Simulation:

What is Ros?

Ros (Robot Operating system) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms.

Why using Ros?

Because creating truly robust, general-purpose robot software is *hard*. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own.

As a result, ROS was built from the ground up to encourage *collaborative* robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work, as is described throughout this site.

The software used for simulation is gazebo, we have chosen gazebo simulator:

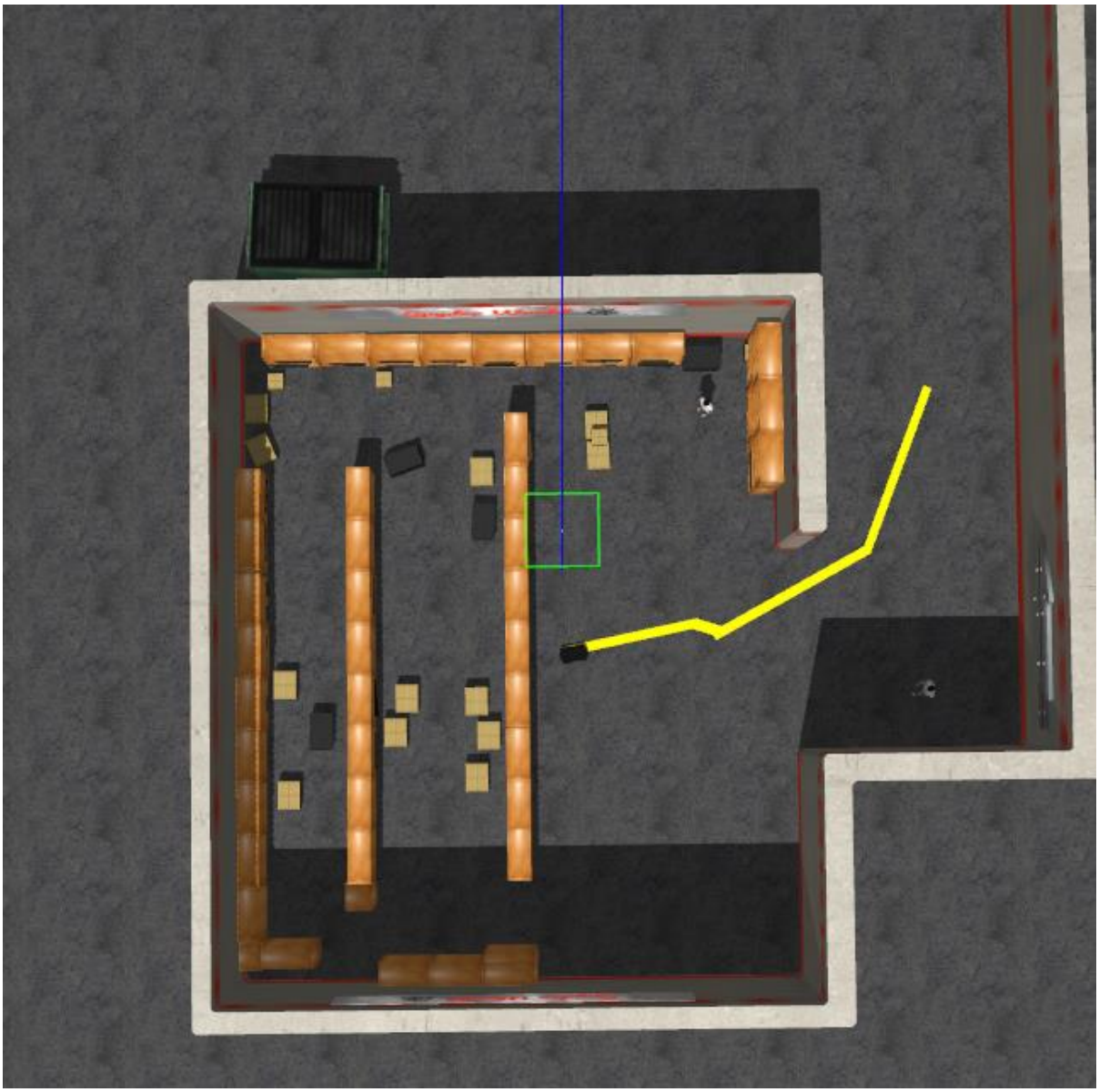
Because it’s a well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Plus, it is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community.

Robot Environment in Gazebo:

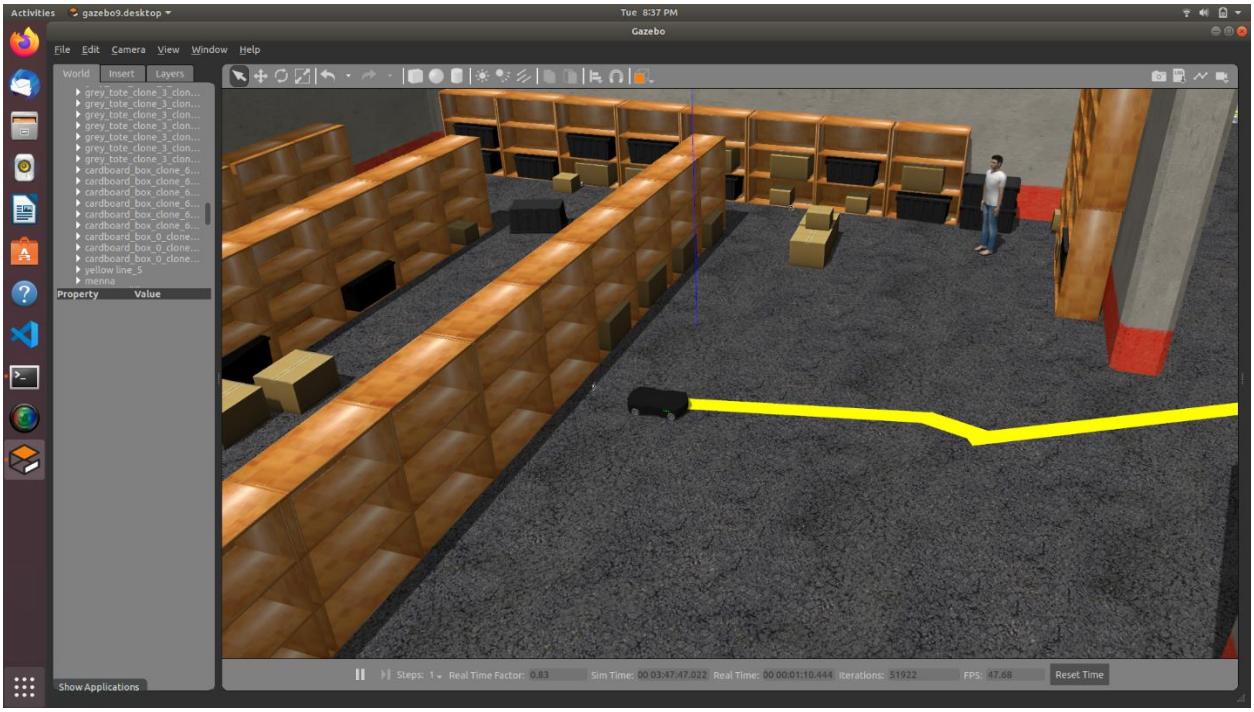
Basically, our robot is operating at Mall or hypermarkets warehouses, which are full of stocks, shelves, and employees everywhere.

Therefore, to simulate our robot in a pretty similar environment we built our own world to match the challenges that will face the robot.

Here is some close up of the environment:



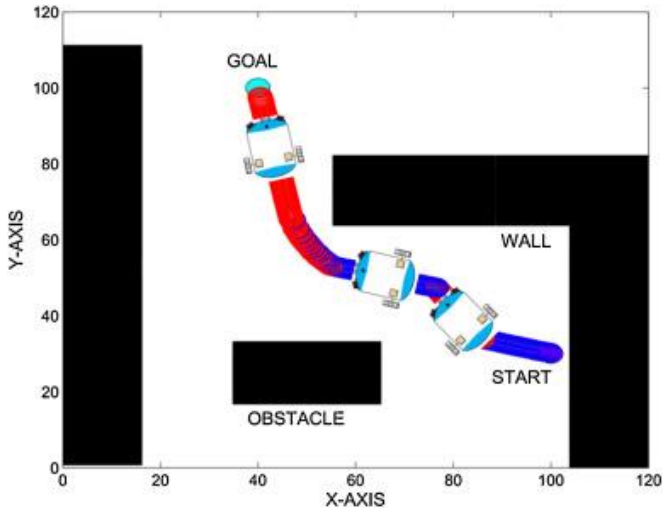

Steps: 1
Real Time Factor: 0.20
Sim Time: 00 03:49:19.113
Real Time: 00 00:04:04.233
Iterations: 144013
FPS: 59.33
Reset Time


Activities
gazebo9.desktop
Tue 8:37 PM
Gazebo
File
Edit
Camera
View
Window
Help
World
Insert
Layers
Property
Value
Show Applications
Steps: 1
Real Time Factor: 0.83
Sim Time: 00 03:47:47.022
Real Time: 00 00:01:10.444
Iterations: 51922
FPS: 47.68
Reset Time

➢ Robot Features

The purpose of adding laser and camera to the robot is to be autonomous by gaining information from both laser scans and camera output , robot is able to:

1- Avoid static and dynamic obstacles.
2- Indoor/outdoor navigation
3- Detect signs (apriltags) and follow certain paths.
4- Follow lines with different colors.
5- Perform tasks and stay away of troubles.



Line Follower using Camera:

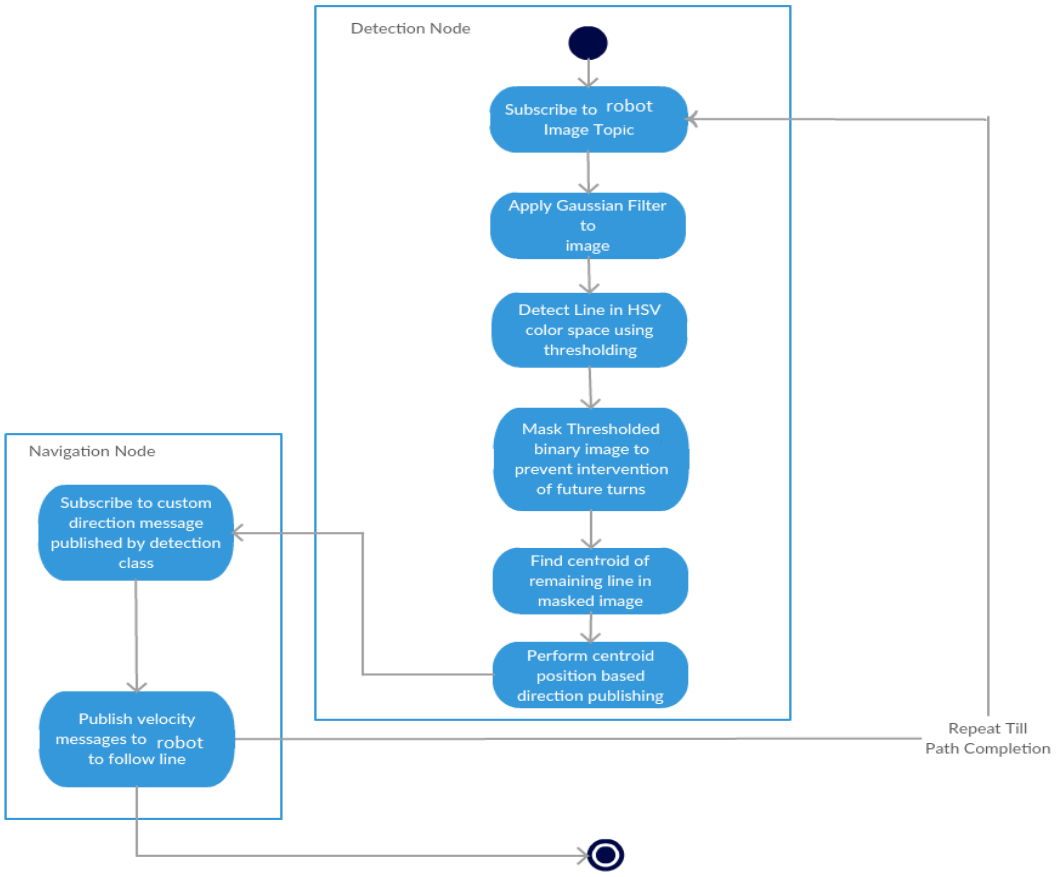
➔ Steps:
1- Get Images from a Camera topic and show them with OpenCV
2- Apply Filters to the Image
3- Move the robot based on the position of the Centroid

This process primarily uses two main classes, one for the detection of the line and one for the navigation of the robot to do so. The detection class first and foremost obtains an image from the robot in the simulated environment, performs color thresholding, masking and centroid detection to communicate the required direction for the robot to move in.

The direction is determined based on the position of the centroid of the detected line with respect to the robot.

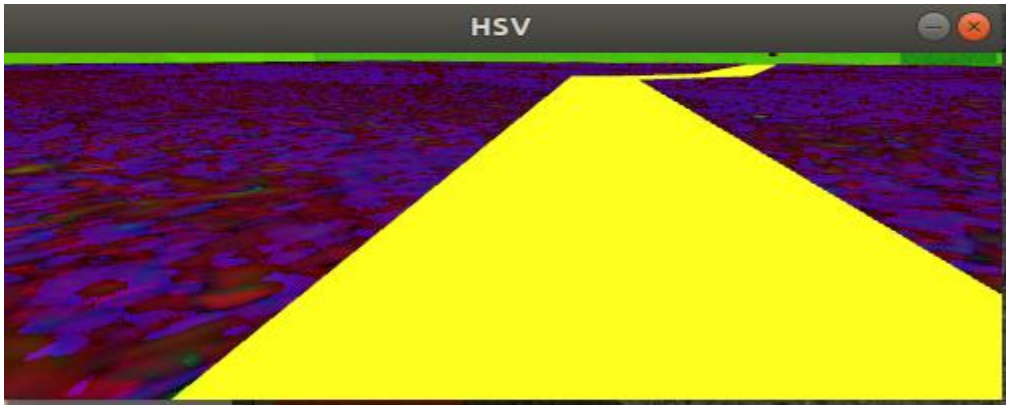
The direction commands are either move forward, turn left, turn right or stop when no line is detected. The direction published by the detection class is used by the navigation class to publish velocity commands to the robot and successfully follow the line throughout the environment. Once the robot reaches the end of the path, it stops.

Illustration Figure:



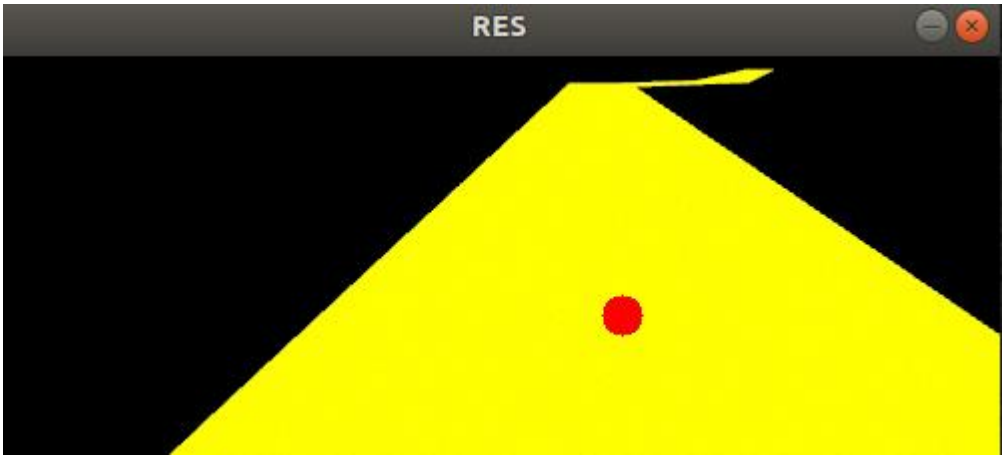
Original Image:

HSV Filter:

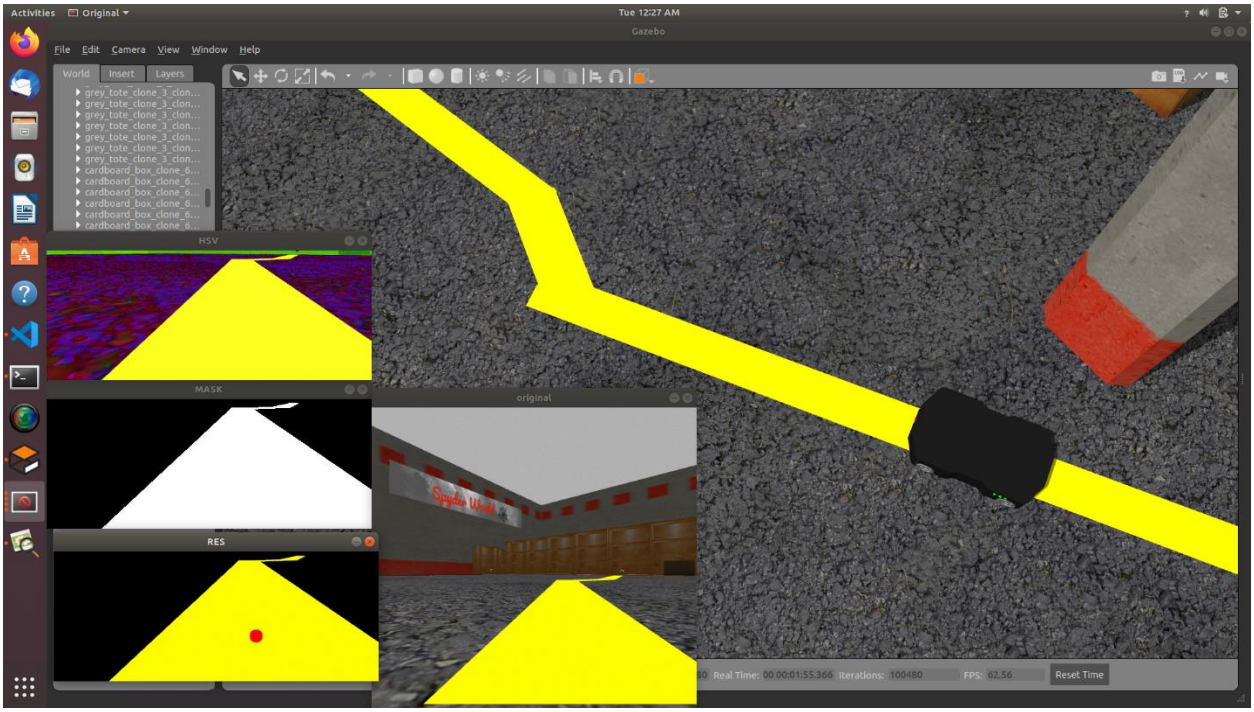


Binary / Mask thresholding:

Final Result:



Scene in Gazebo:

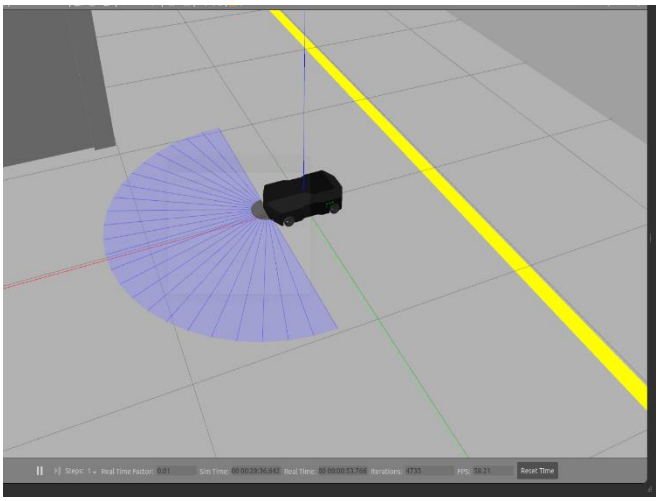


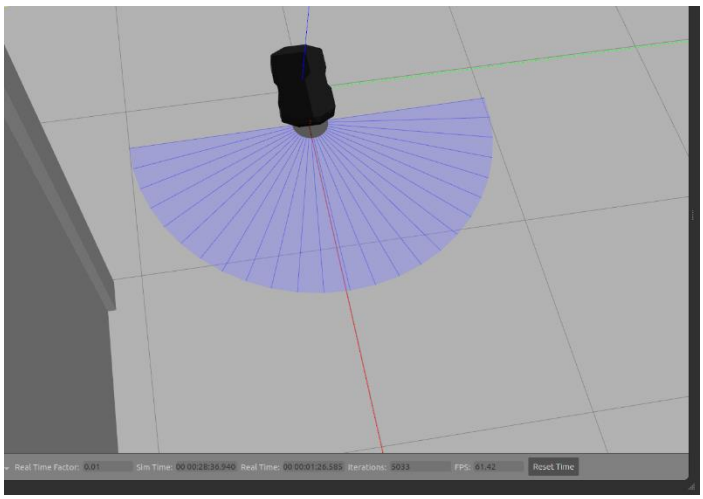
As shown, there are a lot of steps and image processing to conclude the direction which the robot will take.

Shelf Follower/ wall follower using Laser Sensor:

In this process the laser scan has an angle range of -180 ° to 180 ° and distance range is 1 meter.

Here is a visualization of laser rays:





➔ We divided the 25 rays into 5 regions each has 5 rays:

```
regions_ = {
    'right':  min(min(msg.ranges[0:4]), 10),
    'fright': min(min(msg.ranges[5:9]), 10),
    'front':  min(min(msg.ranges[10:14]), 10),
    'fleft':  min(min(msg.ranges[15:19]), 10),
    'left':   min(min(msg.ranges[20:24]), 10),
}
```

➔ The **Five** states that might face the robot:

- For state **Find the Shelf** when:
  1. **No obstacles** detected.
  2. Obstacle on the **left** side.
  3. Obstacles on the **left and right** sides.

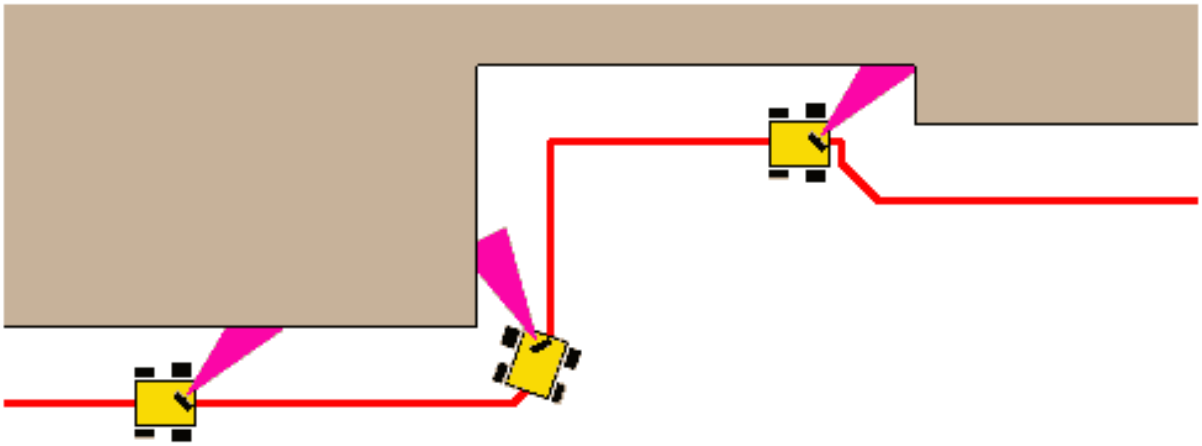
- For the state **Turn left / Left Tag**:
  1. Obstacles in **front** of the robot.
  2. Obstacles in **front and right** of the robot.
  3. Obstacles in **front, left and right** of the robot.
  4. Required to **turn left** to move in a certain path.

- For state **Follow the Shelf** when:
  1. Obstacle detected only on the **right side or left side** of the robot.

- For the state **Turn right / Right Tag**:
  1. Obstacles in **front and left** of the robot.
  2. Required to **turn right** to move in a certain path.

- For the state **Stop Tag:**
  1. Obstacles have a stop tag.
  2. Destination / required goal has reached.

❖ The behavior of Robot against **obstacles or walls**:



Obstacle avoidance using Laser Sensor:

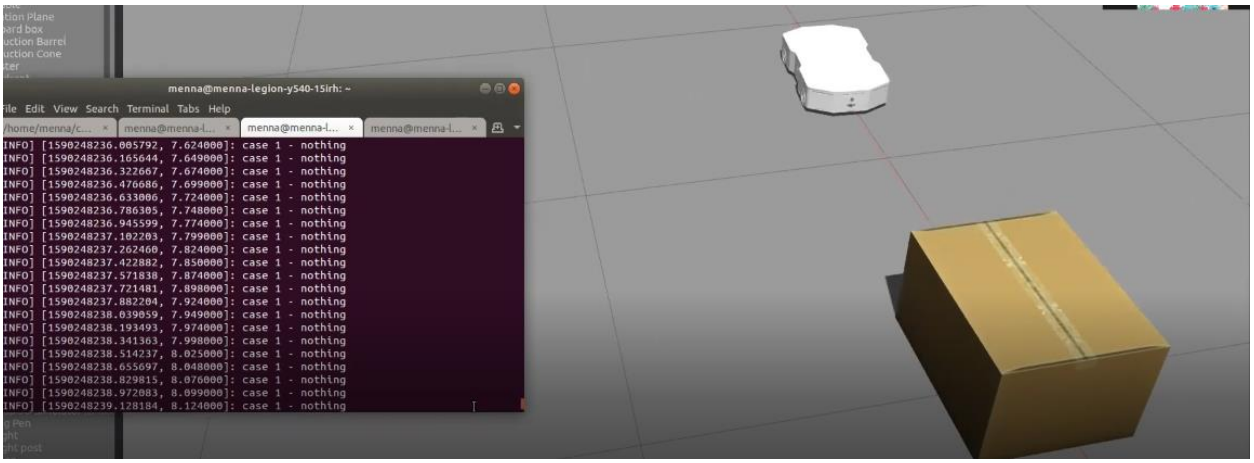
In this process the robot is able to detect and avoid obstacles using laser sensor by configuring 3 regions front, left, and right.
These three regions determine the movement of the robot,
Case 1:
The obstacle is detected in front of the robot → The robot will turn left until the obstacle is gone or no longer detected by laser sensor and then move forward.

Case 2:
The obstacle is detected in the right side of the robot → The robot will turn left until the obstacle is no longer detected and then move forward.

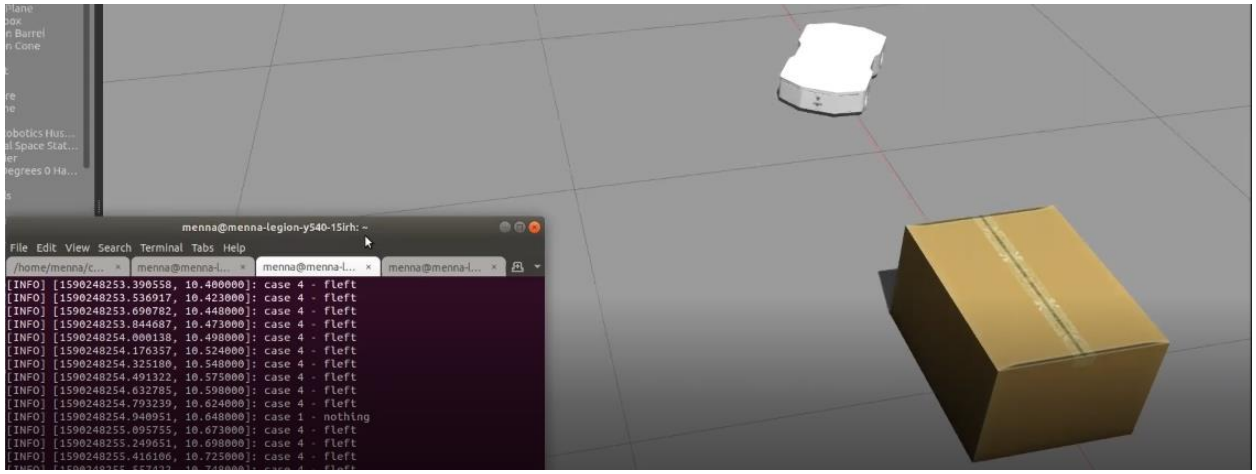
Case 3:
The obstacle is detected in the left side of the robot → The robot will turn right until the obstacle is no longer detected and then move forward.

Illustration:

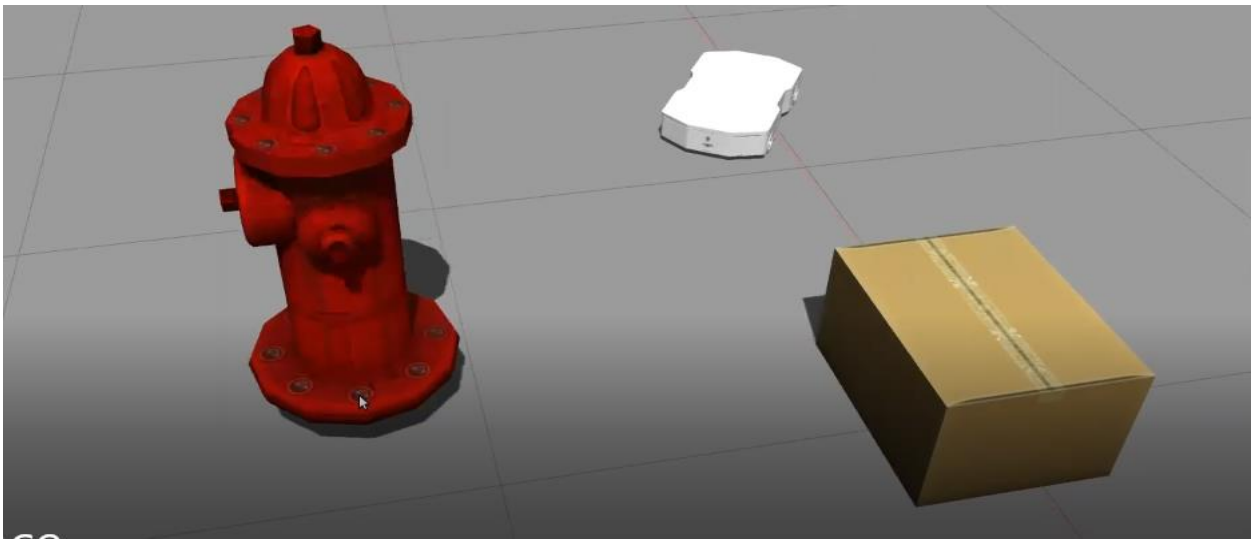
1. The obstacle is in front of the robot

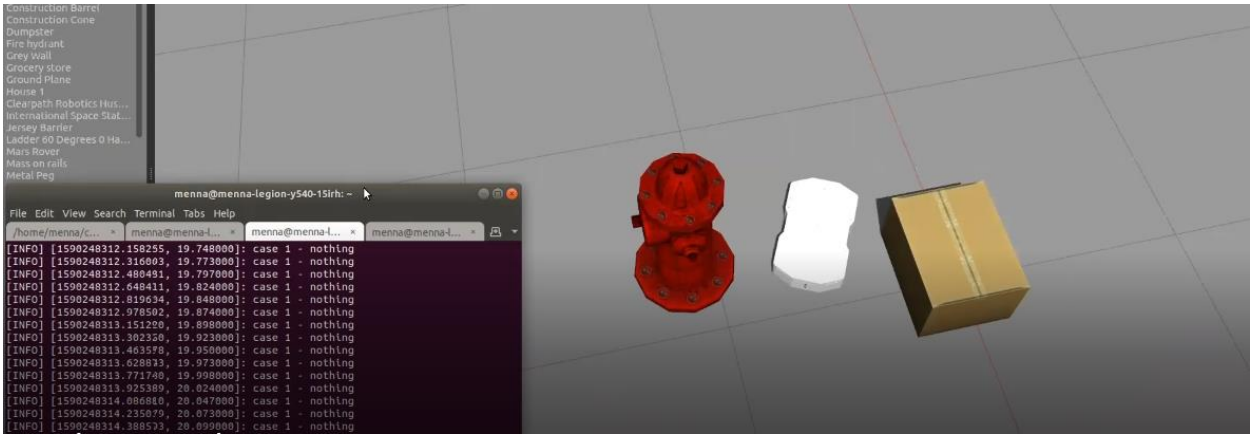


2. Robot starts to turn left to avoid the obstacle

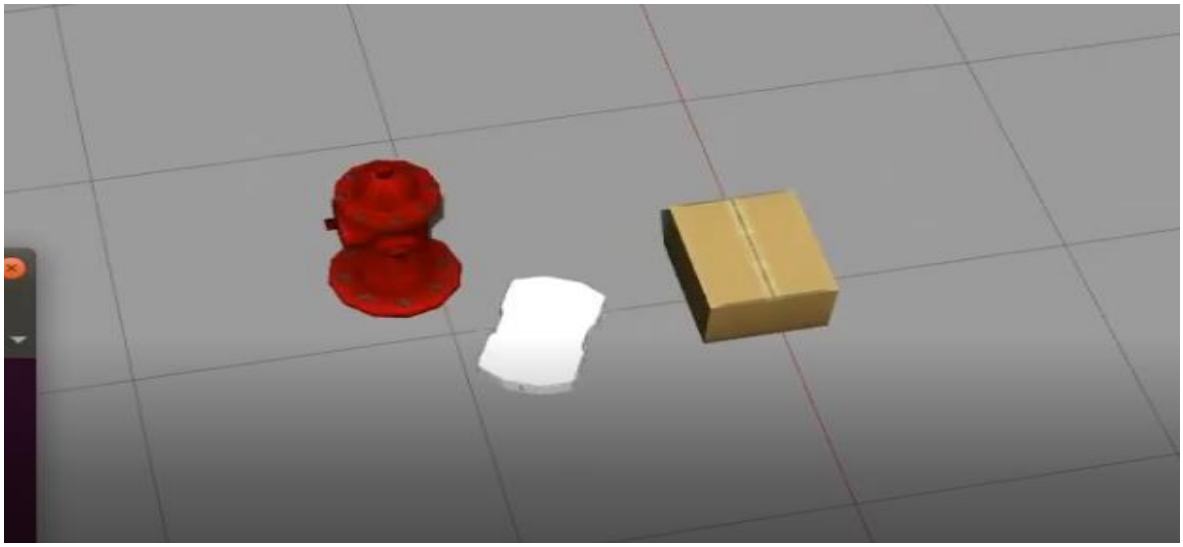


3. To make it more challenging, add another obstacle in the right





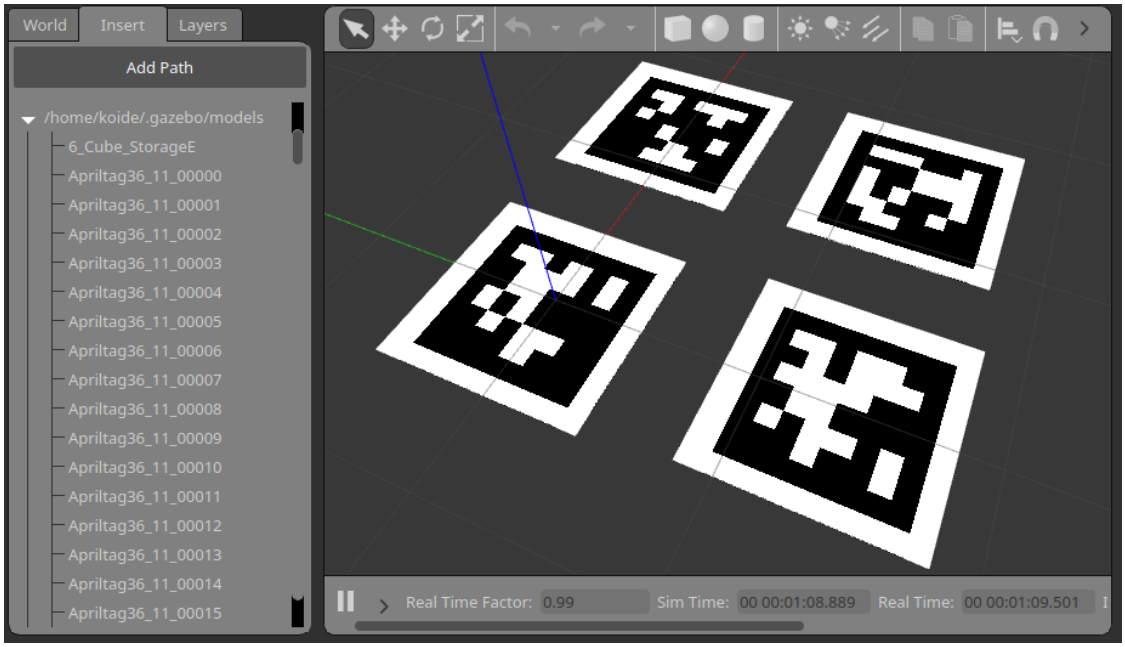
4. The robot moves between the 2 obstacles



Apriltag detection using Camera:

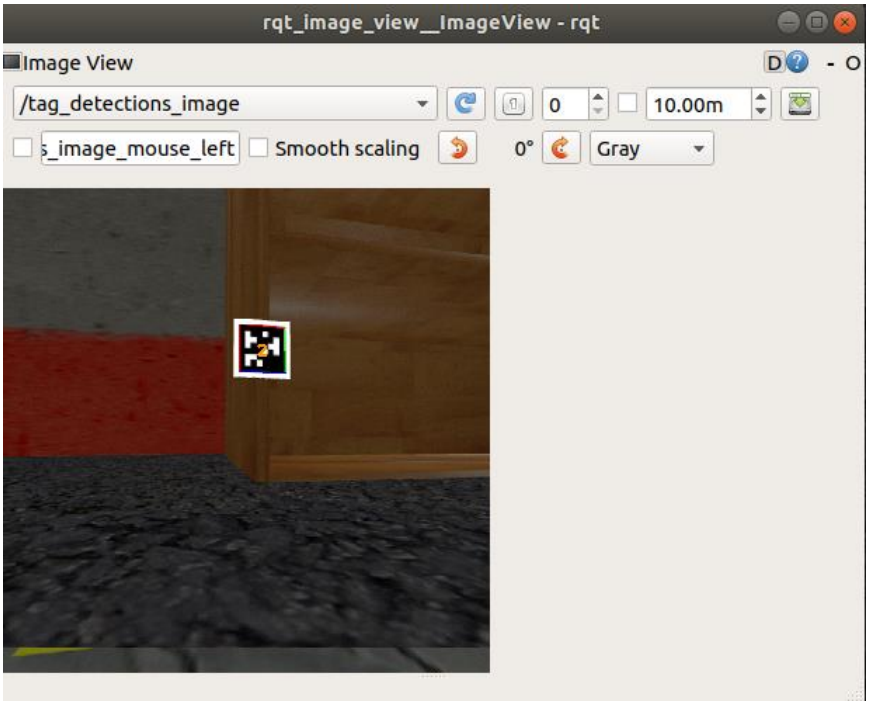
In this process we are using Apriltags to be able to navigate the robot around the environment, tag of id 1 is responsible for making the robot stop when reaching the destination, while tag of id 2 is responsible for turning right in case of wanting the robot to move in a certain path/lane, And finally tag of id 3 is responsible for turning left in case of wanting the robot to move in a certain path/lane.

We are using apriltag family **tag36h11**



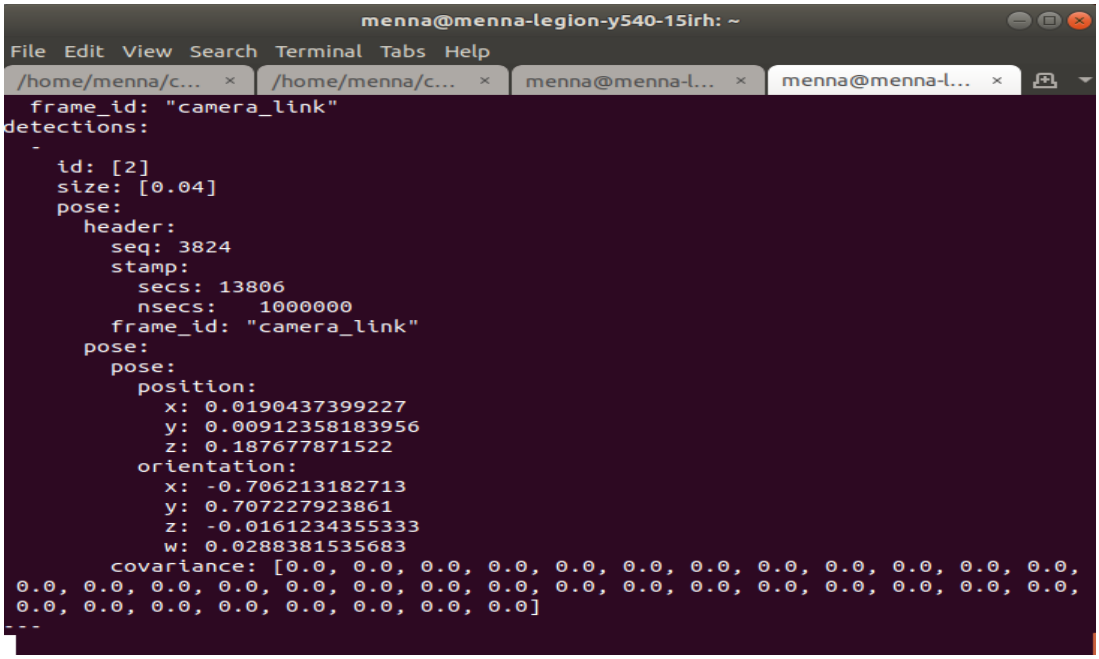
- Overview on how apriltags work:
  Apriltage node subscribes on the image produced by camera and publishes image where tag is detected and assigned to its crosspponding id, also it puplishes the position and orientation of tag.

❖ **Tag detection image:**



❖ **The position and orientation of the tag:**

By detecting the position of tag with respect to camera link the robot can take action based on the distance from the tag



❖ **The scene in Gazebo**

➢ Conceptual Representation

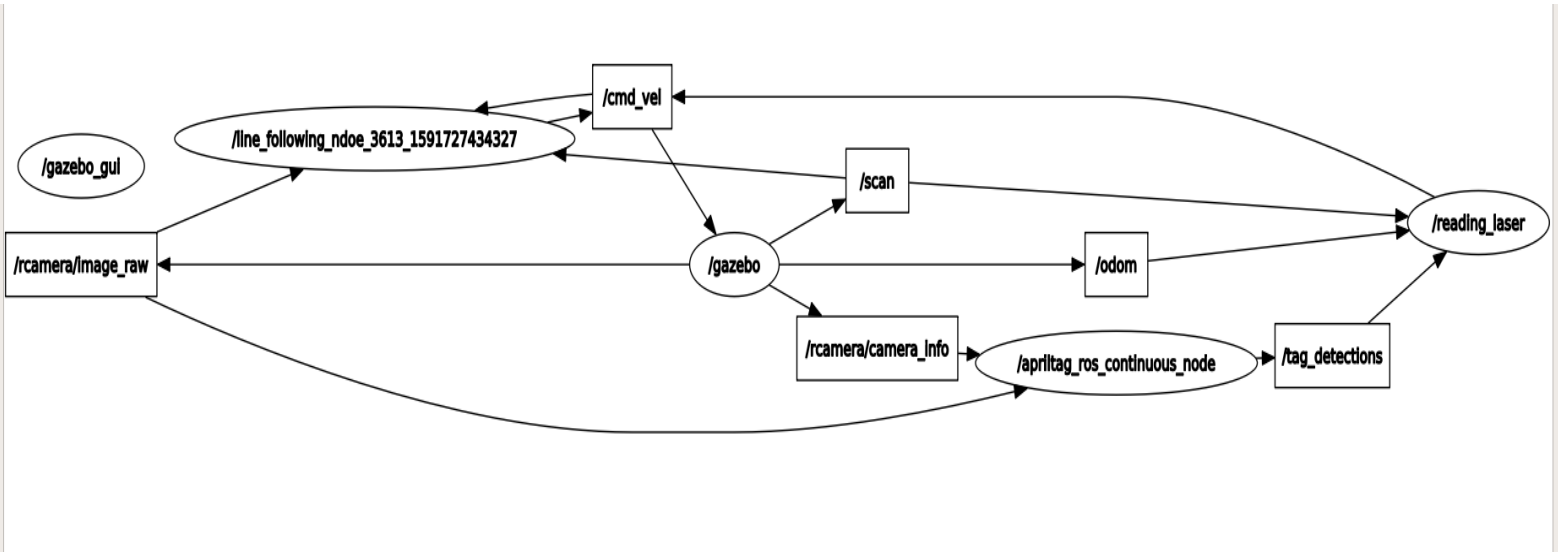
Robot subscribes on laser scans (topic :/scan) , and camera output (topic: /rcamera/image_raw , /rcamera/camera_info) and from both topics it takes action and publishes to (topic : /cmd_vel) which is responsible for robot movement.

Robot also subscribes to odometry (topic: /odom) to determine the exact position and orientation of the robot.

For taking action upon apriltags, the robot subscribes to (topic: /tag_detections) which is generated by (apriltag_ros_contnious_node)

For taking action upon line follower, robot subscribes on (topic: /rcamera/image_raw) and publish the commands on (topic: /cmd_vel) depending on node (line_following_node).

In case of any conflict between apriltags and laser scan readings the priority goes to the apriltags despite what laser scan readings values.



More detailed image for topics used and type of messages: