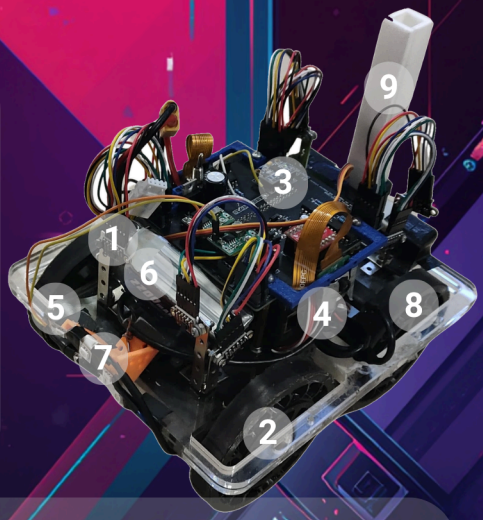
Maze_Buonarroti

ITT M. Buonarroti 2026

RCJ Rescue Maze

1. ToF Distance Sensors
2. DC Motors
3. Custom PCB
4. Raspberry Pi 5
5. Plexiglass Frame
6. Lipo battery
7. CV Cameras
8. 3D Printed Wheels
9. 3D Printed rescue kit dropper



The Team

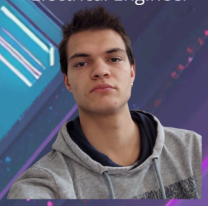
Developer

Matteo Giongo

Capitain

Developer and Mechanic

Federico Litterini



Electrical Engineer

Devid Montibeller

Vice Capitain

The code:

The robot was programmed in C++, a low-level language that allowed us to interact more effectively with the various hardware components of the robot. We implemented hyper-optimized programming and High Performance computing techniques to minimize execution latency. Buffering and multithreading are used to perform 0 delay calculations. Here is the code for our hardware thread to bufferize all of our sensor values and sync the motor outputs to the calculated values.

```
void Robot::syncHardware()
{
    uint8_t current_mask = state.sensors_done;

    for (int i = 0; i < static_cast<int>(Sensor::COUNT); ++i)
    {
        const double val = static_cast<double>(sensors[i]->readRangeContinuousMillimeters());

        const bool targetIdx = (current_mask >> i) & 1U;
        state.sensors[i][!targetIdx] = RobotState::clean_value(val);

        current_mask &= ~(1U << i);
        current_mask |= static_cast<uint8_t>((1 - targetIdx) << i);

        const bool gyroSlot = (state.miscSensorsReady & static_cast<uint8_t>(MiscSensors::GYRO)) != 0;

        float gyroFloatResult = 0.0f;
        gyro.getAngle(static_cast<int>(GyroAxis::X), &gyroFloatResult);

        double deg = static_cast<double>(gyroFloatResult);
        deg += 360.0 * (deg<0.0);

        state.gyroAngles[!gyroSlot] = deg;
        COMPILER_BARRIER();
        state.miscSensorsReady ^= static_cast<uint8_t>(MiscSensors::GYRO);
    }

    COMPILER_BARRIER();
    state.sensors_done = current_mask;

    motors.setSpeed(state.targetSpeedLeft,
                    state.targetSpeedRight,
                    Constants::MAX_MOTOR_SPEED);
}
```