Intelligent Robotics Final Project

Master in Artificial Intelligence 2022-23

Diego Roca Rodríguez

December 20, 2022

Contents

1 Introduction

For this assignment we were asked to develop the controllers for two different robots using the 3D simulation software Webots. This software allows for building worlds and run real-time simulations using market-standard robots programmed using C or Python. The simulator allows for accurate simulations using real-world physics and forces, allowing for this behaviours to be translated into hardware robots and test on them, but as this was a introductory assignment we only performed tests on the simulation. The robots employed for this work are the TurtleBot-3 and the Khepera-4. Details about this robots and their sensors are specified in their sections.

2 Development

As stated earlier, we employed Webots simulation software and Python language for scripting. The decision to use this language instead of C was due to familiarity as it is one of the most used languages nowadays. The main disadvantage that came with the usage of Python instead of C was the inability of use threading in the controller programming, as being a high-level language Python is not as suited as C for this tasks.

The architecture we will follow is a reactive architecture. The robot will take as input the sensors and adjust his behaviour according to the data received. For each one of the robots that we employed the input data taken will be different, but the core concepts of the architecture will still be taken into account. To complement this architecture we will employ a state-machine design pattern. This pattern will change the robot behaviour according to its internal state without the need to change the internal logic of the code. As we wont use threads or an observer pattern both robots will (i) update their sensors on the main loop and after that update their internal state according to those values (ii) run the 'tick' function associated with said state.

This architecture not only is useful for this toy problem, but due to its expandability and scalability we think it suits perfectly the problem that we are asked to deal with. Improvements to this architecture could be use a non-interpreted language (as it is a limit to the performance) like C or employ a threaded architecture where we check the sensors and update state in a separated thread.

3 Turtlebot-3

The first robot that we will talk about is the Turtlebot-3 Burger, as it is the most simple one. The version we are employing is the Turtlebot-3, that was released in 2017 as a more capable version of its predecessors. The main technology of this robot is SLAM (simultaneous location and mapping) allowing the Turtlebot to understand its surroundings. For this practicum we implemented on the robot the "avoid obstacles" and the "fix bumps" behaviours.

3.1 Sensors and Engines

The robot that was employed on the simulation had integrated an LDS-01 Lidar distance sensor, a 2D laser scanner that will be employed for identifying the surroundings. This laser sensor will rotate around the robot and will return a 360-items array with the detected items. According to the experiments performed this sensor will output values inf whenever a "hit" is detected. We will employ it in order to develop the "fix bump" and "avoid obstacles" behaviours. This robot will also have left and right wheel engines in order to allow for its movement.

3.2 Behaviours

We implemented two different behaviours for this robot. As the only capabilities for this robot is to detect its surroundings an move accordingly both behaviours will make use of the Lidar sensor and the wheels.

- Wander avoiding obstacles: The first implemented behaviour will consist in the robot moving freely while avoiding any surrounding obstacles or walls detected by the Lidar. In order to compute the speed of the wheels we will use the 360-array outputted by the sensor and apply the Braitenberg Coefficients. This formula was originally developed to make robots follow points of light or simulate animal-like behaviours but we have seen that it works pretty well for avoiding obstacles or following walls. The main problems of this behaviour was that the speed of the robot needed to be slow in order for it to turn correctly, making it very time-consuming to test the simulations. Improvements to this behaviour could be employ the main directional positions of the Lidar sensor (North, South, East or West) and store their values so whenever they change drastically we can know that the robot has entered or left an item by said side.

- Fix Bumps: The second behaviour came naturally as during the experiments the robot frequently hit walls whenever he had no time to turn. In order to fix this we added a function that parses the "south" item of the Lidar sensor (the one in the front of the robot) and whenever its value reaches inf the robot performs a subroutine of turning back. This subroutine consists on 3 steps:
  - Move backwards for a fixed amount of seconds (and stop if a wall is found by the "north" sensor of the Lidar).
  - Parse the "south east" and "south west" sensors of the Lidar and compute which one has a bigger value (meaning, the one that has a detectable item closer).
  - Turn in the direction ("east" or "west") according to the direction that has no item detected by the Lidar.

3.3 Experiments

In order to test the behaviours implemented in the Turtlebot we designed a test world in the Webots simulator. In said world we included walls with different angles in order to test its capabilities of movement without bumping. We placed the Turtlebot in different positions and in all of them the robot performing accordingly to the theoretical behaviour.

4 Khepera-4

The next robot that we will program is the Khepera-4 one. This robot is a more advanced one than the Turtlebot-3 with a higher am mount of sensors and functionalities. According to their website, the newest version of this robot it even includes a Linux operating system with Bluetooth and WiFi capabilities; however, for this project we will only employ motion sensors. The main sensor that will affect the functionalities we will implement is the camera, allowing us to detect objects by their color scheme and react accordingly. Said functionalities are "detecting food", "avoiding danger" and "retreating from enemies". The robot will also include the behaviours from the Turtlebot-3 "fix bumps" and "avoid obstacles" but this time generalized as "follow walls".

In order to implement the controller for this robot we implemented a state-machine with each one of the behaviours as state. The main class of the robot implemented functions that will be employed by each state to control the robot, such as "turn left", "turn right", "go backwards" or "process sensors".

4.1 Sensors

The Khepera-4 robot includes infrared sensors that will output the distance to a item, a camera and wheel sensors. The infrared sensors are placed in the front part of the robot (front-right, front and front-left) and on its side. Their main use will be following walls and fixing bumping. The camera will is placed in front on the robot.

For our controller we will process the sensors each second in order to avoid lagging the system and whenever a "state" of the robot has finished its execution. The main bottleneck of the sensors will be, as expected, the camera. Each time the sensors are processed the camera will have to (i) process the image as an array, (ii) separate the pixels in each one of the red, green and blue channels, (iii) take the regions of interest (left, center or right) and (iv) count the number of pixels of the desired color in that region. This will allow the robot to have intelligent-like behaviours.

4.2 Behaviours

We implemented 5 different behaviours for the Khepera-4 robot. 3 of them are based on the camera and reacting to the pixels it reads and the 2 remaining will deal with fixing whenever the robot bumps into a wall and gets stuck or wandering follow the walls. The camera behaviours will have priority over the wander behaviour and they will be prioritized as "avoid danger", "retreat from enemy" and finally "go towards food". The behaviours can be summarized as follows:

- Search for food: Whenever the robot finds a green item, meaning, the camera detects an amount of **green** pixels greater than a given threshold it will go towards it. In order to adjust the direction for the robot to move towards the food we will take the green pixels from each side of the camera and adjust the wheels velocity accordingly. Once the robot reaches the green item it will stop as it work has finished. One problem we detected with this behaviour was the residual green pixels that came from the robot surroundings and a solution we came by was filtering the image by connected components in order to detect the food as a "big green blob", however, due to the slowness of the algorithm this was not viable.

- Avoid danger: Whenever the robot finds a blue item, meaning, the camera detects an amount of **blue** pixels greater than a given threshold it will turn until no more blue pixels are on its sight. Again, this behaviour presented problems with the residual pixels of the environment.

- Retreat from enemy: Whenever the robot finds a red item, meaning, the camera detects an amount of **red** pixels greater than a given threshold it will slowly move back and turn around. The same problem that in the previous two steps apply here too.

- Follow walls: The robot in its idle state (whenever is not retreating, avoiding danger or going towards food) it will wander until finding a wall. When he finds a wall we will employ the sensors on its front to adjust the wheels velocity. One problem we found with this behaviour was that whenever the walls performed a hard-turn (i.e. angles of more than 90 degrees) the robot looses the wall. A solution we implemented was to use the side sensors in order to detect whenever a hard turn is found and turn the robot accordingly.

- Fix bumps: The final behaviour implemented will try to fix the stuck state that the robot may be found whenever he cant turn fast enough. This state is triggered whenever the front sensor increases very fast (meaning, is going to crash) or whenever his value goes above a certain threshold (meaning, has crashed). This state will make the robot move backwards by assigning negative velocities to both wheels and will perform a turn based on the front-left and front-right sensors.

4.3 Experiments

We performed experiments using the Webots software. This software allowed us to build a world with realistic lightning and green, blue and red items in order to test the camera. We also added walls in order to test the "follow walls" behaviour and made the robot bump several times to check that he can fix himself correctly. In the apendix A figures 1, 2 and 3 we can see the camera thresholds results for the "avoid danger", "go towards food" and "retreat from enemy" behaviours.

5 Conclusions

We developed controllers for two different robots in the Webots simulator. We employed a reactive design with an under laying state-machine behaviour design. We employed actions based on retreating from items or going towards them while wandering without entering in bump-states, and implemented behaviours for whenever we found our robot into them. We also placed a main focus on image-recognition behaviours, particullary in color-focused algorithms.

APPENDIX A

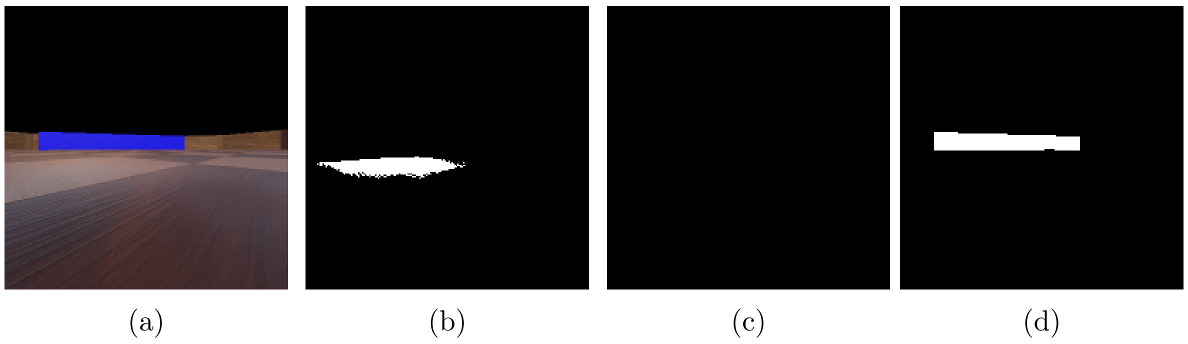

Figure 1: Detection of DANGER (blue color) in all colors (a), red channel (b), green channel (c) and blue channel (d)

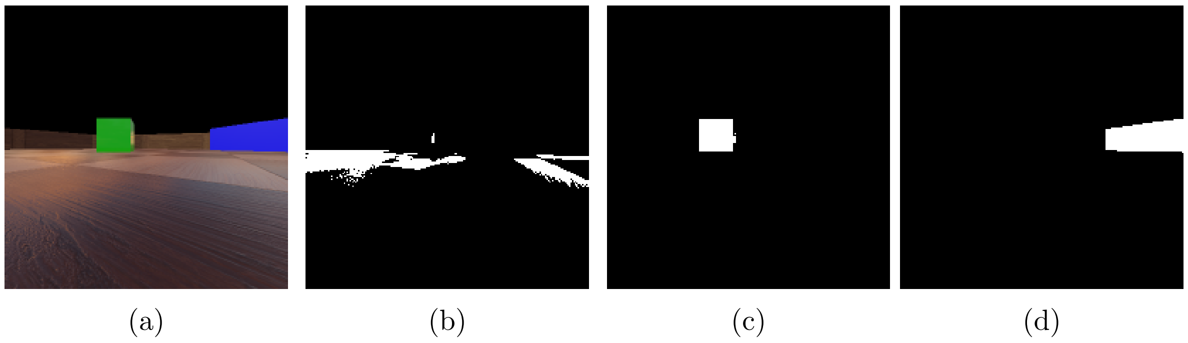

Figure 2: Detection of FOOD (green color) in all colors (a), red channel (b), green channel (c) and blue channel (d)

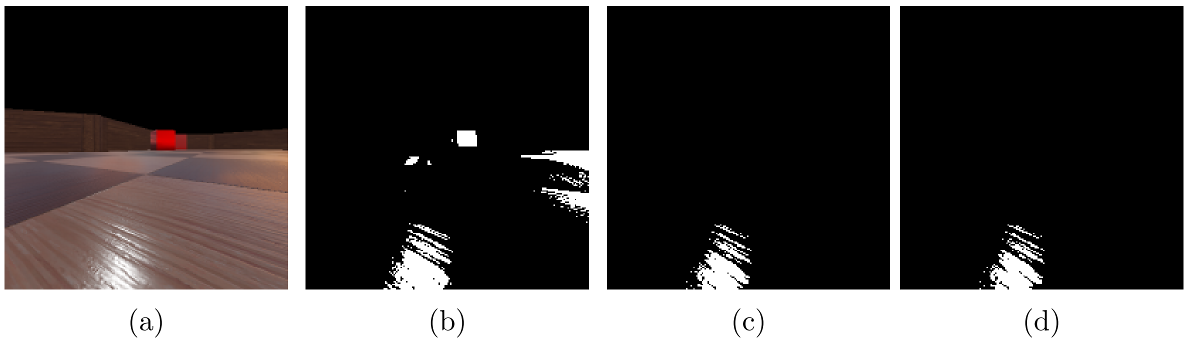

Figure 3: Detection of an ENEMY (red color) in all colors (a), red channel (b), green channel (c) and blue channel (d)