RoboCup Junior - Rescue Simulation

RoboCup Eindhoven 2024

OBJETIVO

TEAM MEMBERS

David Icheng Wang Chou
Responsible for the robot mapping system; worked on designing new maps, while also helping with other functionalities in coding.

Marcos Menezes Nunes
Responsible for organizing the code into the different modules, integrating the code functions, and developing a non tile based navigation system with the RRT* algorithm, taking care of the robot's movement.

Sophia Yara Yano
Responsible for wall token detection, research for computer vision algorithms and design of the hardware and documentation contents(logo and models)

TIME LINE

2024...

2023-October

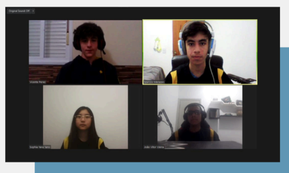
To focus on upcoming exams, João too left the team after the Robocup, and a new member joined in, forming our current team: David Icheng, Marcos Menezes and Sophia Yara.
At the end of last year, with a lot of experience, we participated in LARC/CBR again and became twice champions.

2023-July

After last year's regional competition, Vicente left, leaving the group with João Vitor, Marcos Menezes and Sophia Yara.
With the First Place in the national stage, we qualified for Robocup France Bordeaux, where we got Thirt Place.

2022

In 2022, with more experience, we participated again in LARC/CBR, and we got the First Place in the Rescue Simulation category. With the change from Cospace to Erebus, we learned a lot, improving the minimum path algorithms used in the previous year.



2021

In 2021, we created te team 73 RoBits, from Brazil, São Paulo, for the Cospace Rescue Simulation category, with members: João Vitor Vieira, Marcos Menezes Nunes, Vicente Balbino Perez and Sophia Yara Yano.
Our first competition was LARC/CBR, in which we placed second. After that, we participated in the Asian Pacific Robotics Competition, in Aichi Japan, and got awards for Best Presentation and Superteams Challenge.

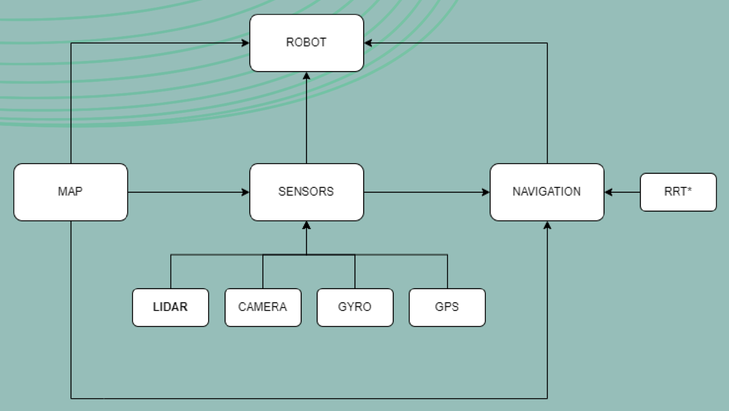
MAIN LOOP

- **Architecture**

The code is divided into many **modules** i.e. python files each of which is composed of multiple classes, specialized in different functions.
It keeps the code organized and helps the debugging process. If a new function gets added to the navigation module, for instance, the team could disable the camera class, making it easier to test this new function. Moreover, if a problem occurs, turning off different modules makes it easy to get to the root of the problem, speeding drastically the debugging process.

- "**explore**" - Find new unexplored areas of the map and go to them.
- "**collect**" - Once a wall sign is seen through the camera, walk to it and try to collect after identifying its type.
- "**LOP**" - Recalibrates the robot.
- "**stuck**" - Divided into two levels. The first one tries to walk back to the last position, marking the current one as explored. If it ends up stuck again for 20 seconds, the second level forces a LOP.
- "**while**" - Forcibly ends the current tick. If called multiple times ends up at a 'LOP' or 'stuck' action.

- **Programming Language:** Python

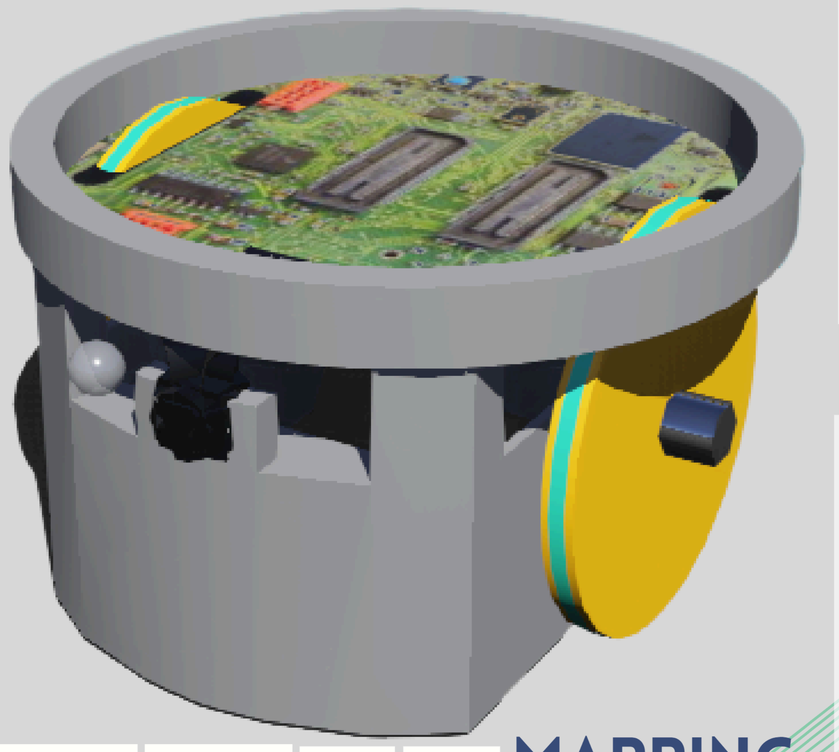


- **Main Loop**

The robot acts based on a list of different **actions**. The main ones are "**explore**" and "**collect**", called **routine** actions. There's also some **emergency** ones triggered by situations that could crash or delay the software, such as a **LOP** (Lack of Progress), being **stuck** at a wall or obstacle, or even entering an **endless loop** that would not tick the simulation.

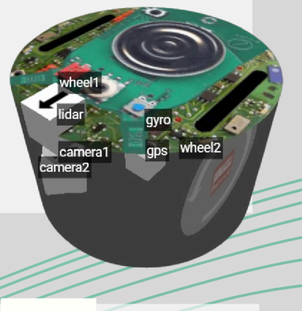
73 ROBITS

BRAZIL



HARDWARE GENERAL CONTENTS

- **Wheels (x2):** Positioned on the right and left side to make the robot move.
- **LiDAR**: Placed in the front-top part, so that it doesn't get blocked by the robot. This replaced **distance sensors** as it's way more versatile with **512 rays** of horizontal resolution instead of 1.
- **GPS:** Placed in the center so that it's distance to the robot's border is consistent, allowing to easily retrieve the robots radius when **navigating**.
- **Gyro:** Also placed at the center to avoid problems with **measuring angles**.
- **Cameras (x2):** Two 120x40 resolution cameras are positioned at the front with an angle between them such their images complete each other, obtaining almost **180° field of view**. This replaced color sensors as their images return every needed color of the map.

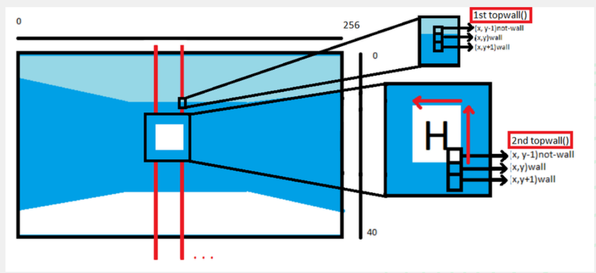


TOKEN DETECTION

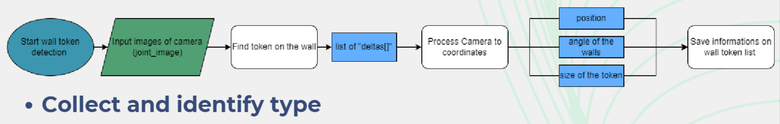
Cameras
LiDAR

- **Identify token on wall and use LiDAR to get position**

The first process is to identify the token on the walls. To do this, the algorithm iterates through all pixels detected by the cameras, verifying three vertical pixels, searching for the pattern **"not wall-wall-wall"**. All pixels satisfying this are then sent as a list to the next processing step.

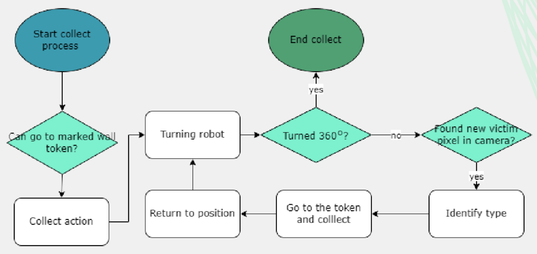


For each pixel position, using trigonometry, the robot can get the angle of the token in relation to the robot, using the closest LiDAR ray to acquire the token world position and its wall's angulation. All this information is stored and used in the collect process.



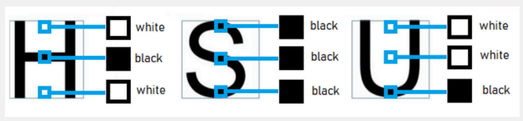
- **Collect and identify type**

At any time, if there's a marked wall token that the robot can directly go to, it stops exploration and gets closer to it, starting the collection process. The robot then rotates until it completes a full turn or the token appears on the camera's image. If the latter happens, the robot identifies its type and emits it.



Victims and hazmats are differentiated by verifying the token's size, colors and key pixels.

To identify the type of victims:



Search for the **highest, lowest and middle** pixels to differentiate the subtypes

To identify the type of hazmats:

Verify the colors of the hazmats, counting the number of each **yellow, black, and red pixels**

MAPPING

Cameras
LiDAR
Gyro
GPS

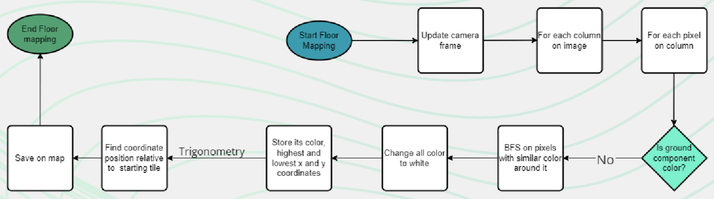
- **Walls**

The mapping of walls and obstacles is done by using **LiDAR rays**. Each of the 512 horizontal rays gives information about the position of the object it encounters, allowing the robot to store it.

As the navigation is not tile based, to achieve the best performance, **the map it uses cannot be not tile based** as well. To achieve this, it is used a resizable NumPy array grid, where each array entry stores a **list of points around a position**. That way, when trying to navigate to somewhere, it's easy to check if there's a wall point that could trouble the movement of the robot.

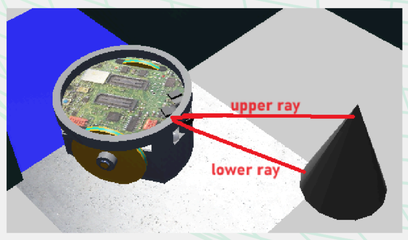
- **Floor Tile**

The robot views the floor through its **cameras**. Since it will not pass through the whole map, the robot determines the connection tiles, blackholes and swamps positions **right when they appear on the camera**. This can be done using their positions and size on the captured frame, in combination with the angulation of the camera.



- **Obstacles**

The robot uses **two vertically different rays of LiDAR** to detect the **shape of of obstacles**. While the ray parallel to the floor detects the point above the robot, the lower one detects the distance to the lower part of the obstacle, making the robot able to **avoid different shaped obstacles**, like conical or spherical ones.



NAVIGATION

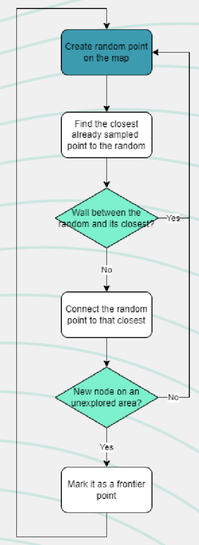
Gyro
GPS

For the navigation and exploration on the "**explore**" action we decided on using a **RRT*** (RRT-Star), a non tile based path planning algorithm (image on the right).

From our past experiences, a tile based navigation using **A*** (A-Star), **Dijsktra**, and alikes. could cause many corner cases, especially around Area 4 and obstacles, like paths with non-trivial angles to traverse.

By continuously turning random coordinates into nodes, the **RRT*** can find points on areas not yet explored. It is simple but highly efficient, quickly finding paths in very complex spaces.

Part of the strategy involves using two different **RRTs,** a **global RRT**, which doesn't reset throughout exploration, and a **local RRT**, reset every time the robot tries to find non explored areas. That increases the amount of points closer to the robot.



The areas labeled as explored are the ones **seen through the camera's images** that don't contain anything useful. This excludes the possibility of exploring areas that don't contain anything new or important to the robot, such as wall signs and colored tiles, decreasing significantly the time to run the full map.

Scan our QR Code to access the robot's source code, performance evaluation spreadsheet and other documents