RoboCup 2025 SALVADOR

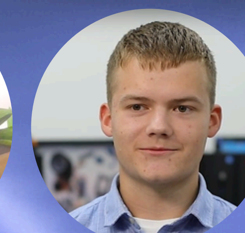
RoboCup Junior

SPESHARI

Rescue simulation erebus

Jakub

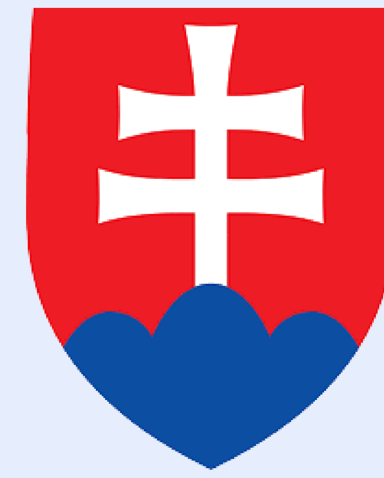
Tomáš


Pavol

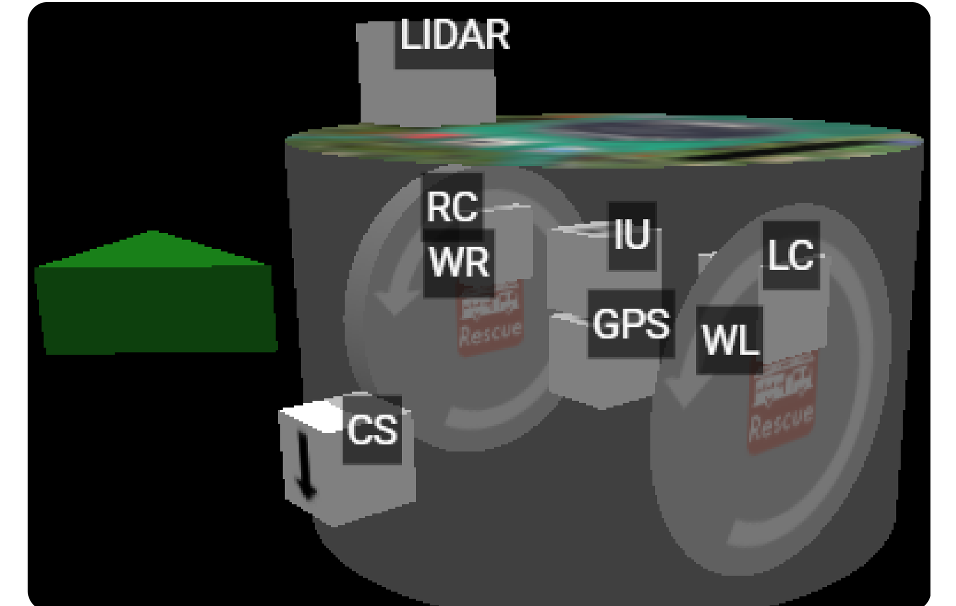
Marko

Jakub is responsible for developing the robot's movement system, including turning, stopping at waypoints, and navigating around obstacles. His logic ensures smooth and adaptive control using data from LIDAR, IMU, and GPS.

Tomáš develops the robot's map-building system using LIDAR, IMU, GPS and cameras. He creates a 2D grid-based model of the world, enabling precise path planning. He also handles sensor data filtering and processing.



SLOVAKIA



Robot cost: 2950 / 3000

Pavol handles victim detection using image processing. Designed and extensively tested algorithm for victim recognition and color-based hazard identification, ensuring reliable performance under varying conditions. Also created script for generating random token rotation.

Marko implements Dijkstra algorithms for route optimization. His work ensures the robot finds the most efficient path to each target while dynamically reacting to changes in the environment and newly discovered obstacles.

1. place – RoboCup Junior World 2025
1. place – RoboCup Junior European 2025
1. place – RoboCup Junior Croatia 2025
1. place – RoboCup Junior Slovakia 2025

LIDAR

Cost: 500

LIDAR is used to continuously scan the surroundings and detect obstacles in all directions. It provides accurate distance measurements that are essential for both real-time navigation and building a reliable map of the environment. Its 360° range and high resolution make it significantly more effective than simple distance sensors.

GPS

Cost: 250

The GPS sensor determines the robot's global position within the simulation world. It is used to anchor the robot's location on a global map and track its movement over time. In combination with the mapping algorithm, GPS allows the robot to localize itself relative to key areas such as the starting zone, map areas and checkpoints. Its use eliminates the cumulative drift that would occur with wheel-based odometry alone.

IMU

Cost: 100

The IMU measures the robot's angular velocity and acceleration, helping determine its orientation (yaw, pitch, and roll). It is especially useful during turns and for precise movemnt itself. By combining IMU data with GPS, the robot maintains smoother and more accurate control over its movement and heading. This sensor also improves motion prediction in the mapping system.

COLOR SENSOR

Cost: 100

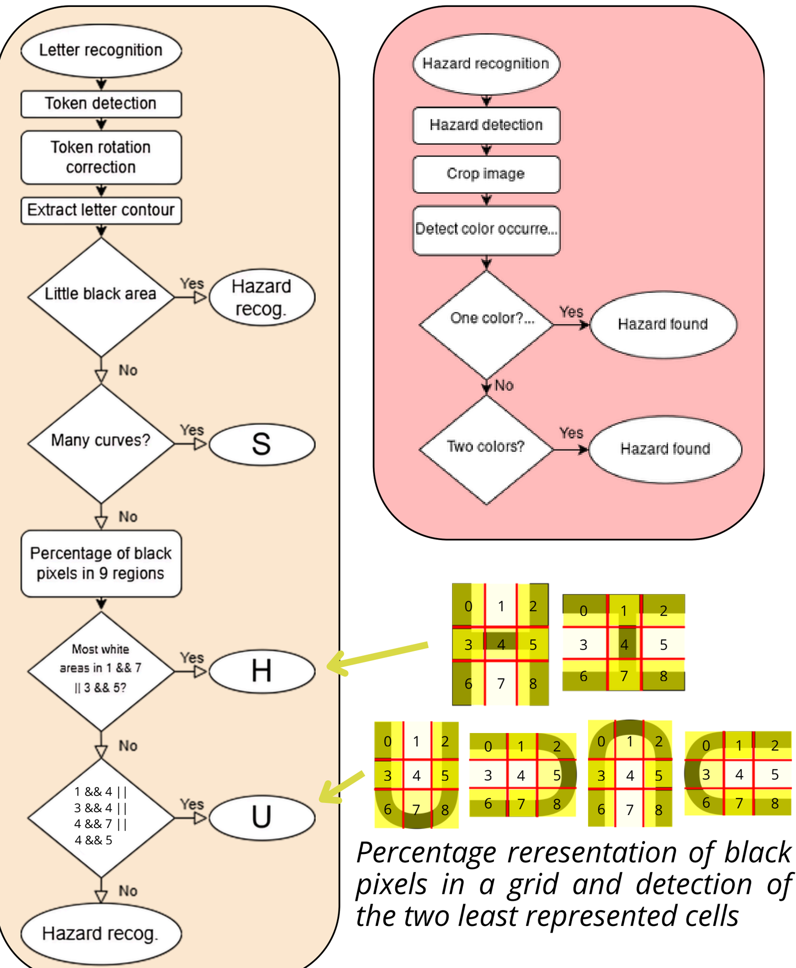
The color sensor detects the surface color directly beneath the robot. It is used to identify special tiles in the environment such as holes, swamps, passages and area passages which are later used for better mapping. This information is used to trigger behaviors such as stopping, reporting, or rerouting.

CAMERA

Cost: 2x 700

Two 64x64 cameras capture images of the space next to the robot and are used for identifying victims, hazards, and the floor from a distance. The image data is processed using OpenCV. These sensors are essential in the Erebus environment, where many victims are identified visually, and no other sensors provide the same level of detailed recognition. Floor detection significantly improves black hole detection success.

Token recognition



Percentage reresentation of black pixels in a grid and detection of the two least represented cells

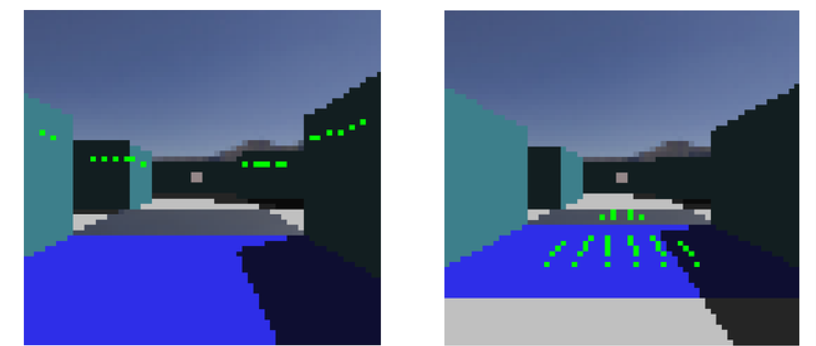
- After extensive research and testing, we developed a complex algorithm with geometric normalization that handles arbitrary rotations, occlusions, and have multiple conditions to ensure detection accuracy — such as the size of color regions, the position of a potential token, and more (see flowchart).
- Altought we initally explored ML solutions we found a purely OpenCV-based approach to be more effective.
- Floor-tile detection leverages both LiDAR space-related data and camera textures to ensure robust identification under varied lighting and viewing angles. It was beneficial for us to create this algorithm, because the color sensor in the center of the robot didn't always detect the black hole, as the robot wasn't always moving in the center of the tiles.
- For testing, we created a simple Python script that loads .wbt files, randomly rotates every token, and saves a copy of the map with the rotated tokens.

STREDNÁ PRIEMYSELNÁ ŠKOLA ELEKTROTECHNICKÁ PREŠOV

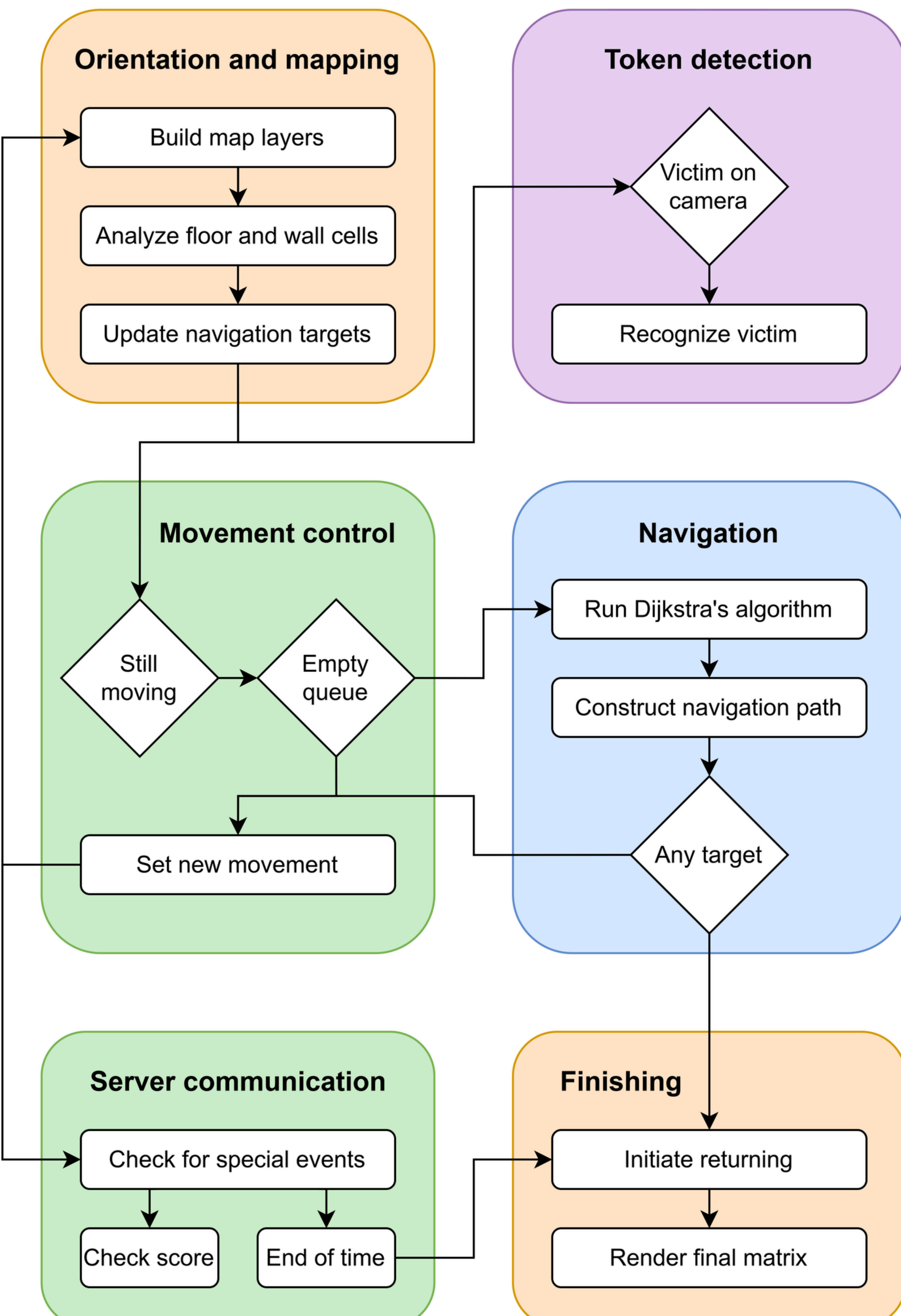
Wall and floor color matching of grid points

The algorithm enhances grid-based barrier representation derived from LiDAR data by classifying each barrier cell as a wall, obstacle, or wall with a token. This classification is based on the color of the camera pixel that corresponds to the real-world position of the barrier, achieved through projection equations that enable precise alignment of directly visible cells with the camera image. The identified wall types can be utilized to filter out obstacles from wall cells during the final matrix rendering or to prioritize exploration of walls containing tokens over plain walls. The same process is also applied to extract information about floor tiles.

Pixel matching feature



Main loop

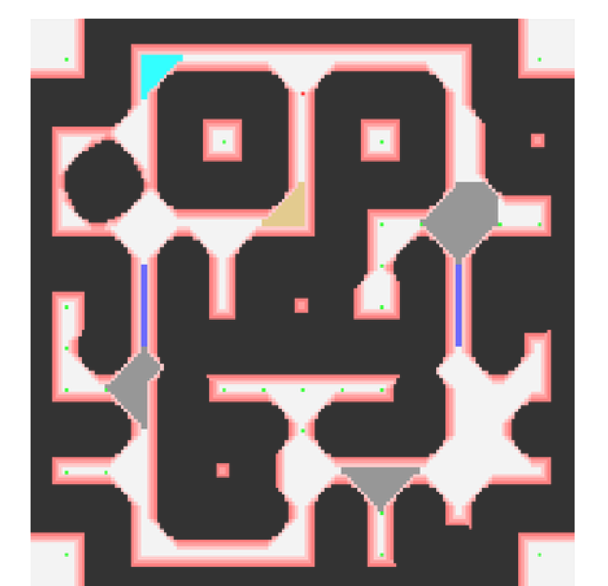


Implementation & Library Support: The entire system is written in modern C++20, with heavy use of the Standard Library, Eigen for high-performance array and linear-algebra operations, and OpenCV for various image-processing tasks. We began programming in January 2025 using Python, which worked for the Slovak round in February. Later, we switched to C++ for better performance and succeeded in the Croatian round in March.

Modular Main Loop: At its core is a task-driven main loop that registers events like sensor reading, mapping updates, path planning, motion control, camera recognition and communication as independent modules. These tasks synchronized via a lightweight event system, ensuring real-time constraints are met.

Multi-Layer Mapping: Taken LiDAR scans are first passed through a simple probabilistic correctness filter: points with a high likelihood are retained and lower-confidence measurements discarded. Those filtered points, along with GPS fixes, IMU orientation and camera vision, are then fused into several overlaid grids: a precise barrier grid distinguishing walls from obstacles; a fine-resolution barrier grid for exact calculations; a floor-tile layer for area perception; and a navigation-optimized grid that abstracts free space into a network graph. Each grid is updated only when new or changed data appear to minimize computational overhead. The final rendered map applies post-processing based on all grid layers to distinguish areas, match wall cells and floor tiles.

Path optimized grid



Path Planning & Navigation: Shortest-path queries are handled by Dijkstra's algorithm on the navigation grid, with a custom cost heuristic that increases penalty as proximity to walls or obstacles grows. Targets are chosen dynamically based on world-map priorities, and filters out low-value areas to focus exploration where it matters most.

Movement control: Once a path is determined, a motion-control module translates it into a queue of drive, rotate, and stop commands. Overall mission time is reduced through smooth velocity adjustments in turns and a lightweight lookahead that anticipates upcoming segments, cutting down on unnecessary braking and re-acceleration.

Server Communication & Game Logic: A dedicated communications module maintains a link with the game server, reporting acquired data for scoring, handling lack-of-progress alerts, querying game scores, and supporting all communication protocols abstracted behind a simple API.