

Berlin United Visual Analytics Tool

<https://github.com/orgs/Visual-Analytics-Toolbox>

Anastasia Prisacaru and Stella A. Schlotter

Adaptive Systems Group
Humboldt-Universität zu Berlin
Unter den Linden 6, D-10099 Berlin, Germany
{prisacaa|schlotts}@informatik.hu-berlin.de

Abstract. Since 2009, our team has curated an extensive repository of multimodal data from RoboCup competitions, including high-frequency robot logs, match videos, and GameController metadata. Historically, this repository functions as a heterogeneous, unstructured collection, making it difficult to isolate specific game events for analysis. This paper introduces a redesigned data management framework that organizes legacy data into a standardized hierarchy and implements a database-backed index for rapid retrieval. By using the GameController logs as a temporal anchor, our tool synchronizes disparate data streams—including robot internal representations, video feeds, and human-recorded audio notes. This unified system allows researchers to instantly correlate a robot’s internal state with ground-truth video and contextual metadata, significantly lowering the barrier for automated game evaluation.

1 Introduction

Since 2009, our team has consistently collected expensive datasets from RoboCup competitions. This corpus includes robot internal logs, external video recordings, qualitative game notes, and GameController logs. Although these assets are publicly hosted on our Log Server¹, the repository was historically managed as an unstructured collection. The lack of organization and occasional mislabeling of files made it prohibitively difficult to isolate specific game events for detailed analysis. To address this, we have restructured the legacy data into an intuitive folder hierarchy and developed a centralized database that serves as a high-speed index. This system allows for rapid querying of specific match moments. To further enhance the system’s utility, we synchronize all data streams, allowing users to instantly correlate a robot’s internal state with its corresponding external camera view at any given timestamp.

2 Outline

In this work, we show a system that facilitates rapid development and debugging in the context of robot soccer. To achieve this goal, data streams from

¹ <https://logs.berlin-united.com/>

different sources are synchronized with each other. All source code is published under <https://github.com/orgs/Visual-Analytics-Toolbox> and documentation in https://docs.berlinunited.org/visual-analytics/vat_setup/.

2.1 Data Overview

Our repository comprises several distinct data streams that, when combined, provide a holistic view of match performance:

- **Robot Logs:** The bulk of our data consists of high-frequency internal logs from each robot. These record the internal state, sensor readings, and decision-making processes.
- **Video Data:** We record one video per halftime. As detailed in our previous work [1], these videos are precisely synchronized with GameController logs.
- **GameController Logs:** These serve as the ground-truth timeline for game states (e.g., set-play transitions, scores, and penalties).
- **Qualitative Notes:** We also include game observations recorded as either handwritten notes or audio snippets. The latter are captured via a custom-developed application that automatically synchronizes the audio recordings with the GameController timeline.

The video data and the GameController Logs have proven to be very useful for post-game analysis, and many teams used our data to develop analysis tools. Based on the video data collected, the Video Analysis Challenge was conducted at RoboCup 2022 [2][3]. The data collected by the robots themselves, however, is rarely used for large-scale analysis, as external video and GameController logs provide sufficient surface-level information. This data originates directly from the robot’s two main software processes: *Cognition* and *Motion* [4]. *Cognition* handles perception and high-level decision-making, processing camera images and managing game behavior, while *Motion* handles joint control, balance, and physical sensor data such as IMU and sonar readings. Within each process, the computations are carried out by modules that exchange data in the form of representations [4]. Representations are instances of C++ classes that mainly store data and provide nearly no functionality [5]. All representations within a process are held in a shared structure called a blackboard, giving every module in that process consistent access to the current state. Because representations are streamable, they can be written or read from a data stream. The representations active in the *Cognition* process are written to `game.log`, while those from the *Motion* process are written to `sensor.log`. In order to add annotation live during a game, a mobile application was developed, that allows the operator to record voice messages during the game. For example, an unexpected fall or a sudden behavioral change.

3 System Overview and Data Flow

Our system simply consists of the aforementioned file storage, a postgres database, and a Rest API. A cron job is responsible for indexing the data from file storage

to the database using the API layer. The database insert can run in two modes. Either the *full* approach where the representation data is saved as json in the db or the *memory map* approach, where only the logfile path is saved for each log, and for each representation we save the start byte + size inside the log file. The data is then queried from the file on every read request. The second approach is slower but needs significantly less storage. The database stores meta information for each half (e.g., teams played, score, event, time, etc.). The log table holds information regarding the data of one robot. Most importantly, the path to the logfiles. The frame tables store the times and frame numbers. Each representation has its own table. When choosing the mmap approach for saving the data, each row in the representation only stores the start byte and the size of the data in the binary logfile.

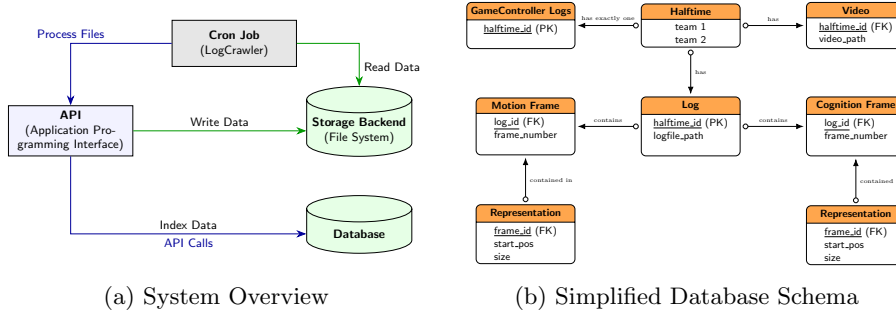


Fig. 1: System Design and Database Structure

4 Data Synchronization

Data synchronization is a critical challenge when working with log data, as multiple processes operate at different levels, frequencies, and timestamps. In our case the `game.log` and `sensor.log` can be synchronized by the timestamp of each logged frame since their timestamps share a common system clock and can be aligned using that shared time reference. A further synchronization challenge emerges when looking at data across multiple robots in a team during the same game half. The robots are booted up at different times, so their internal clocks start at different offsets. Therefore we need to find an external signal as a reference point in time to align the data from all robots in the same team on a shared timeline. This reference is provided by the Game Controller (GC), a dedicated software application running on a computer placed next to the playing field. The GC manages and broadcasts the official game state to all connected devices inside the same wireless network. It broadcasts control messages at 2 Hz [6]. Because this signal is received by all robots simultaneously, the moment of a specific game state transition, such as the "initial" or "finish", can be used

as a shared time anchor to align the logs from all robots to a common game timeline. The video recording is started when camera system receives the GC "ready" game state signal. The robots start logging when they receive this signal, leading to an automatic synchronisation between the robot logs and the video. Our note taking app also receives the GC broadcast messages, so the app can calculate the time elapsed since the last received GC message and use that offset from the button press to assign a precise timestamp to the logged event.

5 Use Cases

Dataset Creation: Indexing log data in a database enables rapid, automated dataset construction. Instead of manual log browsing, metadata queries can filter frames by game states (e.g., `ready`, `playing`), brightness thresholds, or object detections. Negative filters efficiently reduce noise by excluding frames where, for instance, a robot has fallen. Queries can be scoped by event, game, or day, providing precise control over data provenance. These datasets can be used for neural network training, analytical pipelines, or automated pre-annotation. Centralized indexing thus reduces the time between identifying a data need and having a ready-to-use dataset from hours of manual search to a matter of seconds.

Situation Annotations: The infrastructure supports manual tagging via a dedicated marker tool, linking specific events—like missed kicks or falls—to database frames. These annotations serve two roles: for debugging, they act as bookmarks to instantly jump to the robot’s state during an incident; for dataset creation, they provide a queryable dimension (e.g., `ball_lost`). This closes the loop between human field observations and systematic data collection, allowing for targeted extraction of critical competition moments.

References

1. Mellmann, H., Schlotter, S.A., Strobel, P.: robocup.tools - Data Driven Research and Development in Robocup (2019)
2. Röfer, T., Müller-Groh, L., Hasselbring, A., Laue, T.: Automated game statistics for the RoboCup Standard Platform League. In Barros, E., Hanna, J.P., Okada, H., Torta, E., eds.: RoboCup 2024: Robot World Cup XXVII. Volume 15570 of Lecture Notes in Artificial Intelligence., Springer (2025) 281–293
3. Bloisi, D.D., Pennisi, A., Zampino, C., Biancospino, F., Laus, F., Di Stefano, G., Brienza, M., Romano, R.: Mario: Modular and Extensible Architecture for Computing Visual Statistics in Robocup SPL. arXiv preprint arXiv:2209.09987 (2022)
4. Mellmann, H., Xu, Y., Krause, T., Holzhauer, F.: Naoth Software Architecture for an Autonomous Agent. In: International Workshop on Standards and Common Platforms for Robotics (SCPR 2010), Darmstadt (November 2010). (2010)
5. Röfer, T., Laue, T.: On b-human’s Code Releases in the Standard Platform League—Software Architecture and Impact. In: Robot Soccer World Cup. Springer (2013) 648–655
6. RoboCup-HumanoidSoccerLeague: Gamecontroller. <https://github.com/RoboCup-HumanoidSoccerLeague/GameController>