Slam-CV-Navigation

Project Report

EKLAVYA MENTORSHIP PROGRAMME

at

SOCIETY OF ROBOTICS AND AUTOMATION

VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE

MUMBAI

September 2022

ACKNOWLEDGEMENT

The seniors of SRA VJTI really made this learning experience fun and extremely useful to us. The guidance that we received from our mentors helped us build a project which we couldn't have imagined of doing before this.

They not only taught us concepts related to the project but also skills which would help us a long way in our engineering career. We would like to thank SRA VJTI for giving us this golden opportunity and our seniors who took the time and efforts to guide us.

We would also like to specially thank our mentors Aryaman Shardul, Rishabh Bali and Sagar Chotalia for guiding us through the entire duration of the project and helping us achieve our goal.

Advait Dhamorikar

advaitdhamorikar@gmail.com

Dishie Vinchhi

vdishie@gmail.com

Index

1. Project Overview

1.1 Project Idea

The idea of the project was to make a bot able to map its surroundings and localize itself when given landmarks in the same, followed by navigation in a simulated environment.

1.2 Technologies and Tools used:

Technology:

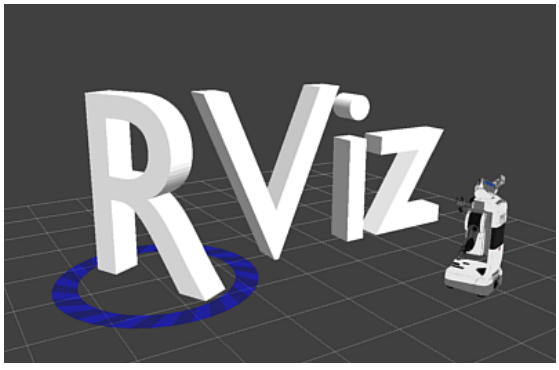
-> ROS Noetic: The Robot Operating System (ROS) versioned Noetic consists of a set of software libraries and tools that help you build robot applications.

Tools:

-> Gazebo: It is an open source 3-D robotics simulator. Gazebo can use multiple high-performance physics engines, such as Open Dynamics Engine(ODE), Bullet, etc with the default one being ODE. It helps us simulate our robot under different conditions which can be quite handy for testing our robot under different environments. It has many plug-ins like sensor plug-ins, world plug-ins etc which help in enhancing the bot's functionalities and creating conditions nearly as good as the real world.

In this project, ROS and Gazebo (version: 11.0) have been used for the simulation.

-> **RViz:** It is a 3-D visualization tool. It is highly configurable. Here it is used for visualizing the line the bot has to follow, through the RGB camera attached on the bot.



2. Introduction

2.1 Project Domains

- SLAM
- Deep Learning
- Computer Vision

2.2 Theory

The project starts off with launching world files (our environment) in the gazebo. It is followed by adding working sensors (LiDAR and depth camera) and plug-ins to our simulated bot using ROS and then launching it in the world. Hence, basic knowledge about ROS and gazebo is important to go about with the project.

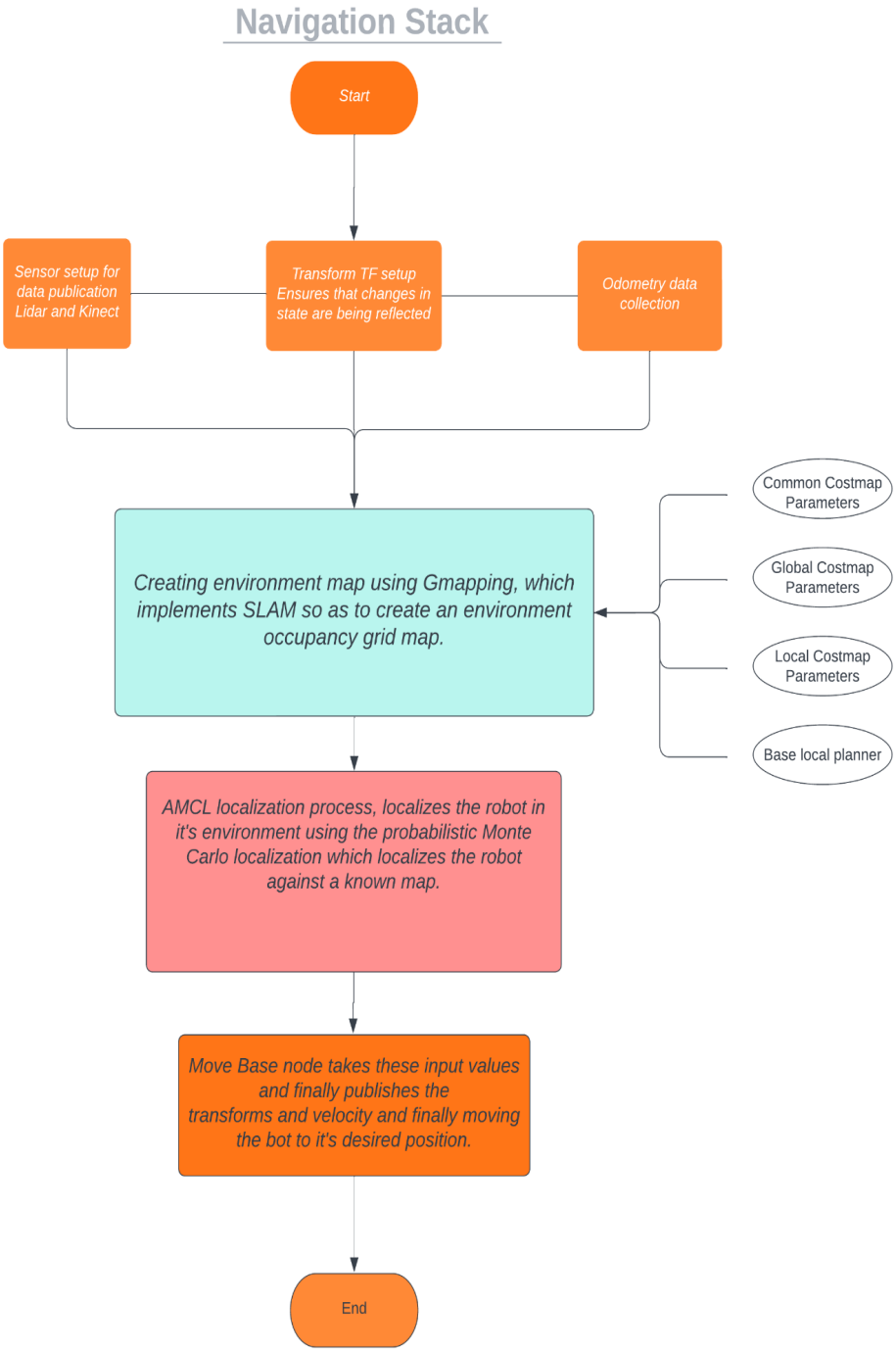
Once the bot is in the required surroundings, the latter is mapped using a package of ROS named gmapping which uses LiDAR data of the bot to sense its environment. The map are created using global and local mapping techniques, In the global planning stage we try to find a collision free kinematically feasible path from start to goal while skipping the dynamic constraints. In the local planning stage, we try dealing with the differential/dynamic constraints that might come up.

The bot is then localized in it’s environment using Monte Carlo localization methodology.

Next, navigation is done using odometry (movement of wheels is used to estimate the position of the bot) and sensor readings, these changes are then passed to the transform TF package.

Object detection is accomplished by implementing the YOLO algorithm. Thus, the bot is now able to identify the objects the depth camera registers.

Basics of python, if known, can be helpful in the extension of the project to achieve object-tracking.

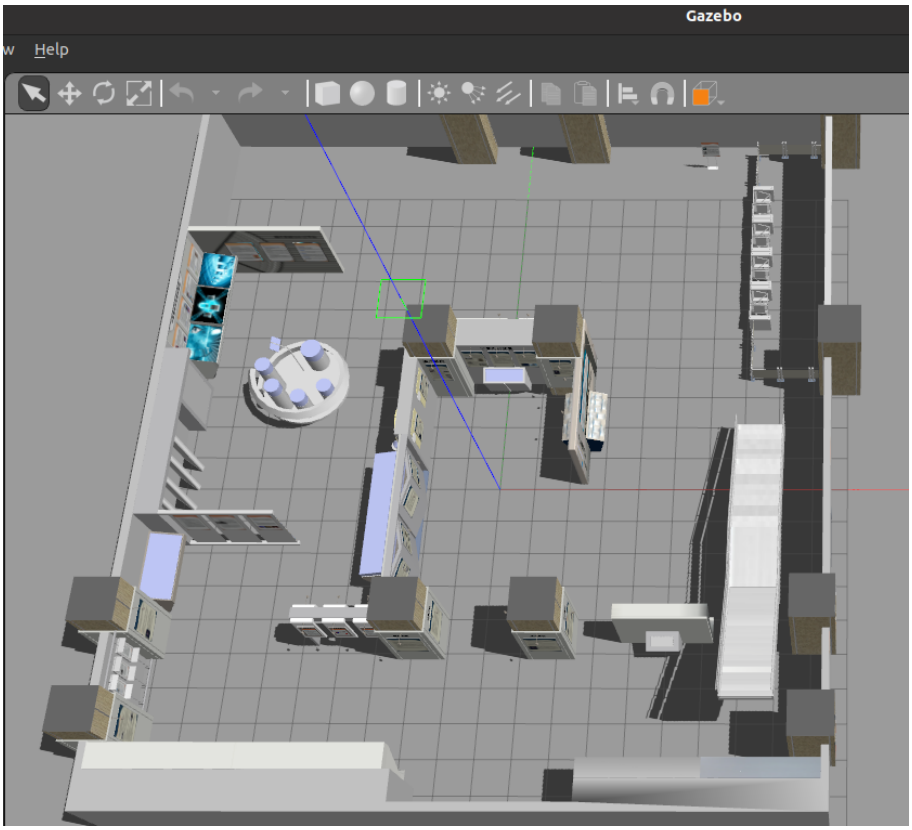

Navigation Stack
Start
Sensor setup for data publication Lidar and Kinect
Transform TF setup Ensures that changes in state are being reflected
Odometry data collection
Creating environment map using Gmapping, which implements SLAM so as to create an environment occupancy grid map.
Common Costmap Parameters
Global Costmap Parameters
Local Costmap Parameters
Base local planner
AMCL localization process, localizes the robot in it's environment using the probabilistic Monte Carlo localization which localizes the robot against a known map.
Move Base node takes these input values and finally publishes the transforms and velocity and finally moving the bot to it's desired position.
End

3. Stages of the project

3.1 Launching world

First, we launch an empty world file in the gazebo. To the launch file, we then add an arguement- world, which launches our environment in the gazebo simulation.

In this project, the world file (sdf format) we used was that of a museum.



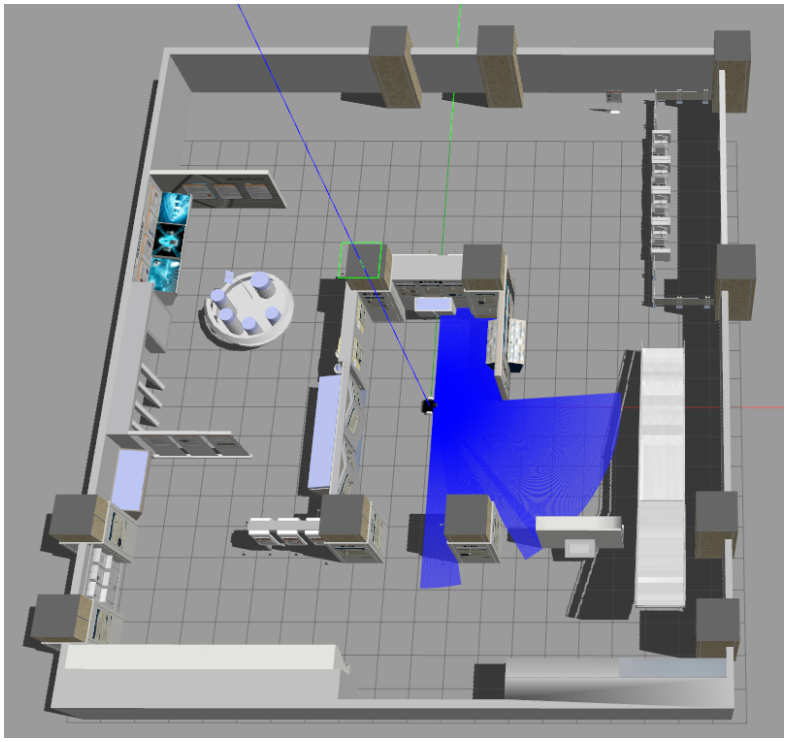
3.2 Launching equipped bot in gazebo

Next, we added the required Gazebo plug-ins (laser and camera) to the urdf files of our bot.

Gazebo plugins give your URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input.

The bot was launched in the museum environment by adding the argument ‘model’ in our launch file.

```
<!-- hokuyo -->
<gazebo reference="hokuyo_link">
  <sensor type="ray" name="head_hokuyo_sensor">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>1</resolution>
          <min_angle>-1.570796</min_angle>
          <max_angle>1.570796</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.10</min>
        <max>6.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/differential_bot/laser/scan</topicName>
      <frameName>hokuyo_link</frameName>
    </plugin>
  </sensor>
</gazebo>
```

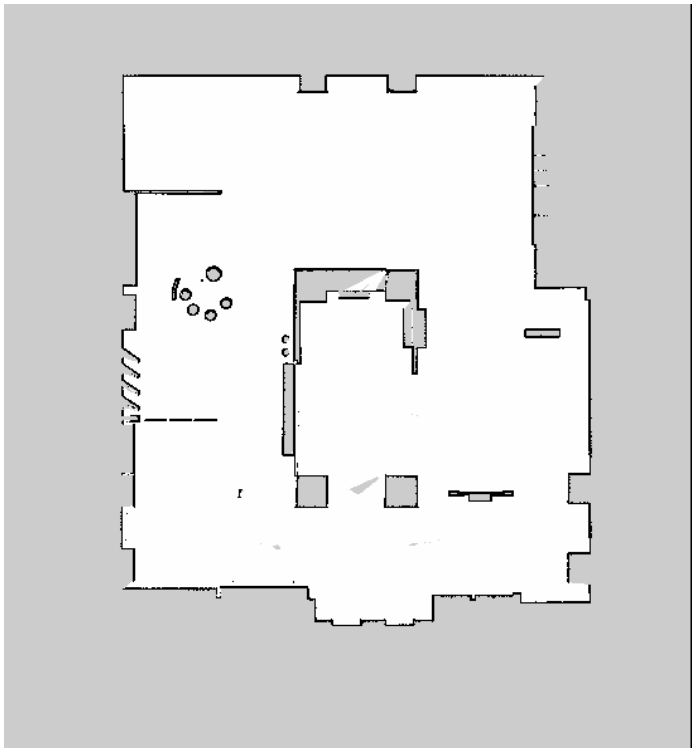


3.3 Mapping of the environment

A map of the museum is obtained by importing a package called Gmapping in ROS.

The gmapping package provides laser-based SLAM, as a ROS node called gmapping. Using gmapping, you can create a 2-D occupancy grid map (like a building floor plan) from laser and pose data collected by a mobile robot.

Once the bot is made to scan the entire museum, a map like below is obtained.



3.4 Localization and Navigation

The bot first localizes itself in the environment using amcl this technique is based off the Monte Carlo localization approach

Using global and local costmaps, we make the bot able to navigate its way in the assigned direction by assessing the environment.

What are costmaps?

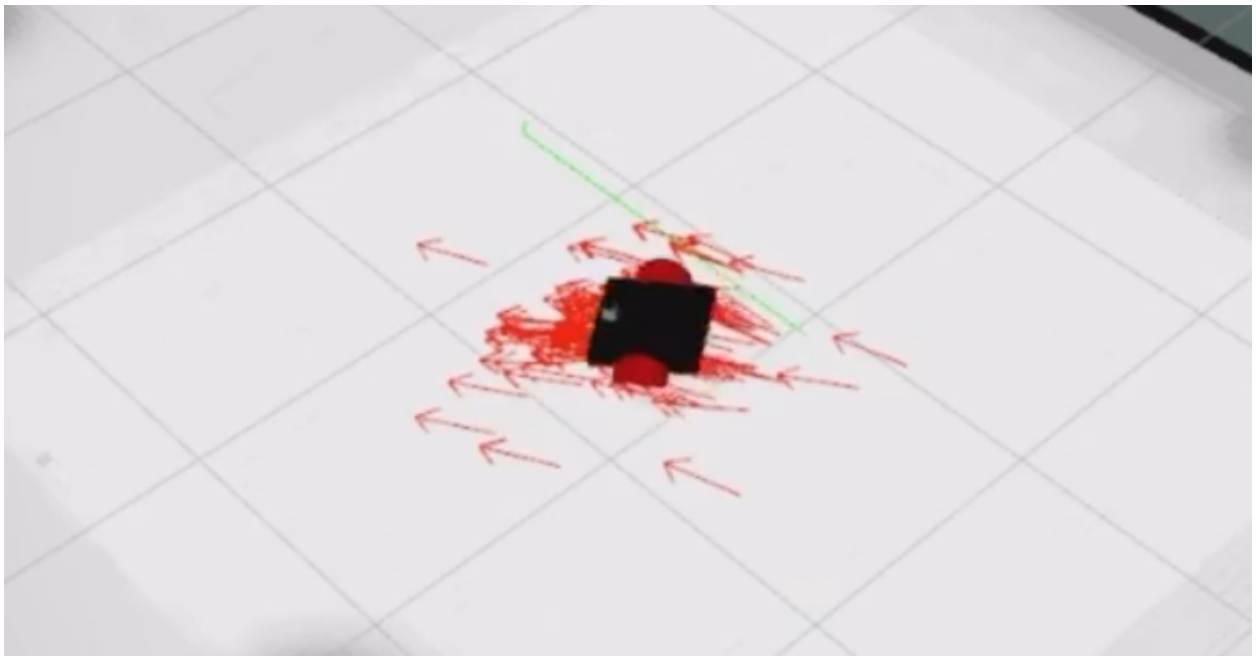
Maps of the environment which are determined using registered sensor values and odometry.

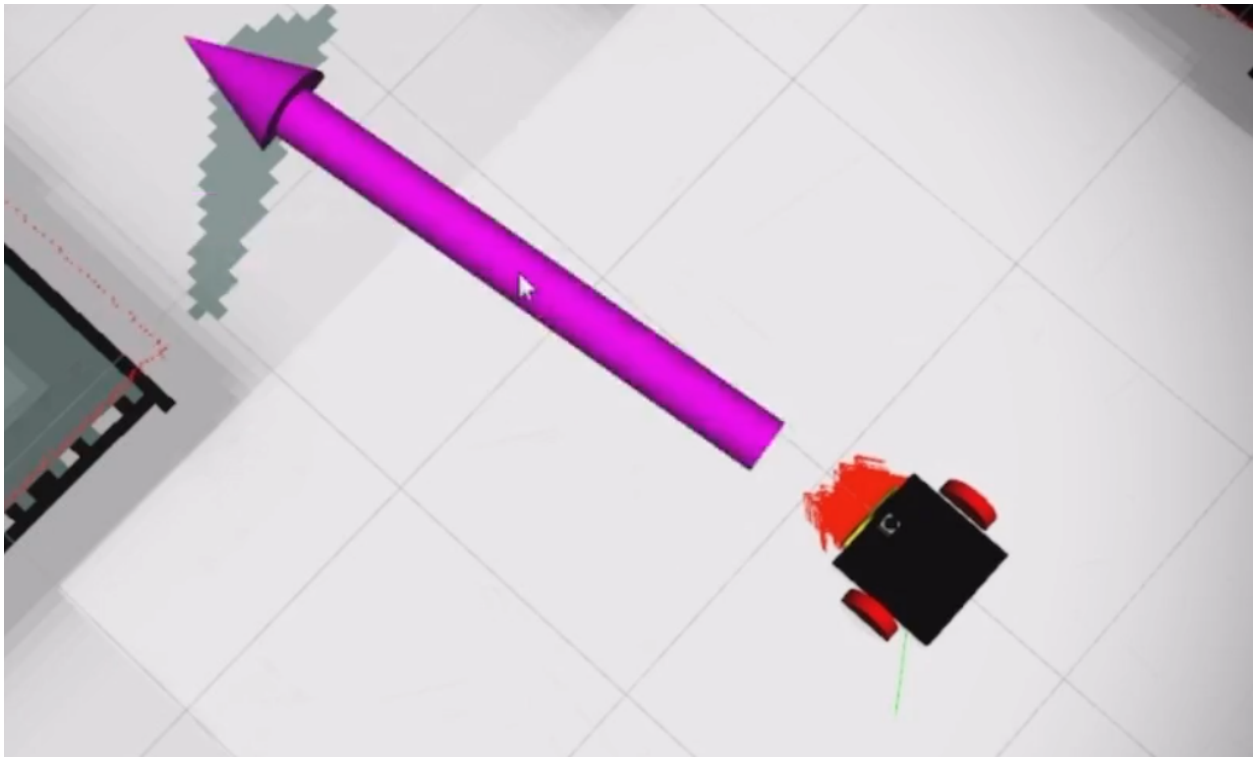
What are global and local costmaps?

Local costmaps use the bot's current location, sensor values to give an idea of its present surroundings.

Global costmaps are generated using previously stored sensor values and odometry readings of the bot.

The parameters of these costmaps can be tuned for more accurate and faster mapping.



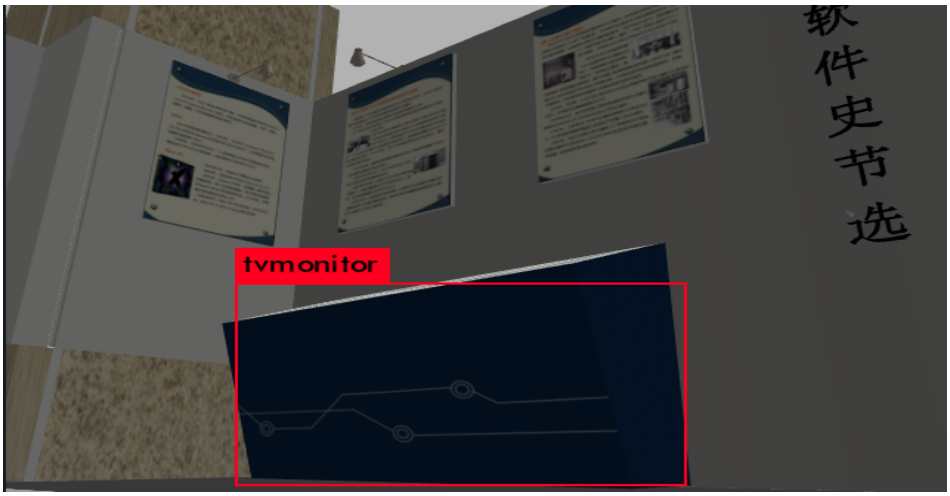


3.5 Object detection

Object detection is used so as to detect a specific object in the frame for the bot to follow. For this project we are utilizing the darknet_ros package

You only look once (YOLO) is a state-of-the-art, real-time object detection system, we use pre-trained model from the YOLO dataset to classify objects in the frame of the bot and thus it can choose which object to follow.

The YOLO output is also utilized further for object following.



3.6 Object following

Object following uses the area of the bounding box which we obtain from the darknet data to determine the distance to the YOLO bounding box.

As the area of the bounding box increases we can know that we are approaching the box and as the area decreases we understand that we are moving away further from the region of interest.

As this data is interpreted values are passed to the bot to adjust it's angular velocity based on the centroid coordinates of the bounding box and linear velocity based on the size of this bounding box.

```
def yolo_readings(msg):
    for box in msg.bounding_boxes:

        parts = {
          'xmin': (box.xmin),
          'xmax': (box.xmax),
          'ymin': (box.ymin),
          'ymax': (box.ymax)
                }
        objname=box.Class
        logic(parts)
```

The code shown above receives the subscribed data from the topic /darknet_ros/bounding_boxes from which it extracts the four coordinates of the edges of the bounding box.

```
def logic(parts):
  linear_speed_x = 0
  angular_speed_z = 0
  msg = Twist()
  height=480
  width=640
  mid_x = (parts['xmax']-parts['xmin'])/2 + parts['xmin']
  mid_y = (parts['ymax']-parts['ymin'])/2 + parts['ymin']
  yolo_width=parts['xmax']-parts['xmin']
  yolo_height=parts['ymax']-parts['ymin']
```

The logic function finds the coordinates of the centroid of this bounding box and compares it with the standard height and width of the frame which has 480x640 dimensions.

```
if mid_x>width/2:
    angular_speed_z = -0.5
  #   prev_angular_speed=angular_speed_z
  #   prev_linear_speed=linear_speed_x

    if distancei > init_dist:
        linear_speed_x = 0.06
      #   prev_angular_speed = angular_speed_z

    elif distancei <= init_dist:
        linear_speed_x = 0
        angular_speed_z = 0
      #   prev_angular_speed = angular_speed_z

    elif distancei == init_dist:
        linear_speed_x = 0
        angular_speed_z = 0
      #   prev_angular_speed = angular_speed_z

elif mid_x < width/2:
    angular_speed_z=0.5
  #   prev_angular_speed = angular_speed_z

    if distancei > init_dist:
        linear_speed_x = 0.06

    elif distancei <= init_dist:
        linear_speed_x = 0
        angular_speed_z = 0

    elif distancei == init_dist:
        linear_speed_x = 0
        angular_speed_z = 0

else:
    rospy.loginfo(parts)
```

It then checks the conditions and checks the position of the bounding box with respect to the standard width and passes angular velocity values to the bot so as to center itself with the bounding box frame.

Further area of the bounding box is calculated and the initial area of the first bounding box is compared to the area of the current bounding box to pass linear velocity values to the bot to move it and follow the object in frame.

4. Project Conclusion and Future Work

4.1 Current working elements

Currently the bot performs the activities of mapping, navigation and object detection, although a different YOLO dataset can be used for giving better results

4.1.1 Mapping

The bot is able to map the environment fairly well with the current configuration

4.1.2 Navigation

The bot is able to localize itself in it's environment and navigate though it can be better tuned to navigate faster

4.1.3 Object detection

The bot does detect objects in the environment from it's sensor readings, however the detected object might not be correctly classified at times, but it can't be controlled by the user as it depends on the dataset being used.

4.14 Object following

At the time of writing this report the bot correctly aligns itself with the frame of the object it is tracking although it can't successfully gauge the depth, but it is being worked upon.

4.2 Problems encountered

We came across some ROS and CMake errors while working on our project many of which have been solved.

Some missing dependencies errors were encountered all of which have been resolved and the respective dependencies can be found on our readme

4.3 Future Work

We are planning to combine and implement object following as well as SLAM of the bot in a combined manner to our project as of now they both work well individually but haven’t been implemented together.

We also hope to implement this project on real hardware and perhaps build lower cost SLAM based utility bots for the masses someday.

5. References

Github Repository

Deep Learning

Andrew Ng's DL course

Neural Networks

Linear Algebra

SLAM

Slam Reference Videos

Slam for Dummies

ROS

Beginner Tutorial

YOLO implementation for ROS

ROS Wiki

Gmapping Implementation tutorial

Special thanks to the authors of these repositories which helped us a lot in this project

https://github.com/PranaliDesai/robomechtrix_ws