RobôCIn Extended Team Description Paper for RoboCup 2023

Aline Oliveira, Cauê Gomes, Cecília Silva, Charles Alves, Danilo Souza, Driele Xavier, Edgleyson Silva, Felipe Martins, Lucas Cavalcanti, Lucas Maciel, Matheus Paixão, Matheus Vasconcelos, Matheus Vinícius, João G. Melo, João P. Moura, José R. Silva, José V. Cruz, Pedro H. Santana, Pedro P. Oliveira, Riei Rodrigues, Roberto Fernandes, Ryan Morais, Tamara Teobaldo, Washington Silva, and Edna Barros

Centro de Informática, Universidade Federal de Pernambuco.
Av. Prof. Moraes Rego, 1235 - Cidade Universitária, Recife - Pernambuco, Brazil.
robocin@cin.ufpe.br
https://robocin.com.br/

Abstract. RobôCIn has participated in RoboCup Small Size League since 2019, won its first world title in 2022 (Division B), and is currently a three-times Latin-American champion. This paper presents our improvements to defend the Small Size League (SSL) division B title in RoboCup 2023 in Bordeaux, France. This paper aims to share some of the academic research that our team developed over the past year. Our team has successfully published 2 articles related to SSL at two high-impact conferences: the 25th RoboCup International Symposium and the 19th IEEE Latin American Robotics Symposium (LARS 2022). Over the last year, we have been continuously migrating from our past codebase to Unification. We will describe the new architecture implemented and some points of software and AI refactoring. In addition, we discuss the process of integrating machined components into the mechanical system, our development for participating in the vision blackout challenge last year and what we are preparing for this year.

Keywords: RobôCIn · RoboCup 2023 · Robotics · Small Size League

1 Hardware

The hardware updates for this year aim at more a reliable motion control, by improving our mechanics project. We have added a brass thread to our aluminium drive transmission support, preventing it from wearing out due to the shaft's friction. Besides the hardware adaptations for the Vision Blackout challenge, which are detailed in Subsection 2.1, our electronics project has remained unchanged, and general hardware specifications can be found in Table 1, with no changes from the 2020 version.

Table 1. Robot Specifications

Robot Version	**v2022**
Driving motors	Maxon EC-45 flat - 50W
Max % ball coverage	19.55%
Microcontroller	STM32F767ZI
Gear Transmission	18 : 60
Gear Type	External Spur
Wheel	3D Printed
Total Weight	2.53 kg
Dribbling motor	Maxon EC-max 22, 25W
Encoder	*MILE 1024 CPT*
Dribbling Gear	50 : 30
Dribbling bar diameter	14mm
Max. kick speed	6.5m/s
Communication Link	nRF24L01+
Battery	LiPo 2200mah 4S 35C

1.1 Drive Transmission Support

To achieve our goal of competing in Division A in the following years, we need to improve our Drive set to minimize the risks on the robot's movement and make the transmission system smoother, by adding an machined aluminium drive transmission support (Figure 1a) to ensure the transmission gears' tolerance. During the competition, some of the Drive sets presented looseness on the wheel shaft, and we partially solved the problem with Tekbond 793.

After the competition, we conducted a material analysis to find the root of this problem. One of the reasons for the gap was that the hardness of our stainless steel wheel shaft was much higher than the hardness of the aluminium drive transmission support. Thus, the forces applied to this structure caused our drive transmission support to wear out.

One solution found to reduce wear in this part was adding a brass thread (Figure 1b), which has a greater hardness than the aluminium from the previous model, equalizing the contact forces.

We also had problems with tolerance when modifying the drive transmission support to add the brass thread. Figure 2 shows the consequences of this inaccuracy in the hole, causing backslash problems for the gearing.

These problems were solved by fabricating new pieces of the drive transmission supports in partnership with the Departamento de Física (DF), which supported us with high-precision machines, guaranteeing reliability for our robots.

(a) (b)

Fig. 1. (a) Old Drive Transmission Support; (b) New Drive Transmission Support;

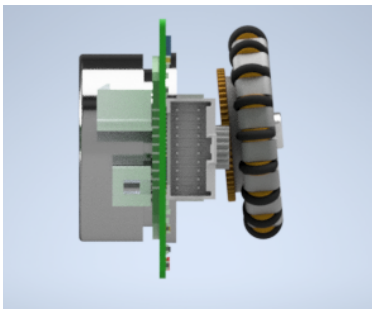


Fig. 2. Uncentric Brass inserted thread

2 Vision Blackout Challenge

For participating in the Vision Blackout Challenge, hardware adaptations were made, new low-level navigation methods were implemented and a complete software infrastructure was built for allowing our robots to execute SSL soccer skills autonomously. Our goal was to create a robust enough infrastructure to implement each robot skills by only creating new Finite State Machines (FSM), all using onboard modules for sensing and processing. With this architecture, we were able to complete 2 of the 4 stages of the challenge in 2022's competition, achieving 2nd place. Also, we shared details of our research on recent papers [6, 11, 12] and open-source project datasets and documentation[1,2].

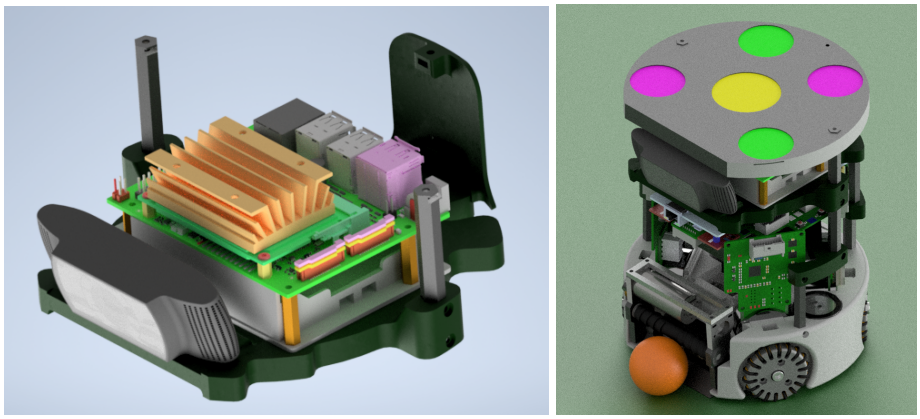
2.1 Hardware Adaptations

Following past approaches in the League, we have added an onboard camera and a compute module for vision processing and decision-making. A Logitech

[1] https://github.com/bebetocf/ssl-dataset
[2] https://github.com/jgocm/ssl-detector

C922 webcam was chosen due to its low-distortion parameters, allowing for easy camera calibration with high precision. As for additional computation, a 4GB NVIDIA Jetson Nano Developer Kit was chosen due to its small size, low power consumption, high throughput on DNN and image processing, and extensive documentation for NVIDIA libraries. Also, its System-on-Module (SoM) architecture leaves room for future improvements, by adapting our electronics to connect the module directly to our mainboard, for instance, saving even more space.

Besides the Jetson Nano and the Logitech camera, we have also added a power supply module, using 4 cells of 18650 batteries, for powering this new subsystem. A new cover plate, which we call the robot's third floor, was designed for mounting those parts onto the robot and it is shown in Figure 3. It also has housings for additional standoffs, enabling us to place a SSL tag on the robot's top, which is useful for experiments and evaluation.



(a) Third floor with Jetson Nano, power supply module and onboard camera.

(b) Vision blackout robot assembled

Fig. 3. Robot hardware adaptations for vision blackout challenge

2.2 Software Workflow

The autonomous SSL robot is mainly operated by two processing modules: the Jetson Nano and the STM32F746ZI, an ARM Cortex-M7 Microcontroller Unit (MCU), also referred to as STM32F7 for simplicity. They communicate through an Ethernet cable using User Datagram Protocol (UDP) Socket packets.

For embedded vision, we use a Logitech C922 camera with 30 frames per second capture rate using 640x480 pixels of resolution. Vision frames are processed by the Jetson Nano running a CNN-based Object Detection model, namely SSDLite MobileNetv2 [9,19], for detecting SSL objects' bounding boxes, which are

used for estimating their relative positions to the robot by using pre-calibrated intrinsic and extrinsic camera parameters, as presented in [11]. The paper also shares details of model retraining and deployment using TensorRT optimizations.

Decision-making is also implemented on the Jetson Nano, which runs Finite State Machines (FSMs) that implement each of the robot's autonomous skills for solving Vision Blackout challenge stages. Objects' relative positions are used as inputs and the FSM computes a target position and orientation, a navigation type, and command flags such as odometry resetting, capacitor charging, and kicking. This information is encoded into a protobuf message and sent to the MCU through the UDP connection.

At the MCU level, for our Target-Point-based navigation, we implement three movement types: Rotate-on-Self (RoS), Drive-to-Point (DtP), and Rotate-in-Point (RiP). The first accounts for rotations around the robot's axis, mainly used for initial ball searching and self-alignment with targets. DtP implements a linear movement with orientation correction, adjusting the robot's translation velocity according to its rotation error and distance to the target. Lastly, RiP executes a circular trajectory around a point, allowing the robot to search for a goal while looking at the ball, for instance.

The MCU also calculates the robot's inertial odometry by computing inverse kinematics from encoder readings, and the trajectory is estimated using gyroscope measurements combined with odometry, allowing it to adjust its path while embedded vision information is not available. Figure 4 illustrates an overview of the proposed architecture, and we present more details in [12].

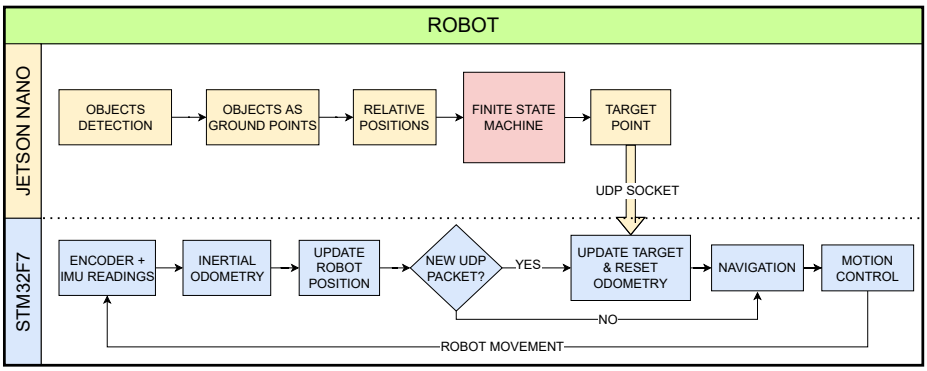


Fig. 4. Overview of the proposed logic diagram in order to build an autonomous RoboCup Small Size League Robot. All the modules are inside the robot, the upper modules run on the Jetson Nano, while the lower ones run on the STM32F7 MCU. Finite State Machines are defined for implementing soccer skills.

2.3 Robot Skills

This architecture was employed to solve the 4 stages of Vision Blackout challenge 2022. However only stages 1 and 2 were fully completed during the competition, showing our solutions were still not robust enough and highlighting many difficulties and necessary improvements, which we discuss in the next subsection.

In more recent experiments [12], for evaluating our system's capabilities and weaknesses, we have executed multiple tries on different scenarios of three common SSL tasks: grabbing a ball (I), scoring on an empty goal (II), and passing the ball (III). The same rules and scoring criteria as the 2022 Vision Blackout challenge [15] were applied in the tasks, except for passing the ball, which excludes scores from the kicker robot from the challenge's stage 4. Also, we consider that the robot has succeeded in the task if conditions for all positive scores are satisfied.

Table 2 shares an overview of the experiments' overall results from [12], showing that the robot was able to stop with the ball touching its dribbler and score a goal in 80% of the attempts on tasks 1 and 2. As for the third task, the ball hit the receiver robot's dribbler on 46.7% of the 15 attempts, although the robot was hit in 80% of them.

Table 2. Autonomous SSL Robot's Overall Performances on Proposed Tasks

Metrics	Task I	Task II	Task III
Min Time (s)	6.09	11.89	9.82
Max Time (s)	10.27	60.00	20.00
Mean Time (s)	7.70	19.01	14.21
Success Rate	12/15	12/15	7/15
Total Score	40/45	54/60	47/60
Penalties	-	8	3

2.4 Major Issues and Ongoing Improvements

Issues from RoboCup One major difficulty we faced at the 2022 Vision Blackout challenge was to detect the ball at high distances, since our object detection approach was only capable of detecting it for up to 5 meters, which led us to failures at 2 of the 3 tries on stages 1 and 2. Also, our self-localization methods were not robust enough for solving stage 3, resulting in low scores and long execution times. As for stage 4, even though the passer robot was able to detect the ball, the kicker one could not move due to communication issues, leading to 3 failures.

Issues from Evaluation Experiments During experiments from [12], which results are reported in Table 2, analysis from embedded vision logs have shown

that most failures were caused by false positive detections from objects outside the field, highlighting the importance of discarding out-of-field information. Also, many penalties were caused due to the robot's inability of detecting field lines, and ball searching was the most time-consuming part of the tasks.

Ongoing Improvements For discarding out-of-field information, we have been developing field boundary detection solutions, which also enable more complex exploration strategies for objects' searching, as we can avoid leaving the field, being also a useful feature for overcoming our major issue from RoboCup: not finding the ball.

Introducing a self-localization solution for SSL robots is also a necessary improvement. It enables planning more efficient paths and avoiding penalties, such as entering the defender's area. In addition, objects' searching, which was shown to be the most time-consuming part of the tasks, could be optimized using localization knowledge for more efficient field exploration. Thus, we are working on a Monte Carlo Localization (MCL) algorithm that fuses our inertial odometry approach with vision information from detected goals and field boundaries relative positions for regressing the robot's pose over time, based on typical approaches from other RoboCup leagues [8, 18].

3 Software

After the team's first participation in RoboCup, in 2019, it was decided to concentrate a good part of the efforts on building a similar codebase that could be reused by other RobôCIn's soccer modalities, other than SSL: Simulation 2D [16] , IEEE Very Small Size Soccer [7]. With the absence of in-person competitions in 2020 and 2021 due to the pandemic, we focused our efforts on improving the software codebase, which we mentioned in the 2022 TDP [20].

One of the main existing problems occurred due to the logic replication and the low interchangeability of developers between categories, even working with the same programming language, C++. Also, with the expansion of SSL-Coach functionalities, the authorial software described in the TDP of 2019 [21], the accumulation of technical debts in the architecture and development infrastructure made it difficult to make improvements, for example, the creation of more elaborate and collaborative plays among the robots.

SSL-Coach is our first stable software version developed for the category, which has a modular architecture, inspired by STP (Skill, Tactics, and Plays) [2]. Due to the variety of demands and a short development period, the information processing steps of this software are characterized by being tightly coupled and difficult to enlarge, which makes it difficult to add new information flows, and requires changes to code snippets not necessarily related to the intended change. Therefore, it was necessary to reduce the complexity of the software to reduce execution errors, that led to points of failure in the initial software architecture. Also, it has been noted as difficult to integrate new team members, transfer knowledge, and renew the team using the existing architecture.

With a survey of the main technical debts to be solved within RoboCIn's categories, a more flexible and modern architecture was modelled, bringing different possibilities of expansion and reuse. In this way, soccer-common was developed, an open-source library, used as a submodule, which aims to concentrate the common global part of the different team modalities, in addition to providing a separation between User Interface (UI) and back-end. Soccer-common has as its main components a library of geometric functions, a graphical interface with drawing support at any point, debugs, and the design of a module.

A module, in the new architecture, consists of the main abstraction capable of executing logic in parallel, providing support for communication with the visualization interface and parameters, and communicating with other modules. Also, a module can be indexed, that is, for occasions where it is intended to have multiple identical execution steps, such as the behavior of a robot. Its conception consists of the main core of our architecture, which we will describe below. From now on, we will call it Unification, the new development base software, which replaced SSL-Coach.

3.1 Architecture

The tight coupling of processing in SSL-Coach is due to how its modules were created and communicate with each other. Each module consists of a thread that executes a singleton, a static global object with a unique instance at any point, and the exchange of information between these modules is done through direct connections by setters and getters, which violates the Single-responsibility principle (SRP) [10], and consequently makes it difficult to change the existing execution flow and call tracking performed.

Since the beginning of software development at RobôCIn, we have used the Qt framework [3]. However, we have hardly explored some of its features, such as Signals & Slots [4], which consists of implementing the Observer Pattern to facilitate communication between components of the Framework. Originally, its use was intended for the communication of objects linked to the UI, avoiding limitations and complications that parallel processing and the implementation of callbacks in C++ may imply.

By combining signals and slots with our projected wrapper for safe shared access, the communication of our modules was implemented as a cascaded publisher-consumer system, where emitter functions are connected in the creation of modules to receivers, confirming each module requirement. As soon as a module receives the input and it is registered, the information is stored in a critical region and waits for the next execution to be effectively consumed.

With the new infrastructure, we have also improved the communication between modules, using more flexible data packages, simplifying modifications and corrections. Previously, the information we shared consisted of an extensive structure, where all the relevant information to be transmitted was present. The distinction of its use was under an enumerator's responsibility. The filling of this structure was not always fully completed, as not all information was relevant to a specific message, which pollutes and makes it difficult to understand.

We solved the problem using a variant type, introduced in C++17 [5], which consists of a type-safe union, capable of aggregating different structures into a single type, enumerating each one of them. It allows us to direct the processing of a message through pattern matching, which simplifies the use of previously needed conditionals.

In this software architecture, the flow of information starts with the vision system that sends position information. Simultaneously, the referee sends stage and command updates. The software then receives these inputs and applies filtering processes. The decision module determines the players' behavior (such as goalkeeper, forward, defender, and others) according to the received referee data, making decisions and using the vision data to identify which player should perform a particular action. The behavior modules then use the decision assignments to execute the intended behavior for each player in separate threads. These threads produce tactics that are processed by the planning and/or navigation modules threads in sequence. Finally, the navigation module process necessary actions for the robots move, with briefed type and parameters. As a result, we achieved the architecture in Figure 5.

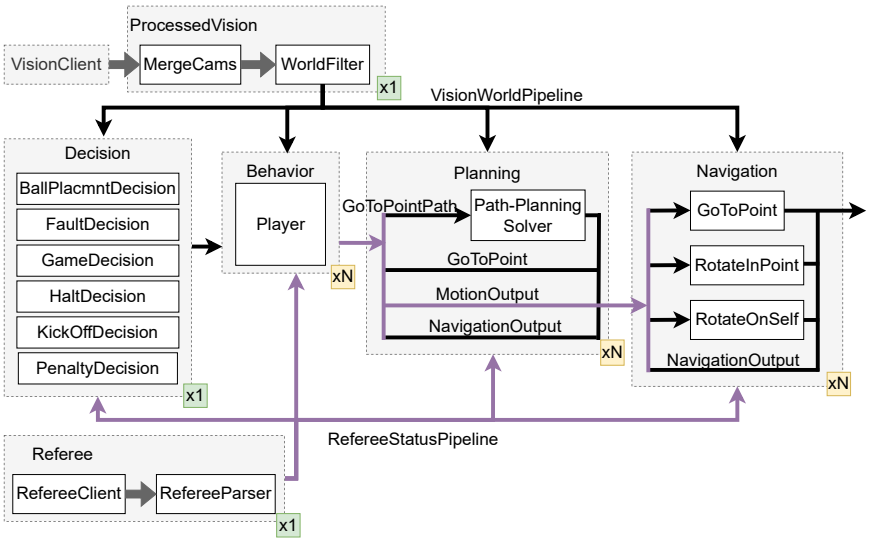


Fig. 5. Overview of SSL-Unification software architecture detailed dataflow. In purple is the data with variant groups type.

3.2 Implementation

We seek to reformulate modules introduced in the 2019 TDP according to the listed technical debts, reducing the number of flags and removing boilerplate code. We will describe the changes made to our code flow below:

DataWorld In SSL-Coach, this component was responsible for receiving vision information obtained from simulators or vision software, performing vision processing and receiving commands from the referee's software. We decided to separate the processing carried out into three modules, each dedicated to one of the respective activities described above.

For the module dedicated to receiving arbitration software commands, we have developed our parser[3] based on the Stage and Command received, allied to the analysis of internal flags, information from the vision and the previous context, aiming at the specialization of game situations, simplifying the strategy carried out later, so each leaf situation at parser tree output, started to be treated in isolation.

The complete referee parser tree, shown in Figure 6, starts from the game's command and the state received from the external referee. At the Game Action division, it decides if the robots must halt, or not. Then, the Game Status transition defines if we are dealing with an in-game situation or a positioning one, such as a preparation for kick-off. Lastly, at the Planning Game division, the parser chooses between states whether the robots must move without touching the ball (Dynamic Formation), execute a predefined play (Planned Tactic), or play the game normally (Game Tactic).

Trainer With the split of the DataWorld component, and the creation of a dedicated module to parse the information received from the arbitration software, the processed commands allowed a strong restructuring in this module, where we currently make quick specific changes for opponents depending on the applied game state.

Over the years we have greatly evolved players' allocation within the decision component, the Trainer, formerly called Decision, starting from a static team with 3 Defenders, 1 Support, and 1 Forward in 2019, to a dynamic allocation based on the position of the ball and risk offered by the enemies' positions. In this way, we are currently able to be an adaptable team, but one that seeks to be extremely offensive by pressing the game to the enemy half, exchanging passes until they get an opportunity to shoot on goal.

Our offensive tactics are made up of a Forward, who is the player in possession or in a direct dispute over the ball, and of a variable number of supporters according to the position of the opposing team's robots, where each supporter seeks to stay in optimal positions, the best within our heuristics to receive a pass from Forward and perform a successful play. Once the ball is passed to the supporter, this will become the player in possession or direct dispute for the ball, switching positions: the supporter receiving the pass will become the Forward, which keeps the attack cycle performed by our team.

Behavior With Unification, one of the team's main goals for the old Player module was to decouple functionalities and simplify state machines. In SSL-Coach, we had behaviors with finite state machines (FSM) of many states and

[3] https://github.com/robocin/soccer-common/wiki/Referee-Parser

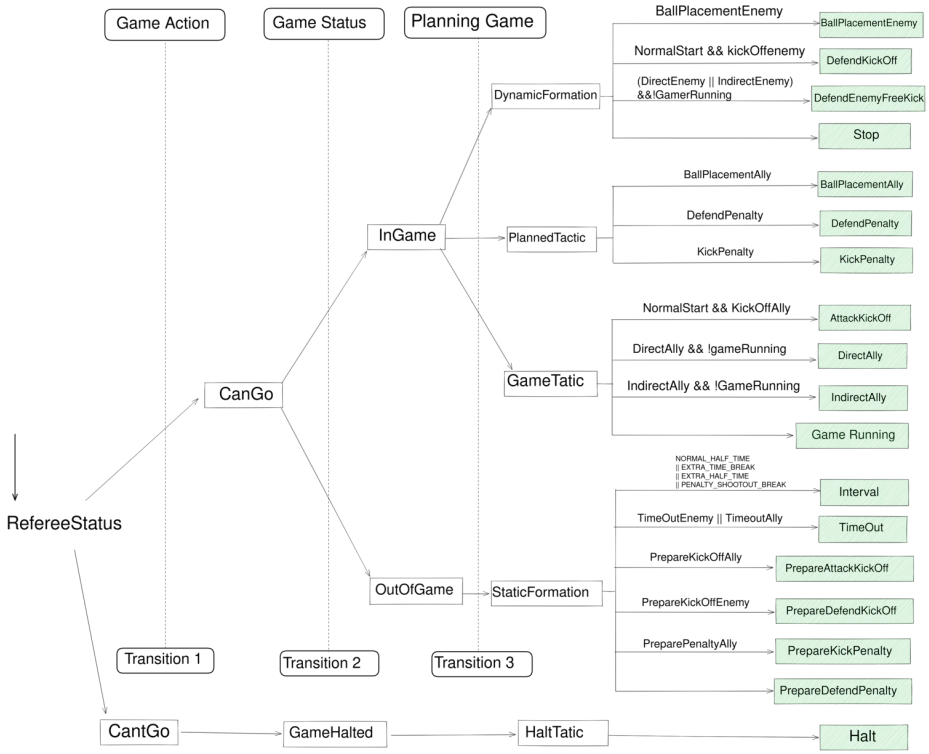


Fig. 6. Complete referee parser tree, showing all possible game states.

with similar logic functions done in several different ways within the behavior itself, which made it difficult to understand the transitions to debug and make corrections.

Previously, as described, each state corresponded to an enumerator, and the state processing nodes, functions, which are incapable of storing contexts, switched by a large number of conditionals. The input for each processing node consisted of a pair $< state, context >$, with the information needed for all existing states, which made it particularly difficult to distinguish information relating only to specific states. With the architecture update, some ways to improve the implementation of an FSM for the desired purposes were also studied. Similar to the messages used for communication between modules, the machine states started to consist of a variant, while the processing nodes of these states became classes. With this change, the state processing nodes now have greater independence, with contexts capable of restarting as a transition to a new state is performed.

Also, in the current architecture, we started to apply the concept of SkillBook coming from the STP [2], and to define the attacker as a set of tactics that involve interacting with the ball (be it kicking to goal, giving a pass or take a

penalty) that was previously all together in a single FSM, making it complex and disproportionately large to deal with various situations.

In order to facilitate maintenance and future improvements, it was decided to extract the previously existing Planning and Navigation modules within the Behavior component, thus enabling the alternation of algorithms used, as we will explain below.

3.3 Path-Planning

One of the changes made to the architecture was the creation of a dedicated module for path planning. After that, we became capable of exploring path optimizations and switching the used algorithm.

This year, we changed our path-planning algorithm and optimized our low-level control. Until then, we used an evolved version of the visibility graph presented on the 2019 TDP [21], due to a bunch of changes realized over the years aimed to optimize and handle corner cases. However, it has become really difficult to maintain it given the increased code complexity.

Current Problems One of our major issues in past competitions was the high number of fouls due to crashing and robot distance to forbidden locations, as shown in Table 3. Due to the yellow cards arising from those fouls, we were frequently forced to play with 5 or 4 players, which reduced massively our offensive power given the reduced number of players to compose the attack. This analysis led us to optimizations on the path-planning algorithm, since the majority of those fouls were avoidable.

Table 3. Collision and invasion detected during matches [22] at RoboCup 2019, 2021 and 2022

Referee foul event	Amount
ATTACKER_TOO_CLOSE_TO_DEFENSE_AREA	22
BOT_CRASH_UNIQUE	93
DEFENDER_TOO_CLOSE_TO_KICK_POINT	69

With limitations concerning Visibility Graph's nature, those related to the generated path stand out. Despite being the shortest euclidean path, it's not time-optimal for omnidirectional robots with an abrupt change in direction and velocity, as presented by Balkon et al. [1]. Furthermore, because the algorithm does not take into account the agent's momentum/direction, which is the whole robot's state with velocity and acceleration rather than solely position, there is a dissonance in its execution between the calculated path and the robot's real trajectory, as shown in Figure 7. Moreover, the available margin for navigation error is minimal due to the generated path being tangential to the obstacles. Hence,

both factors culminate in the high amount of collision and invasion, since the expansion of obstacles' boundaries is not a direct guarantee of decreased collisions in general, besides being a solution that greatly hurts our team performance.

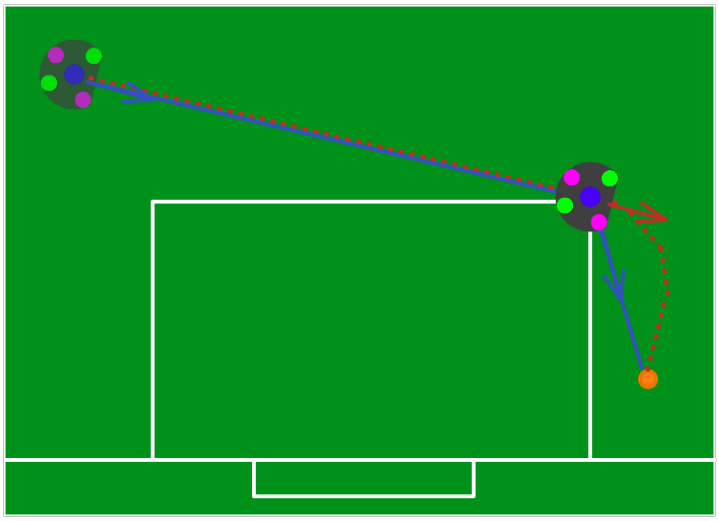


Fig. 7. Dissonance between the planned path in blue and the trajectory executed in red, with emphasis on the forces acting in the change of edge of the graph.

Also, bigger obstacles reinforce some limitations with our implementation, given that as they are solely a set of points connected in the scene's graph, then it is not possible to increase the complexity of the polygons used as obstacles further than triangles and rectangles without harming the execution time. Likewise, we are not able to properly handle the escape from an obstacle, whether it is the start or target position. So, with a more generic solution, we are susceptible to dealing with a lot of corner cases, which result in both bad placements for ball disputes and defense area invasion.

Desired Key Improvements Therefore, we listed the following sought improvements for the new algorithm:

- Fewer number of collisions, allowing better velocity and movement.
- A generated path harmonic to the real robot's trajectory.
- Robust algorithm for real-time and dynamic scenarios as those of SSL.
- Possibility of an obstacle model that appraises the movement's dynamic, considering time as a factor to determine possible collisions.
- Obstacles that can be differentiated from each other for a greater fidelity of representation of the world.
- Possibility of simulating the robot's movement to feed estimate the robot reaching range.

Solutions Adopted Unlike the analyzed options in TDP 2019 [21], this time, we chose to study and compare Sampling and Trajectory based algorithms, which are classes of Path Planning Algorithms that had proven some robustness in terms of Motion Planning for SSL. Despite the popularity of RRT-based algorithms (RRT*, RRT-Connect, ERRT) among SSL teams, we opted for the Bang Bang trajectory [14] given that its traditional implementation already suffices all of our requirements, which has a strong integration between pure planning with a series of points and the proper navigation, since it computes the robot's action velocity along the path.

Bang Bang trajectory-based path-plannings were studied and adopted by reference teams, being reported as important for the achieved results by Tigers [13,23] and Er-Force [17,24]. Both implementations are open-source and demonstrate distinct ways of dealing with the implementation of the algorithm. While Tigers bet on an approach based on selecting intermediate points from a constellation of points around a given origin connected by trajectory segments to the target, the Er-Force goes with a more open search approach seeking through the trajectory time and orientation.

Each implementation has its advantages and drawbacks, and we sought to validate both approaches and their code bases. We converted Tigers' implementation to C++, but the achieved performance and some discrepant behavior to the java version made us adopt Er-Force's base, which was already developed in C++ and had an execution time lower than 1ms. But, we found some situations where the generated path was trajectory-based due to heavy optimizations which reduced the search's number of iterations. Therefore, we merged into the algorithm some of our ideas as well as some from the Tigers' solution at the cost of increased time complexity, but still contained in a time frame.

Correlation with Navigation Path planning and navigation are inherently related. A path planning that doesn't take into account the robot's dynamics causes a big discrepancy in the obtained result. Mainly in the Visibility Graph, which has sharp changes in the velocity and direction on the generated path. Then, from solely an ΔS the navigation needs to predict the output for the robot to fulfill and generate all the movement's state transitions that affect the planning result, but none of this feedback is propagated into the next path planning.

Aiming to close this control loop, trajectory-based algorithms are fed with the robot's current state with its position and velocity. But the software relies on data from the vision system where this current state corresponds to a robot's past state that was captured by the currently received frame. Furthermore, an SSL Robot is capable of changing its velocity in such a way that it is difficult for vision processing to keep up. Thus, vision-based only approaches limit the robot state transition rate.

The limitation of detection of the current state by the vision mainly impacts the ability to control the acceleration and deceleration of the robot when the path is being adjusted throughout the cycles since it cannot reach the expected state.

Seeking to mitigate this effect, we developed methods for estimating the current state of the robot based on the current information in the frame of the vision, the processing delay of the vision and the speed commands sent to the robot during this delay, starting from the state seen in the vision and we apply the commands sent to the robot by replicating their performance time, thus estimating its real current state. Another more effective approach to this problem, eliminating assumptions about the current state of the robot, would be to calculate its current trajectory segment itself, performing embedded navigation, a solution adopted by the Tigers team that we intend to invest in the following years.

4 Acknowledgement

First, we would like to thank our advisors and the Centro de Informática (CIn) - UFPE for all the support and knowledge during these years of project and development. We also would like to thank all our sponsors: CESAR, Microsoft, Veroli, HSBS, Moura, and Mathworks.

References

1. Balkcom, D.J., Kavathekar, P.A., Mason, M.T.: Time-optimal trajectories for an omni-directional vehicle. The International Journal of Robotics Research **25**(10), 985–999 (2006). https://doi.org/10.1177/0278364906069166
2. Browning, B., Bruce, J., Bowling, M., Veloso, M.: Stp: Skills, tactics, and plays for multi-robot control in adversarial environments. Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering **219**(1), 33–52 (2005)
3. Company, Q.: Qt framework, `https://www.qt.io/product/framework`
4. Company, Q.: Qt signals & slots, `https://doc.qt.io/qt-6/signalsandslots.html#advanced-signals-and-slots-usage`
5. Cpp-Reference: std::variant, `https://en.cppreference.com/w/cpp/utility/variant`
6. Fernandes, R., Rodrigues, W.M., Barros, E.: Dataset and benchmarking of real-time embedded object detection for robocup ssl. In: Alami, R., Biswas, J., Cakmak, M., Obst, O. (eds.) RoboCup 2021: Robot World Cup XXIV. pp. 53–64. Springer International Publishing, Cham (2022)
7. IEEE: Very small size soccer, `https://ieeevss.github.io/vss/index.html`
8. Laue, T., Röfer, T.: Particle filter-based state estimation in a competitive and uncertain environment. In: Proceedings of the 6th International Workshop on Embedded Systems. Vaasa, Finland (2007)
9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. CoRR **abs/1512.02325** (2015), `http://arxiv.org/abs/1512.02325`
10. Martin, R.C.: Clean Architecture: A Craftsman's Guide to Software Structure and Design. Prentice Hall Press, USA, 1st edn. (2017)
11. Melo, J.G., Barros, E.: An embedded monocular vision approach for ground-aware objects detection and position estimation (2022). https://doi.org/10.48550/ARXIV.2207.09851, `https://arxiv.org/abs/2207.09851`

12. Melo, J.G., Martins, F., Cavalcanti, L., Fernandes, R., Araújo, V., Joaquim, R., Monteiro, J.G., Barros, E.: Towards an autonomous robocup small size league robot. In: 2022 Latin American Robotics Symposium (LARS), 2022 Brazilian Symposium on Robotics (SBR), and 2022 Workshop on Robotics in Education (WRE). pp. 1–6 (2022). https://doi.org/10.1109/LARS/SBR/WRE56824.2022.9996004
13. Ommer, N., Ryll, A., Geiger, M.: Tigers mannheim (team interacting and game evolving robots) extended team description for robocup 2019 (2019), roboCup Small Size League, Mannheim, Germany, 2019
14. Purwin, O., D'Andrea, R.: Trajectory generation for four wheeled omnidirectional vehicles. In: Proceedings of the 2005, American Control Conference, 2005. pp. 4979–4984 vol. 7 (2005). https://doi.org/10.1109/ACC.2005.1470795
15. RoboCup: RoboCup 2022 SSL Vision Blackout technical challenge rules, `https://robocup-ssl.github.io/technical-challenge-rules/2022-ssl-vision-blackout-rules.pdf`
16. RoboCup: Simulation 2d, `https://ssim.robocup.org/`
17. Robotics Erlangen e.V. Team: Open Source Framework (2022), `https://github.com/robotics-erlangen/framework`
18. Röfer, T., Jüngel, M.: Fast and robust edge-based localization in the sony four-legged robot league. In: Polani, D., Browning, B., Bonarini, A., Yoshida, K. (eds.) RoboCup 2003: Robot Soccer World Cup VII. pp. 262–273. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
19. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. CoRR **abs/1801.04381** (2018), `http://arxiv.org/abs/1801.04381`
20. Silva, C., Alves, C., Silva, E., Martins, F., Cavalcanti, L., Maciel, L., Vinícius, M., Monteiro, J.G., Moura, J.P., Cruz, J.V., Santana, P.H., Sousa, R., Rodrigues, R., Fernandes, R., Morais, R., Araújo, V., Silva, W., Barros, E.: Robôcin extended team description paper for robocup 2022 (2022), robocup Small Size League, Recife, Brazil, 2022
21. Silva, C., Martins, F., Machado, J.G., Cavalcanti, L., Sousa, R., Fernandes, R., Araújo, V., Silva, V., Barros, E., Bassani, H.F., de Mattos Neto, P.S.G., Ren, T.I.: Robôcin 2019 team description paper (2019), robocup Small Size League, Recife, Brazil, 2019
22. Tigers: Robocup match metabase, `https://metabase.tigers-mannheim.de/public/dashboard/2a77fd2f-b8f9-4b34-8b6c-67bdf084b2a8?tournament=RoboCup2021&tournament=RoboCup2019&tournament=RoboCup2022`
23. Tigers Mannheim Team: Open Source Software and Hardware (2019), `https://tigers-mannheim.de/index.php?id=65`
24. Wendler, A., Heineken, T.: Er-force 2020 extended team description paper (2020), roboCup Small Size League, Erlangen, Germany, 2020