

*Programing for Robotics*  
*Final Project*

**Rover for libraries to identify if the book  
available is on the shelf**

**Authors:** Lucas Resende Gomes  
Nemo Palmer

# Summary

**Abstract**

**I – Introduction**

## Abstract:

The use of robots in the engineering industry is not only essential but a requirement in modern and competitive companies and fields of work, however most of its use have been directed to the manufacturing, to do repetitive and simple movements. The purpose of this project is to implement a robot in a different environment than it would normally be integrated, in a library to identify if the book is in its designed position or not. The robot with the use of a camera and a LIDAR will go to the shelf and search for the book's identification, if it finds the book it means that it is on the designed position, if not it means that someone might be using the book inside the library, or someone misplaced the book in other shelf.

## 1- Introduction:

Integrating machines to the industry is an implementation firstly made after the 1<sup>st</sup> industrial revolution, like using the loom to do faster and better fabrics or the implementation of the vapor-based machines. However, there are differences between a machine and a robot, according to "Robot Science", a machine is a simple device that might be designed to complete the given command, a robot is a single combination of many machines in one body, that gets information from external sources and gives back a response based in those perception and previous information.

The machines and simple robots started being added in the manufacturing industry, doing simples and repetitive movements, that a human could also do, but the technology and robots kept evolving to a point where the robot was faster and with more accuracy than a human worker.

However, the integration of the robots got "stuck" in the industries for a long time, mostly because of its costs and because people still had a prejudice with the robots in their "every-day lifestyle". Nowadays this is no longer a major problem, people are getting more and more used to the robots in their lives (Grandys F. Prassida and Uly Asfari, 2022), as part of the changes from the 4.0 industry to the 5.0 industry.

To simplify and make the world work faster and smarter, the Stanford University created the Robot Operating System (ROS) in 2007, since then the works in the robotic field are mostly done in this system, making the projects closer to each other, unifying the languages of programming in one system. This way if someone in Brazil and someone in France could share their work, problems, and solutions (Keenan Wyrobek)



*Robot Operating System (ROS) logo.*

According to the organization website, ROS is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source. ROS is also the platform that we used to make our "Library Book Checker" robot.

## 2- Our Project

### a. The problem to solve

According to the ministry of culture in France, there are more than 16.000 public libraries in France, therefore there are millions of books to be organized and sorted in their respective shelves and places. All libraries have one major problem, organizing all those books, not because of space or software, but because everyday hundreds of people go to the library, take a book, and put them in a place that isn't their right spot. Doing this, not only the person is compromising the organization, but also hiding the book from the system, so even though the book is inside of the library, it will be extremely hard to find it.

The scientists of Singapore already have a prototype of a robot that can identify the misplaced books in the library, the robotic librarian was developed by A\*STAR, and it's called AuRoSS, that stands for Autonomous Robotic Shelf Scanning system, the research was made by Rejun Li, Zhiyong Huang, E. Kurniawan and Chin Keong Ho.



*The AuRoSS Robot at work*

However, this technology is very expensive and may not be accessible for smaller libraries or bookshops, that can also have the same problem, because many people go inside them, look at the book and replaces then without carrying about its original placement. Because of this, when another costumer goes inside searching for that book intending to buy it, he wont be able to find the book.

## b. Our solution

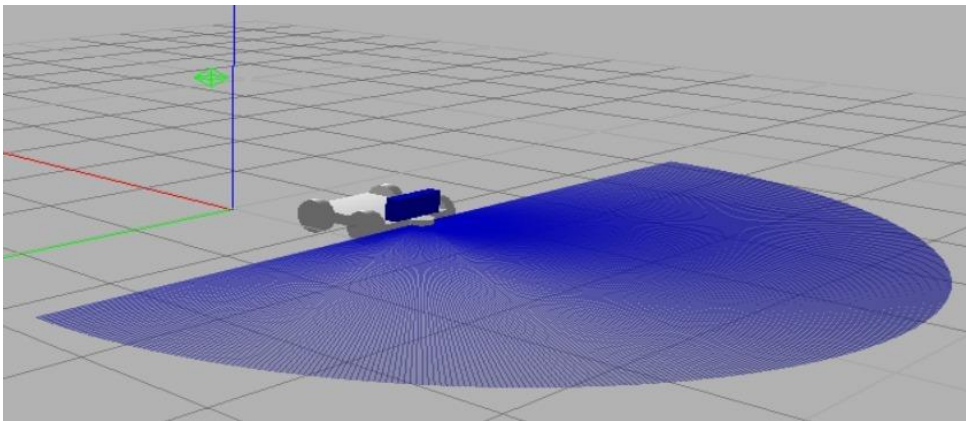
The solution that we had was to create a robot that could identify a QR Code each book and make the robot go to the shelf were the book is supposed to be and using a camera check if the book is there or not.

## 3- Our First Robot

Working with ROS, Python and Gazebo, we had to create a few things to make a simulation possible. We had to create an environment, that was going to be our library/bookstore, a robot that could go around the environment and take the camera, a LIDAR to avoid hitting walls and make the autonomous riding possible, and a camera to identify the QR Code and check.

### a. The Rover

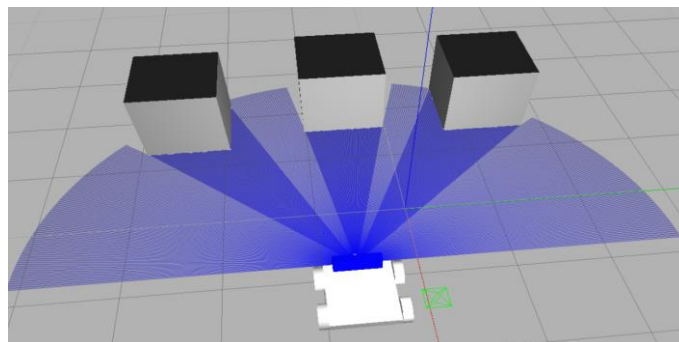
The first thing we created was the Rover, that is the robot that would run our simulation.



*The Rover*

This is a simple Rover robot, have a chassis, 4 wheels, a Lidar, and a camera. The whole robot is white with exception of the camera, that is the blue box at the top of the robot. The blue semi-circle is the Lidar searching for possible obstacles and a safer path to follow.

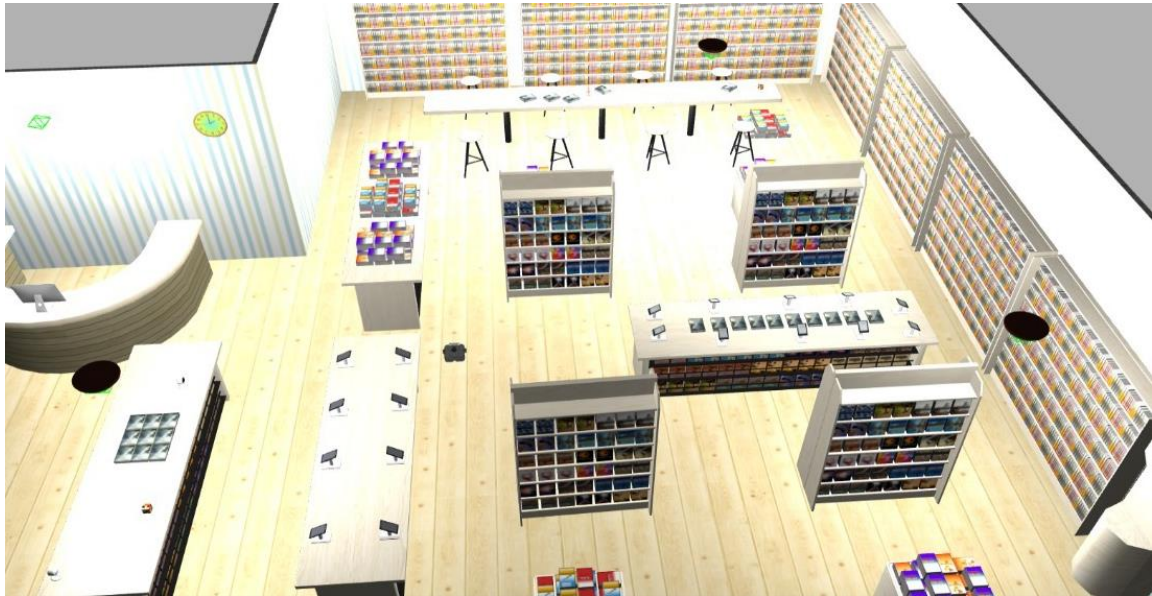
The Rover had simple geometries and joints, and the plugins for the camera and for the Lidar. It can make turns by decelerating the wheels of one side and accelerating the wheels of the other side.



*Rover's Lidar identifying obstacles*

## b. The Environment

For the environment we selected a bookstore, so we could have a smaller space with a more controllable simulation.



*The Bookstore Environment*

The environment, as shown, have many shelves and books at disposable, the idea is to place a QR Code in the cover of the book to check the positioning of it.

## c. The Camera

The camera system was very simple, it would identify a QR Code, take a picture of it, and then search for a compatible one in the designated shelf. To do so we used the following code:

### Project Camera Following

```
In [1]: import numpy as np
import cv2

cap = cv2.VideoCapture(0)
while True:
    success, img = cap.read()
    if success:
        # Get height and width of webcam frame
        height, width = img.shape[:2]
        # Define ROI Box Dimensions
        top_left_x = width / 3
        top_left_y = (height / 2) * (height / 4)
        bottom_right_x = (width / 3) * 2
        bottom_right_y = (height / 2) - (height / 4)
        top_left_x = int(top_left_x)
        top_left_y = int(top_left_y)
        bottom_right_x = int(bottom_right_x)
        bottom_right_y = int(bottom_right_y)
        # Display the image with the ROI rectangle
        cv2.rectangle(img, (top_left_x, top_left_y), (bottom_right_x, bottom_right_y), (0, 0, 255), 3)
        # Display image
        cv2.imshow("Template", img)
        if cv2.waitKey(1) == 13:
            frame = img
            break
        cv2.destroyAllWindows()
    else:
        break
    cv2.destroyAllWindows()

r, h, c, w = 300, 50, 300, 50
track_window = (c, r, w, h)

# Crop region of interest for tracking and convert it to HSV color space
roi = frame[r:r+h, c:c+w]
hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
```



```
# Create a mask between the HSV bounds
lower_red = np.array([0,0,0])
upper_red = np.array([60,255,255])
mask = cv2.inRange(hsv_roi, lower_red, upper_red)

# Obtain the color histogram of the ROI
roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0,180])

# Normalize values to lie between the range 0, 255
cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)

# Setup the termination criteria
term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )

while True:

    # Read webcam frame
    ret, frame = cap.read()

    if ret == True:
        # Convert to HSV
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        # Calculate the histogram back projection
        dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
        # apply Camshift to get the new location
        ret, track_window = cv2.CamShift(dst, track_window, term_crit)
        # Draw it on image using polyLines to represent the adaptive box
        pts = cv2.boxPoints(ret)
        pts = np.int0(pts)
        img2 = cv2.polylines(frame,[pts],True, 255,2)

        cv2.imshow('Camshift Tracking', img2)
        if cv2.waitKey(1) == 13: #13 is the Enter Key
            break
    else:
        break
cv2.destroyAllWindows()
cap.release()
```

*The code*



*The camera locks an image in its memory*



*The Camera now identifies and follows the locked image*

The importance of the camera tracking the locked image is so that whenever the image shows up in the camera range, the robot will be able to locate it and move in that direction.

## 4- Problems

Even though we had all of the parts working separately, when we assemble all of them together, we couldn't make them work properly.

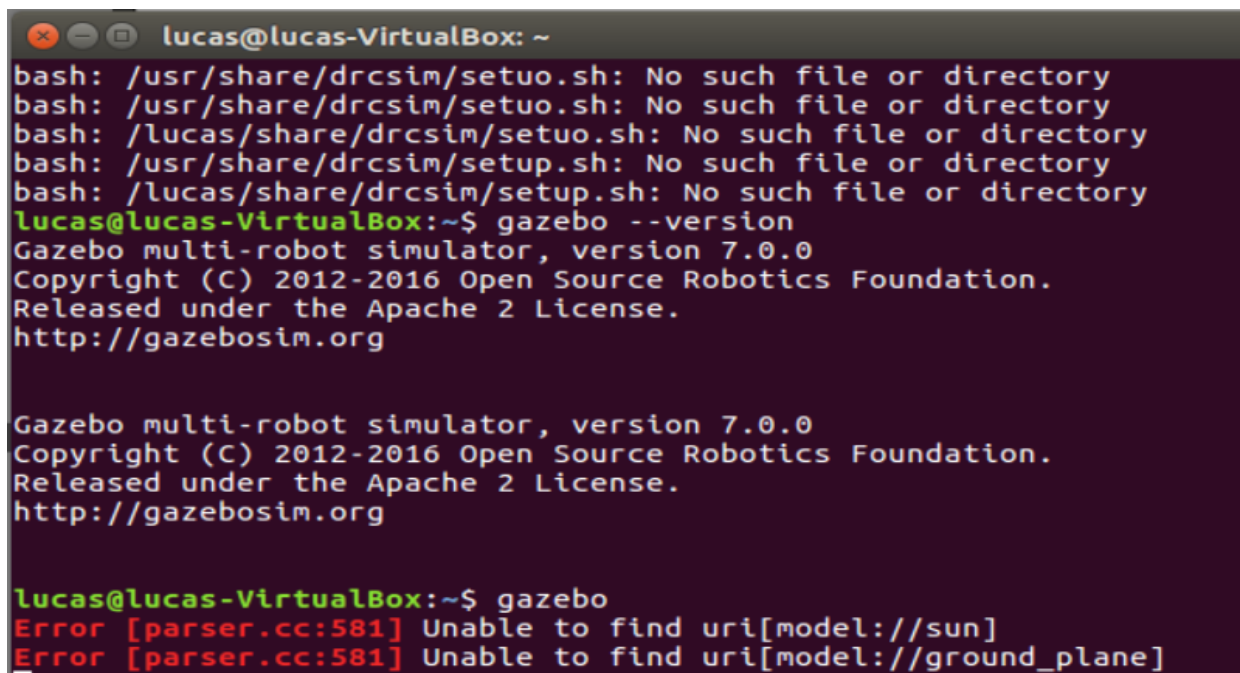
The first problem we had was using Gazebo 7 in Ubuntu 16.04. The problem was that the gazebo couldn't get the files of the ground and sun, so the only way to place something in the world was: make it statical or disable the physics of the world.

The second problem we had was with ROS2 Foxy in Ubuntu 20.04, where it couldn't find the files of the rover to launch and chased when the environment was launched.

Another problem was when we tried to use the QR Code recognition system, that when we opened the gazebo, it crashed and showed a lot of errors in the terminal.

The last of the problems, now in Ubuntu 20.04 using ROS Noetic, was that I couldn't spawn the rover in the environment selected.

So, with all these problems our solution was to use a turtlebot robot inside the environment instead of the rover, and it worked. However, we couldn't integrate the QR Code system to the turtlebot.



```

lucas@lucas-VirtualBox: ~
bash: /usr/share/drctsim/setuo.sh: No such file or directory
bash: /usr/share/drctsim/setuo.sh: No such file or directory
bash: /lucas/share/drctsim/setuo.sh: No such file or directory
bash: /usr/share/drctsim/setup.sh: No such file or directory
bash: /lucas/share/drctsim/setup.sh: No such file or directory
lucas@lucas-VirtualBox:~$ gazebo --version
Gazebo multi-robot simulator, version 7.0.0
Copyright (C) 2012-2016 Open Source Robotics Foundation.
Released under the Apache 2 License.
http://gazebo.org

Gazebo multi-robot simulator, version 7.0.0
Copyright (C) 2012-2016 Open Source Robotics Foundation.
Released under the Apache 2 License.
http://gazebo.org

lucas@lucas-VirtualBox:~$ gazebo
Error [parser.cc:581] Unable to find uri[model://sun]
Error [parser.cc:581] Unable to find uri[model://ground_plane]
```

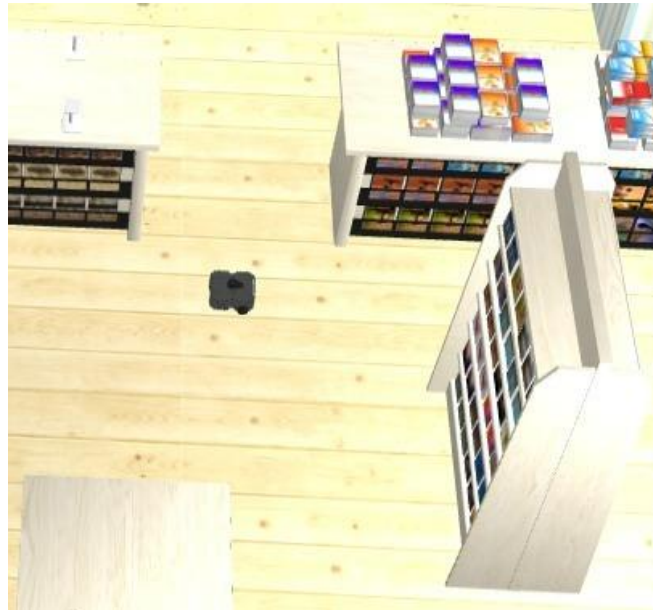
*One of the problems found during the project*



## 5- Solutions and Conclusion

### a. Turtlebot

To continue the work with a good environment, we made a choice to abandon the rover that we spend – a lot of – time doing, so that we could use an robot inside of the bookstore and use the camera and the Lidar.

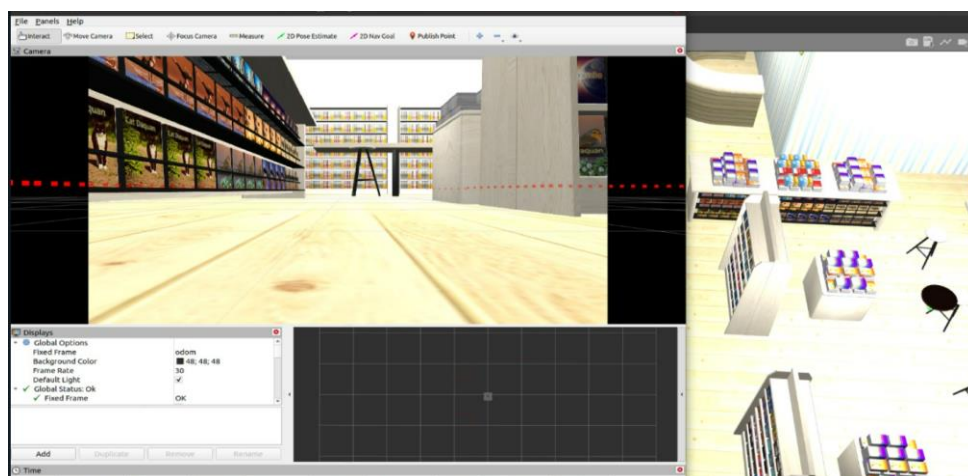


*The Turtlebot in the Bookstore*

The turtle bot model that we used was the Waffle, because had similar dimensions to the rover that we had created.

### b. Turtlebot Camera

We also used the prompts in the turtlebot to look inside the bookstore and control it.



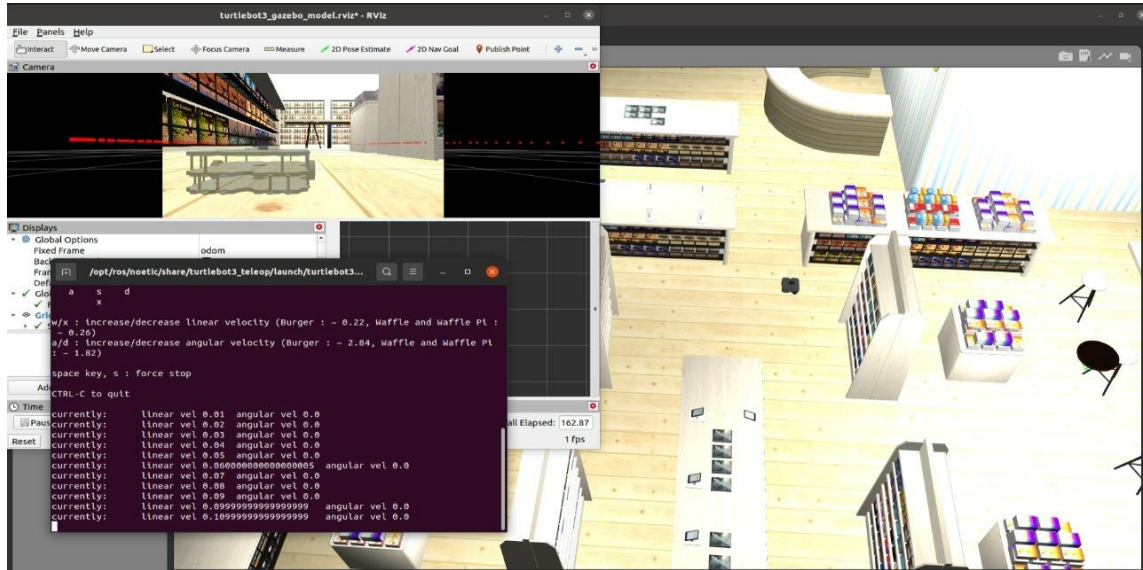
*Turtlebot rviz with the camera images*

### c. Controlling system

To control the robot and make it drives around the store we had three possibilities:

#### - **Manually controlling**

The first and most simple method was to just control the robot using the keyboard A W S D X keys.



*At the bottom left is possible to see the terminal with the commands.*

#### - **Autonomous driving**

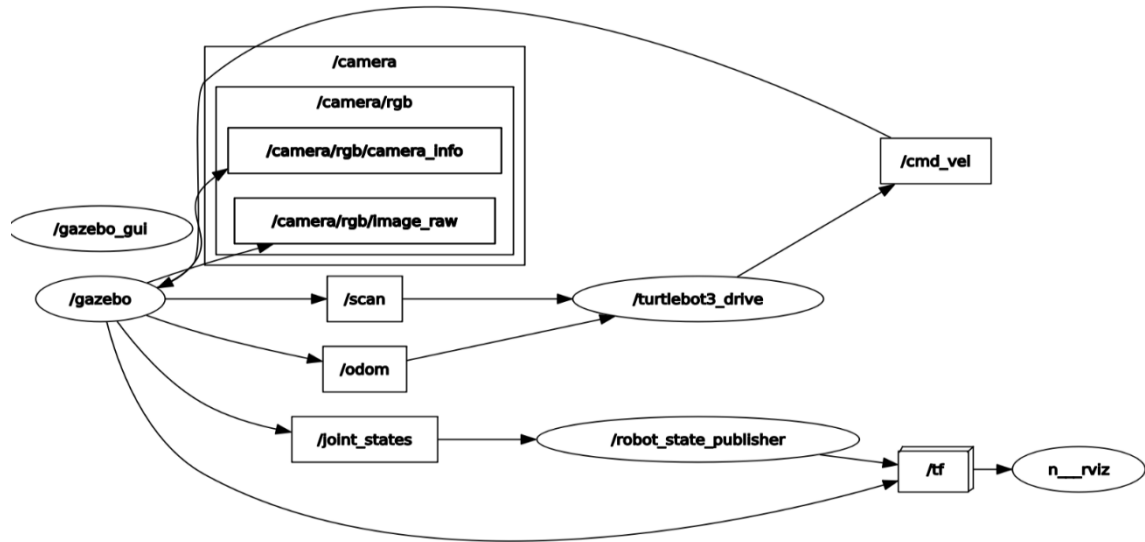
To do the autonomous driving we had to add a node that would make the robot avoid hitting the obstacles, to do so we used the Lidar. This way the rover could drive around the store without bumping into tables, chairs, and shelves.

This is a good option to make a recognition of the room, so that we can latter to the next step of autonomous driving, follow a pre-existing route.

#### - **Following a route**

To do this part is important to have done a recognition of the room, this way we know all the coordinates of the obstacles and a safe path to the shelf we want to go to. To make a route we need a sequence of x, y and z coordinates to determine were the robot needs to go, were to stops and were to move next. Placing all of the coordinates in order, creates the path that the robot can follow, making the process more professional and practical for a bookstore or library.

#### d. QRT of the program



After running all the nodes and lines of our Library Book checker robot, this is the QRT Framework that we get. Here we can see how each node talks to each other, the publishing and subscribing happening and everything working together.

## 6- Possible enhancements

To really achieve our first goal, we need to implement the QR Code camera code in the process, by making it subscribe to the turtlebot camera and lock the image shown there and place the QR Codes inside of our bookstore environment. All of this is possible to be made with more time and research, that as requested, we will try to do them by Sunday.

## 7- References

- 1- <https://www.robotsscience.com/resource/difference-between-a-robot-and-a-machine/#:~:text=Robots%20are%20self%2Dgoverning%20machines,like%20turning%20on%20an%20umbrella.>
- 2- <https://www.sciencedirect.com/science/article/pii/S1877050921023425?via%3Dihub>
- 3- <https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics>
- 4- <https://www.ros.org/>
- 5- Renjun Li, Zhiyong Huang, E. Kurniawan and Chin Keong Ho, "AuRoSS: An Autonomous Robotic Shelf Scanning system," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 6100-6105, doi: 10.1109/IROS.2015.7354246.
- 6- <https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/>