Software Survey 2026

Team name

GeoHBots

Which division(s) are you applying for? If your used software differs between divisions, please fill out the survey once per division.

Small Size (height < 110 cm, weight < 15 kg)

Is your software fully or partially OpenSource? If so, where can it be found?

Yes. Our software is released under the Apache License 2.0.It can be found with this Github URL:https://github.com/cigratte4/GeoHBots_Soccer.git

Are you using any software developed by other teams? If so, list every component that you are reusing and the team that originally developed it.

Third-party / external software components used:

• BehaviorTree.CPP — high-level behavior control and task sequencing

• Rerun SDK — runtime visualization, log recording, and post-match analysis

• backward_ros — stack trace and crash backtracing for debugging

• Eigen3 and OpenCV — numerical computation and vision post-processing

• Platform/perception interfaces (e.g., booster_interface, vision_interface, game_controller_interface)

Are you using any datasets in your research? If you are using your own datasets, are they public?

Yes. We use self-collected and annotated field images and match recordings to train/validate the perception pipeline used by vision_interface (ball/goal/line-related detections).

Please list the scientific publications your team has made since the last application to RoboCup (or if not applicable in the last 2 years).

None.

Are there any other contributions you would like to share with the RoboCup community?

We maintain a modular ROS 2 codebase with clear interfaces between perception, localization, behavior, and motion. We also provide comprehensive runtime logging and visualization (via Rerun) to support rapid debugging, replay, and quantitative evaluation.

Which approach are you using to generate the robot walking motion?

High-level walking commands are generated in Cartesian space (vx, vy, vtheta) and sent to the locomotion controller through the platform interface. Motion targets (e.g., walk-to pose and ball alignment) are computed using geometric reasoning and closed-loop feedback.

Which approach are you using to generate other motions of the robot (e.g. kicking, standing up)?

Discrete motions such as kicking, getting up, and special actions are triggered through dedicated APIs/messages. Motion selection (e.g., kick type and direction) is decided by the behavior layer and executed by the low-level controller.

Do you have a kinematic or dynamic model of your robot? If so, how did you create it (e.g. measure physical robot, export from CAD model)?

Yes. The system maintains a kinematic model using tf2 frames (URDF-based), and uses Eigen for coordinate transforms and kinematic computations needed for motion control and perception-to-world projection.

What approaches are you using in your robot's visual perception?

Our perception pipeline combines deep-learning detections (ball/goal/robots) with geometric post-processing. The vision node produces detections which are fused with camera calibration and field geometry to estimate bearings and distances. OpenCV is used for image pre/post-processing and filtering.

Are you planning with objects in Cartesian or image space? If you are using Cartesian space, how do you transform between the image space and cartesian space?

Cartesian space. Detections in image space are transformed to robot/world coordinates using calibration parameters and tf2 transforms, enabling planning and behavior decisions in the field coordinate system.

Do you have some form of active vision (i.e. moving the robots camera based on information known about the world)?

Yes. The robot actively controls head yaw/pitch to keep the ball and key landmarks inside the camera field of view. Head motion is coordinated with walking and search behaviors to improve detection continuity and localization confidence.

What approach are you using to localize your robot?

Localization estimates the robot pose (x, y, theta) in the field frame. It integrates odometry/IMU cues with visual observations of field landmarks, and manages transforms via tf2 to provide consistent robot□field coordinate conversions to all modules.

Is your team performing team communication? Which communication protocol are you using?

Yes. We integrate with the official GameController via game_controller_interface and exchange team state through ROS 2 messages (e.g., role, ball estimate, and intent) to support role assignment and cooperative behaviors.

What approach are you using for navigation? Are you avoiding obstacles?

Navigation is executed as closed-loop walking toward target poses with reactive obstacle avoidance. Obstacle information (from vision or range sensors) is used to estimate collision risk and adjust velocity commands in real time while maintaining task progress.

How is the behavior of your robots structured? (e.g. Behaviour Trees)

High-level decision-making is implemented using BehaviorTree.CPP. Behaviors are composed in XML behavior trees, with shared state stored in a blackboard. Runtime parameters are configured through YAML files, enabling fast tuning and modular reuse.

Are you simulating your robot? If so, which simulator are you using and for what purpose do you use simulations?

Yes. We verify modules in simulation when applicable (e.g., Webots or Gazebo) and also use log-based replay for regression testing. The project is built with ament_cmake and follows

ROS2 package conventions.

What operating system is running on your robot and which middleware are you using (for example Ubuntu 22.04 and ROS2 Galactic)?

Ubuntu Linux (primarily 22.04) and ROS 2 (rclcpp-based).

Is there anything else you would like to share that did not fit any previous question?

None.