Obstacle Run SLAM

This project was developed in a ROS environment using the Robotis OP3 Framework, Python 2.7, OpenCV 3 and the Unified-Ros-Platform.

Vision

The “Piano Detector” vision from last assignment was adapted to the purposes of this assignment. Object detection now returns more shape descriptions like position in a region of the screen. In addition, it will return a list of all the objects of the same colour instead of just the biggest, this will help to track multiple cups in the screen. The yellow line is also treated as an object.

Navigation

The robot decides where to turn depending on the amount of objects accumulated in a region of the screen, if there are the same amount of obstacles in the sides then it will go forward. If a line is found, then it will compare the closest obstacle to the line and see if it is feasible to go that way or not, if not, it will turn the opposite direction for more time. This is also getting some help from localization.

The head tilt is set to a very low angle so it sees as less noise from outside as possible.

Localization

There was a publisher added for FIRA 2019 in the walking module that will return a boolean each time the robot does one step (from just one leg). Taking this value, we can count how many steps it does in a given time, therefore, each direction (forward, left, right) will have a counter of steps to keep track of where it is on the map.

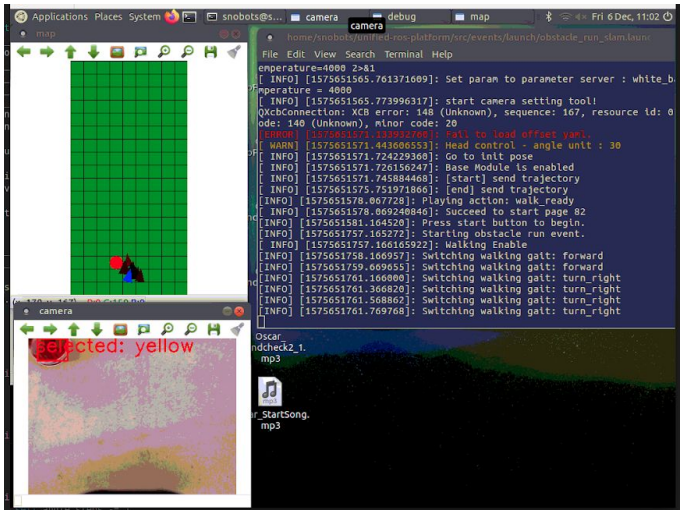
The map is divided into cells of 20 pixels, and the robot will update every 5 cells, which is roughly the distance in centimeters it will move every two steps forward. In the case of degrees, it will update every step with 15 degrees to the map.

The robot’s position (x, y) and angle is published to the map after the amount of steps explained before and it will be used to draw the robot in the map. The angle is also used to decide where to go in case of encountering a yellow line, if the angle of the line tell us to go to a wrong direction we will double check with the angle recorded by the localization and make it turn the correct way. All the values from localization will be published to the slam module, which is inside vision, plus a string telling the particles whether to relocate the particles or not, depending on the line’s angle.

SLAM

The map will generate 20 particles at the very beginning in the same position of the robot. After each step, a random value in a range will be assigned as a motion error. For example, after moving to the left, a random value between -20 to -10 will be chosen for the angle and from 0 to 1 for forward. All these values were chosen after a manual analysis of how the robot tends to move after some steps in all directions. When localization tells the slam that a yellow line was saw, all the particles will move to the correct side of the map and grouped around the coordinates provided by localization.

This module will also receive the obstacles' position and place them on the map according to the region of the screen and translated into a rough distance in pixels, which was also measured manually. When seen an obstacle more than ones, it will check in a range inside the grid and decide if it is a new obstacle or not by accounting the robot position.



Results

Robot navigation is pretty accurate, mostly 80% successful. Less accuracy is encountered when the battery is low, the left leg gets loose or the obstacles are very close to each other (exactly 40cm apart). Localization had always an error in coordinates of 40-60 pixels and +/- 5 degrees. Most of the time, the particles will be more accurate in position than localization, but not by too much in the direction.