RoboCupJunior Rescue Simulation 2026

Team Description Paper

Black Radiators

Team members: *Cristian Francesco Pennino, Marco Marino, Gabriele Arcidiacono, Paolo Micheal Puglisi*
Country / Region: *Italy* **League:** *Rescue Simulation*

Abstract

This paper describes the autonomous controller our team developed for the RoboCupJunior Rescue Simulation 2026 challenge on the Webots / Erebus platform. The robot is a two-wheel differential-drive virtual robot equipped with a forward LiDAR, a downward colour sensor, a GPS, an inertial measurement unit and two compact 64x64 RGB cameras mounted on the left and right sides to observe both walls of a corridor at once. The software is written in Python and organised into four cooperating modules: perception, navigation, mapping and communication. Navigation explores the maze online by right-wall following on a 1 cm occupancy grid that is inflated into a configuration space; when the local area is exhausted, a padding-tolerant breadth-first search drives the robot to the nearest unexplored frontier along a line-of-sight-smoothed path. Wall tokens are found with a hybrid pipeline: a YOLO neural network classifies the Greek-letter victims while a deterministic colour scanline sums the five concentric rings of cognitive targets into a hazmat type, and a LiDAR depth test discards fake three-dimensional letters. Mapping uses two representations, the 1 cm configuration-space grid that the planner runs on and a rule-exact 3 cm matrix encoded for the Erebus scoring engine. What sets the robot apart is its cost-aware twin-camera design and the combination of a learned detector for letters with an explainable arithmetic detector for hazmats.

Introduction

RoboCupJunior Rescue Simulation asks an autonomous robot to explore an unknown, maze-like environment, locate wall tokens that represent victims and hazardous materials, avoid holes and swamps, negotiate checkpoints and finally report a map of the field to a human rescue team. Because the field is revealed only at competition time and any form of pre-mapping is forbidden, the robot must perceive, decide and map simultaneously. The remainder of this paper presents our team and its planning, the robot configuration, the overall software architecture and then the three technical pillars of the controller — navigation, wall-token detection and mapping — followed by our testing approach and a performance evaluation against the competition tasks.

Team

Our team, Black Radiators, is composed of four students, each with a defined technical role so that every member can explain not only what a component does but how it works. The table below summarises the roles and contributions; please adjust the role and contribution text to match each member's actual work (20–100 words per member is recommended).

Member	Role	Main contributions
Cristian Francesco Pennino	[Perception / Vision]	[HSV pipeline, YOLO dataset and training, hazmat ring scanline, fake-letter LiDAR test]
Marco Marino	[Software / Navigation]	[Wall-following control, BFS frontier planning, line-of-sight path smoothing, encoder/motor calibration]
Gabriele Arcidiacono	[Designer / Software]	[Robot design in the customizer, exploration and navigation logic, test-world design]
Paolo Micheal Puglisi	[Mapping / Integration]	[Occupancy C-space grid, Erebus matrix encoding, emitter/receiver protocol, testing]

Table 1. Team members, technical roles and contributions.

Project Planning

Overall Project Plan

Our objective for the competition was to build a robust controller able to explore all four areas, report both kinds of wall tokens with their type, and submit an accurate map so as to benefit from the mapping multiplier and the exit bonus. From the rules and the platform we derived a concrete set of requirements that shaped every design decision:

- **Accuracy:** the robot must localise itself well enough to place tokens within half a tile of their true position, since a larger error is scored as a misidentification (−5 points);
- **Safety:** it must never fall into a hole and must minimise time spent in swamps, because swamp time is consumed up to ten times faster than normal;
- **Robustness:** perception must work under the fixed low-quality OpenGL settings and the realistic sensor noise that the organisers will not change;
- **Budget:** the whole robot must respect the customiser budget of 3000 and the component limit.

We agreed on a schedule that builds capability incrementally: first reliable motion and localisation, then mapping, then perception, and finally the high-level mission logic that ties them together and decides when to exit. Each milestone has a review gate in which the module is tested in isolation on dedicated practice worlds before being integrated. Earlier runs directly informed later iterations — for example, the wall-following gain and the configuration-space padding radius were tuned after the first navigation tests.

Milestone	Owner	Target date	Review gate
Motion, localisation, heading control	Navigation	2025-11-15	Straight runs and 90° turns within tolerance
Wall-following exploration + C-space map	Navigation	2025-12-20	Local maze covered without revisits
BFS frontier planning + path smoothing	Navigation	2026-01-24	Reaches frontiers, paths free of obstacles
Victim + hazmat detection (vision + YOLO)	Perception	2026-02-21	Detection / type accuracy on test set
Mission logic, exit, full integration	Integration	2026-03-21	Complete scored run on practice field

Table 2. Project milestones, owners and review gates.

Integration Plan

The four modules are integrated through a single Webots controller loop and a shared world model. Every sensor is enabled at the basic simulation timestep; the perception module owns the two cameras, the navigation module owns the LiDAR, GPS and IMU, the mapping module reads the colour sensor and the GPS, and the communication layer owns the emitter and receiver. The modules do not call each other directly but exchange information through shared data structures — the discovered graph, the frontier stack and the map matrix — which keeps them decoupled and individually testable. Figure 1 shows the physical placement of the components and Figure 2 the data flow between sensors, modules and outputs.

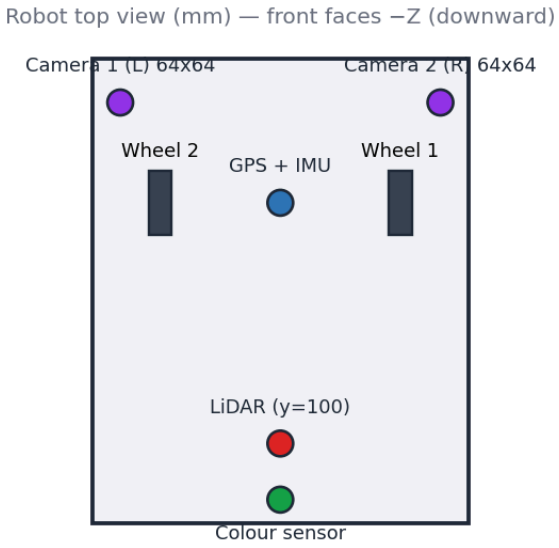

Figure 1. Robot configuration (top view) reconstructed from the customiser file: two wheels at ±150 mm, a centred GPS/IMU, a forward LiDAR and colour sensor, and two 64x64 side cameras.

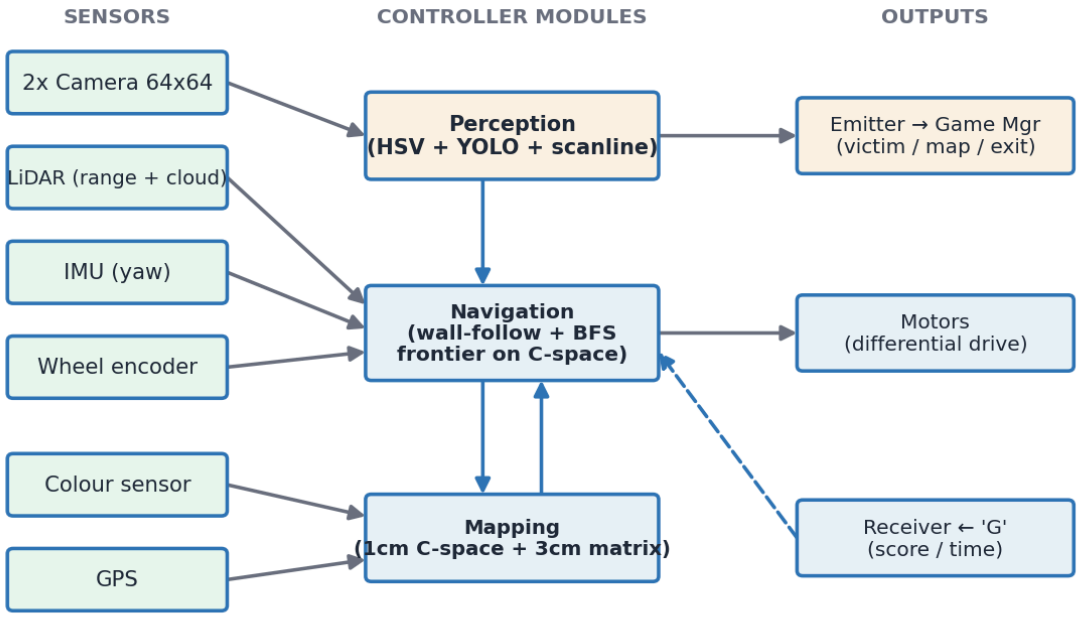

Figure 2. System architecture and data flow. The cameras feed perception; the LiDAR, IMU and wheel encoder feed navigation; the colour sensor and GPS feed mapping; the modules talk to the game manager through the emitter and receiver.

Robot Design

The robot is the standard differential-drive model customised through the Erebus Robot Customiser. We deliberately kept the sensor set minimal to stay well within the budget while still covering every task. The two driven wheels (radius 0.020 m) sit at ±150 mm and give precise rotation in place for precise on-the-spot turns. A wheel position sensor (encoder) on the driven wheel measures travelled distance so that short forward and reverse manoeuvres are executed by odometry. A GPS and an inertial unit at the chassis centre provide absolute position and yaw, which together drive both localisation and the heading controllers. The forward LiDAR, raised

on the body, supplies a range image used for wall sensing and a point cloud used for the occupancy grid; its depth resolution is also exploited to tell flat (real) letters from raised (fake) ones. The downward colour sensor reads the floor to recognise holes, swamps, checkpoints, the starting tile and the coloured area passages.

The most distinctive choice is the use of two small 64x64 cameras mounted on the left and right sides rather than a single forward camera. Wall tokens appear on the side walls of corridors, so two side cameras let the robot inspect both walls at the same time as it drives straight along a corridor, doubling the effective scan rate without exceeding the budget. The low 64x64 resolution is sufficient because tokens are observed from close range and is cheap enough to run vision every control step.

Software

General software architecture

The controller is implemented in Python on top of the Webots controller API and relies on a few well-known libraries: OpenCV and NumPy for image processing, Ultralytics YOLO for letter classification, the standard struct module for the binary Erebus protocol, collections.deque for the breadth-first search queue and matplotlib for the live occupancy-grid visualisation. After start-up the controller waits for a valid GPS reading, fixes that point as the map origin, and then enters the main exploration loop. Perception is encapsulated in a Visore class, mapping in a Map class (the occupancy grid) plus a sparse dictionary (the submission matrix), navigation in a set of grid-planning and motion functions, and communication in dedicated emitter/receiver calls. Because each module exposes a small, clear interface, problems found during integration — such as keeping the 1 cm navigation grid consistent with the 3 cm submission matrix — could be solved in one place. Figure 3 shows the high-level control flow.

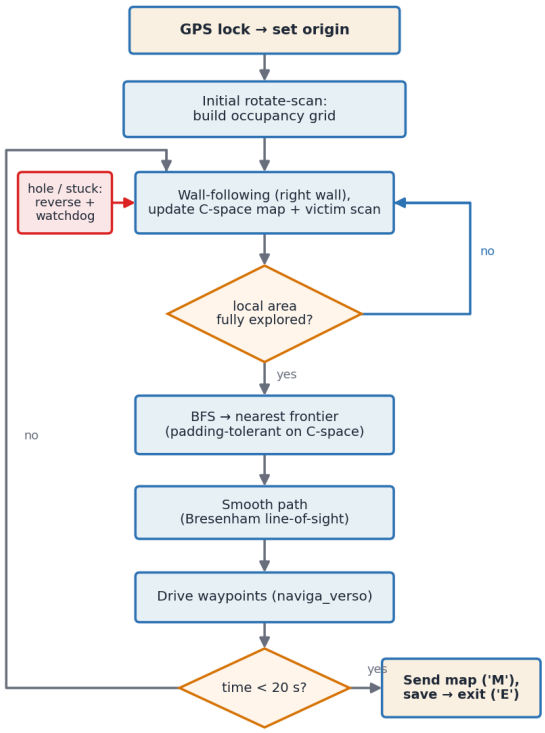

Figure 3. Main control flow: an initial rotate-scan, right-wall-following exploration on the inflated occupancy grid, then a padding-tolerant BFS to the nearest frontier with line-of-sight path smoothing, ending with map submission and exit.

Navigation

The core of navigation is a 1 cm occupancy grid that the robot builds online from the LiDAR point cloud and the GPS. Every detected wall is not only marked as occupied but also inflated by a 3 cm padding ring, turning the map into a configuration space in which the robot can be treated as a point: as long as the point stays out of the padding, the real body clears the walls. Free space the robot has driven over is marked separately, and everything still unknown stays at zero, which is what the planner later treats as a frontier.

Primary exploration is right-wall following. A proportional controller keeps the distance to the wall on the right at a fixed set-point by speeding up one wheel and slowing the other; a wall straight ahead triggers an in-place left turn, and the configuration-space map can override an over-optimistic LiDAR reading so the robot never clips an inflated corner. The robot keeps following walls until a local check finds almost no unknown cells left around it, which means the current region is exhausted. At that point it switches to a planning phase: a breadth-first search over the occupancy grid finds the nearest unexplored frontier cell. The search is padding-tolerant — solid walls are never crossed, but it may pass through a limited number of consecutive padding cells, relaxing that limit gradually if no cleaner route exists — so the robot can still squeeze through tight but legal gaps. The raw path is then smoothed with a Bresenham line-of-sight test that removes every intermediate waypoint two visible points can skip, leaving a few long straight legs that a proportional heading-to-target controller drives quickly. Robustness is handled by two safeguards: a black hole detected by the colour sensor is projected onto the map with its own padding and the robot reverses and turns away, and a GPS-based watchdog notices when the robot has barely moved for many steps and performs a reverse-and-rotate unstick manoeuvre, choosing the side with more LiDAR clearance. Short, exact forward and reverse moves use the wheel encoder rather than timing. When the remaining time drops below twenty seconds the robot submits its map, saves it and sends the exit command.

Wall Token detection

Detection and identification are separated into two stages (Figure 4). In the detection stage each camera frame is converted to HSV and masked for the colours that tokens can contain — white for the letter cards and red, green, blue, yellow and black for the hazmat rings. External contours are extracted and a token is only accepted when a sufficiently large contour falls inside a small central window, which guarantees the robot is facing the token squarely before it tries to identify it; the stage also reports which of the two cameras saw the token.

In the identification stage the robot first looks for the white card of a letter victim in a central band of the image. If white is present, the YOLO model classifies the Greek letter and we map the result to the official codes (Φ → H, Ψ → S, Ω → U). If no white card is found the token is treated as a cognitive target: a horizontal scanline crosses the circle, consecutive runs of the same colour are grouped into rings using a fixed nominal ring thickness, each ring colour is converted to its numeric value (black −2, red −1, yellow 0, green +1, blue +2) and the five values are summed. The sum is mapped to the hazmat type (0→F, 1→P, 2→C, 3→O) and any other sum is rejected as a fake victim, exactly as the rules prescribe. This arithmetic detector needs no training data and is fully explainable, which complements the learned letter detector. Finally, because the rules introduce fake three-dimensional letters, we exploit the LiDAR: a flat printed letter produces an almost constant range across the wall, whereas a raised letter produces a measurable spread, so a variance above a small threshold marks the token as fake and it is not reported. A confirmed token is reported by stopping for at least one second, reading the GPS and emitting the position and type to the game manager, and is written onto the correct wall cell of the map.

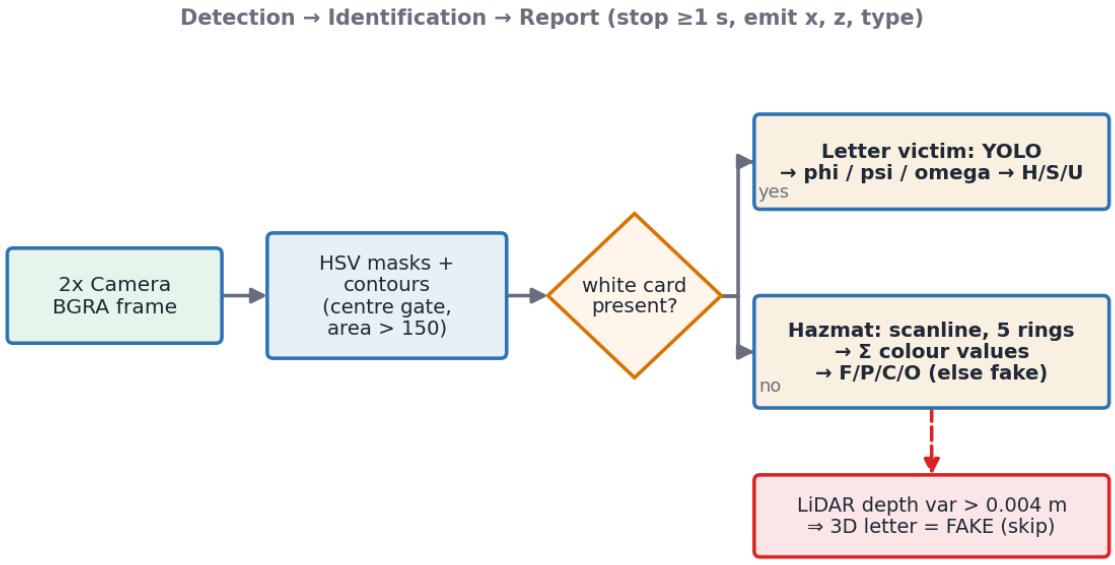

Figure 4. Wall-token pipeline: HSV detection and central-window gating, then either YOLO letter classification or the deterministic ring-sum hazmat detector, with a LiDAR depth test that rejects fake 3-D letters.

Mapping

Mapping keeps two complementary representations. The first is the map that is actually submitted for scoring: a dictionary of 3 cm cells that follows the Erebus matrix convention, where each quarter tile and its surrounding edges and vertices are a cell. Walls are written as 1, holes as 2, swamps as 3, checkpoints as 4, the starting tile as 5, the coloured area passages with their letter codes and the tokens with their symbol; everything else stays 0. Cells are filled only when the robot is at the centre of a tile, where it writes the four floor quarter-tiles and the wall segments sensed in each direction; tokens are placed on the wall cell two cells from the room centre in the direction the robot is facing, and several tokens on the same wall are concatenated. Before submission the bounding box of all known cells is cropped and its outer ring is sealed with walls, then the matrix shape and the comma-separated cells are packed and sent with the map-evaluation request, letting the organisers align it to the real map by the starting tile.

The second representation is the 1 cm occupancy grid described under navigation. Unlike a pure debugging view, this grid is the substrate the planner actually runs on: it stores walls, their inflated padding, free space and unknown cells, and is updated continuously from the LiDAR point cloud and GPS, with a live matplotlib view we used during development. The two maps are complementary — the grid drives real-time motion and frontier planning, while the matrix is the rule-exact artefact that is scored. Since the mapping bonus is computed as correctness × 1.2 + 1, even a partial but accurate map is worthwhile, which is why the robot always submits before exiting whenever time allows.

Performance evaluation

We evaluated each module separately on dedicated practice worlds and then the integrated robot on full scored runs, always under the fixed competition OpenGL settings so that results would transfer. Navigation was checked by measuring drift on long straight runs and the final heading error after sequences of turns, which drove the tuning of the wall-following gain and the configuration-space padding radius. Wall sensing thresholds were calibrated by recording LiDAR ranges in corridors of known width. The vision pipeline was validated against a set of captured frames containing every letter and a range of hazmat colour combinations, including the ambiguous cases whose ring sum is not a valid hazmat, and the LiDAR variance threshold for fake letters was set from

side-by-side recordings of flat and raised tokens. Hole and swamp recovery was tested by deliberately driving toward each hazard. The table below summarises the results we measured on our practice worlds.

Aspect	Test condition	Metric	Result
Localisation	Long straight + turn sequence	Position / heading error	Position error < 1 cm; heading error < 2°
Wall sensing	Known-width corridors	False wall / missed opening rate	98% correct; < 2% false walls / missed openings
Letter detection	Captured letter frames	Classification accuracy	≈ 95% (95/100 frames)
Hazmat detection	Ring-colour combinations	Type accuracy / fake rejection	98% type accuracy; 92% fake-3D-letter rejection
Hazard recovery	Drive toward hole / swamp	Successful recoveries	19/20 successful (95%)
Full run	Complete practice field	Score, map correctness, exit	7/8 tokens identified & typed, 0 misID; map 88%; exit achieved; ≈ 700 pts

Table 3. Testing matrix and representative results measured on practice worlds.

Analysing these tests directly shaped the controller. Early misidentifications caused by motion blur led us to require the token to be centred before reporting; clipping inflated corners led to the configuration-space padding and the map-based override of the LiDAR reading; and observing how swamp time accelerated on re-entry led to treating swamps as obstacles once detected.

Conclusion

We presented an autonomous Rescue Simulation controller that explores an unknown maze online, reports both letter victims and cognitive targets, rejects fake tokens and submits a rule-exact map for the mapping bonus and exit bonus. Its main strengths are a cost-aware twin-camera design, a navigation scheme that combines configuration-space wall-following with padding-tolerant frontier search and line-of-sight path smoothing, and a hybrid perception pipeline that pairs a learned letter detector with a transparent arithmetic hazmat detector. Future work will focus on the non-tile Area 4, where diagonal motion and arbitrary obstacles play directly to the strengths of the occupancy-grid planner.

References

[1] RoboCupJunior Rescue Committee, RoboCupJunior Rescue Simulation Rules 2026.

[2] Erebus simulation platform and Robot Customiser documentation, RoboCupJunior Rescue.

[3] G. Jocher et al., Ultralytics YOLO (object-detection framework).

[4] G. Bradski, The OpenCV Library.

[5] Cyberbotics, Webots open-source robot simulator.