Improving Robot Coordination Using a Shared Game State

Amir Elmesiry*, Florian Kummerer, Christina Schönthaler, Vanja Drakulic
Höhere Technische Bundes Lehr- und Versuchsanstalt Wiener Neustadt
Department of Computer Science
2700 Wiener Neustadt, Austria
*Corresponding author's email: mrelmesiry@gmail.com

***Abstract*—This paper introduces a shared *game state* to improve coordination and dynamic strategy adjustment in the Botball competition. Traditional time-based models often fail due to environmental uncertainties, such as robot positioning errors or dynamic properties of the game table. The proposed *game state* framework synchronizes real-time information about the game table and robot statuses, enabling adaptive decision-making and sharing of knowledge about the environment. Experiments on a custom game table show the shared *game state* outperforms the conventional method in consistency, average points and fail-safety.**

I. Introduction

In the Botball competition, robots need to be almost perfectly synchronized to avoid interfering with each other. Furthermore, they may support each other by completing tasks needed by each other. Coordination between robots has been explored in previous ECER research papers, for example, using peer-to-peer communication to distribute tasks between robots [1]. However, consistent coordination between robots is difficult to achieve, as there are various factors that can negatively affect it, such as incorrectly built game tables or inaccuracies in robot navigation [2]. Therefore, this paper proposes a more robust system compared to the usual time-based model. Rather than coordinating the robots linearly following one course of action, information concerning a run and all its active tasks are managed and shared between both machines. This shared data is referred to as the *game state*. It enables useful features such as dynamic strategy adjustment based on the current state of the run and a robot being able to use information provided by the other machine. By replicating various scenarios on a simplified custom game table this paper compares the conventional approach with the *game state*. This work investigates whether a shared *game state* framework improves coordination robustness and overall scoring performance compared to a traditional time-based coordination strategy in Botball robotics competitions. We hypothesize that sharing real-time information between robots enables more adaptive task selection and therefore leads to higher average scores and improved resilience to failures.

II. Game State

The *game state* consists of the *table state*, which captures the physical state of the game table, and the *phase state*, holding the execution status of each *phase*. It is inspired by multi-robot task allocation frameworks [3]. Both components of the *game state* are synchronized between the two robots acting as a distributed state machine [4].

The full implementation of the *game state* framework is available under the GNU Affero General Public License v3 (AGPL-3.0) [5].

A. Phase

A *phase* is defined as a sequence of related actions to be executed once by one robot. It contains certain conditions which must be met to mark the phase as successfully completed and may require additional circumstances to be valid before execution. All phases are declared in a single JSON file that is stored on the primary robot and will be sent to the secondary robot before the initialization phase. Each phase has a status, which can be one of the following:

- *Ready* — All pre-conditions are met
- *Blocked* — The conditions are not yet met, but may be met in the future
- *Running* — Currently executing
- *Timed Out* — There is not enough game time left to execute this phase
- *Done* — This phase has already been completed

Additionally, both bots define a local registry which maps the ID of every phase that may be executed by a specific robot to a source code function. On selection of a phase, the executing robot sets the status to *running* and invokes the function corresponding to the ID of the selected phase. Each phase belongs to and is executed by exactly one robot.

B. Phase State

The phase state contains metadata for all phases, including the current phase of each robot and all open phases. A phase is considered open if its status is neither *done* nor *timed out* (see Fig. 1).

C. Phase Selection Algorithm

To decide which phase ϕ gets executed next, each phase is assigned a score S_ϕ as defined in

$$S_\phi = K_p \cdot p_\phi - K_t \cdot t_\phi + K_{pt} \cdot pt_\phi \tag{1}$$

where p_ϕ is the maximum amount of points that can be earned, t_ϕ is the time required to complete the phase.

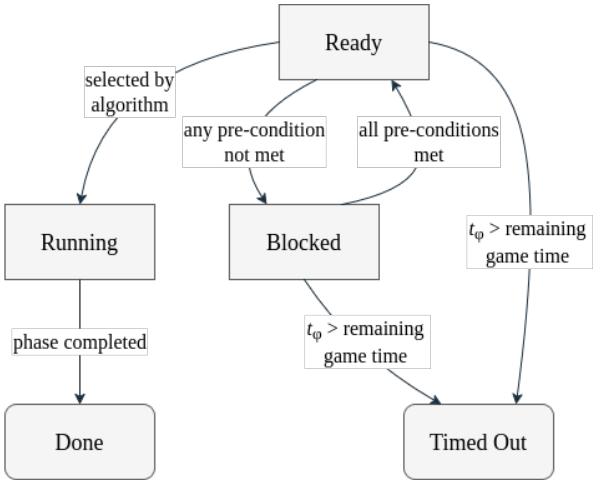


Fig. 1. State transition diagram of *phase* statuses

The *potential* pt_ϕ represents how many other phases become available afterward as defined in Algorithm 1. K_p, K_t, K_{pt} are weighting coefficients that are responsible for determining the priority of points, time and potential. The algorithm then selects the phase ϕ with the highest score.

In the experiments for the *game state* model, the coefficients were empirically chosen as $K_p = 1.0$, $K_t = 0.6$, $K_{pt} = 0.4$. These values prioritize scoring opportunities while still penalizing long execution times and rewarding phases that unlock additional tasks.

Algorithm 1: Compute Phase Potential

Input: unlockedPhases
Output: phasePotential
$totalPoints \leftarrow 0$;
$phaseCount \leftarrow 0$;
for each $phase$ in $unlockedPhases$ **do**
  $totalPoints \leftarrow totalPoints + phase.points$;
  $phaseCount \leftarrow phaseCount + 1$;
if $phaseCount > 0$ **then**
  $phasePotential \leftarrow totalPoints / phaseCount$;
else
  $phasePotential \leftarrow 0$;
return $phasePotential$

D. Table State

The second component of the *game state*, the *table state* aims to describe the current state of the Botball game table. It enables all bots to easily read information that may be specific to each run (e.g., a random color sequence) or might change during a run (e.g. position of game pieces), provided they have been previously gathered by either robot. The game table is modeled in a JSON config file which is parsed before game start by the primary robot. Data on the game table can be represented as any primitive data type or arrays of primitive data types, and are used to determine if all preconditions of a phase are met.

E. Communication

The *game state* needs to be shared and communicated between both robots, thus a custom communication protocol operating at the 7th layer of the OSI model [6] was implemented. The choice of Wi-Fi and TCP for the protocol was justified by a previous research paper comparing various different communication methods for Botball [7]. The protocol begins with a Client-Server communication model. The primary robot acts as a server during start up, parses and validates all configuration files and serves them to the secondary robot, the client. When both robots have a synchronized *game state*, they execute their respective initialization phases and the protocol switches to a peer-to-peer communication model.

There are a total of three requests in the protocol.

1) request state: Upon receiving this request, the server sends the entire *game state* to the client. This marks the beginning of the initialization phases *init_a* and *init_b* which enable positioning and the calibration of any sensors. Light start triggers the execution of the first phase and starts the game timer.

2) update table: Each robot can modify the *table state* at any time during a run. In order to achieve this, the robot that is making a change will send a request to its peer, which will then update its local version.

3) update phase: Similar to updating the *table state*, once the status of any phase changes, a request is sent to notify the robot's peer that it needs to update its local phase registry.

III. Weaknesses and Sources of Error

A. Newly introduced error sources

The *game state* adds complexity, enabling bugs to hide more effectively in the source code, such as actions with the wrong phase ID. The phase selection algorithm introduces the task of selecting the appropriate coefficients for p_ϕ, t_ϕ, and pt_ϕ. Setting these values incorrectly could negatively impact the outcome of the runs. Another issue is Wi-Fi signal congestion. If a system such as the one outlined in this paper is used for a Botball competition, there may be a lot of signal interference.

B. General error sources

When working with sensors one must consider that small variations in input data are the norm. Hence, a working balance of trust in sensor data and software processing has to be met. Apart from the aforementioned problem there is always the inconsistency of various game table builds.

IV. Test Environment

To test the effectiveness of the *game state* approach, a custom game table has been developed based on the compAIR mat [8]. A simplified environment was chosen intentionally, as the benefits of the *game state* increase with the complexity of the environment.

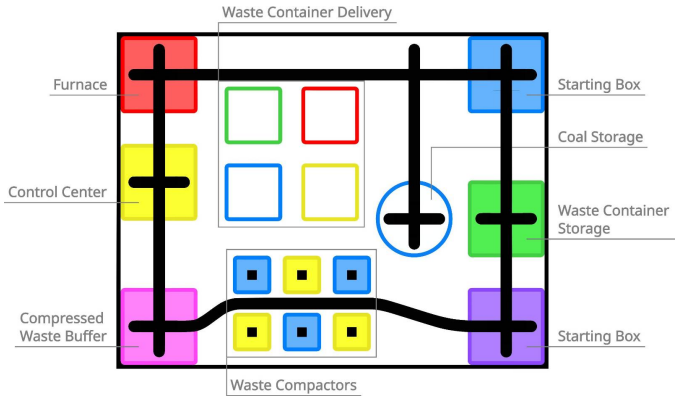


Fig. 2. Game table with captions

A. Game Table

The table (Fig. 2) represents a virtual incineration plant consisting of nine areas:

- Purple Starting Box — starting area for one robot
- Blue Starting Box — starting area for the other robot
- Waste Compactor — containing six compressed waste cubes represented by green cubes
- Compressed Waste Buffer — used for temporary storage of compressed waste cubes
- Control Center — providing the stacking order for waste containers
- Furnace — deposit for compressed waste cubes
- Waste Container Delivery — having an equal colored block in the yellow and blue box, standing for a waste container
- Coal Storage — containing a single piece of coal represented by a rock
- Waste Container Storage — waste container stacking zone

B. Rules

- The robots have 90 seconds to complete as many tasks as possible.
- Robots may only drive on black lines, except in the Waste Compactor and Waste Container Delivery area.
- Waste Container Delivery area may only be entered from the line between the Furnace and the blue Starting Box.
- Waste container stacking information may only be gathered from within the Control Center.
- In order to burn waste the Furnace first needs a piece of coal.
- All compressed waste cubes delivered earlier than the coal do not count towards the points.
- The robots may drive into the Furnace, but they must leave through the same border they entered from.

C. Scoring

The score a robot achieves is the most important metric of the tests conducted. All scoring rules are shown in Table I.

TABLE I
SCORING CRITERIA FOR GAME PIECES AND LOCATIONS.

Game Piece	Location / State	Points
Compressed Waste	Waste Buffer	10 each
Compressed Waste	Furnace (Post-Coal)	20 each
Waste container	Unstacked	10 each
Waste container	Stacked (General)	25 each
Waste container	Stacked & Sorted	50 each
Coal	Furnace	Condition

V. Data Collection And Analysis

To test the effectiveness of both coordination strategies, two specifically designed robots were placed on the custom game table, each designed to perform a different role, as would be the case in a typical Botball competition. Several scenarios, including hardware failures, were implemented to test the effectiveness of both coordination strategies. Points scored and the time remaining were used as the key metrics to measure the effectiveness of the robots.

A. Primary Robot

The primary robot was fitted with pliers to move small cubes and a camera to detect the waste container stacking order (see Fig. 3(a)).

Starting at the purple Starting Box, the robot drives to the Waste Compactors to collect the first row of compressed waste cubes, then follows the black line to the Control Center. In the *game state* approach, the robot uses the camera at the Control Center to obtain the stacking order of the waste container and share it with the secondary robot. The primary robot then checks if the coal is already in the Furnace. If not, the cubes are stored in the Compressed Waste Buffer, otherwise they are delivered to the Furnace. In the time-based approach, the robot skips the scanning of the waste container stacking order and goes to the Furnace to deliver the cubes unconditionally. In both cases the primary robot drives to the Waste Compactors to collect the last row of compressed waste cubes and deliver them to the Compressed Waste Buffer.

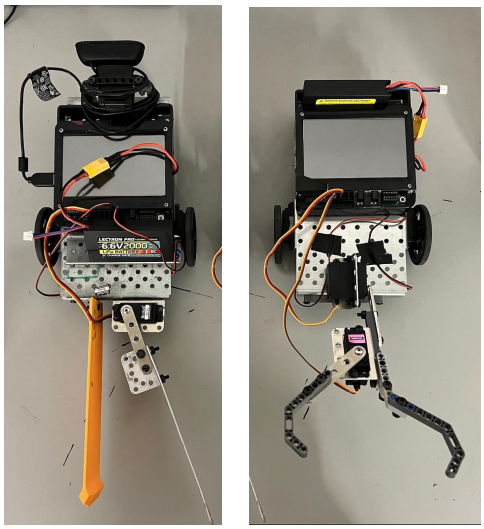
B. Secondary Robot

The secondary robot was equipped with a gripper arm responsible for grabbing and stacking cubes. (Fig. 3(b)).

Starting from the blue Starting Box, the first step is to drive to the Coal Storage area and collect the coal. It is then delivered to the Furnace. The robot then either waits for the waste container stacking order or, using the time-based model, always picks up the blue waste container, which it then drops of in the Waste Container Storage. Afterward, it drives back to collect and place the remaining waste container.

C. Test Cases

To evaluate the robustness of both coordination strategies, three test cases were designed, reflecting typical Botball tournament scenarios. Baseline performance was assessed by running both the time-bound and *game state* approaches ten times each. Hardware resilience was tested through five trials



(a) Primary robot (b) Secondary robot

Fig. 3. Robots used for the experiment

with a manually introduced failure on a single robot and five with simultaneous failures on both bots.

All hardware failures were simulated by either unplugging the motors or causing the robots to intentionally drop a game piece in front of them. Botball robots use wheel encoders that emit discrete pulses, called ticks, proportional to wheel rotation, which the software uses to estimate distance traveled. In case motors are stuck or unplugged the tick count does not increase. While driving, the robots recognized that the tick count had not increased and that they had not yet reached their destination. Later on, when the game piece was removed or the motors were plugged in again, the robots continued to drive but arrived too late, failing to complete the task that the other robot depended on within the given time.

D. Evaluation Metrics

To evaluate both coordination strategies, the average score, its standard deviation, average remaining time and its deviation were recorded for each scenario.

E. Results and Analysis

The results of all conducted test cases were recorded. Tables II and III summarize the collected data for the conventional strategy and *game state* respectively.

TABLE II
TIME-BASED STRATEGY METRICS ACROSS SCENARIOS

Metric	Exp.[1]	Fail P.[2]	Fail S.[3]	Both[4]
Avg Points	165	160	130	100
Avg Time Left (s)	16.2	16.6	3.6	15
Avg Point Dev.	25	24	0	24
Avg Time Dev. (s)	0.64	0.88	2.48	0.8

[1]Expected Operation, [2]Primary Robot Failure, [3]Secondary Robot Failure, [4]Simultaneous Failure.

TABLE III
GAME STATE STRATEGY METRICS ACROSS SCENARIOS

Metric	Exp.[1]	Fail P.[2]	Fail S.[3]	Both[4]
Avg Points	190	160	160	130
Avg Time Left (s)	17.4	16.6	3.6	17
Avg Point Dev.	0	24	0	24
Avg Time Dev. (s)	0.72	0.88	2.48	0.8

[1]Expected Operation, [2]Primary Robot Failure, [3]Secondary Robot Failure, [4]Simultaneous Failure.

Fig. 4. Comparison of Average Points Between Time-Based and Game State Approach.

Under normal operation, the *game state* strategy achieved an average of 190 points, compared to 165 points for the conventional time-based strategy. When the primary robot failed, both approaches achieved the same average score of 160 points, indicating similar resilience in this scenario.

The largest performance difference occurs with a failure of the secondary robot. In this case, the *game state* approach allows the primary robot to detect that coal has not been delivered and adapt its strategy accordingly enabling it to earn a higher score (160 points). In contrast, the time-based strategy continues executing pre-defined actions without considering the missing prerequisites, which leads to wasted time and a lower score (130 points).

These results highlight that the main advantage of the *game state* approach lies in its ability to dynamically react to incomplete tasks and missing prerequisites during runtime.

When both robots experienced failures simultaneously, the *game state* strategy still achieved 130 points, whereas the conventional approach achieved 100 points. The lowest points were achieved using the time-based approach when both robots experienced failures. Depending on the stacking order, these could reach a total of 80 points.

Figure 4 visualizes these differences. The *game state* strategy demonstrates a higher baseline performance and equal or lower score variability across all scenarios.

VI. DISCUSSION

The results show that the *game state* approach outperforms the time-based model in terms of average scores and robustness. The biggest advantage of the *game state* approach is that

it is more adaptable, as the sharing of information in real-time removes any guesswork from the process, as was the case in the time-based model. This is highlighted in the results from the tests under normal conditions, where the information available to the robots enabled them to make optimal decisions at every step, resulting in a perfect score of 190 with zero deviation, as opposed to the results from the time-based model, where the robots would inevitably have to speculate when stacking the waste containers in the correct order.

In the event of failure, the adaptability of the *game state* is equally valuable. When the secondary robot failed, the primary robot was able to detect the absence of coal. The *table state* was then adjusted accordingly. However, the time-based strategy continued blindly, resulting in an average loss of 30 points. The only case in which the two methods were equivalent was when the primary robot had a failure. The advantage of the *game state* method is that it uses real-time communication to find the optimal stacking order, however if the information is not available, it defaults to retrieving the blue waste container first. This way, a minimum amount of points is able to be procured.

Nonetheless, the *game state* method doesn't come without trade-offs. The complexity added by the *table state* and the definition of the phases increases the likelihood of bugs, which may not be easily identified in a distributed system. Additionally, the requirement of a stable Wi-Fi connection introduces a failure case that does not exist with the time-based approach. In a competition setting where signal interference is high, a disconnection could render the coordination framework useless, if not even a burden. Lastly, the choice of coefficients used in the phase selection algorithm may not yield optimal performance, potentially leading to underperformance compared to the time-based approach.

VII. Conclusion

This paper introduces a shared *game state* model for coordinating robots in the Botball competition. Experiments on a custom game table demonstrated that, by sharing certain information about each other and the current stage of the run, the robots were able to achieve a higher score than would have been possible using the time-based approach due to time restrictions. The robots were also able to dynamically adjust their next task in order to avoid interfering with each other or performing tasks for which the preconditions had not been met. While the approach increases complexity and relies on stable communication between robots, the results show that shared state coordination is a possible method for improving reliability and fail-safety in robotic competitions.

Possible future work could involve the use of automated parameter tuning for the phase selection algorithm. Another extension of the shared *game state* model could be the integration of a strong hardware failure detection system. This would enable the system to adjust its strategy immediately if a failure is detected, rather than continuing to operate on incorrect assumptions. While the current project implemented a basic form of motor error detection by tracking the increments of the encoder ticks, a more sophisticated system could extend this to other components, such as servos and analog sensors, to enable the system to detect and respond to a wider range of hardware failures autonomously.

As the proposed coordination strategy requires communication between multiple robots, security aspects must also be taken into account. Previous studies [9] have shown the potential for security vulnerabilities and the ability to exploit these vulnerabilities of the Wombat [10]. Similar security issues could also be resolved to improve the system's reliability and prevent malicious interference.

Acknowledgment

The authors would like to thank robo4you for providing the resources and support necessary for producing this publication. Furthermore, they would also like to express their deep gratitude to Konstantin Lindorfer, Alexander Gugumuck and Dr. Michael Stifter for their countless amount of feedback and helping improve this paper. The authors are also thanking Ing. Jakob Eichberger, BEd BSc MSc, as well as their mentors and colleagues for their great guidance and feedback.

References

[1] Yasin Sahin, Sven Oberwalder, Fabian Hitzenberger, Karl Dopplinger, Raphael Ackerl and Julian Kerer, "Task Administration via Peer-to-Peer Communication," https://ecer.pria.at/archive/ecer-2023/papers/Task_Administration_via_Peer-to-Peer_Communication.pdf (accessed on 2026-03-12)

[2] B. Klauniger et al., "Enhancement of Accuracy in Botball Navigation" https://ecer.pria.at/archive/ecer-2023/papers/Enhancement_of_Accuracy_in_Botball_Navigation.pdf (accessed on 2026-03-12)

[3] Brian P. Gerkey and Maja J. Matarić, "A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004. https://robotics.stanford.edu/~gerkey/research/final_papers/mrta-taxonomy.pdf (accessed on 2026-03-12)

[4] Leslie Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978. https://doi.org/10.1145/359545.359563 (accessed on 2026-03-12)

[5] Amir Elmesiry, "Botball - Game State" https://github.com/Amir-jpg-png/Botball-Game-State/

[6] International Telecommunication Union (ITU), "Information technology — Open Systems Interconnection — Basic Reference Model: The basic model" https://www.itu.int/rec/T-REC-X.200-199407-I/ (accessed on 2026-02-08)

[7] J. Sztavinovszki et al., "Comparing the Viability of Different Communication Methods for Botball" https://ecer.pria.at/archive/ecer-2024/papers/Comparing_the_Viability_of_Different_Communication_Methods_for_Botball.pdf (accessed on 2026-03-11)

[8] robo4you, "Wettbewerb für MINT und Robotik" https://comp-air.at (accessed on 2026-01-24)

[9] J. Gosling et al., "Spellz: Attacking robots in cyberspace — Blue Team/Red Team Approach" https://ecer.pria.at/archive/ecer-2025/papers/Spellz_Attacking_robots_in_cyberspace_Blue_Team_Red_Team_Approach.pdf (accessed on 2026-03-09)

[10] KIPR (KISS Institute for Practical Robotics), "KISS Hardware/Software" https://www.kipr.org/kipr/hardware-software (accessed on 2026-03-09)