Embectrical[1], Software and Magic[2]. This paper will outline the integral components that each department has worked on, with an emphasis on describing their innovations.

[1] Embectrical is a concatenation of the words embedded and electrical. This term was adopted from the Project MARCH Dream Team based in Delft.

[2] The magic department focuses on applying machine-learning to perform smart and adaptive control.

2 Hardware

2.1 Wheels

The previous design of the wheels were found to be hard to maintain during the competition due to the difficulty with disassembling the wheel and replacing the sub-wheels.

The old wheel design that is suitable for commercial locking assembly for a 4mm shaft would cost a minimum of 1500 euros. Thus, the wheel and sub-wheel attachment was redesigned as seen in Figure 1.

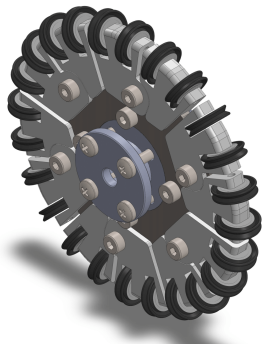
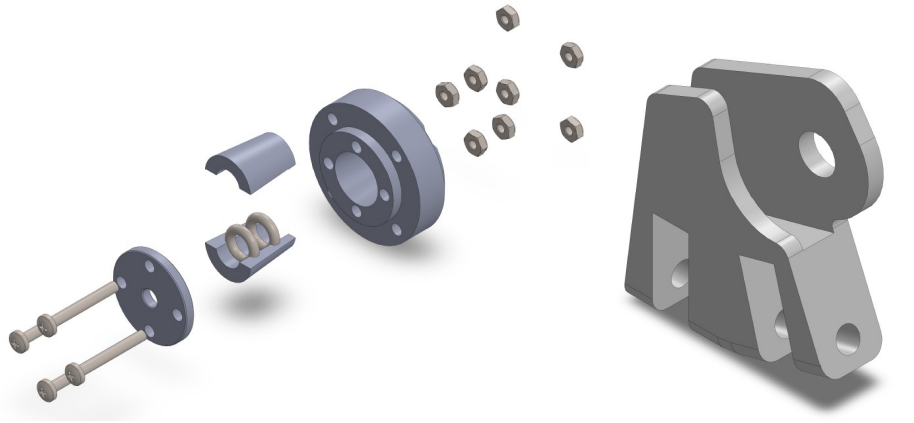


Fig. 1: Wheel Assembly

Figure 1 illustrates the whole wheel contains 6 sub-sections, with each 4 sub-wheels. This design allows for the removal of portions of sub-wheels faster without disassembling the whole wheel. 6 sub-sections makes it so that there are not too many loose parts to assemble, and less good subwheels wasted when a section needs to be replaced. As needed, the sub-sections are connected each using a single nut and bolt, with the whole wheel being connected to the shaft using a 3D-printed locking assembly.

The locking assembly illustrated in figure 2a grabs the shaft using two O-rings, which are squished using the movement of the angled surfaces in the assembly by the force of the bolt. Worded differently, the larger of the two parts moves to the screw plate and pushes the two semi-cylindrical parts into the shaft.

The 6 sub-sections of the wheels, as seen in figure 2b, are made of two identical parts each, reducing the assembly and repair complexity. Each sub-section has four independent sub-wheels that are connected to the part in figure 2b using 2 mm rods.



(a) Locking Assembly nuts and bolts (b) Locking Assembly centerpiece

Fig. 2: Locking Assembly

The wheels were mounted to the robot to validate the mechanical boundaries as defined in the competition rules remain respected. They still need to be manufactured and tested, to see how well they drive.

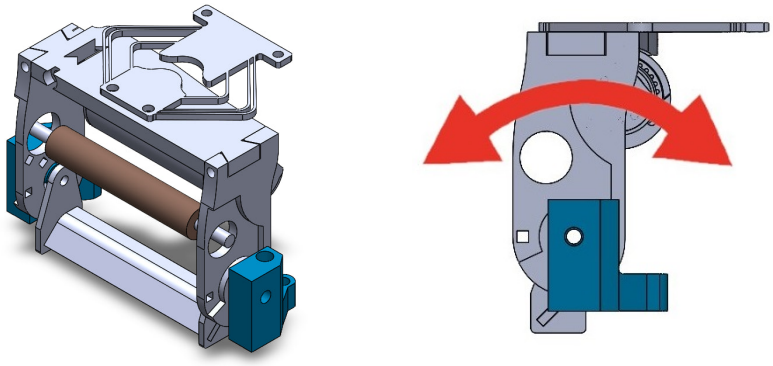
2.2 Chipper and Dribbler Assembly

For this year's competition, a full redesign of the dribbler assembly was done, to both improve the overall dribbling capabilities and accommodate the addition of the chipper. The dribbler and chipper assembly can be seen in Figure 3a.

This new design features a single connection point between the moving section of the dribbler assembly and the mounts on the base-plate. This design choice establishes a well-defined rotation point which allows the dribbler to rotate freely around that pivot point. This is visualized in Figure 3b

This design intentionally constrains the dribbler assembly to a single degree of freedom. This means any energy absorbed during impacts or while dribbling is effectively converted to motion around the pivot, and it was decided to make use of a spring to regular the motion and provide the stiffness in this degree of freedom. Said spring is a compliant 3D printed spring. This can be seen in Figure 4 where the deformed and undeformed shape of this spring is shown. This was inspired by the UBC Thunderbots 2023 design [3]. The implementation was significantly altered to create a more robust and less complex system.

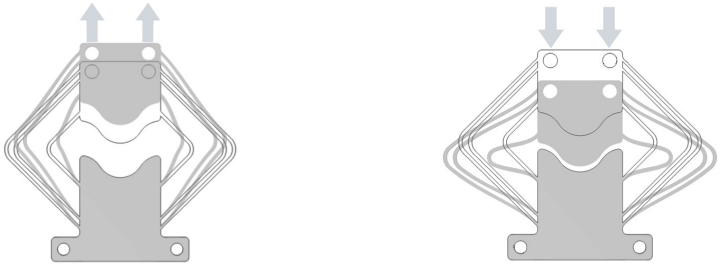
In the conventional approach, the process of getting to the perfect stiffness involves specifying and ordering custom-fabricated springs. This is a process that



(a) An isometric view of the current dribbler design

(b) A side view of the current dribbler design indicating the degree of freedom

Fig. 3: February 2025 dribbler design



(a) In tension

(b) In compression

Fig. 4: Compliant spring deflection example

is both expensive and time-consuming in the case of almost all suppliers. This alone hinders rapid, affordable and feasible prototyping and iterative testing. In contrast by making use of the compliant 3D printed spring it is possible to obtain any desired stiffness in a short time.

When a new stiffness is desired, the compliant/deformable spring parameters such as the length and width of the compliant/deformable parts of the spring along with its thickness are adjusted. This is then followed by a preliminary finite element analysis (FEM) to validate the design. Eventually, the design is printed which takes less than 10 minutes. This allows for rapid prototyping to find the best stiffness and additionally allows the team to customize the stiffness and rebound of the dribbler to different performance requirements in case of different settings. Additionally, the use of 3D printing allows a wide selection of

materials to further tune the springs rigidity and durability. One of the main contenders for alternate materials at the moment is TPU, as this is known for its elasticity. This new design is being manufactured and tested for competition viability.

2.3 Polycarbonate Baseplate

The baseplate used in the 2024 and 2025 robots is double layered and made of polycarbonate, a novelty within the competition. This choice was made to accommodate for the required gap in the baseplate to allow the DF45L024048 motors to drive the wheels directly. The common solution of a single-layered metal baseplate could not be accommodated, due to the lack of CNC capabilities within the team. The double-layered baseplate, which can be found in figure 5, allowed us to laser cut both 2 mm thick layers separately, with the top layer having additional cuts for the motors, the batteries, the fan and the case, as can be seen in figure 5b.

The original design used wood for the baseplate layers. However, the structural integrity and reliability was not sufficient. The most promising alternative found was polycarbonate, which held up extremely well during play tests and the 2024 competition in Eindhoven.

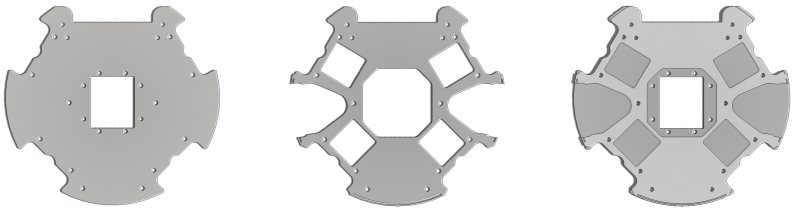
The 2 plates are held together by the screws on the bottom and the assemblies connected to the baseplate on the top, basically sandwiching the plates together. Thus, if the robot is not assembled, the plates are loose. This is beneficial, since the layers can be replaced separately in case of design updates, or damages. There has been nearly zero damage done on the baseplates during the competition, they held be exceptionally well and can be reused for this year's competition.

The polycarbonate does have some downsides. Mainly, the density of the polycarbonate used is $1.21 g/cm^3$, which leads to the center of gravity of the robot being higher compared to most teams that use a metal base plate. The goal is to compensate for this through the use of the fan.

3 Embedded & Electrical

3.1 New motor drivers

The team is still using the same B-G431B-ESC1 motor drivers as last year, however the carrier boards have been redesigned.



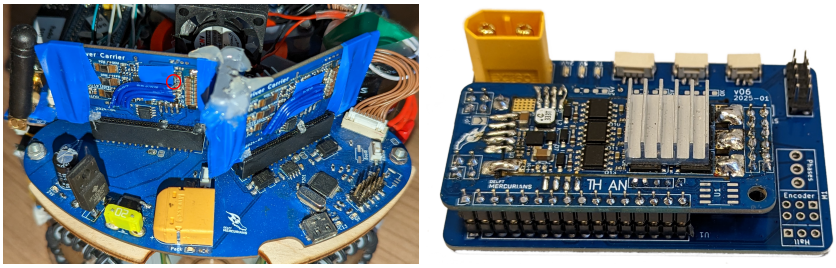
(a) Bottom layer of the base plate

(b) Top layer of the baseplate, with additional cut-outs for the battery packs, wheel motors, fan and the case

(c) Baseplate layers combined

Fig. 5: Double-layered polycarbonate baseplate

The previous robot used PCIe connectors to hold the motor drivers, to allow easy replacement of the motor drivers. However, these connectors were not suitable for a moving robot, the motor drivers would frequently disconnect when the robot accelerated. This issue was mitigated at the competition using hot glue, but it was clear that the carrier boards should be redesigned with a suitable connector.



(a) Old motor drivers, on the power boards

(b) New motor drivers, on the test board

Fig. 6: Old and new motor driver designs

The new boards use standard pin headers and are mounted horizontally instead of vertically, increasing the stability of the motor drivers and also allowing for better airflow and cooling.

An issue was encountered where the motor drivers would get excessively hot even when not using the motors, which was likely caused by the on-board 5V regulator. This issue was solved by removing the regulator on each of the boards (circled in red in Figure 6a), and supplying 5V externally from the power boards.

3.2 Power boards

The so-called power boards contain most of the high power distribution circuitry, and carry the motor driver boards. The motor driver boards are separate, so that they can be swapped out in case of failure.

The power boards also allow for a unique two-battery strategy. To run at full power, the robots need two 3S battery packs to be connected in series. However, the robots can still run off of one 3S battery, enabling hot swapping of the batteries without power interruption. This decreases the time needed to swap the batteries during a match, by removing the need to wait for the robot to start up. It may be even more useful if slow-to-boot subsystems, such as a Raspberry Pi, are added in the future. Figure 7 shows a basic circuit for implementing the dual battery setup. The notable parts are the two buttons (A and B) for switching on the circuit initially, as well as the two optocouplers, which allow a microcontroller to keep the circuits on.

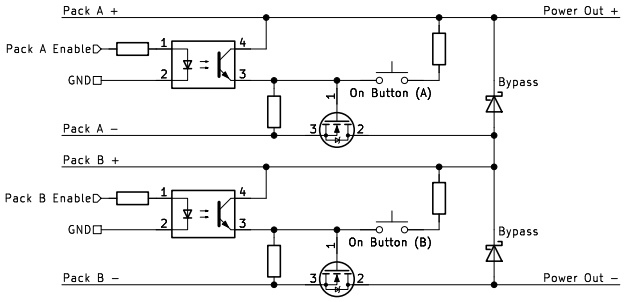


Fig. 7: Dual Pack Circuit (Simplified)

For coordination, there is a small microcontroller, whose primary function is to latch the battery pack switching circuitry on. It also monitors the pack voltages and switches the downforce fan on/off. In theory, the battery pack voltages can be balanced by selectively shutting off the lower voltage battery pack, however this still needs to be implemented.

Finally, the team is experimenting with a strategy to reprogram the motor drivers in place. Using multiplexers, the UART and SWD lines to the motor drivers can individually be addressed, allowing reprogramming and debugging of all motor drivers through a single USB connection. Due to bandwidth limitations, the SWD connection is not stable, however this will be addressed in future revisions of the PCBs.

4 Software

Last year the team developed their own open-source software platform, Dies [3], which provides a framework for AI development. It consists of a core written in Rust, which includes a physics simulator, networking, vision data processing, game state management, and development tools. Strategies, the high-level logic that governs the player's behaviors, are also written in Rust and are run in a separate process, allowing for hot reloading and easy debugging.

Following the experience from last year's competition, it was recognized that large scale improvements were needed in the team's software architecture. The main areas of focus were chosen to be:

- Improving low-level control
- Adopting a more structured approach to strategy design
- Improving the developer experience for common tasks

4.1 Low-level Control

Low-level control is responsible for driving individual robots to position and heading setpoints determined by the higher-level strategy by issuing velocity commands. There were several challenges in this aspect. Last year's robots were prone to overshooting position setpoints, oscillating around heading setpoints and were generally inaccurate, especially when it came to maneuvers like passing that required precision.

Position Control For position control, a minimum-time path (MTP) controller was implemented, which operates in three distinct modes. For longer distances, the controller first enters an acceleration phase, commanding maximum acceleration until reaching either the maximum allowed velocity or transitioning to the next phase. See Figure 8 for an illustration.

One issue is that this rapid acceleration can cause the wheels to slip – this is being addressed by implementing jerk limits on the embedded controller.

The controller includes a "carefulness" parameter that can be tuned to trade off between aggressive and conservative behavior. When in proportional mode, the deceleration rate is amplified compared to acceleration to account for inertia, and this amplification increases with the carefulness parameter:

$$a_{\text{decel}} = 5(1 + k_{\text{care}})a_{\text{accel}}.$$

[3] https://github.com/DelftMercurians/Dies

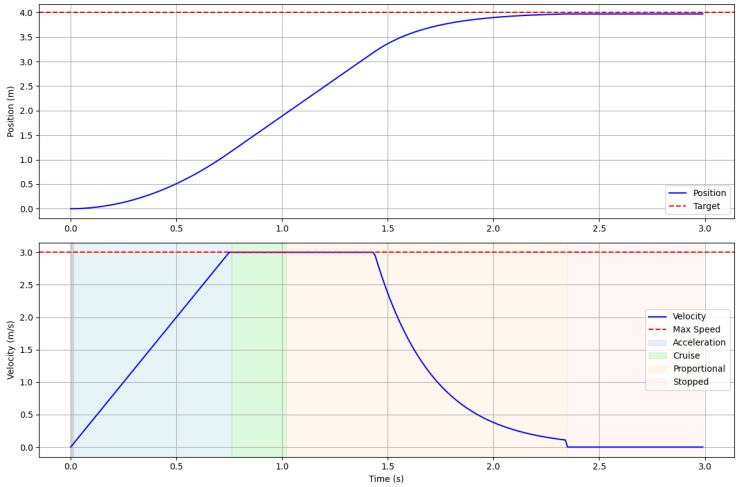


Fig. 8: The bottom plot shows the distinct control phases: acceleration (blue region) where velocity increases linearly, a brief cruise period (green region) at maximum velocity, followed by the proportional control phase (red region) for careful deceleration, and finally the stopped state when reaching the target. The top plot shows the resulting smooth position trajectory reaching the 4-meter target position.

This helps prevent overshooting while still allowing for rapid approach to the target. The revised design has significantly improved the controller's positioning accuracy compared to last year's implementation.

Heading Control The heading control system employs a dual approach. The primary system utilizes onboard PI controllers that leverage high-frequency IMU feedback, receiving periodic absolute heading references from the central system. As a fallback mechanism, a server-side PI controller was maintain which can take over if the IMU fails, though this alternative exhibits lower accuracy due to communication delays.

Time Delay Compensation A significant challenge in control accuracy is vision system delay. This is addressed using Kalman filters for all tracking, with their internal models compensating for time delays. The current state estimate is given by $\hat{x} = x_{\text{meas}} + \hat{\dot{x}} \cdot \tau$, where τ is the dynamically calculated time delay based on vision system timestamps, x_{meas} is the measured state, and $\hat{\dot{x}}$ is the velocity estimate.

While more advanced approaches like Model Predictive Control (MPC) were considered, it was decided that the implementation complexity outweighs the

potential benefits. However, there is plan to experiment with MPC in future iterations as the system matures.

4.2 Strategy Design

For the strategy layer, the team has transitioned to a behavior tree architecture that provides a structured approach to robot decision-making and coordination. Behavior trees allow us to compose complex behaviors from simple, reusable components while maintaining clarity and debuggability.

The core of the team's strategy system is built around the concept of "situations" - composable conditions that represent a robot's understanding of the game state from its perspective. These situations help answer questions like whether a robot has a clear shot on goal, if there are teammates available for a pass, or if there is a defensive emergency requiring attention.

The team's implementation evaluates behavior trees on each vision frame update, with nodes returning one of three states: Success, Failure, or Running. This polling-based approach allows for reactive behavior adaptation as game conditions change. For example, a node executing a passing behavior might return Running while the pass is in progress, Success when completed, or Failure if the passing lane becomes blocked.

Situations can be composed using logical operators to create more complex conditions. For instance, a "good scoring opportunity" might be composed from simpler situations:

$$\text{good\_shot} = \text{has\_ball} \wedge \text{clear\_shot\_to\_goal} \wedge \neg\text{heavily\_defended}$$

The behavior trees consist of several types of nodes that can be composed to create sophisticated behaviors:

- Select nodes try actions in priority order until one succeeds
- Sequence nodes execute a series of actions in order
- Action nodes perform the actual robot behaviors
- Guard nodes gate execution of subtrees based on situations

Active nodes in the tree can maintain state between ticks, enabling behaviors that develop over time. For example, a passing sequence might track its progress through multiple stages: preparation, execution, and confirmation. Each node can also register debug visualizations, making it clear which branches of the tree are active and why certain decisions are being made.

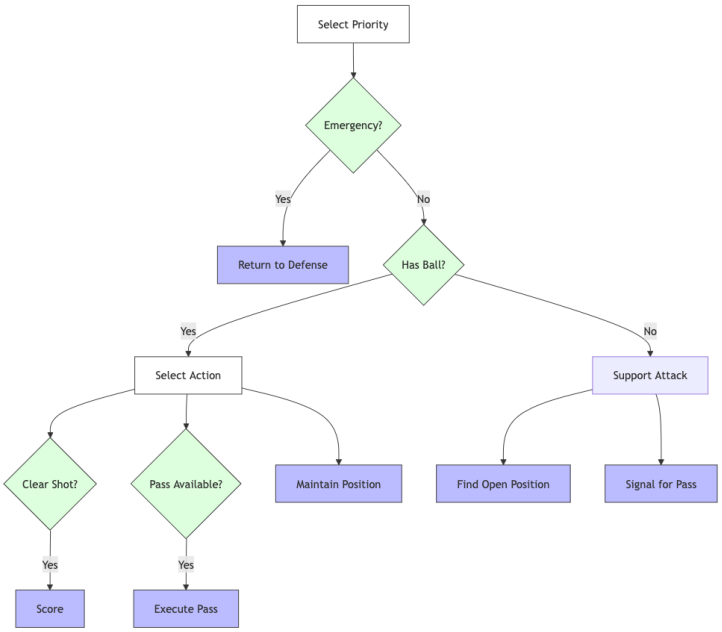


Fig. 9: Example of a simplified behavior tree for an attacking role. Select nodes (white) choose between alternative behaviors based on conditions (green), which gate the execution of actions (blue). The tree evaluates conditions in priority order, running emergency responses first, then attempting to score or pass if possible, and falling back to support positioning.

Team coordination is achieved through a combination of explicit and implicit mechanisms. Robots can broadcast their intentions (like preparing to pass or seeking a pass) through a shared world model, allowing for loose coordination without tight coupling. For more structured plays, robots can explicitly coordinate through a play system that defines roles and interactions, though the exact formulation of this system is still under development.

The behavior tree framework as seen in 9 allows for composition at multiple levels. Individual trees can be composed into larger trees, promoting code reuse. For example, a midfielder role might combine the passing behavior from the attacker role with additional defensive responsibilities:

$$\text{midfielder} = \text{select}(\text{defend\_if\_emergency}, \text{pass\_if\_possible}, \text{maintain\_formation})$$

The strategy system integrates closely with the visualization tools, allowing developers to see the active branches of behavior trees, the conditions being evaluated, and the reasoning behind robot decisions. Each situation can include visualization elements like ranges, vectors, or text overlays that help explain the

robot's decision-making process. This visibility is invaluable for strategy development and debugging.

4.3 Developer Experience and Debugging Tooling

Developer experience (DX) has emerged as a critical focus for the team. As a partially distributed group with members having limited availability due to academic and professional commitments, reducing friction in the development process directly impacts the team's ability to retain contributors and maintain continuous progress. Good DX and robust simulation capabilities enable remote development without requiring physical access to robots or field space – a significant advantage for the team's distributed structure.

Rich Debugging Interface The new web-based UI incorporates comprehensive debugging tools that have dramatically improved development workflow. The interface supports real-time visualization of robot states, planned paths, and behavior tree decisions through an overlay system that can render geometric primitives directly on the field view. A flexible logging system allows developers to track and plot arbitrary values over time.

A particularly powerful feature is the built-in plotter with its custom scripting language. This tool enables developers to create complex visualizations and analysis of robot behavior on the fly, without requiring code changes or system restarts. The scripting language supports mathematical operations, filtering, and aggregation of any logged values, making it invaluable for tuning controllers and debugging complex behaviors.

Custom Simulation Environment While several established simulators exist in the SSL community, the team made the strategic decision to develop their own simulation environment. This choice was driven by several key factors:

1. Improved sim-to-real fidelity through precise modeling of specific robot characteristics
2. Seamless integration with the team's codebase, eliminating external dependencies and reducing setup complexity
3. The ability to run faster than real-time, enabling rapid testing and validation cycles

The simulator achieves up to 10x real-time speed while maintaining physics accuracy, allowing developers to quickly iterate on strategies and control algorithms.

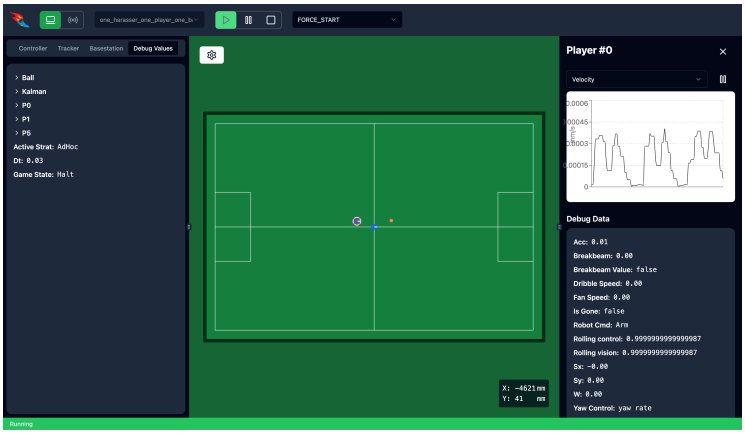


Fig. 10: The debug interface showing real-time visualization of robot states, and paths. The built-in plotter (right) uses a custom scripting language for flexible data analysis.

Automated Testing Framework The team is still in the early stages of developing a comprehensive automated testing framework. The framework leverages the simulator's faster-than-realtime capabilities to run extensive test suites that validate both low-level controls and high-level strategies. Tests can be defined using a declarative syntax that specifies initial conditions, expected behaviors, and success criteria.

The combination of these tools – rich debugging interface, custom simulator, and automated testing – has significantly improved development efficiency. Team members can now contribute meaningfully without requiring physical access to robots, and the time from concept to validated implementation has been dramatically reduced.

5 Magic

A defining feature of strategies implemented within RoboCup SSL competitions is their basis in classical control. When talking with other teams and examining their code, it was discovered that high-level strategies use finite state machines (FSMs) to specify the behavior of the robots in each particular field configuration. FSMs can represent strategies of arbitrary complexities, and debugging tools are abundant. However, the FSM-based strategies are problematic in the long term. The main issue lies in the accumulating cost of supporting large state machines. As the number of possible states becomes substantial, the interdependence of robot behaviors makes debugging harder and discourages developers from improving the strategy. The solution to this problem is to replace the complex conditionals in state transitions with data-driven methods. An example

would be evaluating which robot should be passed to based on the self-play history and not the set of human-defined rules.

Magic department aims to alleviate FSM issues and ultimately incorporate data-driven decision-making into the strategy.

5.1 Simulator

One of the necessary features of implementing a successful data-driven algorithm is the ability to self-play. In most modern supervised learning, one gets a high-quality dataset with labeled data; in this case, such a dataset is not available. The recordings of past games are available, but most of the games are of low quality, and overall, the amount of data is too small to be naively useful.

Thus, the team is interested in implementing self-play. Since the team is *severely* compute-limited, a simulator that runs on a GPU is required [4]. While there are several available industry solutions [5] [6], there is not a single simulator that would natively support 2D objects.

Since JAX [7] is preferred as the main machine learning framework, it was decided to implement the simulator similarly to Brax [8]. The reason Brax is not applicable in its raw state is the imperfect handling of multiple body systems, leading to a quadratic compilation time. In preliminary experiments, Brax was compiled for more than an hour to run a simple policy on 12 robots. Thus, the frameworkw as abandoned. Instead, a solution with lower complexity, simpler physics, and faster compilation times was implemented.

The name of the simulator is cotix [9], not yet in a production-ready state.

5.2 Algorithms

Under the assumption of an existing simulator, algorithms now have to be designed. As was already mentioned, JAX [7] is preferred over alternatives, thanks to the compute provided by Google TPU Research Grant and overall better suitability for reinforcement learning. While very little implementation has been done at present, the aim is to have A0C [10] implemented before the competition. Preliminary experiments with Proximal Policy Optimization [11] show some promise, but the results are generally insufficient to be used within main framework. The results are well aligned with the previous attempts done by [12].

6 Conclusion & Future Work

In the past year the team has revised every sub system from the experience in the last year's competition, and made great improvements in their design. The team has also implemented some of the changes, with more working in progress.

In hardware, a new wheel design was made to be modular and easier to maintain during the competition. A new assembly method was also designed to reduce cost and improve ease of assembly. The new design is extremely promising but it still has to be manufactured and tested. The overall dribbler assembly was also redone to be more reliable and also accommodate for the chipper. It should also comply to the competition regulation better. Lastly, a new material was used for the baseplate, which has proved to be extremely strong and reliable, so it will be used again for this year's competition with an updated design to accommodate the hardware changes.

In terms of electronics, the motor drivers were updated to solve the overheating problem. The carrier boards were also redesigned to be much more stable and not lose contact with the power board when the robot is running. Furthermore, the power board is also being redesigned to support reprogramming and debugging all the motor drivers at the same time. In addition, there were some issues with the battery connectors in last year's competition, so they will also be redesigned to improve ease of use.

For software, a ton of improvements were done in framework to improve the strategies, developer experience and speed, and overall stability of the system. A more structured approach to strategies was established, which allows for more complex yet comprehensible strategies to be made. Low level control was improve greatly to improve position and heading control flexibility and accuracy, and to compensate for time delays. Debugging, UI, and other visualization tools were implemented to improve the overall devleoper experience to write better code and expose issues better. Furthermore, a custom simulation environment was implemented to test strategies and controls faster, with automated testing in the works to speed the testing process even more. This also improves the sim-to-real fidelity.

Finally, in the Magic department great progress was made in the lightweight simulator to enable training reinforcement learning models. The aim for the algorithm was also established.

The team has done a ton of improvements on all aspects of the robot, they are positive that these improvements would bring a lot of value to how the robots play and would bring an edge to them during the competition.

7 Acknowledgements

The team would like to thank the TU Delft Robotics Institute, Cognitive Robotics department of Mechanical Engineering of TU Delft, as well as the FAST grant for supporting the team's activities in the year of 2025.

Further thanks to all the members that contributed to the team: Thomas Hettasch, Tim Verburg, Zhengyang Lu, Alexander Nitters, Thijs Houben, Zhengyang Lu, Nianlei Zhang, George Sotirchos, Leila Hashemi, Irs van de Vijver, Evan Freeman, Kevin Do Cao, Martijn Boonen, Mohammad Kian Ebrahimi, Ece Sinanoglu, Mahdi Mârroufi, Balint Magyar, Guillem Ribes Espurz, Teodor Neagoe, Yohan Le Gars, Renyi Yang, Ivan Lopez Broceno, Yousef El Bakri, Vilohit Kaza, Roman Knyazhitskiy, Martino Manaresi, Sasan Salmani Pour Avval, Junxiang Qi, Joona Rengers, Jasper Brink, Sandro Karhula, Giannis Pantidis.

References

1. "Delft Mercurians TDP 2024," [Online; accessed 5. Feb. 2025]. [Online]. Available: https://delftmercurians.nl/documents/TDP_2024.pdf
2. "Delft Mercurians," [Online; accessed 12. Feb. 2024]. [Online]. Available: https://delftmercurians.nl/about/
3. A. Senthilkumarb, A. Sidhuf, A. Balamuralia, D. Sturnc, D. A. D. Tof, F. Muhstaqb, F. Cremad, H. Bryantb, H. Rovnerg, J. Lewb, K. Wakabad, N. Zareiana, O. Levyf, R. Khanf, R. Caod, R. Nedjabatb, T. Kongb, S. Ajmald, S. Lye, and Y. Zhou, "Ubc thunderbots 2023 team description paper."
4. J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox, "Gpu-accelerated robotic simulation for distributed reinforcement learning," in *Conference on Robot Learning (CoRL)*, 2018. [Online]. Available: https://arxiv.org/abs/1810.05762
5. NVIDIA, "Isaac sim: Nvidia's robotics simulation platform," https://developer.nvidia.com/isaac-sim, 2023, accessed: 2025-02-09.
6. E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 5026–5033.
7. J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax
8. E. Frey, A. Raichuk, S. Girgin, O. Bachem, S. Bohez, O. Pietquin, M. Geist, and D. Duckworth, "Brax: A differentiable physics engine for large scale rigid body simulation," *arXiv preprint arXiv:2106.13281*, 2021.
9. R. Knyazhitskiy, "cotix: continuous-time differentiable simulator in jax," 2025. [Online]. Available: https://github.com/DelftMercurians/cotix
10. T. M. Moerland, J. Broekens, A. Plaat, and C. M. Jonker, "A0c: Alpha zero in continuous action space," 2018.

11. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
12. F. B. Martins, M. G. Machado, H. F. Bassani, P. H. M. Braga, and E. S. Barros, "rsoccer: A framework for studying reinforcement learning in small and very small size robot soccer," 2021. [Online]. Available: https://arxiv.org/abs/2106.12895