ROBOTIC 2020

Marta García Tornero., Irene Melchor Félix.
mgarciajr@alumnos.unex.es , irmelchor@alumnos.unex.es
University of Extremadura

***Resumen*—To perform this practice, the main tasks that have been carried out have been three, which have been served to start the robot, make it move by itself, and finally avoid obstacles.**

I. Introduction

First of all, the student has been provided with a project on GitHub. This project consists of a map with a robot. When executing the project for the first time, the robot is static on the map, and the next step that has been carried out has been its start-up. In order to move the robot around the map, certain components have been installed that allow the robot to move through two options:

- The first way that has been provided is through a joystick, connected by the USB port of the computer.
- The second option using the computer keyboard.

In this case, the two options were tested to check the operation of the robot, with each device that was give us. Next, the MyFirstComp component was created, which will be explained later, to make the robot circulate alone in the room. Once the robot has been configured, the vacuum cleaner component has been started, this will serve to measure the percentage of sweep of the robot, in the desired time. Finally, a first sweep was carried out, thus verifying that the robot only cleaned 20 percent of the room, so our objective is to increase this percentage with a series of methods that we will see later.

II. Initial compute Method

Once in situation, the first thing to create is the MyFirstComp component, this component will allow the robot to move on its own without using the joystick or the computer keyboard, as mentioned before. Next, the compute method found in the SpecificWorker.cpp class of said component is modified. In this method the calls to the other methods that have been implemented will be made.

```
void SpecificWorker::compute(){
  const float threshold = 300;
  float rot = 0.6;

  initializeMatrix();

  try{
    RoboCompLaser::TLaserData
    ldata = laser_proxy->getLaserData();
    std::sort(ldata.begin(), ldata.end(),
    [](RoboCompLaser::TData a,
    RoboCompLaser::TData b)
    { return a.dist < b.dist; });
    turnMethod(ldata, rot, threshold);
  }catch (const Ice::Exception &ex){
    std::cout << ex << std::endl;
  }
}
```

III. InitializeMatrix Method

The first method created in SpecificWorker.cpp is to initialize matrix. This method consists of "transforming"the map through which the robot moves into a matrix, which would go from -2500 to 2500, but since it is more complicated to use negative values, it is decided to add 2500 to each value, so the matrix ranges from 0 to 5000.

All the cells of said matrix are initialized to a false value, so that, as the robot goes through them, they change their value to true, in this way the robot will know through the cellsthat it has already passed.

```
void SpecificWorker::initializeMatrix(){
  for (long int i = 0; i < 5000; i++){
    for (long int j = 0; j < 5000; j++){
        map[i][j] = false;
    }
  }
}
```

IV. turnMethod Method

Inside SpecificWorker.cpp a method is created that checks if the distance in front of the robot is less than or equal to the threshold created, (in this case it has been initialized to 300), and, if so, the robot turns at a random angle and call the checkMatrix method, which will be talk about it later. If the distance is greater than the threshold, then the robot continues forward at the highest possible speed. The code that has been implemented for this method is the following:

```
void SpecificWorker::turnMethod
(RoboCompLaser::TLaserData ldata,
float rot, float threshold){
```

```
  if (ldata.front().dist <= threshold){
    srand(time(NULL));
    rot=(rand()%(1570-0000))/1000.0;

    differentialrobot_proxy->
    setSpeedBase(5,rot);
        usleep(rand()%(1500000-100000+1)
        +100000);

    int x = 0, z = 0;
    float alpha = 0;
    this->differentialrobot_proxy->
    getBasePose(x, z, alpha);
    checkMatrix(x, z, alpha);
  }else{
    differentialrobot_proxy->
    setSpeedBase(1000, 0);
  }
}
```

V. checkMatrix Method

The checkMatrix method is also found in the SpecificWorker.cpp class. This method checks, by means of a vector of neighbors, the 8 adjacent cells of the robot, in this way, it will know if all its neighbors have been visited. At the moment in which a neighbor that has not been visited is found, the robot will go in that direction, thus making sure, do not visit a cell that has already been visited previously. If it has already passed through any of the cells, the robot continues forward, and if not, it turns.

```
void SpecificWorker::checkMatrix
(int x, int z, float alpha){
  x = x + 2500;
  z = z + 2500;
  bool enc=false;

  vec vecinos[8] = {
    vecinos[0] = {x - 1, z - 1},
    vecinos[1] = {x, z - 1},
    vecinos[2] = {x + 1, z - 1},
    vecinos[3] = {x - 1, z},
    vecinos[4] = {x + 1, z},
    vecinos[5] = {x - 1, z + 1},
    vecinos[6] = {x, z + 1},
    vecinos[7] = {x + 1, z + 1}};

  if (map[x][z] == false){
    map[x][z] = true;

    for (int k = 0; k < 8 && !enc; k++){
      if (map[x][z] !=
      map[vecinos[k].v1][vecinos[k].v2]){
       differentialrobot_proxy->
       setSpeedBase(500, 0.8);
       enc=true;
       }else{
        differentialrobot_proxy->
        setSpeedBase(1000, 0);
       }
    }
  }else{
    std::cout << "ERROR" << std::endl;
  }
}
```

VI. Final Compute Method

The compute method has varied a lot throughout these weeks, since at first it simply consisted of an if-else within the try-catch it contains. Later, a switch was created but, after several tests with the robot, it was eliminated to optimize execution, since with the current implementation it has been proven that it is not the most optimal. Another variation that the current code has undergone was to call a method that would make the robot go forward, but it was also eliminated due to slowdown problems. Finally, within the try-catch, the turnMethod method is called, which is responsible for calling the checkMatrix method, both explained previously.

VII. Mistakes

At first, the compute method created and initialized the threshold variable to 200, but when modifying the code and adding new methods, this variable has been changed to a higher value, in this case 300, as mentioned above, because If this value was kept at 200, the robot, when moving forward and turning, would go through some walls or some of the objects that were found along the way.

Another mistake that was made was not slowing down before the robot made a turn, so when approaching a wall or obstacle, the speed was too high and the robot did not have time to turn without colliding with the object. To solve this problem, the robot has been slowed down when turning.

VIII. Executions

After carrying out the entire implementation, 5 executions of 120 seconds each have been carried out to study the behavior of the robot. As can be seen in each of the captures, the times oscillate between 32.32 percent and 35.52 percent. So the average obtained between the five executions is 33.856 percent.

IX. Conclusion

As a conclusion, it can be added that the percentages obtained can be improved, so we will work on it throughout the sessions. We believe that to be the first contact with the subject, it is not bad at all, even so we will try to achieve the highest percentage possible.

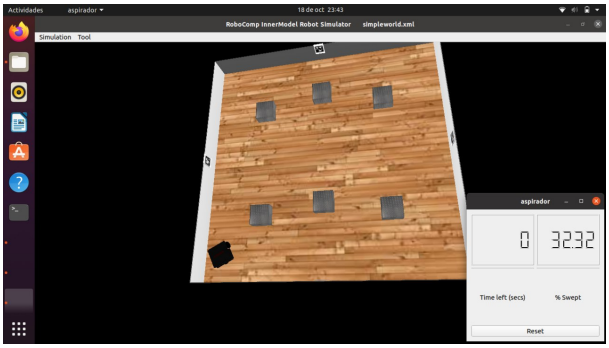

Figura 1. 32,32 percent

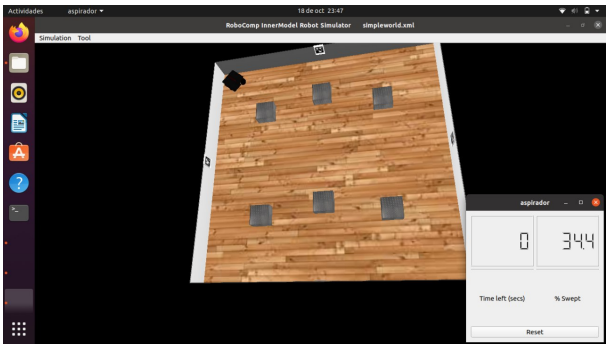

Figura 2. 34,4 percent

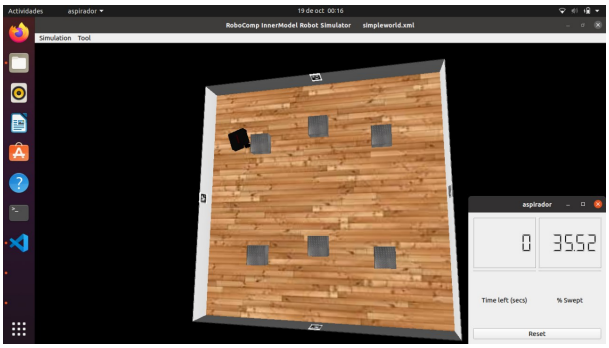

Figura 3. 35,52 percent

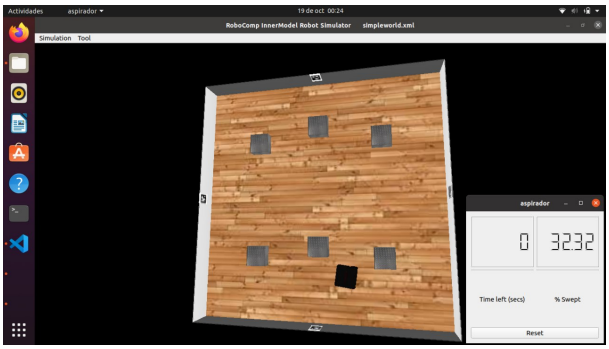

Figura 4. 32,32 percent

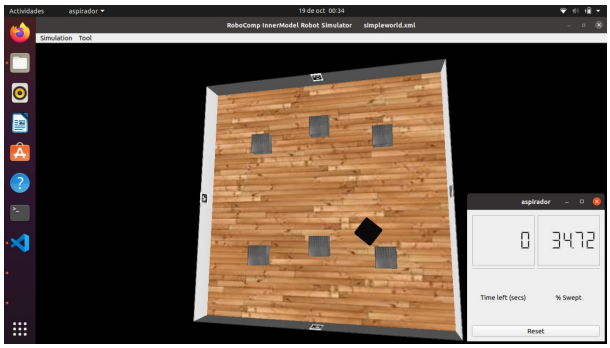

Figura 5. 34,72 percent