Motion planning and target searching in complex environments

Salvatore Zammuto

Robotics project - Master's Degree in Computer Engineering - University of Palermo

Abstract

This report contains the presentation of the Robotics exam project of the Master's Degree in Computer Engineering at the University of Palermo. The showcased works consists in the construction, programming, and simulation of a robot that uses the information coming from its surrounding environment to navigate to a target location through a complex environment. The robot mainly relies on the wheels' position sensors and LIDAR devices to navigate through a maze and avoid obstacles using the surrounding walls as reference points. Once out of the maze, the simulation is designed to trigger the Simultaneous Localization And Mapping (SLAM) mode, where the robot is capable of evaluating the path to follow according tho the video feedback of the camera mounted on it, which localizes and outlines a track through the detection of known landmarks within its field of view. Each of the modes (aiming at the target, avoiding obstacles, and SLAM) is also autonomously computed, as is the strategy that triggers the mode to be activated based on the robot's state and its surroundings.

1 Introduction

"I have thrust myself into this maze,
Happily to wive and thrive as best I may."

-William Shakespeare

Since the birth of the first robots, the goal-oriented autonomous navigation has always been a fundamental aspect of the field [9]. This is actually the very first motivation for the creation of earliest robotics systems: nowadays, there is an ample literature on the navigation problem in all sorts of environments, with a number of different approaches for it, form the probabilistic robotics, Artificial Intelligence techniques [3] in terms both of computer vision and robot position and path estimation. The methodology presented here arises from all of such methodologies in order to present an algorithm that enables the robot to both autonomously move around a maze with walls and obstacles and perform the mapping of the path to follow even when the main structure of the environment is missing; in this situation, where the SLAM action is activated, localization occurs through the landmarks recognized by the robot's camera.

In addition, the robot is also able to establish which of the different scenarios it is in, and to consequently apply the appropriate strategy for each of them.

1.1 Materials and methods

The project was entirely built on Webots (vR2022a) robot simulation software [11] belonging to Cyberbotics Ltd., a subsidiary company of the EPFL University. Webots is a multiplatform and open source application that includes all of the required tools to the creation and simulation of the robots and the world they live in. Each aspect of the simulation can be customized: from the physics engine, to the environment textures and the robot's components. Webots also supports ROS (Robot Operating System) and multiple APIs for programming the controllers in languages like C, C++, Python, Java, and Matlab.

In our case, the robot's controller is entirely written in Python (v3.8), taking advantage of its numerous libraries for scientific computing and support to the complex computations required by this particular type of problem.

2 The world's components

2.1 The robot

The chosen robot's model is GCTronic's e-puck [1]. The robot has been 'manually' assembled in Webots, i.e., the available default PROTO was not used directly, rather, it was built upon a generic Robot node resembling the exterior and technical features of an e-puck (implemented as child nodes). This choice offers bigger customization capabilities, especially in terms of the embedding of auxiliary devices as the LIDAR and camera, which would be otherwise limited both in compatibility and number.

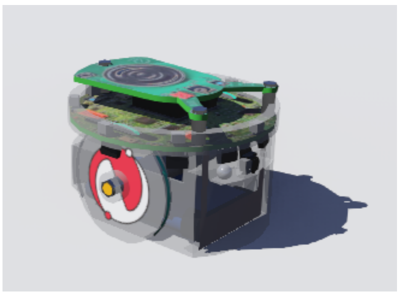
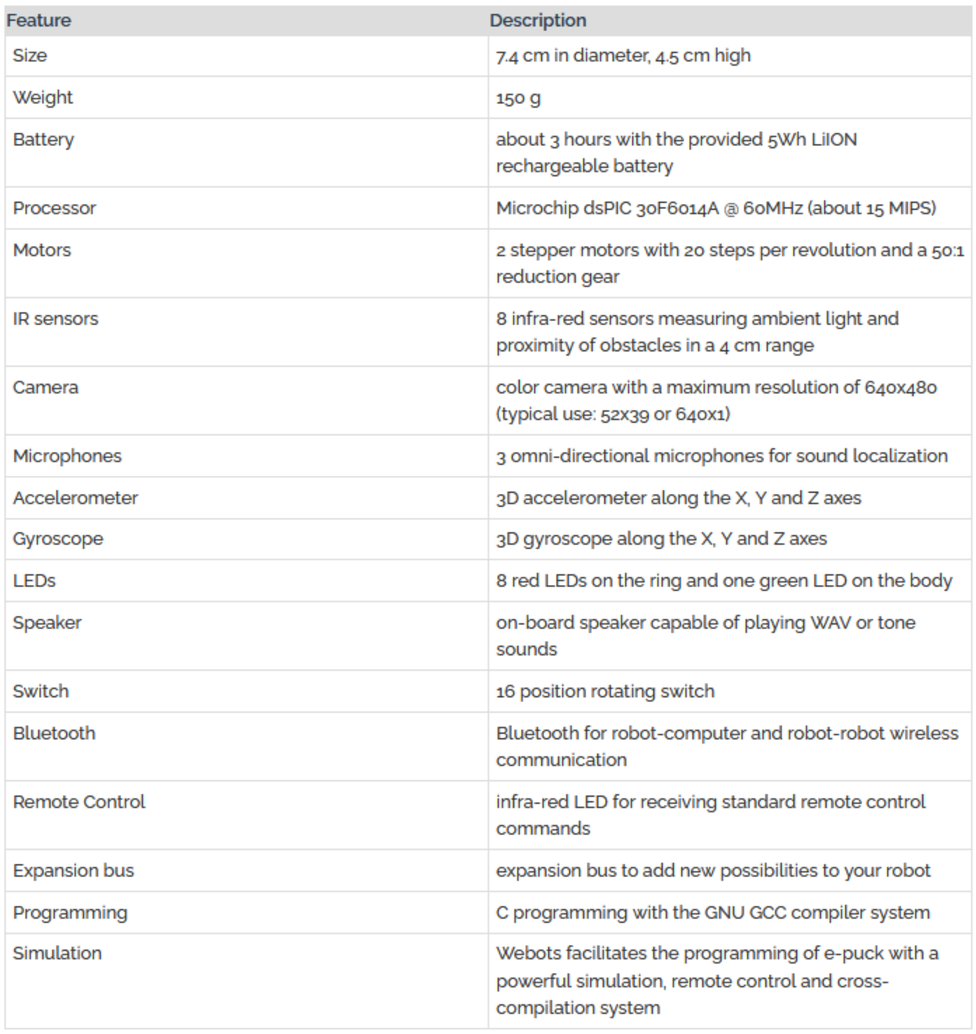


Figure 1: E-puck. Despite its structural simplicity, the robots presents the support for a wide variety of devices, from the motors for differential driving, to the proximity and position sensors.

The characteristics of the robot have therefore been explicitly implemented in the Webots model, on the basis of the information provided by the software's documentation [2] (Figures 2 and 3).

The flexibility of the e-puck ease our way into the implementation of complex procedures mainly due to the reduced computational burden of simulation, and therefore allow a better focus on the visualization of the action for the designed



Feature	Description
Size	7.4 cm in diameter, 4.5 cm high
Weight	150 g
Battery	about 3 hours with the provided 5Wh LiION rechargeable battery
Processor	Microchip dsPIC 30F6014A @ 60MHz (about 15 MIPS)
Motors	2 stepper motors with 20 steps per revolution and a 50:1 reduction gear
IR sensors	8 infra-red sensors measuring ambient light and proximity of obstacles in a 4 cm range
Camera	color camera with a maximum resolution of 640x480 (typical use: 52x39 or 640x1)
Microphones	3 omni-directional microphones for sound localization
Accelerometer	3D accelerometer along the X, Y and Z axes
Gyroscope	3D gyroscope along the X, Y and Z axes
LEDs	8 red LEDs on the ring and one green LED on the body
Speaker	on-board speaker capable of playing WAV or tone sounds
Switch	16 position rotating switch
Bluetooth	Bluetooth for robot-computer and robot-robot wireless communication
Remote Control	infra-red LED for receiving standard remote control commands
Expansion bus	expansion bus to add new possibilities to your robot
Programming	C programming with the GNU GCC compiler system
Simulation	Webots facilitates the programming of e-puck with a powerful simulation, remote control and cross-compilation system

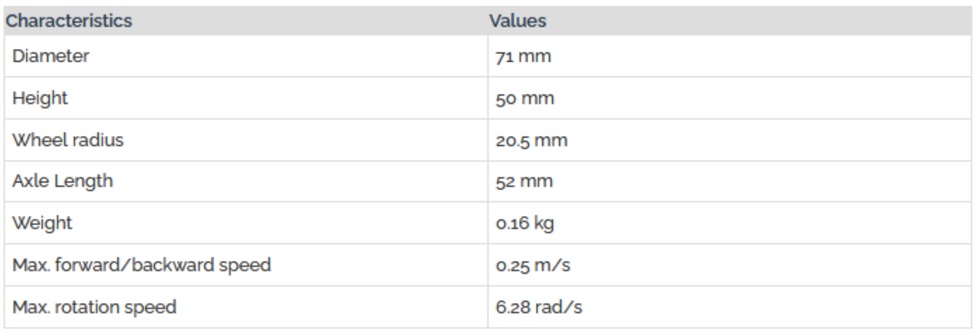
Figure 2: Components and functionalities natively supported by the e-puck model.

algorithms, still maintaining shared similarities with the real, practical scenarios of autonomous navigation tasks.

2.1.1 Devices

As we have just seen the devices supported by the e-puck are multiple, from the motors to the proximity sensors (Figure 4), to the camera, accelerometer and gyroscope.
Particularly , those used in this project are:

- motors: required for the rectilinear and rotational movements of the robot;
- position sensors: one for each wheel, they register their rotational velocity



Characteristics	Values
Diameter	71 mm
Height	50 mm
Wheel radius	20.5 mm
Axle Length	52 mm
Weight	0.16 kg
Max. forward/backward speed	0.25 m/s
Max. rotation speed	6.28 rad/s

Figure 3: Physical features of the e-puck.

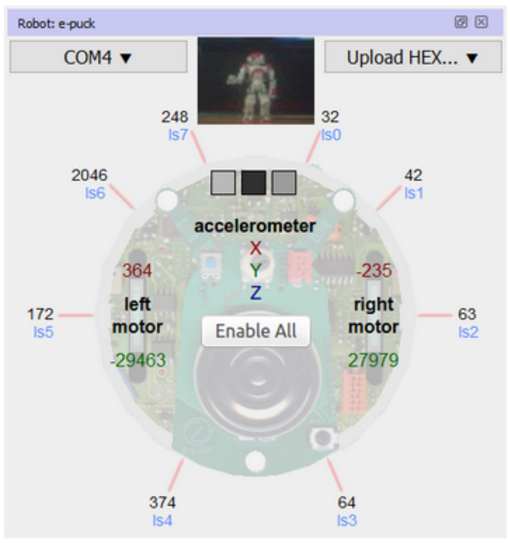


Figure 4: Positioning of the motors and proximity sensors in the robot's chassis.

allowing an estimation of the travelled distance;

- LIDAR (Light Detection And Ranging): outlines the environment within its field of action through laser scans;
- camera: provides the real-time video feedback of the frontal perspective of the robots; its object recognition feature enables to estimate the distance from the landmark and the path they delineate according to their relative position in the captured image.

Out of all of those devices, the LIDAR is the only one that is not supported by the standard e-puck model, but that can nonetheless be added to our generic Robot node. The LIDAR offers a wider range and better precision of the default position sensors, and, above all, it allows to establish a density metric on the laser feedback in terms of the number of rays belonging to a certain beam, which in turn results in a more accurate obstacle detection and avoidance strategy. The

LIDAR of our Webots world is set with a maximum resolution of 512 and field of view equal to π.

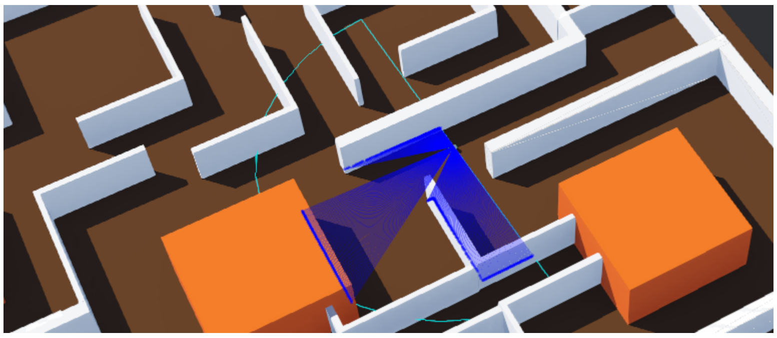


Figure 5: Example of LIDAR scan inside the maze.

2.2 The environment

The complex environment where the robot needs to navigate is composed of a main maze that is linked to the target zone through an open corridor (Figure 7). The robot start from on end of the maze (Figure 6) and has to exit it in order to reach the rubber duck located in a block that isolated from the structure (Figure 9). In the world, there is also an area that is free (of walls) (Figure 8) just outside the maze, that connects its exit to the block where the target is located.

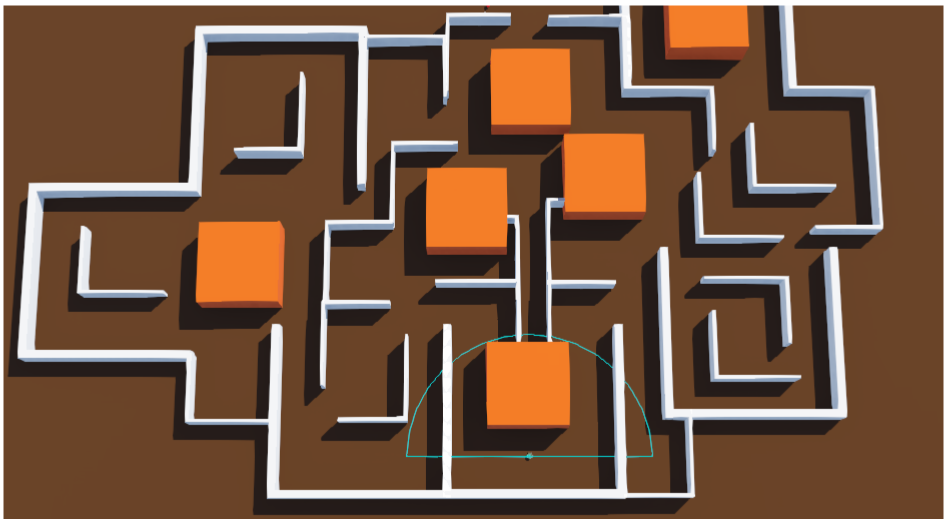


Figure 6: Detail of the main maze.

This is the area where the robot needs to trigger the SLAM mode to locate itself based on the landmarks recognized by the camera, as the LIDAR information, being mainly used for obstacle avoidance tasks, are not sufficient to determine a final path in an open environment as that of Figure 8.

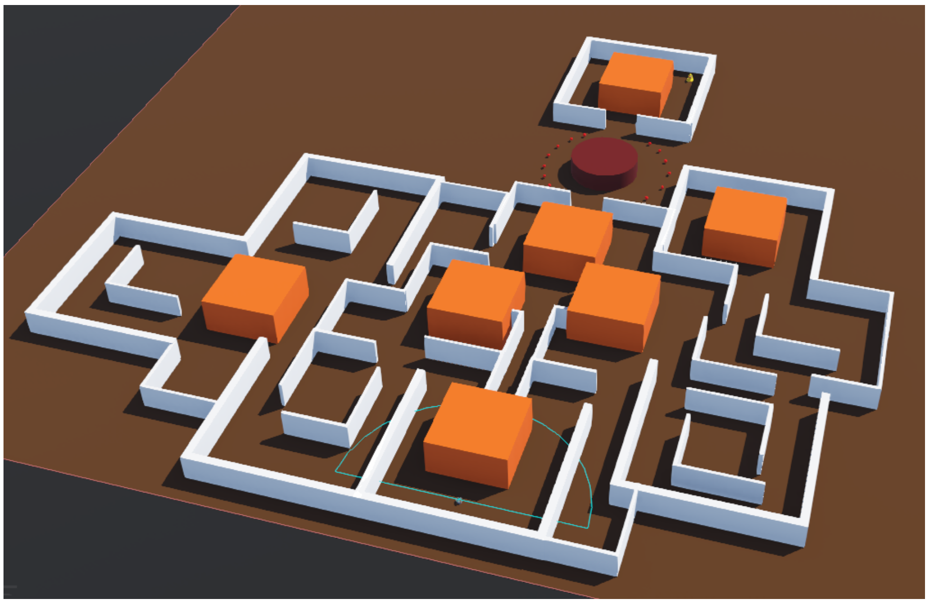


Figure 7: Aerial view of the robot's environment. The robot (for which the LIDAR's field of view is visualized) starts from the lower section of the maze outlined by white wall and orange obstacles; once it gets out, it enters the SLAM zone, with a cylindrical red obstacle and spheres (the landmarks). Surpassing this area results is the robot entering the final block where the target (rubber duck) is located.

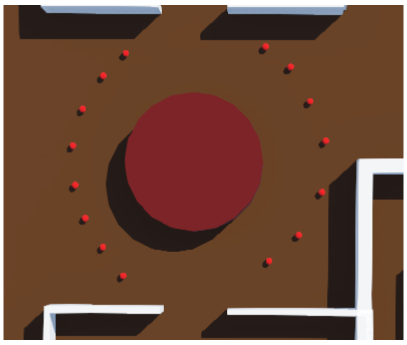


Figure 8: SLAM zone.

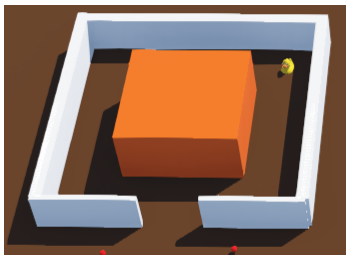


Figure 9: Final block that contains the target position, that of the duck.

3 General description of the behavior algorithm

As previously mentioned, if the main task of the task of the robot is that of reaching the target position, the different scenarios it can found itself in need a separate management as of the behavior to follow in each one of them. These behaviors (action modes) are 4 in total, defined as:

- DEFAULT: the robot moves in a straight line towards the goal, potentially after having rotated in order to get to the right direction;
- AVOID_OBSTACLE: the LIDAR detected an excessive proximity to an obstacle, which therefore needs to be avoided by stopping (to prevent collision) and directing towards an obstacle-free zone;
- FLANK_OBSTACLE: the robot has not the obstacle in front anymore and can proceed to flanking with the objective of surpassing it;
- SLAM: the moment that the robot exits the maze and the camera detects a minimum number (3) of landmarks, the SLAM mode performs localization and path planning based on the reference points.

The controller contains all of the information about the strategy to apply ech of those behaviors, which are triggered according to a constant evaluation on the system's state. The mode choice, which also describes the robot's general behavior during its autonomous navigation, is described in the pseudocode 1.

Algorithm 1 navigate and reach target

```
initialize robots' motors, devices and localization variables
initialize ACTION_TYPE to DEFAULT
while simulation is running do
    parse and store current values of position, orientation and velocity
    parse and process LIDAR data
    if near obstacle is detected then
        go into AVOID_OBSTACLE mode
    end if
    if ACTION_TYPE == DEFAULT then
        if robot is not oriented towards goal then
            rotate towards goal direction
        end if
        go forward
    end if
    if ACTION_TYPE == AVOID_OBSTACLE then
        determine direction of rotation
        rotate away from obstacle
        if robot is parallel to obstacle then
            go into FLANK_OBSTACLE mode
        end if
    end if
    if ACTION_TYPE == FLANK_OBSTACLE then
        go forward
        if lidar data shows obstacle is now distant then
            go back on track
            go into DEFAULT mode
        end if
    end if
    if ACTION_TYPE == SLAM then
        parse recognized objects form camera data
        do predict stage of EKF algorithm
        for each recognized landmark do
            get relative position to object with camera and lidar data
            update stage of EKF
        end for
        if recognized landmark is not detected as obstacle then
            determine path using landmark as reference point
            move to computed location
            if last landmark is detected then
                go into DEFAULT mode
            end if
        end if
    end if
    if SLAM landmarks are recognized by camera then
        go into SLAM mode
    end if
end while
```

4 Sensor data processing

At the start of each i-th iteration of the simulation, before determining the action mode and activating it, a preliminary preprocessing phase is required, where all of the sensor data are retrieved and elaborated as to dispose of real-time parameters that can be used to determine the state of the system in the current and following iterations.

4.1 Position, orientation and velocity

Once the information about current position, orientation and velocity of the robot are read, the first, step, propaedeticus for the initialization of the DEFAULT mode and those of obstacle avoidance, is that referred to the the evaluation of the $\dot{\theta}$ angle, between the robot's frontal direction and the line that connects it to the target. Particularly, this parameter is determined starting from θ, evaluated on the coordinates of the robot and those of the target:

$$\theta = \text{atan2}\left(\frac{y_{target} - y_{robot}}{x_{target} - x_{robot}}\right) \tag{1}$$

In the case that the computed value in (1) was negative, it will be appropriately normalized by summing 2π to it so that it is brought back to the $[0, 2\pi]$ interval.

Once θ is found, knowing (from the sensors) the initial orientation of the robot θ_s, $\dot{\theta}$ can be found by difference; this is the angle that the robot needs to rotate on itself in order to get aligned with the target:

$$\dot{\theta} = \theta - \theta_s \tag{2}$$

This $\dot{\theta}$ is therefore the rotation angle of the robot in the DEFAULT mode.

It also comes in handy to compute the distance travelled by the robot and its position according to the *dead reckoning*, through the current parameters' values ($\overline{ps}_i$) from the position sensors (one for each wheel) and the previous ones ($\overline{ps}_{i-1}$). We start from their difference,

$$\Delta\overline{ps} = \overline{ps}_i - \overline{ps}_{i-1} \tag{3}$$

and the travelled distance is computed based on r_W, the robot's wheel radius, which in the e-puck is equal to 20.5 mm:

$$\Delta x = \Delta\overline{ps} \cdot r_W \tag{4}$$

Similarly, it is also possible to estimate the reached position $\overline{x}'$ from the current one and knowing the covered distance, as a function of the velocity $\overline{v}$ registered by the sensors and the time interval of one iteration dt, corresponding to the timestep of the Webots world, set at 32 ms:

$$\overline{x}' = \overline{x} + \overline{v} \cdot dt \tag{5}$$

The same exact line of thought goes for the robot's orientation.

4.2 LIDAR data

The LIDAR data consist in the angle and distance of each of the scan rays [10]. Once read and stored, they undergo a slightly more elaborate processing phase, aimed at extracting additional information on the morphology of the external environment, therefore improving the precision of the movements that encode the complex behaviors of the robot.

Specifically, the raw data from the LIDAR are fed to a Hough Transform [8] that determines the exact correspondence between the 'useful' rays (those associated with a non-infinite distance, i.e., outside of the field of view of the device) and the corresponding angles. The most frequent associations are those used to mark the distances detected by the robot. The implementation (within the controller) of the HT was realized based on the source code of the OpenCV Python module [6].

The transformed data are once again refined, excluding erroneous or duplicate results through a single-linkage threshold clustering of the function *fclusterdata* of the module *scipy.cluster.hierarchy*. For each result, flat clusters are computed and the corresponding centroid is found, that related to a specific wall detected by the LIDAR.

The controller then proceeds on iterating on each extracted centroid (wall), and, in the case of proximity detection to one of them, stops the robot as to prevent the imminent collision, subsequently triggering the first of the two obstacle avoidance strategies, the AVOID_OBSTACLE mode.

5 Action modes

The following paragraphs will describe in detail each of the individual robot's behavior for every scenario, focusing on the implementation of the subroutines that compose the robot's navigation and localization.

5.1 DEFAULT

The DEFAULT mode is the main one, which gets fired at the start of the simulation and every time no other 'special' situation presents itself. In this behavior, the robot, that is aware of the direction of its target, moves in a straight line with maximum speed towards it. In the case that the frontal direction of motion does not overlay with the one that connects the robot and the target, the machine rotates around its own central axis of an angle $\dot{\theta}$ evaluated with expression 2 in order to subsequently enable the aforementioned straight motion, as described in Figure 10.

The algorithmic procedure that implements this mode, described in [4], is depicted in the flow chart of Figure 11.

In an environment that is free of impediments, just as that of 10, this is already a complete behavior that allows the robot to safely reach its objective. In complex scenarios, just as ours, other modes need to be incorporated as to deal with the presence of obstacle and absence of reference points; those modes will intertwine with the DEFAULT one.

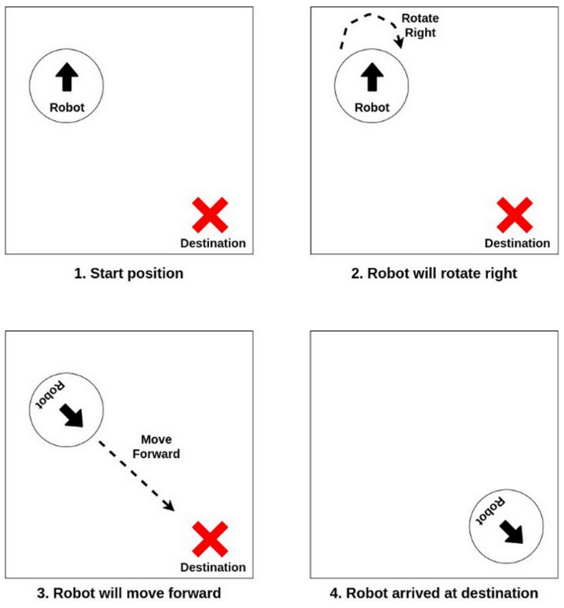


Figure 10: Visualization of the DEFAULT behavior of the robot for target reaching.

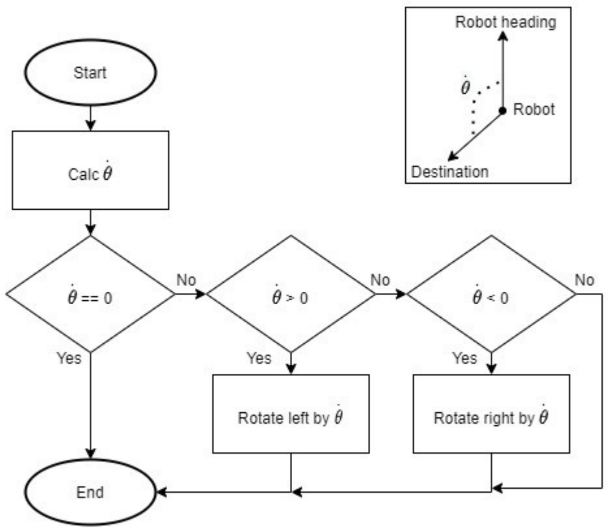


Figure 11: DEFAULT mode algorithm.

5.2 AVOID_OBSTACLE

This mode is triggered when the robot has detected an obstacle or wall in front of itself and needs to change direction in order to avoid it.

Once entered in AVOID_OBSTACLE, the controller evaluates the (normalized) angle θ_{obst} between the LIDAR centroid of the obstacle and θ_s as their difference and simultaneously extracts the LIDAR values from the terminations of its field of view (if the device resolution is set to 512, as in our case, those are the values in the intervals $[0, 30]$ e $[482, 512]$). This is done with the objective of determining the best steering direction, interpreted as the one that brings the robot into a 'freer' area with respect to the other. The sensor reading are averaged and their values compared as to obtain the rotation direction.

As the robot steers, the θ_{obst} decreases. As soon as this angle is 0 (or at least below a certain threshold), it means that the frontal direction of the robot is now parallel to the obstacle and it can proceed flanking it, i.e., go into the FLANK_OBSTACLE mode.

5.3 FLANK_OBSTACLE

Here, the obstacle is no longer in front of the robot, that now needs to surpass it to get back to the DEFAULT behavior.

The robot proceeds at maximum velocity flanking the obstacle and cross-referencing the information coming from the LIDAR about the proximity to it and the other detected impediments, with a special attention to the closest one. Once a sufficient distance from the obstacle is detected, the DEFAULT mode is once again triggered. In this situation, the HT, by construction, may (erroneously) detect an early overstep of the obstacle; this problem is solved by forcing the robot to move in a straight line for a certain number of fixed, additional iterations, after which it can be considered safe to assume that the robot has indeed gone over the impediment.

5.4 SLAM

During each of the previous action modes, the robot also constantly monitors the video feedback of the camera mounted on it, checking if it captures recognizable objects. As long as it is located within the maze, the LIDAR and the other sensors are enough for the autonomous navigation; however, as soon as the robot peeks at the exit, it starts picking up the landmarks from the camera, which now become the new reference points, necessary to assure self-orientation, motion planning and obstacle avoidance in this special area of the world.

For each landmark (red sphere), we calculate the distance of the robot from it based on its position on the frame captured by the camera. Particularly, if w_c and fov are respectively the frame width and the field of view of the camera, and x_h is the horizontal position of the object in the image, we first evaluate the angle between the object and the robot as

$$\operatorname{atan2}\left(\frac{1}{2}w_c - x_h, \frac{\frac{1}{2}w_c}{\tan\left(\frac{1}{2}fov\right)}\right) \tag{6}$$

and then this values is used to extract, from the LIDAR reading the distance of interest. Out of all the objects, we also store separately the most distant one in the field of view, which is the one that assumes the temporary role of target to be reached. This determines a path of the robot that has the same shape of the geometric locus that connects the landmarks, allowing the machine to easily go around the circular object. An example of video feedback with activated object detection is shown in Figure 12.

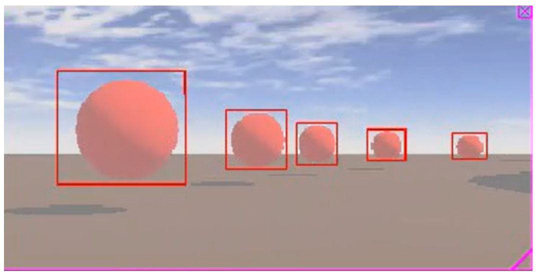


Figure 12: Landmarks detected by the camera in the SLAM stage.

The position of the references is also needed for the localization of the robot, that exploits the Extended Kalman Filter algorithm (EKF) with a prediction phase at the start of the SLAM iteration and several update stages associated with the relative landmark position estimation.

5.4.1 EKF

The Extended Kalman Filter algorithm [5] is one of the most common procedures of the nonlinear estimation of the shape (expected value and variance) of the distribution of variables such as the position and orientation of a robot. The complete algorithm is applied in two different occurrences: the first performs a prediction on the future state according to the available current information, the other fixes the prediction and updates the system's state, together with the uncertainty on the estimate, as a function of additional information that gets fed as input.

In our case, the prediction is done in the early stage of the SLAM behavior, providing as input $\hat{\mathbf{x}}$, which contains the robot's 2D coordinates and the orientation of the robot in the current iteration, Σ_x, the covariance matrix relative to the position uncertainty, $\mathbf{u}$, the array containing the current velocity and orientation (control signals) and Σ_n, the uncertainty on those last parameters. The new coordinates are then evaluated as

$$\hat{\mathbf{x}}_0 + \mathbf{u}_0 \cdot \cos\hat{\mathbf{x}}_2 \cdot dt \tag{7}$$

$$\hat{\mathbf{x}}_1 + \mathbf{u}_0 \cdot \sin\hat{\mathbf{x}}_2 \cdot dt \tag{8}$$

$$\hat{\mathbf{x}}_2 = \hat{\mathbf{x}}_2 + \mathbf{u}_1 \cdot dt \tag{9}$$

The associated uncertainty is instead updated starting from the matrices ϕ and G, computed on the velocity v as

$$\phi = \begin{bmatrix} 1 & 0 & -v \cdot \sin \hat{\mathbf{x}}_2 \cdot dt \\ 0 & 1 & v \cdot \cos \hat{\mathbf{x}}_2 \cdot dt \\ 0 & 0 & 1 \end{bmatrix}, \quad G = \begin{bmatrix} \cos \hat{\mathbf{x}}_2 \cdot dt & 0 \\ \sin \hat{\mathbf{x}}_2 \cdot dt & 0 \\ 0 & dt \end{bmatrix} \tag{10}$$

from which Σ_x is updated with

$$\Sigma_x = \phi \cdot \Sigma_x \cdot \phi^T + G \cdot \Sigma_n \cdot G^T \tag{11}$$

The update stage of the EKF also works on $\hat{\mathbf{x}}$ and Σ_x, but this time, it takes as input also the updated $\mathbf{z}$ measure and Σ_z uncertainty, together with p_l, the position of the landmark.

Defining the (transpose) rotation matrix $RM_T(\alpha)$ on an angle α as

$$RM_T(\alpha) = \begin{bmatrix} \cos \alpha & -sin\alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}^T \tag{12}$$

the update of the parameters starts from the measures by evaluating

$$r = z - RM_T(\hat{\mathbf{x}}_2) \cdot (p_l - \hat{\mathbf{x}}) \tag{13}$$

$$H = \begin{bmatrix} RM_T(\hat{\mathbf{x}}_2) \\ RM_T(\hat{\mathbf{x}}_2) \cdot \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \cdot (p_l - \hat{\mathbf{x}}) \end{bmatrix} \tag{14}$$

therefore, respectively,

$$S = H \cdot \Sigma_x \cdot H^T + \Sigma_z, \;\; K = \Sigma_x \cdot H^T \cdot S^{-1} \tag{15}$$

where we can compute the final position update as

$$\hat{\mathbf{x}} = \hat{\mathbf{x}} + K \cdot r \tag{16}$$

along with the associated uncertainty,

$$\Sigma_x = \Sigma_x - (\Sigma_x \cdot H^T \cdot S^{-1} \cdot H \cdot \Sigma_x) \tag{17}$$

Once the variables are updated with the EKF, the robot gets directed towards the farthest landmark, and another iteration takes place.

6 Conclusions and future works

The realization of the project has dealt with the programming of a robot's controller implementing a complete strategy of motion planning aimed at reaching a target position inside a complex world. Within the planned path, all of the possible scenarios have been considered, from free motion, to obstacle avoidance, to the lack of main reference points as the maze's walls; for each of those, a strategy to be carried out. The navigation problem is arguably the most important one in the field of robotics and, however conceptually simple it may appear, it is never trivial the detailed treatment of each of its individual aspects. The generalized nature of the environment and of the robot device also allow multiple applications of a methodology as the one presented in this work: every situation where a robot needs to move in an intricate world is a candidate.

As for future works, it is possible to conceptualize the integration of even more peculiar situations that the robot can find itself into, or even try to established the developed techniques as a starting point for more complex ones, which are composed, at least in part, of stages of autonomous navigation. At the same time, the project can lay the foundations for the research on implementation of physical devices driven by even more intricate algorithms such as that of [7], where the global movement and the derived path are determined by a quantum algorithm; enabling a potential practical demonstration of quantum supremacy over its classical counterpart.

References

[1] E-Puck - GCtronic wiki.

[2] Webots: epuck model.

[3] Mohammad Abdeh, Fatih Abut, and fatih akay. Autonomous navigation in search and rescue simulated environment using deep reinforcement learning. *Balkan Journal of Electrical and Computer Engineering*, 9:92, 04 2021.

[4] Albert Alfrianta. Webots Series: Move Your Robot to Specific Coordinates, August 2021.

[5] Tamer Basar. *Contributions to the Theory of Optimal Control*, pages 147–166. 2001.

[6] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[7] Antonio Chella, Salvatore Gaglio, Giovanni Pilato, Filippo Vella, and Salvatore Zammuto. A quantum planner for robot motion. *Mathematics*, 10(14), 2022.

[8] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15(1):11–15, jan 1972.

[9] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3):189–208, 1971.

[10] Uwe Stilla and Boris Jutzi. *Topographic Laser Ranging and Scanning: Principles and Processing*, pages 215–234. 01 2008.

[11] Webots. http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.