HULKs RoboCup SPL Team TUHH, Germany

Advancing Humanoid Robotics with Rust: An Open Framework for Runtime Efficiency

Maximilian Schmidt, Hendrik Sieck, Ole Felber, Konrad Valentin Nölle, Luis Scheuch, Patrick Göttsch

Robotics Framework in Rust

Efficient and **adaptable** software frameworks are crucial for advancing the capabilities of teams participating in RoboCup and robotics in general. Our novel framework emphasizes **modularity** and **parallelization** while optimizing for **minimal runtime overhead** and complexity.

Implemented in **Rust**, we combine **computational efficiency** with **safety**, with **performance** comparable to other leading compiled languages. Unique features like ownership and borrowing contribute to more **concise and comprehensible** code compared to languages like C. Rust has an extensive **package ecosystem** which simplifies the use of **external libraries**, enhancing **flexibility** and **extensibility**.

Features

- Parallelization
- Motion Cycle Priority
- Low Runtime Overhead
- Separation of Robotics and Framework Domain
- Adaptive Computation
- Temporally-Ordered Data
- Abstraction over Hardware
- Data Structure Serialization, Deserialization, and Reflection
- Tooling Support
- Rust

Framework Components

```
CyclerManifest {
    name: "Vision",
    kind: CyclerKind::Perception,
    instances: vec!["Top", "Bottom"],
    setup_nodes: vec!["vision::image_receiver"],
    nodes: vec![
        "vision::ball_detection",
        "vision::camera_matrix_extractor",
        "vision::feet_detection",
        "vision::field_border_detection",
        "vision::field_color_detection",
        "vision::image_segmenter",
        "vision::limb_projector",
        "vision::line_detection",
        "vision::segment_filter",
    ],
}
```

Real-Time vs. Perception

multiple instances of the same processing pipeline

event-based adaptive computation

Nodes:
- small units of computation
- sorted topologically
- at compile-time

Each node is defined by its **inputs**, **outputs**, **self-contained state**, and its implementation. Nodes produce **strongly typed** outputs, each uniquely identified by a distinct name, forming a comprehensive **dependency graph**. The framework **sorts the execution of nodes topologically**, ensuring that each node is executed only after its dependencies have been produced. A cyclic dependency between nodes results in a **compile-time error**.

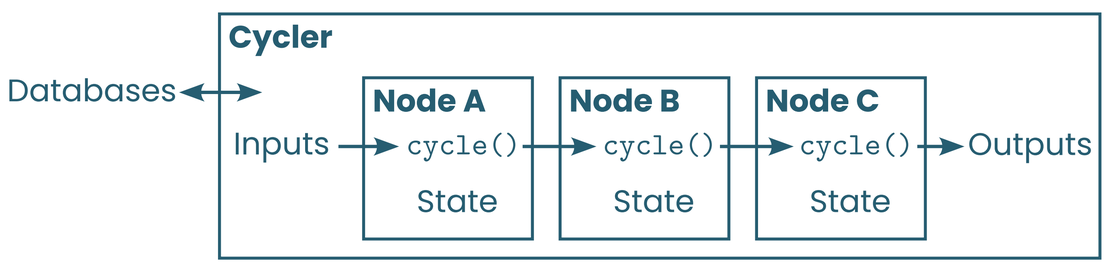

Figure 2: Architecture overview of a cycler composed by its nodes.

Interleaved Temporally-Ordered Access

Some nodes like Kalman-Filters require their inputs to be temporally ordered. Problems arise when multiple threads produce updates at different speed.

Inefficient Solution: delaying processing of fast cycles
Our Solution: queuing futures of pending cycles

- Cyclers announce start of a pending cycle
- Each entry is associated with a timestamp

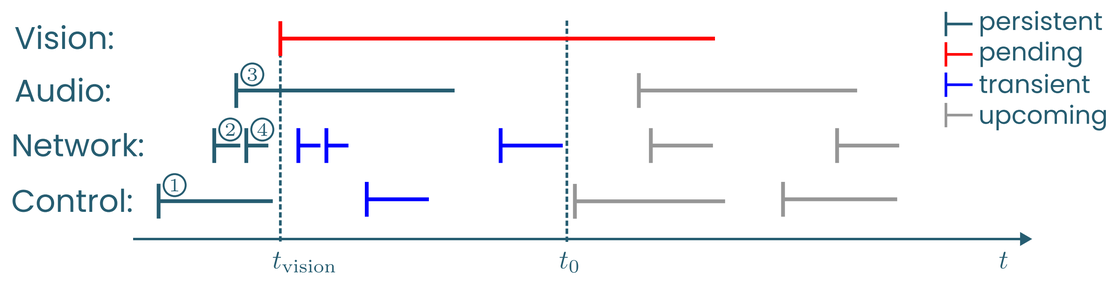

Figure 3: Timing of multiple cyclers running in parallel.

Recording and Replay

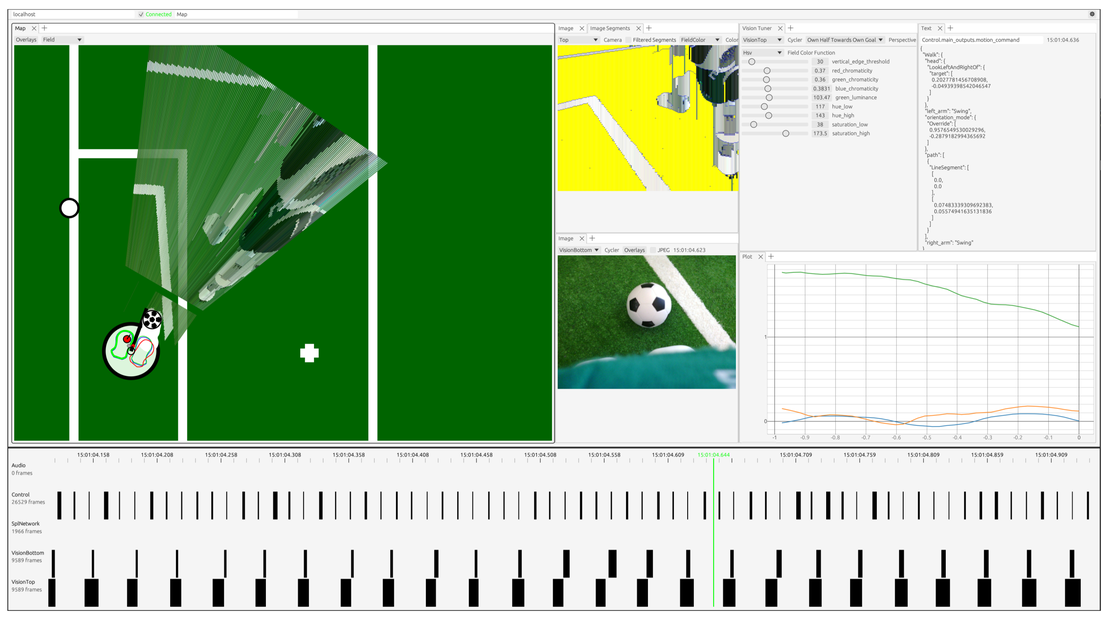

Figure 4: Our Visualization Tooling replaying a recorded competition.

Case Study: HULKs Robotics Domain

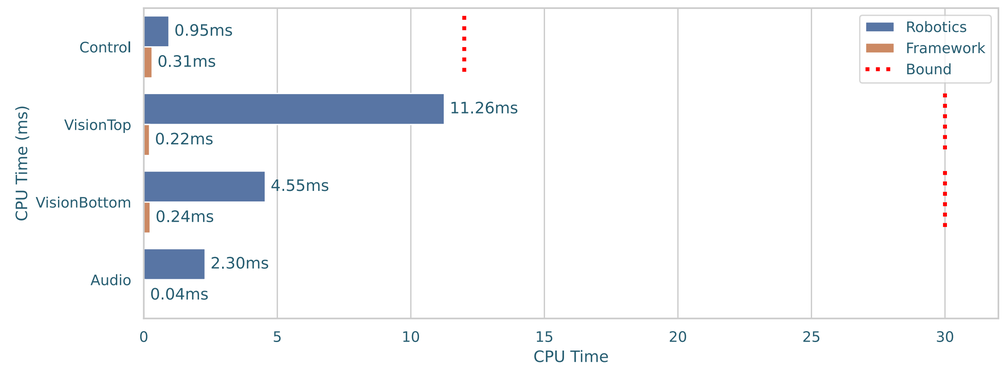

Figure 5: Profile of the CPU time in the HULKs code base across different cyclers.

Adoption in the RoboCup SPL

Since its **open-source** release, our framework has been rapidly **gaining popularity**. For the 2023 RoboCup, the **Dutch Nao Team** adopted the HULKs framework and achieved the 4th place. Team **Rinobots** from Brazil uses the framework as well.

TUHH
Hamburg University of Technology

HULKs
RoboCup SPL Team • TU Hamburg, Germany

Technische Universität Hamburg
Am Schwarzenberg-Campus 1
21073 Hamburg
hulks@tuhh.de

github.com/hulks/hulk

hulks.de