Robótica 2021-2022

1st José Luis García Salas

2nd Enrique Moreno Ávila

I. Introduction

Nowadays, the research field of robotics is a must-know discipline for all the engineers who are working in technologies like Internet of Things (IoT), automation of tasks, or home automation for people with reduced mobility. The use of robots for making the daily issues easier and natural to make life more simple is becoming so common, that in actual days is strange the house that does not have electronics integrated to work with assistant like Amazon's Alexxa or have the refrigerator or cleaner bot connected to the phones with the purpose of track its activity and route.

In this subject, following the line of work of previous years, we are going to learn how the fundamentals of robotics functions, as well as making our own simple programs to make the simulated robot obeys our commands and instructions with the objective of transfer our own software to a real robot facilitated by the department of RoboLab from the EPCC in the UNEX.

To make this software, we will use a library written in C maintained by RoboLab in EPCC, Aston University and many other collaborators called Robocomp, an open-source framework based in the idea of communicate components through public interfaces.

In this letter, we are going to explain step by step the procedure to make a good software for robot control, as well as the other technologies and tools used in the making.

II. First Class

Here we are going to introduce all the new software tools we use in the design of the robot software as well as we make for the first lesson of revise C/C++ knowledge.

A. Software Tools

We run all the software in a Ubuntu 20.04.3 Focal LTS. Because of the Robocomp library y heavily based on C, we are going to write our code on it, using the GCC/G++ compiler 10.3.0 version because it brings the last dependencies and libraries to work with. Generally, in the Ubuntu version we are using, the GCC/G++ compiler are installed in the 9.3.0 version. We had to update it installing the 10.3.0 version executing **sudo apt-get install gcc-10 g++-10** to install it, and then switching the version of the compiler which the S.O. is using with **sudo update-alternatives –install /usr/bin/gcc gcc /usr/bin/gcc-10 2 0 –slave /usr/bin/g++ g++ /usr/bin/g++-10**.

Thanks to the department of RoboLab and the EPCC

For running and coding in C/C++, we use CLion, an IDE developed and maintained by JetBrains, it is one of the best for this situations due to its capacity of build by itself all the dependencies between the libraries and files, as well as good maintainance with other associated compiling tools like CMake. One good alternative for this is the code editor Visual Studio Code (VSC).

Once we installed all this, we install the first important tool for the interfaces and libraries needed for the robot behaviour, it is called Qt5 software. This is the main toolkit to use the QT libraries related to C/C++, and despite the fact that we are using CLion in first place, QtToolkit have its own IDE called QtCreator and most important, QtDesigner, which is an interface designer for tools like timer counters and lcd displays and the main tools we are going to use for the visual part. As we mentioned, the main library we are going to use is Qt, specifically related to signals and slots, this library fits perfectly with the methodology of components, we can use objects declared as QTWidgets as the future components of the robots, in which each slot is a function triggered by a signal declared previously and connected by the **connect()** method.

For the version control, we use Git, a software of code-version-control which allows us to maintain different workflows and previous status of the code in case we find an error or we make a critical mistake. With Git, we use the repository GitHub, in which we upload from our local repository of Git all the versions of the code for working everywhere in remote control without the need to do it only in classes.

The last tool used before we star to install all Robocomp library is Overleaf, Overleaf is a online text editor which allows us to have the documentation stored in the cloud, and modify the document simultaneously making the parallel documentation possible and we can review and improve our own mistakes or changing things with the security of being both of us at the same time.

Once all of these tools are installed, it is possible start installing all the Robocomp related software in the next classes and the new ZeroC middleware for the communication between interfaces.

B. QTimer and timerSimple

In this lesson we are given 2 different projects, both of them implementing the same thing, a simple timer with a counter of elapsed time and a slider to modify the period for the timer itself.

The first project, called ejemplo1, is the version of the timer implemented using QtTimer library. The main difficult here was to learn the functionality of signals and slots, and how they interconnect, despite this learning proccess, Qt libraries allow to make this very easy and simple to understand.

The second project, called timerSimple, is the same version of the timer, but using the std library of C/C++, the main goal of this project is to review and refresh the knowledge of C/C++ after so long time without use it. This timerSimple version has been heavier and deeper than its other version, we think that the most difficult part is understand the declaration and use of the thread and how is used and synchronized with other threads without hindering each other.

For the visual interface of the timer, there are two ways to modify and built it, we can go directly to the .ui file located in the project, it is a xml file and it can be modified adding manually the new objects like buttons or lcd displays and horizontal slayers. The other way, and it is the more simple way to do it, is open the .ui file with the tool of QtSoftware called QtDesigner, it is a graphic interface and you can drop the object in visual mode, making so much easier the interaction with the timer and interface.

As a conclusion to this practice, we think that the most hard part of it was the use and manage of the std library, probably because the long time we do not use C/C++, despite this, we adapt quickly and it has been an easy review practice.

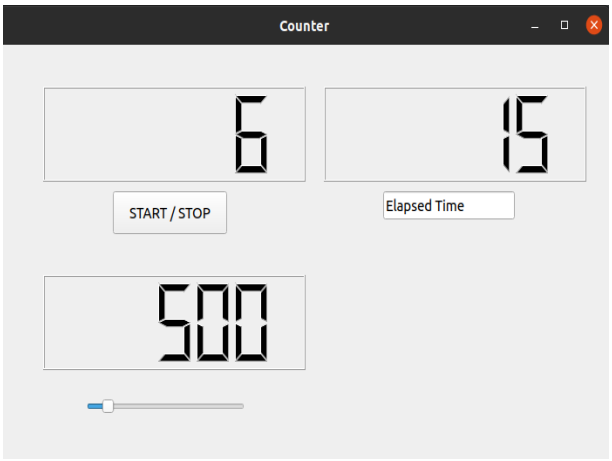

Fig. 1. Graphic representation from the interface of the timer.

III. Second Class

In this section, we are going to explain which software was necessary to install and design components, as well as describe how the behavior of the component was modified and designed with the tools and technologies mentioned in previous section .

A. Robocomp Software

As we said previously, Robocomp software is an open-source framework based in the idea of communicate components through public interfaces, all of them created by members of robotic research community as Robolab in EPCC university or Aston university.

In a first moment, we realized that exist many tutorials to learn how to design and modify components such as the behavior of them. With this in mind, we followed the tutorials given by the teacher of the subject and present in virtual campus, this tutorials taught us how to install all **robocomp** libraries and dependencies, as well as the functioning of **CoppeliaSim robot simulator**, the environment which we are going to use by the rest of the classes.

The first thing we are going to install is robocomp, for this purpose we had to access to the following repository on github: **https://github.com/robocomp/robocomp** and when we went to the documentation we saw a brief abstract about robocomp software about what is, how is make it and other type of information. Next, we have requeriments will be necessary to install it and run it. A brief can be with a command got it in github, it will update all dependencies and all certificates of the S.O. . If this command works, we will be able to put the next command, which will install the rest of robocomp. Before finish we have a recommendation about packages that could be necessary to run robocomp.

The next thing we will do is install CoppeliaSim and Pyrep. CoppeliaSim is a simulator used to work with robocomp in physically complex scenes, so CoppeliaSim will be used to realise the behavior of components designed by us or examples that we want to see it what they do. In the other hand, we have Pyrep, that it's a standar remote python API of CoppeliaSim. Once, we have explain what is CoppeliaSim and Pyrep in a few words, we proceed to install it. To install it, we have to access to this github: **https://github.com/robocomp/robocomp/blob/development /doc/robocomp-pyrep.md** and we can see a documentation where there is a brief abstract about CoppeliaSim and Pyrep about what is and why are chosen, and then a guide step by step about the installation. To install CoppeliaSim with download it in the official web page and install it and then add three lines(that they are in the tutorial) necessary in the path, it will be ready. To install Pyrep we will continue in the same github, and we can see that we will clone a github, where it's Pyrep Github, and then with two commands of pip3 we will have installed. Of course, it will be necessary to add three lines to the path.

To have totally ready CoppeliaSim we will go to the directory of robocomp and we will do a git clone to **"https://github.com/pbustos/beta-robotica-class"**, and in the directories within, we will start the middleware with the **rcnode &**, and then we will change to the directory **"¬/robocomp/components/beta-robotica-class/pyrep/giraff"** and run **"./run.sh"**, with this we will have CoppeliaSim started.

Then, we can see four tutorials that explain the interface language that there is in the component, the language definition of the component, many example of coding and the code generator, that will be the start of the next section, because it explains the basics of how to create a component and run it.

B. Making components.

As we said before, the code generator tutorial is the start of how to make a component. In the tutorial, we can see that to create and design components we will use **robocompdsl**, which is a new tool used in robocomp to to automatically generate components and modify their main properties once they have been generated (e.g., communication requirements, UI type). The language used is CDSL that allows to specify components and IDSL to speify interfaces. The first thing will we do is generate a CDSL template file that gives a example or guide of what we have to put about we want the component do. The second thing is go to the directory of innermodel inside of robocomp to start the enviroment to see the component, and then open the source file of the component to change the behavior in the section of compute to make that the component do what we want. Last, we will compile the component and we will run it.

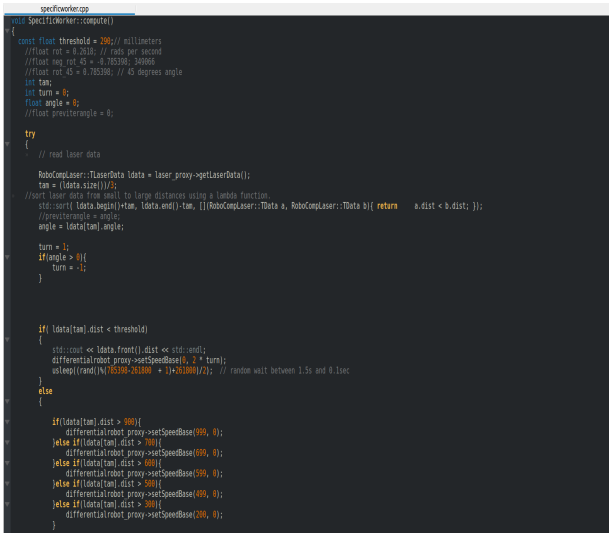

Fig. 2. Graphic representation from the code of the robot.

From now on we are going to continue the second class that consists basically in create a new component, and change his behavior so that this component be a ”Roomba”. Also, there are requirements to this component, it will be necessary that this ”Roomba” can sweep the room in the least possible time, of course it has to avoid hitting walls and it hasn't stay stuck in the room, as minimun. It has to be efficient sweeping the room, to be competitive in the market. So, in conclusion, the component has to work at least 10 minutes with requirements said before. First, we will make a template above functions we will use, we do this with **the robocompdsl**. Once we have the template, we will modify it, we add **the differentialrobot** to have a robot that move in the room, and we use **the robocomplaser** to calculate the distance of the robot to the wall and make a behavior to avoid hit the wall, as example. Second, we have modified the source file to put the behavior in the section compute. Here, we have to think the distance where we want to rotate the robot to avoid hit with the wall or other thing. We realized it's related with the angle and the speed, a good strategy it could be put lower threshold and speed, and the angle may be little. One one hand, we can get a robot that in the most of cases it won't get stuck and it will be able to avoid walls, and we realized that the robot do a path as a elipsis. One the other hand, we get that sometimes the robot get stuck in some part of the room or it doesn't avoid the wall. The first thing to resolve it, we divided the data vector in three parts due to we don't want the first data or the last data of the laser because it doesn't contribute information about the distance. Other thing to change is to reduce the speed when the robot is near the wall or other thing and set a lower speed when is nearest, with this idea we will get that the robot can rotate with the same speed any obstacle. Also, emphasize that the method **”setSpeedBase()”** is basically to set an angular speed, so you must think a good speed due to the robot has a correct rotation, we use 0.2 rad/s. To have a good rotation is neccesary to put a correct angle, in this case, we use between 45º and 15º. In conclusion, the result is a robot that always rotate a similar angle, even though at first the robot can clean a lot, when it's cleanning 1 minute, you can realize that the robot doesn't clean so much due to it starts to pass to sections are cleaned, and as result it doesn't clean a lot. After 3 minutes, we can realize that the robot get cleaned a 40 percent and it isn't bad in a first but it can better.

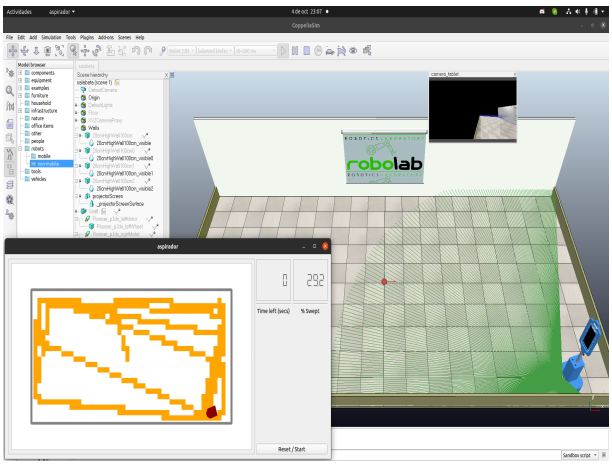

Fig. 3. Graphic representation from the result of the first algorithm

Seeing the problems of cleanning with the robot, we have though another algorithm to have a better result. To have a efficient result we have though that maybe trying with a form of spiral, we can have a better results. To do a spiral, the first thing we can think is that the threshold with respect the wall must be low to make go the robot the nearest to the wall, and then when it has doing a lap, the robot must have a threshold a few high. A few moments later, we'll see that the robot will tour the room with a spiral form. After this explanation, the thing that we have to put in our code respect the code before, basically when the robot arrives to a section that is cleaned, there is to reduce it's threshold and it has to continue. The angle will be always the same, because the robot has to rotate

45º to avoid the walls or a object, and another 45º if the robot must continue the same path. Last, the strategy of reducing the speed when the robot is near of a wall or an object is correct because the best strategy is when the robot is far of an obstacle the robot has a high speed, and when it is nearest of an obstacle it must have a lower speed to contribute with a good rotation.

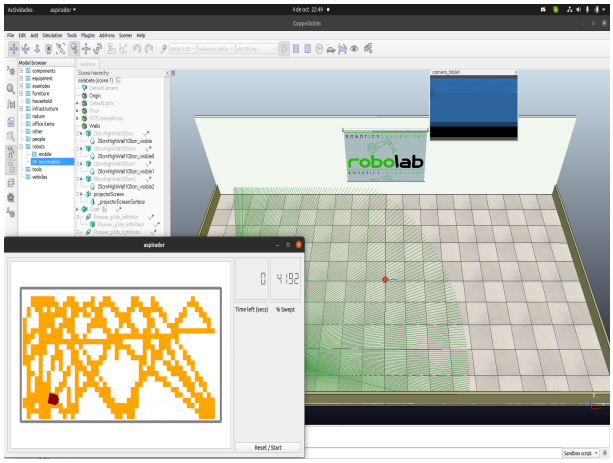

Fig. 4. Graphic representation from the result of the second algorithm

As a conclusion of this practice, the part most difficult was installing robocomp and it's software, because it's normal many requirements that are needed, it's not installed and it can appear many errors, so the installation can get much time and in many occasions the error is not easy to deal with it, but once you get installed all, you can work with robocomp without any problems and it's interesting design the component and change his behavior and realize of the possibilities can have.

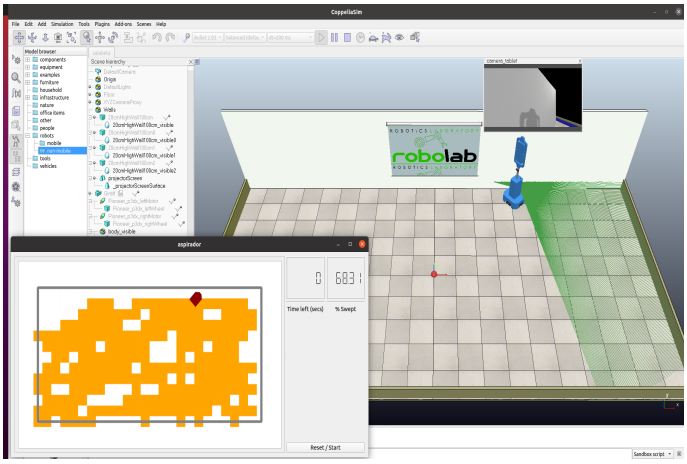

Fig. 5. Final pathing with vacuum modified