Race car Design and Obstacle Avoidance

Project Report

Eklavya Mentorship Program

At

SOCIETY OF ROBOTICS AND AUTOMATION,
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE, MUMBAI

SEPTEMBER 2021

Acknowledgements

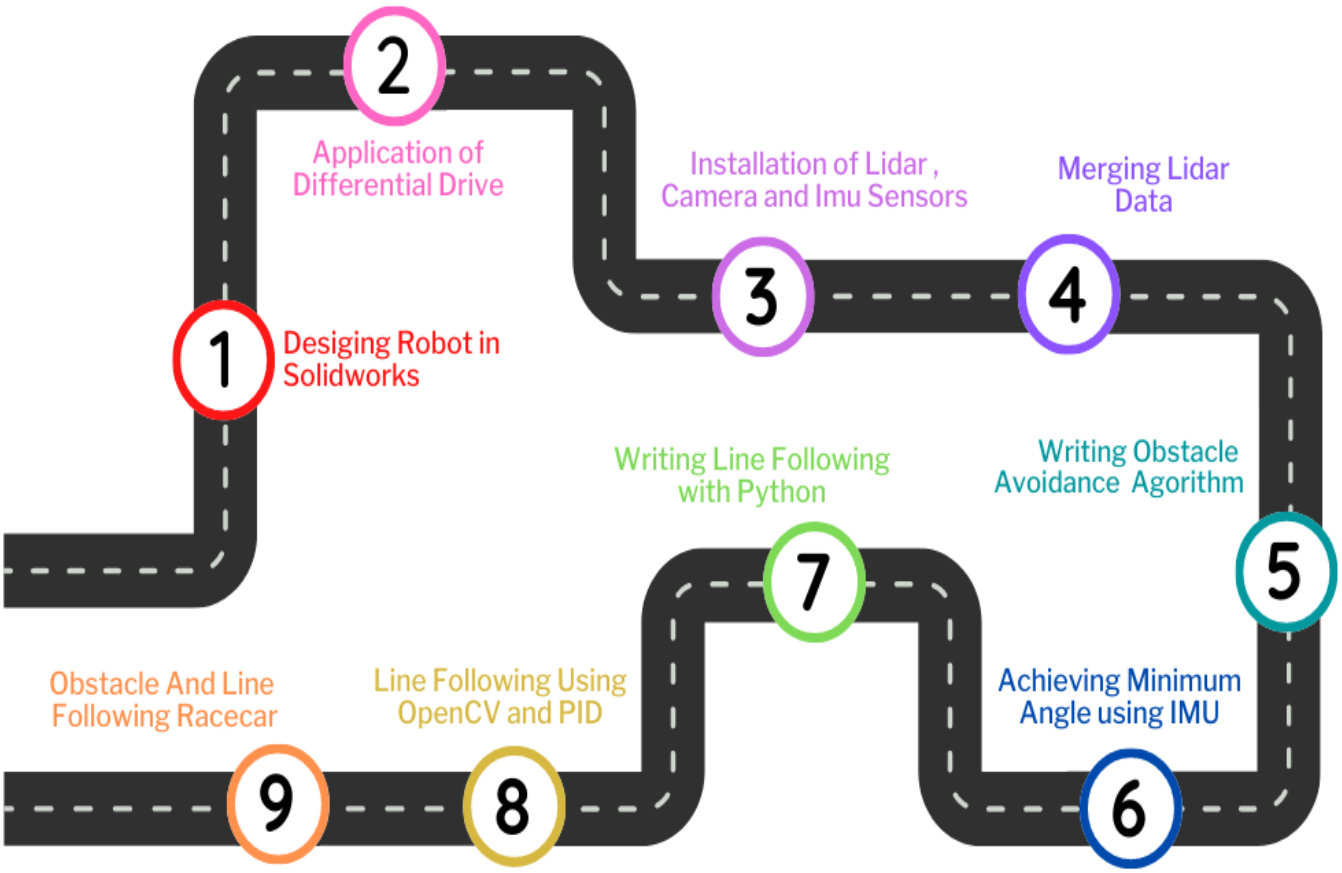
This Project is dedicated to our Mentors **Toshan Luktuke**, **Mark Koothor** and **Aryaman Shardul**, to whom we are extremely grateful, without them this project would just be on paper. We would also like to thank all the members of SRA VJTI for their timely support as well as for organizing Eklavya and giving us a chance to work on this project.

Sameer Gupta
sameergupta4873@gmail.com

Yash Rajput
yashrajput9232@gmail.com

Table of Contents

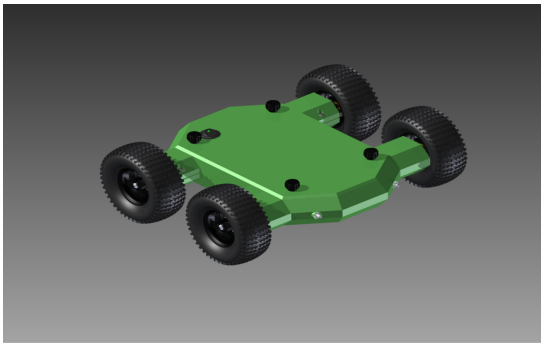
Gist of the Project



1. Introduction

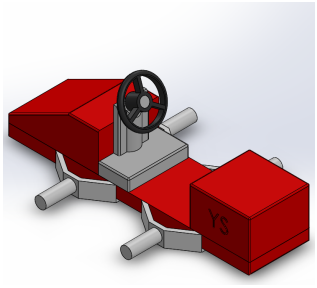
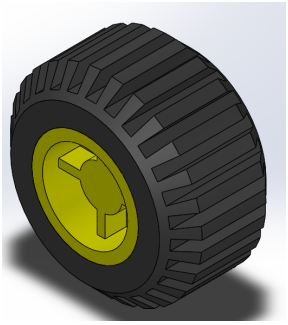
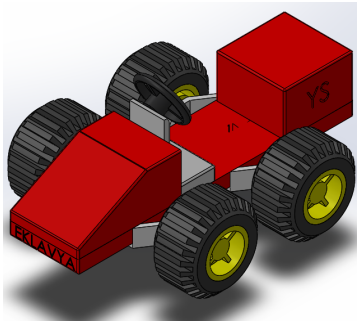
1.1 Designing of Race car on solidworks

SolidWorks is a solid modeling computer-aided design and computer-aided engineering application
Race car was designed on the basis of different measurements such as wheelbase, track width and ground clearance. For the design of the vehicle we had 2 options: Gyro car and Quadracycle A gyrocar is a two-wheeled automobile. The difference between a bicycle or motorcycle and a gyrocar is that in a bike, dynamic balance is provided by the rider, and in some cases by the geometry and mass distribution of the bike itself, and the gyroscopic effects from the wheel. Quadracycle is a European Union vehicle category for four-wheeled microcars, which allows these vehicles to be designed to less stringent requirements when compared to regular cars. We decided to proceed with quadracycle to have better stability and to use diff_drive_controller.



The vehicle is made by the assembly of different parts designed on SW (.sldprt file) and then assembled for the final cad model (.sldasm), while designing the wheel base, track width, ground clearance were kept in mind and changed as per requirements. We decided to refer to this video for the cad model and design according to our needs.

Our cad model has mainly 2 parts:

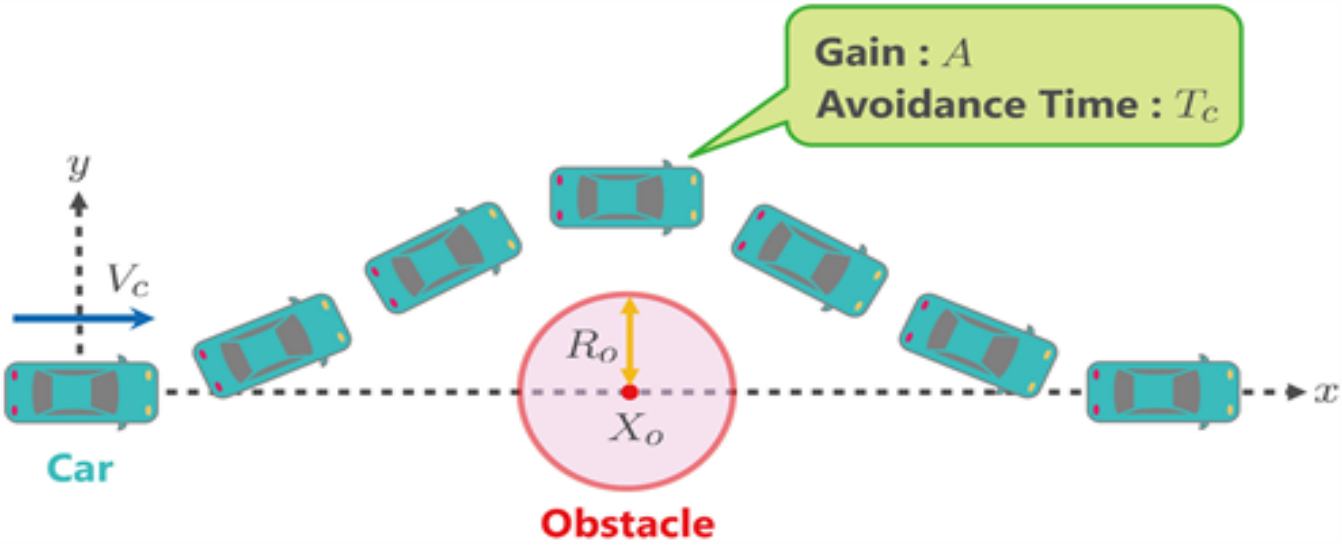
Base Link	Wheel Link	Joint	Final Assembly
		Continuous Joints	

1.2 Exporting URDF -

Universal Robot Description Format (URDF) is an XML format for representing the robot model. It is basically a collection of files which are needed for the physical description of the robot. ROS reads these files and tells the computer about how the robot is going to move, how much it weighs etc.

1.3 Obstacle Avoidance

In robotics, **obstacle avoidance** is the task of satisfying some control objective subject to non-intersection or non-collision position constraints. What is critical about the obstacle avoidance concept in this area is the growing need of usage of unmanned aerial vehicles in urban areas for especially military applications where it can be very useful in city wars. Normally obstacle avoidance is considered to be distinct from path planning in that one is usually implemented as a reactive control law while the other involves the pre-computation of an obstacle-free path which a controller will then guide a robot along. With recent advances in the autonomous vehicles sector, a good and dependable obstacle avoidance feature of a driverless platform is also required to have a robust obstacle detection module.



- Obstacle avoidance Methods
  - The Conventional Potential Field Method
  - The Follow-the-Gap Method
  - The Advanced Fuzzy Potential Field Method
  - The Obstacle Dependent Gaussian Potential Field

- Why did we choose The Obstacle Dependent Gaussian Potential Field ?

1. As there were drawbacks of the other three methods we prefer the ODG-PF algorithm.

2. Drawbacks :-

    a. The Conventional Potential Field Method :
        i. the conventional potential field method has a drawback in that it becomes easily stuck at local minima
    b. The Follow-the-Gap Method:
        i. FGM will be in danger of a collision if an obstacle is between the angle of the maximum gap and the angle toward the goal.
    c. The Advanced Fuzzy Potential Field Method :
        i. AFPFM was better than PFM in all cases, but when it was large, there were collisions. The value should be much larger (by 10 times, for simplicity) than PFM. The reason for this is that there is a coefficient in calculating repulsive field and there are also weights of TS Fuzzy in AFPFM which are a maximum of 15 times larger than those in PFM when an obstacle is placed close to the front of the vehicle.

3. Pros of ODG-PF :

    a. The simulation of ODG-PF performed the best in all cases. We performed the same experiments several times and there were no collisions. It always avoided the obstacles successfully, and its heading angle finally returned to theta goal.

1.4 Ros, Gazebo, RViz

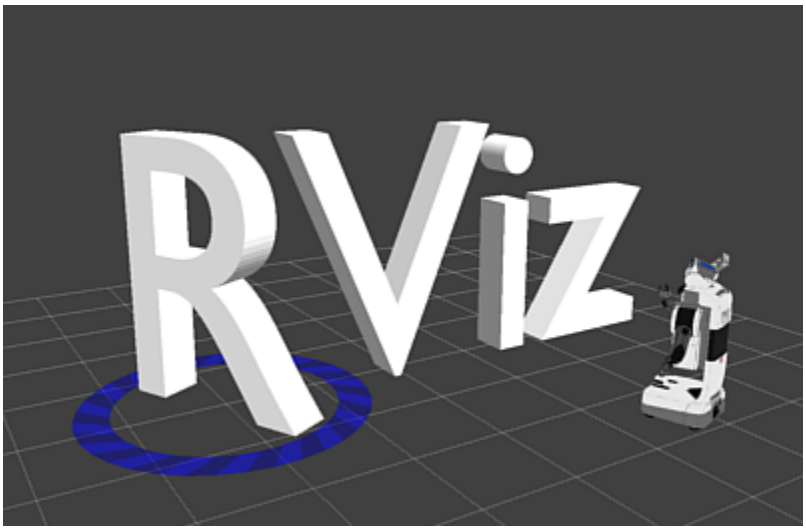
1.3.1 What is ROS (**R**obot **O**perating **S**ystem)?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

1.3.2 What is Gazebosim?

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI systems using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. At your fingertips is a robust physics engine, high-quality graphics, and convenient programmatic and graphical interfaces. Best of all, Gazebo is free with a vibrant community.

1.3.3 What is RViz ?



RVIZ is a ROS graphical interface that allows you to visualize a lot of information, using plugins for many kinds of available topics.

1.5 Gazebo Plugins

Gazebo plugins give your URDF models greater functionality and can tie in ROS messages and service calls for sensor output and motor input. In this tutorial we explain both how to set up pre-existing plugins and how to create your own custom plugins that can work with ROS.

Plugin Types

Gazebo supports several plugin types, and all of them can be connected to ROS, but only a few types can be referenced through a URDF file:

1. ModelPlugins, to provide access to the physics::Model API
2. SensorPlugins, to provide access to the sensors::Sensor API
3. VisualPlugins, to provide access to the rendering::Visual API

1.6 Sensors and Data

There are several robotics sensors that are supported by official ROS packages and many more supported by the ROS community.

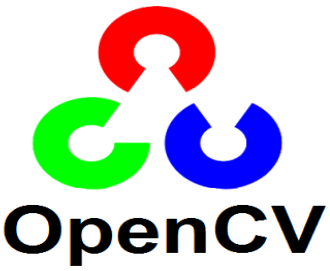
Type of Sensors

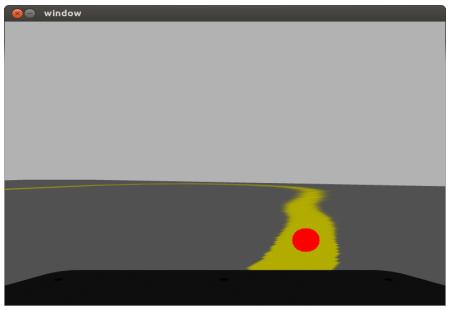
1. 2D range finders
2. 3D Sensors
3. Cameras
4. Pose Estimation (GPS + IMU)
5. Sensor Interfaces

Sensors used in The Projects

1. Lidar / Hokuyo Sensor
2. Camera Sensor
3. IMU Sensor

1.7 Line Following and OpenCV



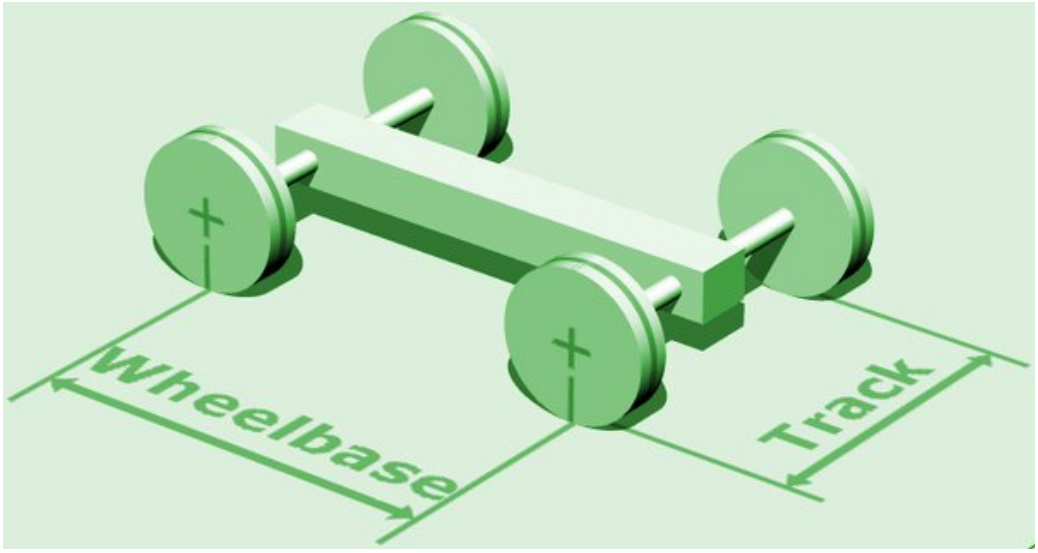


A line following robot is an automated device programmed to follow a specific path. Some of the existing techniques used in controlling line following robots are by IR sensor, LDR sensors, etc. This research work aims to develop image processing-based line following robots. A camera is used to obtain an image of track and then it is converted into a bitmap image. Least Square Method is used to follow the predefined path. By calculating slope of line, angle by which robot should turn is determined. Thus, by using image processing techniques, the following robot is guided along the desired path.

2. In-Depth Analysis of this Project

2.1 Designing in Solidworks and URDF

For designing any vehicle we basically need 3 dimensions. Track width (TW), Wheelbase (WB) and ground clearance (GC), TW is the length between the 2 wheels, WB is the length between the front and rear tires, GC is the length between the bottom point of tire and the bottom point of chassis.

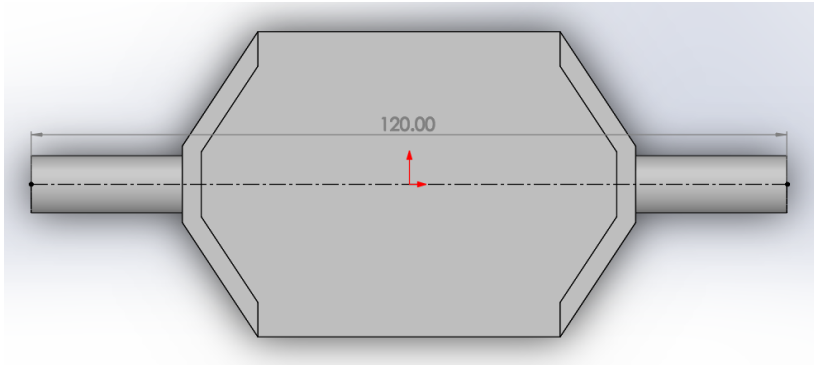


The dimensions of our solidworks design was determined on the basis of the length of the road/ track. The length of our track was __ hence the TW of our vehicle is 120 cm. The WB in our vehicle is 68 cm, the WB is slightly shorter than the original WB, we decided to shorten the length to improve the movement of tires (In case of longer WB the tires were under a lot of friction which made the turning difficult).

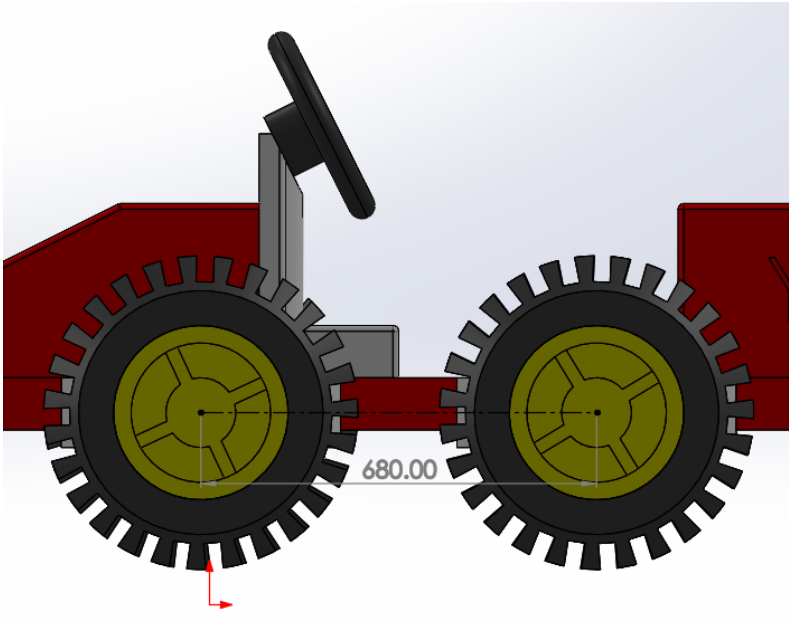
The GC of our vehicle is roughly 24 cm, GC of our vehicle was calculated by trial and error, GC of vehicle was changed by changing the radius of our tires.

Following dimensions are in mm

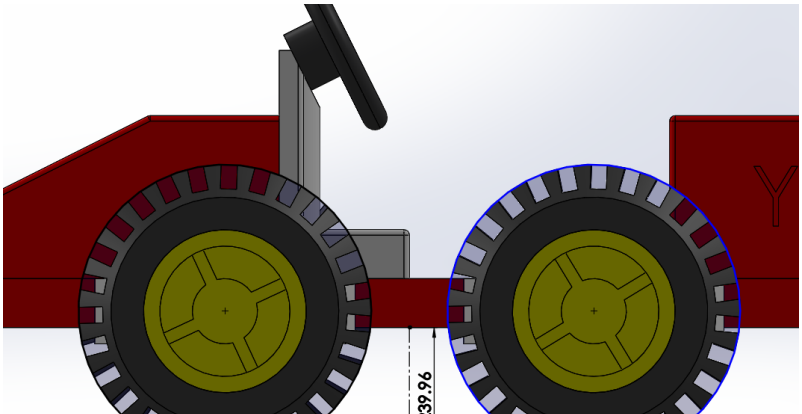
Track Width:



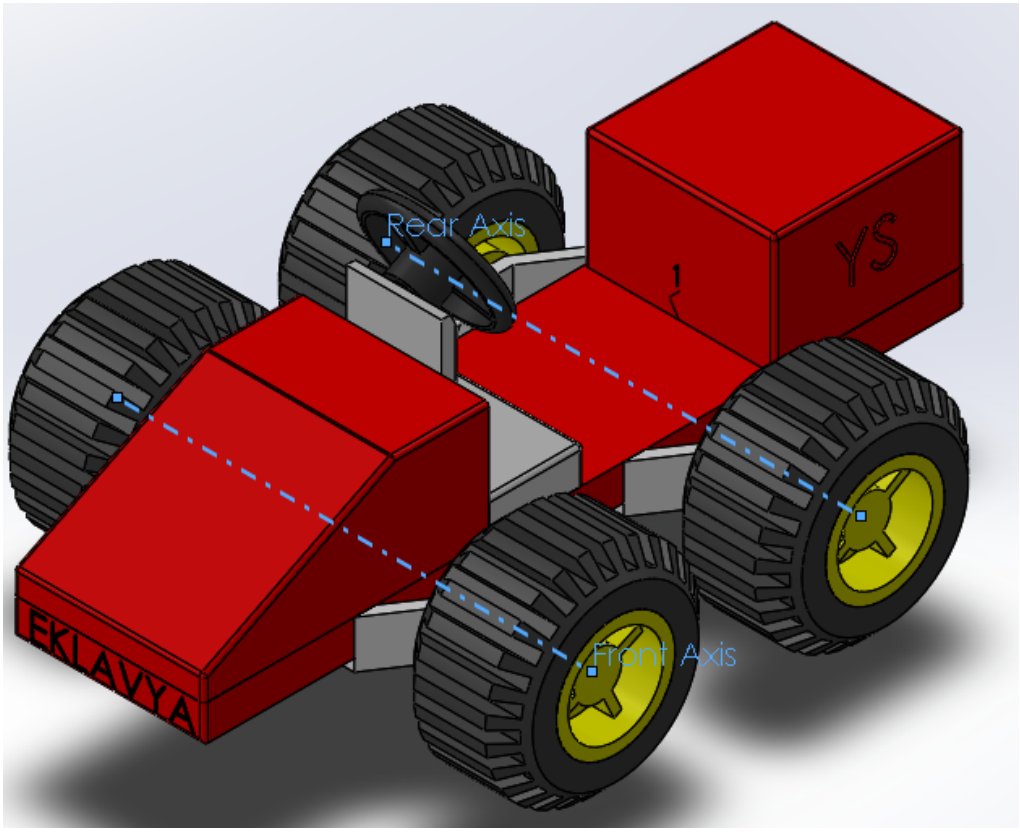
Wheelbase:

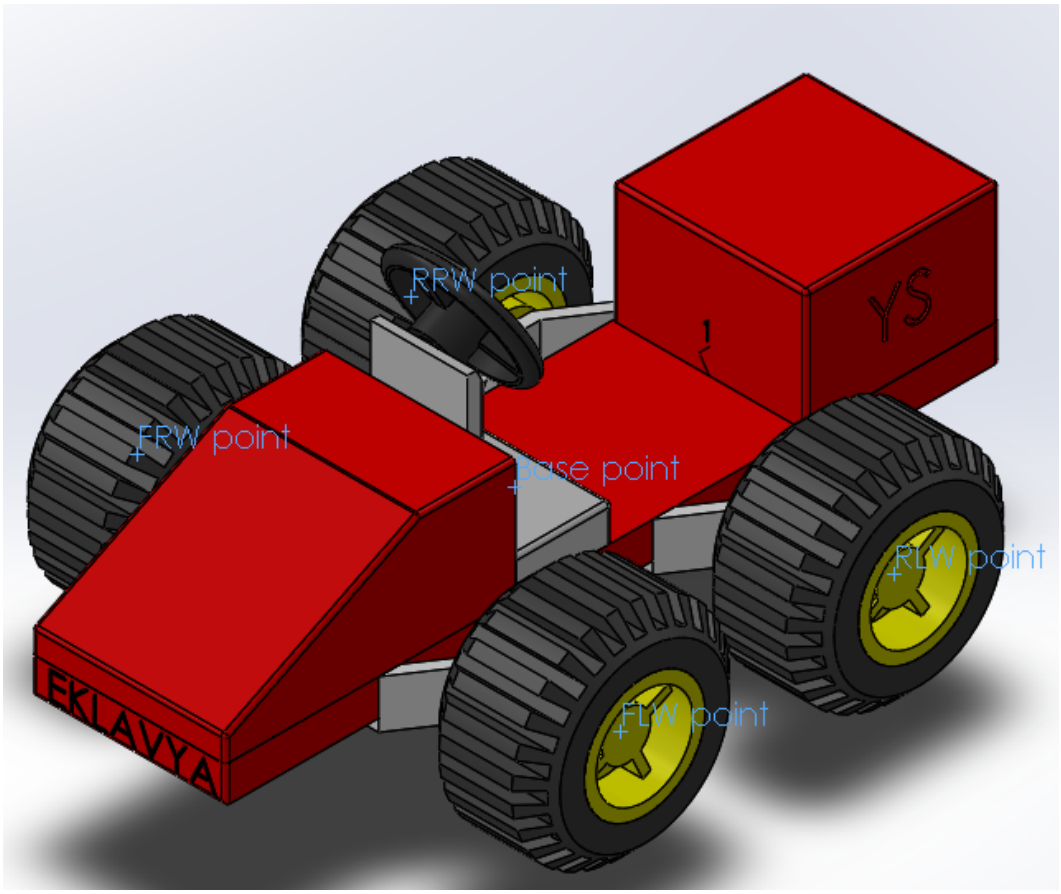


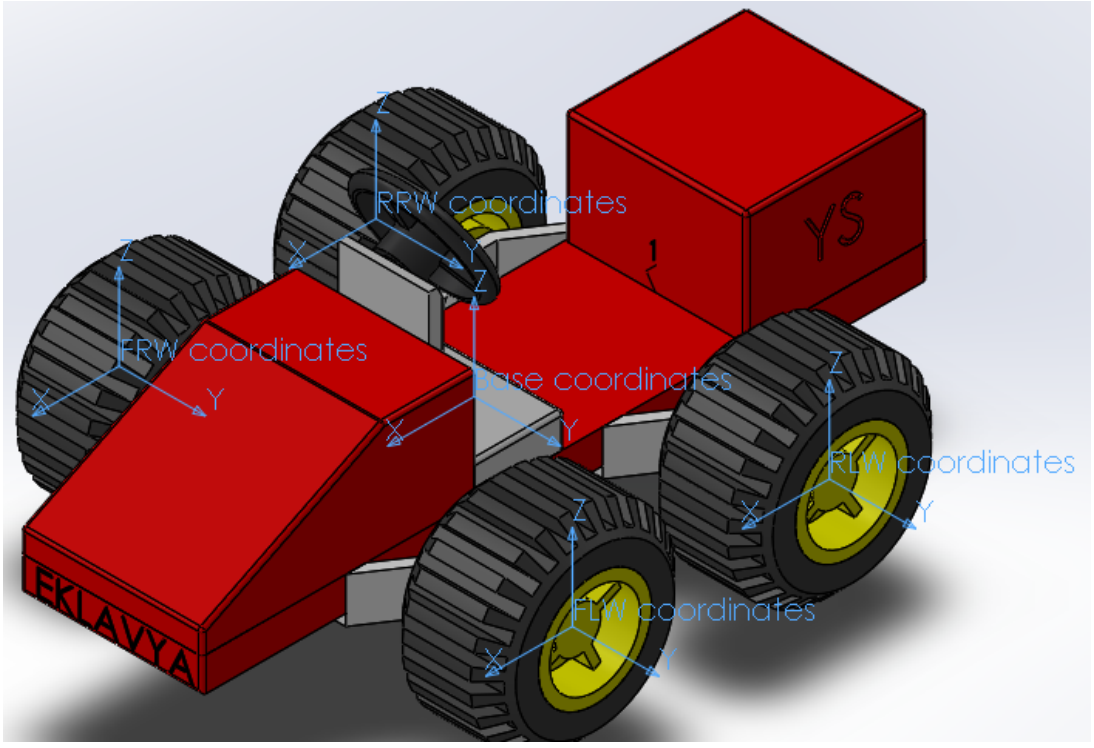
Ground clearance:



After the model is ready we need to assign a few axes, points, coordinate systems for base link and other 4 tires. In total we assigned 5 coordinate systems: Base coordinate system, RRW coordinate system. RLW coordinate system, FRW coordinate system, FLW coordinate system. For the rotation of wheels we need an axis about which the wheels will rotate. 2 points are assigned on the center of the wheels joining them will give you the axis of rotation. Such 4 points are assigned on each wheel. Joining them will give you the front axis for the FRW and FLW and the rear axis for RRW and RLW.

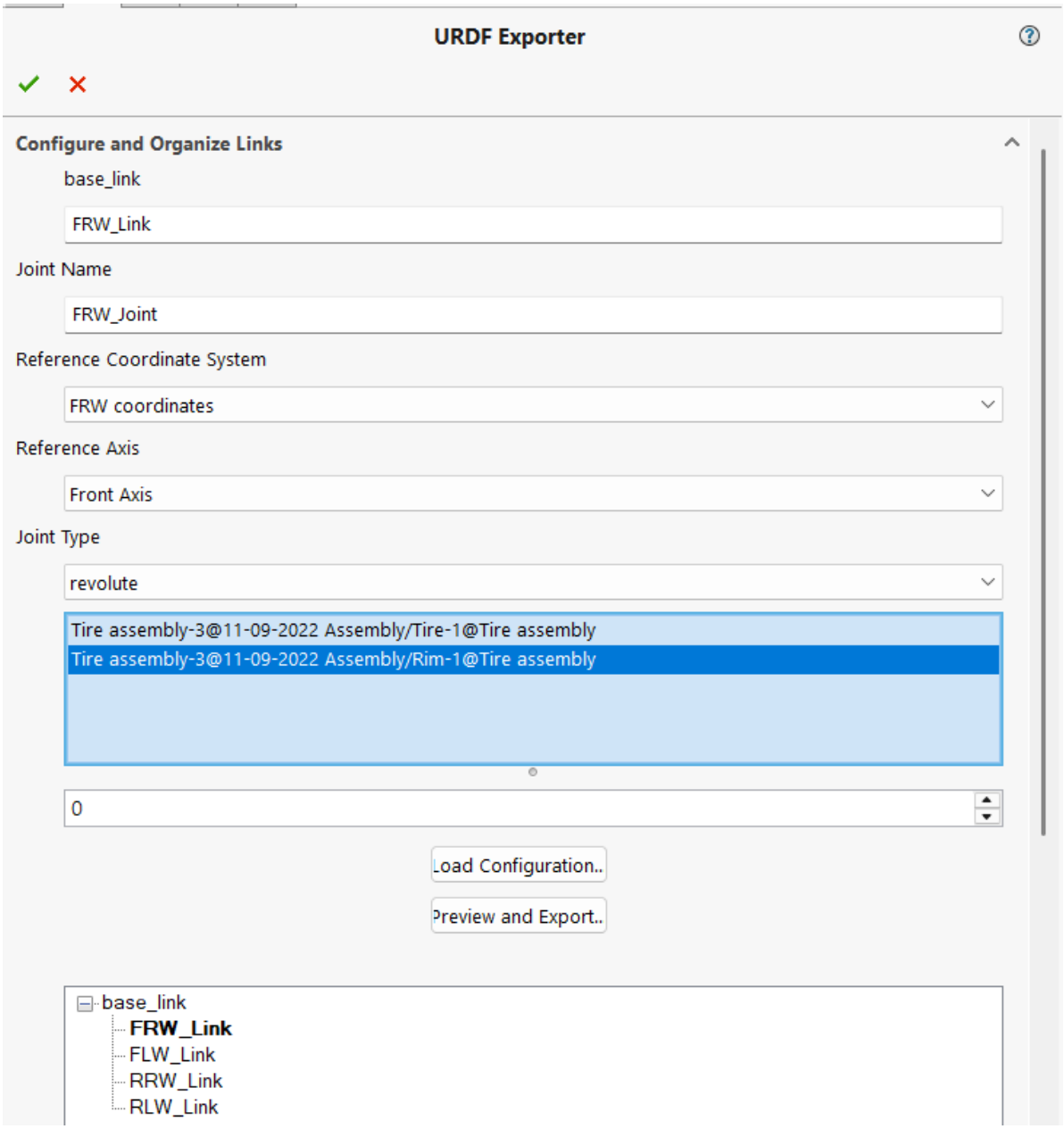



RRW point
YS
1
FRW point
base point
RLW point
FKLAVYA
FLW point

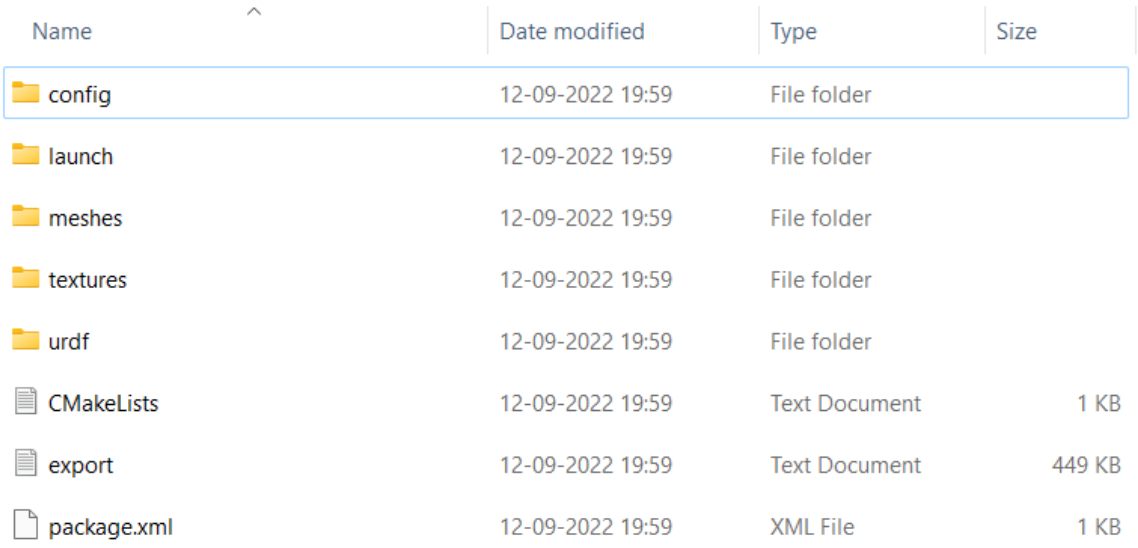

Z
RRW coordinates
YS
1
Z
Y
X
Z
FRW coordinates
Y
X
base coordinates
Z
Y
X
RLW coordinates
Z
Y
X
FKLAVYA
Z
FLW coordinates
Y
X

After the coordinate systems, points and axes have been assigned. Go to Tools -> Export as URDF. Select the base link as all the parts except the wheels, assign the base coordinate system to it, similarly add 4 child links for the wheels. For the child links add the reference coordinate system, reference axis and the joint type (continuous for our case). Repeat this process for all 4 wheels

For example:



After you're done with adding all the links click on the **Preview and Export..**, after checking all the joints click on **Next** and then **Export URDF and Meshes..** This will create folder on the desired location.



Name	Date modified	Type	Size
config	12-09-2022 19:59	File folder	
launch	12-09-2022 19:59	File folder	
meshes	12-09-2022 19:59	File folder	
textures	12-09-2022 19:59	File folder	
urdf	12-09-2022 19:59	File folder	
CMakeLists	12-09-2022 19:59	Text Document	1 KB
export	12-09-2022 19:59	Text Document	449 KB
package.xml	12-09-2022 19:59	XML File	1 KB

2.2 Simulating URDF in Simulator

SW has tools available to export an assembly to a URDF model.The URDF (Universal Robot Description Format) model is a collection of files that describe a robot's physical description to ROS. These files are used by a program called ROS to tell the computer what the robot actually looks like in real life. The URDF file is a .xml file which describes the parameters of the bot to ROS. Gazebo can then be used to open this URDF file into a simulated world using a launch file.

- ROS Terminologies:

  The best resource for learning ROS is through the docs . Let's look at some basic concepts and how they've been implemented in our project.

  1. Catkin Workspace:

     A catkin workspace is a folder where you modify, build, and install catkin packages.

  2. ROS Package:

     Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module.

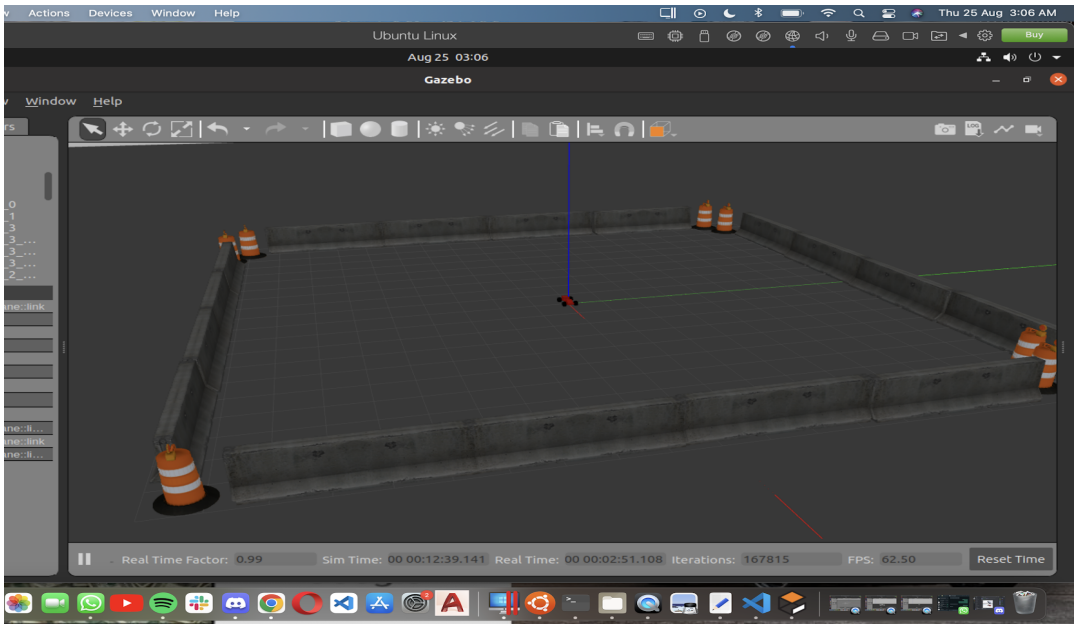
  3. ROS Node, Topics and Messages:

     10 Nodes, topics and messages are the basic components of the communication system in ROS. Nodes are points which can either publish data or subscribe to a topic to read its data. Topics are the pathways through which the data is transferred. A single topic may subscribe or publish to multiple nodes. Messages are the actual data which is sent. There are many in-built message types(like string,int,float,etc.) but custom types(like prop_spped used in our pkg)can also be used.

Full tutorial available here.

2.3 Creating World file

The term world in gazebo is used to describe a collection of robots, objects and global parameters including the sky, ambient light, and physics properties. A typical world will have all the necessary global properties of the simulation environment including the lighting, physics, wind, gravity and the robots.

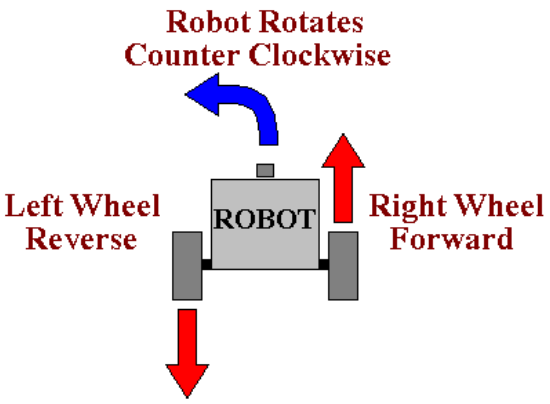



```
    <!-- World File -->
    <arg name="world_file" default="$(find racecar_eklavya)/worlds/world01.world"/>

    <!-- Launch Gazebo World -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
      <arg name="use_sim_time" value="true"/>
      <arg name="debug" value="false"/>
      <arg name="gui" value="true" />
      <arg name="world_name" value="$(arg world_file)"/>
    </include>
```

2.4 Application of Differential Drive

The differential drive is a two-wheeled drive system with independent actuators for each wheel. The name refers to the fact that the motion vector of the robot is the sum of the independent wheel motions. The drive wheels are usually placed on each side of the robot and toward the front.




```
  <gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <legacyMode>false</legacyMode>
    <updateRate>40</updateRate>
    <leftJoint>RLW_Joint</leftJoint>
    <rightJoint>RRW_Joint</rightJoint>
    <wheelSeparation>0.3</wheelSeparation>
    <wheelDiameter>0.18</wheelDiameter>
    <torque>10000</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>chassis</robotBaseFrame>
  </plugin>
</gazebo>
```

2.5 Adding Lidar, Camera and IMU Sensors

What is lidar?

Lidar is an acronym for “light detection and ranging.” It is sometimes called “laser scanning” or “3D scanning.” The technology uses eye-safe laser beams to create a 3D representation of the surveyed environment. Lidar is used in many industries, including automotive, infrastructure, robotics, trucking, UAV/drones, industrial, mapping, and many more. Because lidar is its own light source, the technology offers strong performance in a wide variety of lighting and weather conditions.

How does Lidar Work?

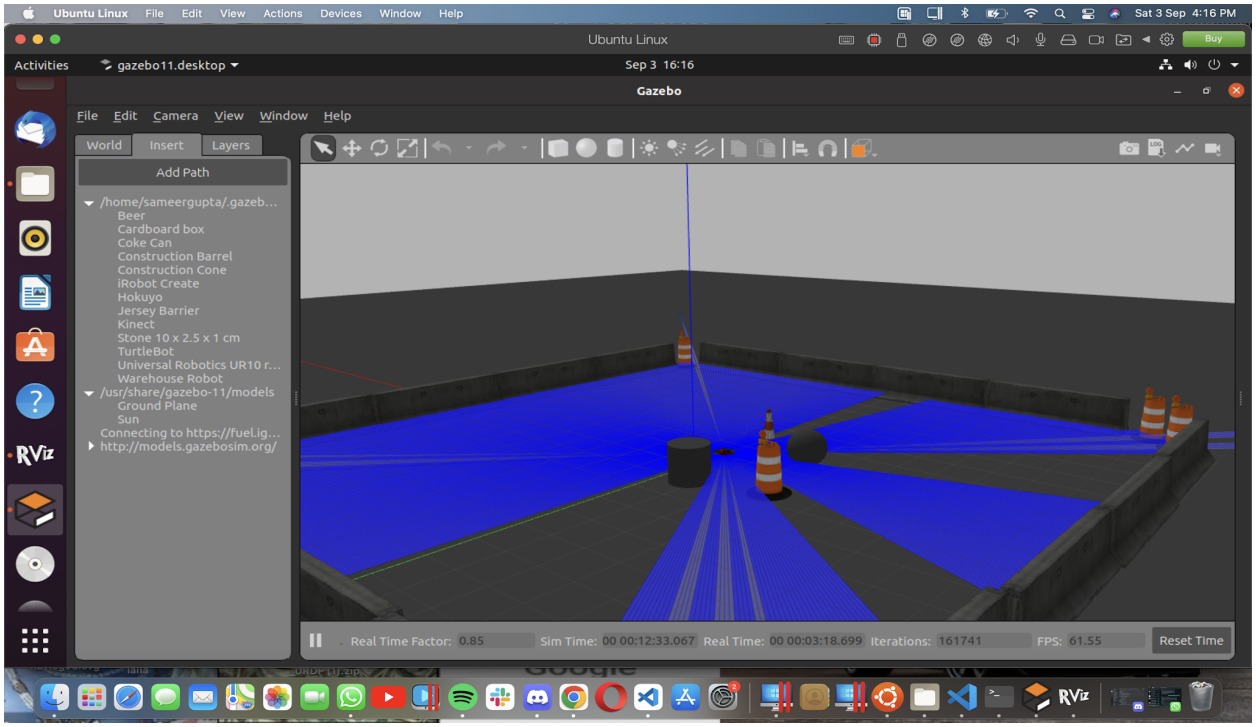
A typical lidar sensor emits pulsed light waves from a laser into the environment. These pulses bounce off surrounding objects and return to the sensor. The sensor uses the time it took for each pulse to return to the sensor to calculate the distance it traveled. Repeating this process millions of times per second creates a real-time 3D map of the environment. An onboard computer can utilize this 3D map of the surrounding environment for navigation

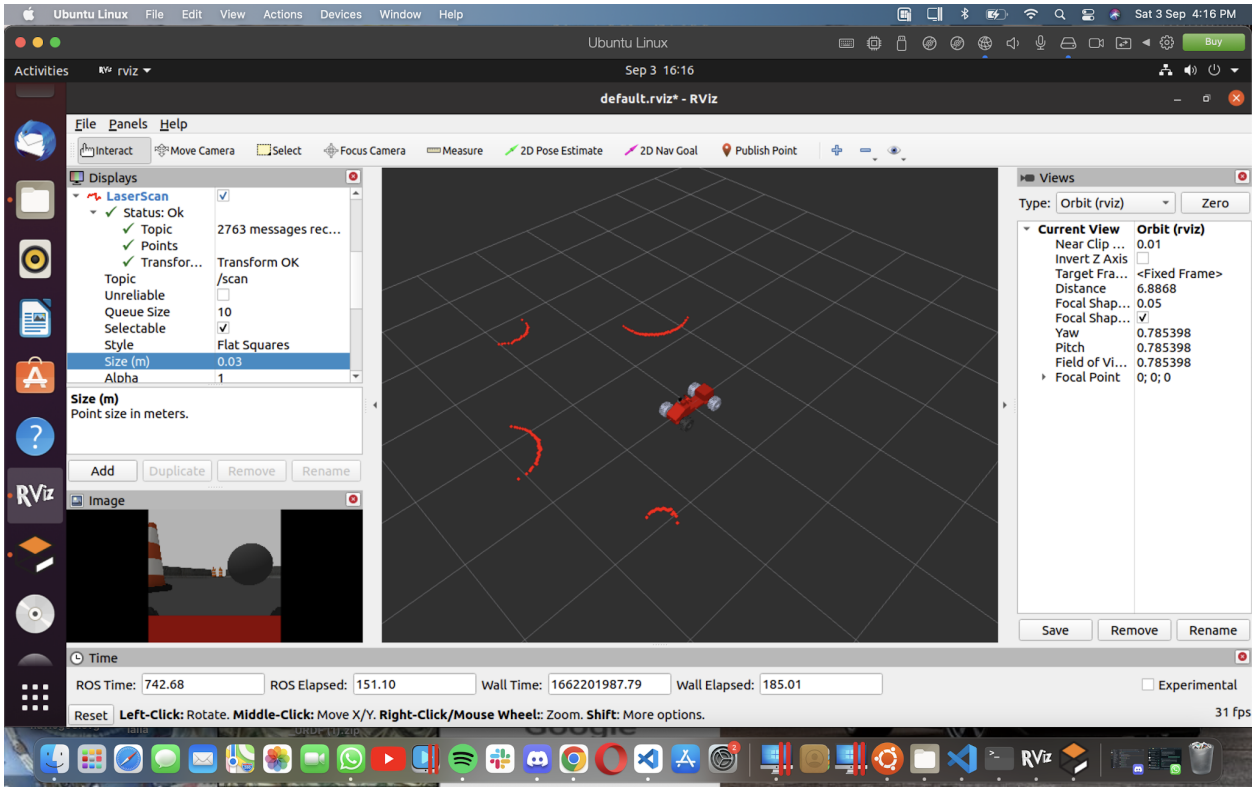
Why is lidar needed for Autonomous Vehicles?

Lidar sensors are a key component in autonomous vehicles, providing a high-resolution 3D view of their surroundings. Lidar enables autonomous vehicles to “see” by generating and measuring millions of data points in real time, creating a precise map of its ever-changing surroundings for safe navigation. Lidar’s distance accuracy allows the vehicle’s system to identify and avoid objects at up to 300 meters in a wide variety of weather and lighting conditions. Velodyne Lidar Inc. is the market-leader in 3D lidar technology with over 450 customers globally. As a result, Velodyne’s sensors have been tested, validated, and utilized by companies worldwide

- Lidar Plugin

```
<!-- hokuyo -->
<gazebo reference="hokuyo_1">
  <sensor type="ray" name="head_hokuyo_sensor_1">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>2</resolution>
          <min_angle>-1.57079632679</min_angle>
          <max_angle>1.57079632679</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.3</min>
        <max>10.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser
             achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
             reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/scan_1</topicName>
      <frameName>hokuyo_1</frameName>
    </plugin>
  </sensor>
</gazebo>
```


Ubuntu Linux
Activities
gazebo11.desktop
Sep 3 16:16
Gazebo
File Edit Camera View Window Help
World Insert Layers
Add Path
/home/sameergupta/.gazeb...
Beer
Cardboard box
Coke Can
Construction Barrel
Construction Cone
iRobot Create
Hokuyo
Jersey Barrier
Kinect
Stone 10 x 2.5 x 1 cm
TurtleBot
Universal Robotics UR10 r...
Warehouse Robot
/usr/share/gazebo-11/models
Ground Plane
Sun
Connecting to https://fuel.ig...
http://models.gazebosim.org/
Real Time Factor: 0.85
Sim Time: 00 00:12:33.067
Real Time: 00 00:03:18.699
Iterations: 161741
FPS: 61.55
Reset Time


Ubuntu Linux
Activities
rviz
Sep 3 16:16
default.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
LaserScan
Status: Ok
Topic
2763 messages rec...
Points
Transfor...
Transform OK
Topic
/scan
Unreliable
Queue Size
10
Selectable
Style
Flat Squares
Size (m)
0.03
Alpha
1
Size (m)
Point size in meters.
Add
Duplicate
Remove
Rename
Image
Views
Type: Orbit (rviz)
Zero
Current View
Orbit (rviz)
Near Clip ...
0.01
Invert Z Axis
Target Fra...
<Fixed Frame>
Distance
6.8868
Focal Shap...
0.05
Focal Shap...
Yaw
0.785398
Pitch
0.785398
Field of Vi...
0.785398
Focal Point
0; 0; 0
Save
Remove
Rename
Time
ROS Time: 742.68
ROS Elapsed: 151.10
Wall Time: 1662201987.79
Wall Elapsed: 185.01
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
31 fps

Camera Plugin :

provides ROS interface for simulating cameras such as `wge100_camera` by publishing the CameraInfo and Image ROS messages as described in `sensor_msgs`.

```
<!-- camera -->
<gazebo reference="camera">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>camera</cameraName>
      <imageTopicName>rgb/image_raw</imageTopicName>
      <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
      <frameName>camera</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

IMU Sensor :

simulates an Inertial Motion Unit sensor, the main differences from **IMU** (GazeboRosIMU) are: - inheritance from SensorPlugin instead of ModelPlugin, - measurements are given by gazebo ImuSensor instead of being computed by the ros plugin, - gravity is included in inertial measurements. - set `initialOrientationAsReference` to `false` to comply with REP 145.

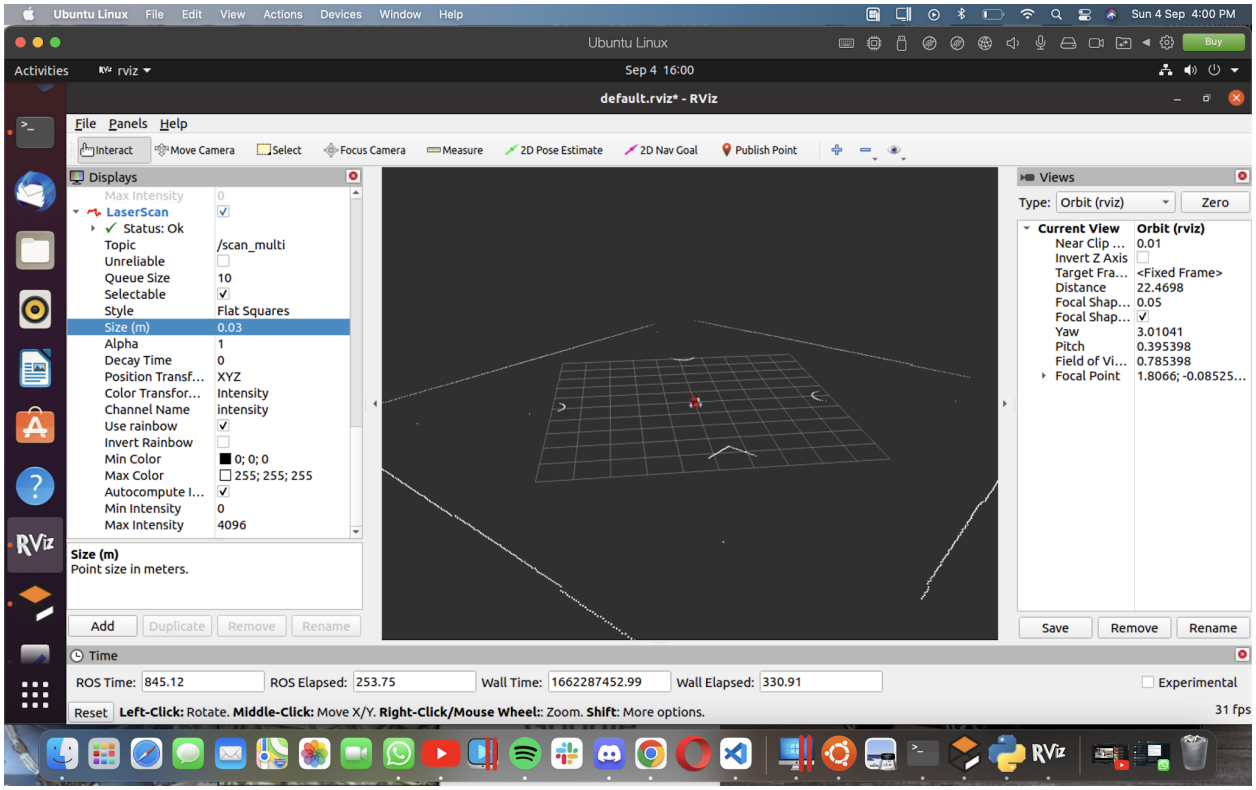
```
<!-- imu -->
<gazebo reference="imu_link">
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize>
    <topic>__default_topic__</topic>
    <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
      <topicName>imu</topicName>
      <bodyName>imu_link</bodyName>
      <updateRateHZ>10.0</updateRateHZ>
      <gaussianNoise>0.0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <frameName>imu_link</frameName>
      <initialOrientationAsReference>false</initialOrientationAsReference>
    </plugin>
    <pose>0 0 0 0 0 0</pose>
  </sensor>
</gazebo>
```

2.6 Merging Lidar Sensors Data

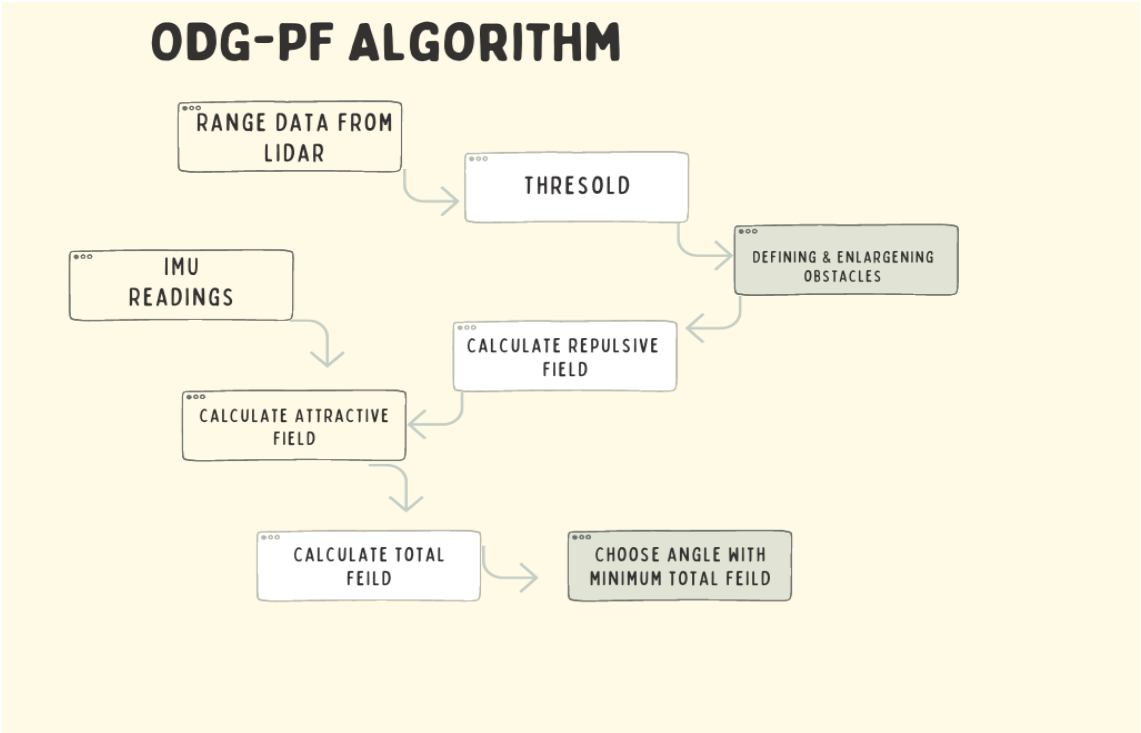
As of now we have two Lidar sensors mounted and we're using the ira_laser_tool plugin.
Ira Laser Tool :

> Laserscan_merger allows to easily and dynamically (rqt_reconfigure) merge multiple, same time, single scanning plane, laser scans into a single one; this is very useful for using applications like gmapping, amcl, pamcl on vehicles with multiple single scanning plane laser scanners, as these applications require just one laser scan as input. The scanning planes need to be approximately the same. The resulting scan will appear generated from a single scanner dis-regarding actual occlusions as seen from the merged scans; for instance, consider the case of 2 scanners mounted on the 2 front corners A and B of a rectangular vehicle (we live in 2D); each scanner gives out a 270degs scan, from along the long side of the vehicle going backward to toward the other scanner. The merged scan will appear as generated from a virtual scanner positioned in C, halfway between A and B, and measuring the same measures of the merges scans, regardless of the occlusions that would apply to a real scanner positioned in C.

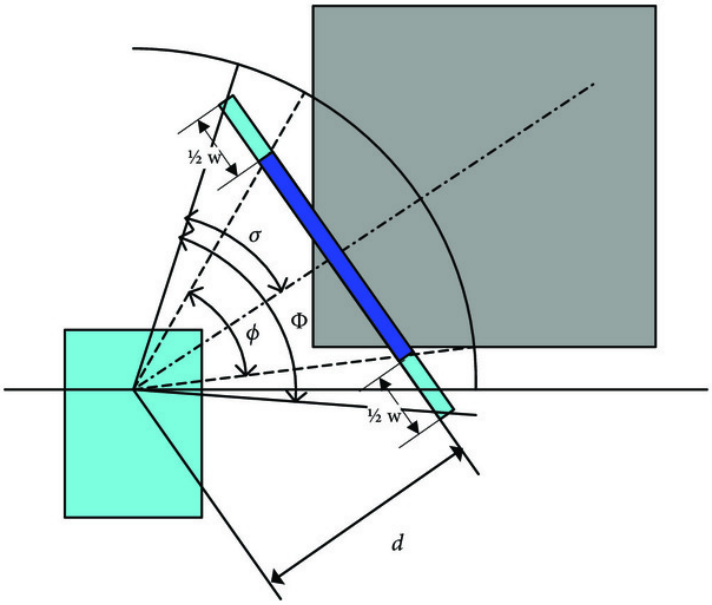
```
<launch>
        <node pkg="ira_laser_tools" name="laserscan_multi_merger" type="laserscan_multi_merger" output="screen">
        <param name="destination_frame" value="chassis"/>
        <param name="cloud_destination_topic" value="/merged_cloud"/>
        <param name="scan_destination_topic" value="/scan_multi"/>
        <param name="laserscan_topics" value ="/scan_1 /scan_2" /> <!-- LIST OF THE LASER SCAN TOPICS TO SUBSCRIBE -->
        <param name="angle_min" value="-1.57079632679"/>
        <param name="angle_max" value="1.57079632679"/>
        <param name="angle_increment" value="0.0058"/>
        <param name="scan_time" value="0.0333333"/>
        <param name="range_min" value="0.30"/>
        <param name="range_max" value="10.0"/>
        </node>
</launch>
```



2.7 ODG-PF Algorithm



- To understand the code better we need the understanding of Attractive, Repulsive and Total Potential field formulas and variables required in formula for calculation.
- Following are the formulas and variables required:
    1. Repulsive Field:



If we consider the vehicle's width, we need to recalculate angle as

$$\Phi_k = 2\sigma_k = 2 \cdot \text{atan2}\left(d_k \tan\frac{\phi_k}{2} + \frac{w_{robot}}{2}, d_k \right)$$

Φ_k is obtained from each obstacle's average distance and the occupied angle of obstacle.

Gaussian likelihood functions (repulsive fields) of the obstacles are calculated as

$$f_k(\theta_i) = A_k \exp\left(-\frac{(\theta_k - \theta_i)^2}{2\sigma_k^2}\right)$$

Where,

$$A_k = \tilde{d}_k \exp\left(\frac{1}{2}\right)$$

where $\tilde{d}_k = d_{max} - d_k$. d_{max} is the maximum detection range of the range sensor and σ_k is half of the angle occupied by the k_{th} obstacle.

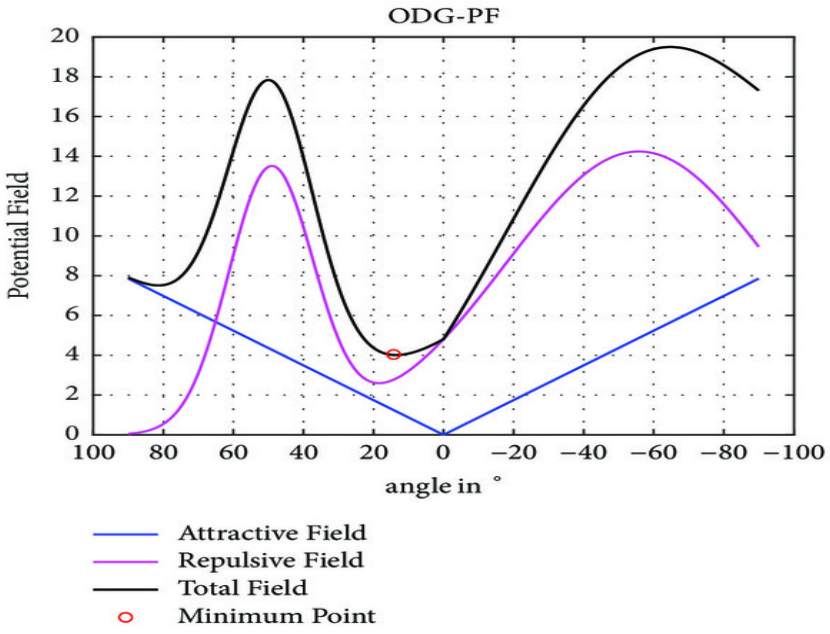
This is also a function of angle θ_i. The attractive field is calculated as

$$f_{att}(\theta_i) = \gamma \left|\theta_{goal} - \theta_i\right|. \quad (11)$$

The total field is calculated by adding these two fields. Since these two fields are functions of θ_i, the total field is also a function of θ_i:

$$f_{total}(\theta_i) = f_{rep}(\theta_i) + f_{att}(\theta_i). \quad (12)$$

The value γ was chosen as 5.0 by executing several simulations.



All the variables required to calculate potential fields:

- vehicle ‘s width : 145mm + 160mm + 145mm = 450mm = 0.45m
- d_k (average distance of obstacle)
- Φ_k (angle occupied by the obstacle)
- As we have Φ_k and d_k we have σ_k
- d_{max} is the maximum length of the lidar data.
- Θ_k is all the angles in the lidar data.

0. Importing libraries

```
from math import atan2 ,tan ,exp
from shutil import move
from statistics import mean
import sys
from unicodedata import name
import numpy as np
import rospy
from sensor_msgs.msg import LaserScan , Imu
from geometry_msgs.msg import Twist
from tf.transformations import euler_from_quaternion
```

1. We call the listener() function whenever node is executed,

```
def listener():
    rospy.init_node("listener",anonymous=True)
    rospy.Subscriber("/imu", Imu , imuCallback)
    rospy.Subscriber("/scan_multi", LaserScan , laserCallback)
    rospy.spin()
```

   a. Imu subscriber calls imuCallback function, which will be returning yaw angle of the racecar to know it’s orientation.
   b. LaserScan calls laserCallback function, it’s our spot line function to avoid obstacles.

2. imuCallback

```
def imuCallback(msg):
        orinetation_list = [msg.orientation.x,msg.orientation.y,msg.orientation.z,msg.orientation.w]
        (roll,pitch,yaw) = euler_from_quaternion(orinetation_list)
        roll = roll * (180/3.14159265)
        pitch = pitch * (180/3.14159265)
        yaw = yaw * (180/3.14159265)
```

3. laserCallback

   a. Initializing the numpy arrays to find total field for all lidar angles(542).

```
def laserCallback(msg ):
    # final feilds
    total_net_feild = np.zeros(542)
    total_repulsive_feild = np.zeros(542)
    total_attractive_feild = np.zeros(542)
    #laser reading
    laser_readings = msg.ranges
```

   b. Detecting nearby(2m) obstacles.

```
start = 0
end = -1
obstacle_endpoints = set()
for i in range(start,len(laser_readings)-1):
    if( laser_readings[i]  > 2 and laser_readings[i+1] <=2):
        start = i+1
    elif(laser_readings[i]  <= 2 and laser_readings[i+1] > 2 ):
        end = i
    if(start <= end and end != -1):
        obstacle_endpoints.add((start,end))
```

c. Calculating repulsive field:

1. Calling repulsive field function :

```
for elements in obstacle_endpoints :
    total_repulsive_feild= np.add(total_repulsive_feild,calculate_repulsive_feild(laser_readings, elements[0],elements[1]))
```

2. calculate_repulsive_feild() function :

```
def calculate_repulsive_feild(laser_readings, start,end):
    # variables for repulsive feild:-
    #calculating average distance of obstacle -> d_k:-
    d_k = mean(laser_readings[start:end+1])
    #calculating  angle occupied by obstacle -> fi_k:-
    # as for 542 readings of lidar ==> pi radians of area is covered for 1 reading ==> 0.0058 radians :-
    fi_k = 0.0058 * (end - start)
    # as we now have d_k and fi_k we can calculate -> sigma_k :-
    global robot_width # obstacle enlargening
    sigma_k = atan2( (d_k * tan(fi_k/2)) + robot_width /2 , d_k )
    # calculating A_k :-
    global max_range
    d_k_dash = max_range - d_k
    A_k = d_k_dash * exp(- 1 / 2)
    #calculating center angle -> theta_k :-
    theta_k = (((start+end) / 2 ) * 0.0058) - 1.57079632679
    # calculating repulsive feild for all the laser reading of the given obstacle
    repulsive_feild = []
    for theta_i in range(542) :
        repulsive_feild_function = A_k * ( exp (- (((theta_k - ((theta_i*0.0058)-1.57079632679))**2)/(2*(sigma_k**2)))))
        repulsive_feild.append(repulsive_feild_function)
    return repulsive_feild
```

$$\Phi_k = 2\sigma_k = 2 \cdot \operatorname{atan2}\left(d_k \tan\frac{\phi_k}{2} + \frac{w_{robot}}{2}, d_k \right)$$

Gaussian likelihood functions (repulsive fields) of the obstacles are calculated as

$$f_k(\theta_i) = A_k \exp\left(-\frac{(\theta_k - \theta_i)^2}{2\sigma_k^2} \right)$$

d. Calculate the attractive field:

i. Calling attractive field function:

```
global yaw
global theta_goal
total_attractive_feild = calculate_attractive_feild(theta_goal)
```

ii. calculate_attractive_feild() function:

```
def calculate_attractive_feild(theta_goal):
    global gamma
    attractive_feild = []
    for theta_i in range(542) :
        attractive_feild_function = (gamma)*abs(theta_goal - ((theta_i*0.0058)-1.57079632679))
        attractive_feild.append(attractive_feild_function)
    return attractive_feild
```

This is also a function of angle θ_i. The attractive field is calculated as

$$f_{att}(\theta_i) = \gamma \left| \theta_{goal} - \theta_i \right|. \tag{11}$$

The total field is calculated by adding these two fields. Since these two fields are functions of θ_i, the total field is also a function of θ_i:

$$f_{total}(\theta_i) = f_{rep}(\theta_i) + f_{att}(\theta_i). \tag{12}$$

e. Calculate total field:

```
# finally total net feild
total_net_feild = np.add(total_attractive_feild,total_repulsive_feild)
```

f. Calculate angle of minimum total field:

```
# minima angle for which net feild is minimum

min_value_index  = np.where(total_net_feild == np.amin(total_net_feild))
angle_for_min_radians = (0.0058*(min_value_index[0][0]) - 1.57079632679)
angle_for_min_degrees = ((angle_for_min_radians*(180)) / 3.14159265359)
```

g. Achieving minimum total field angle using yaw and twist message:

```
# publishing spin to achieve the angle_for_min

pub = rospy.Publisher("/cmd_vel" , Twist)
move = Twist()
move.angular.z = 0

if(abs(angle_for_min_degrees - yaw ) > 0.5):
    if(angle_for_min_degrees > yaw ):
        move.angular.z = 6
    elif( yaw  > angle_for_min_degrees ):
        move.angular.z = -6
else:
    move.angular.z = 0
    move.linear.x = 0.5


pub.publish(move)
```

2.8 Line Following and OpenCV:

- Steps:
    - Subscribe the camera sensor topic to alter the image captured by the sensor.

```
def __init__(self):
  self.bridge = cv_bridge.CvBridge()
  #cv2.namedWindow("window", 1)
  self.image_sub = rospy.Subscriber('/camera/rgb/image_raw',
                                    Image, self.image_callback)
```

    - The image obtained is rgb, which is needed to be converted into hsv image to more precise definition of low and high shades of a color.We try to calculate the mask of the hsv image to detect a certain color.

```
def image_callback(self, msg):
  image = self.bridge.imgmsg_to_cv2(msg,desired_encoding='bgr8')
  hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
  lower_red = numpy.array([ 160,  50,  50])
  upper_red = numpy.array([180, 255, 255])
  mask = cv2.inRange(hsv, lower_red, upper_red)
```

    - After color detection we find the contour of the red line, which eventually can be used to find the centroid of the contour.

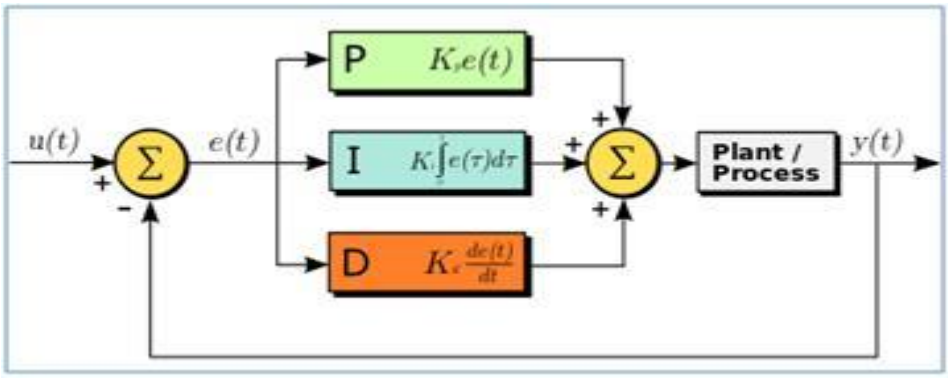
```
h, w, d = image.shape
search_top = 3*h//4
search_bot = 3*h//4 + 20
mask[0:search_top, 0:w] = 0
mask[search_bot:h, 0:w] = 0
M = cv2.moments(mask)
```

- Now, if the racecar is misaligned the centroid of the contour would be displaced from the actual center of the captured image.The errors found in the misalignment are in **pixels,** and the error would be resolved using PID controller.

```
if M['m00'] > 0:
  cx = int(M['m10']/M['m00'])
  cy = int(M['m01']/M['m00'])
  cv2.circle(image, (cx, cy), 20, (0,0,255), -1)
  # BEGIN CONTROL
  global prev_err, err_diff,error_area,k_p, k_i , k_d
  err = cx - w/2
  error_area += err
  err_diff = err - prev_err
  prev_err = err
  correction = k_p*err + k_d*err_diff + k_i*error_area
  print(f"Error : {err}    Error_Area : {error_area} Error_Diff : {err_diff}")
  print(correction)
```

2.9 Control Systems and PID

- **What is a PID Controller?**



The term PID stands for proportional integral derivative and it is one kind of device used to control different process variables like pressure, flow, temperature, and speed in industrial applications. In this controller, a control loop feedback device is used to regulate all the process variables.

This type of control is used to drive a system in the direction of an objective location otherwise level. It is almost everywhere for temperature control and used in scientific processes, automation & myriad chemicals. In this controller, closed-loop feedback is used to maintain the real output from a method close to the objective, otherwise output at the fixed point if possible. In this article, the PID controller design with control modes used in them like P, I & D are discussed.

- Control System:

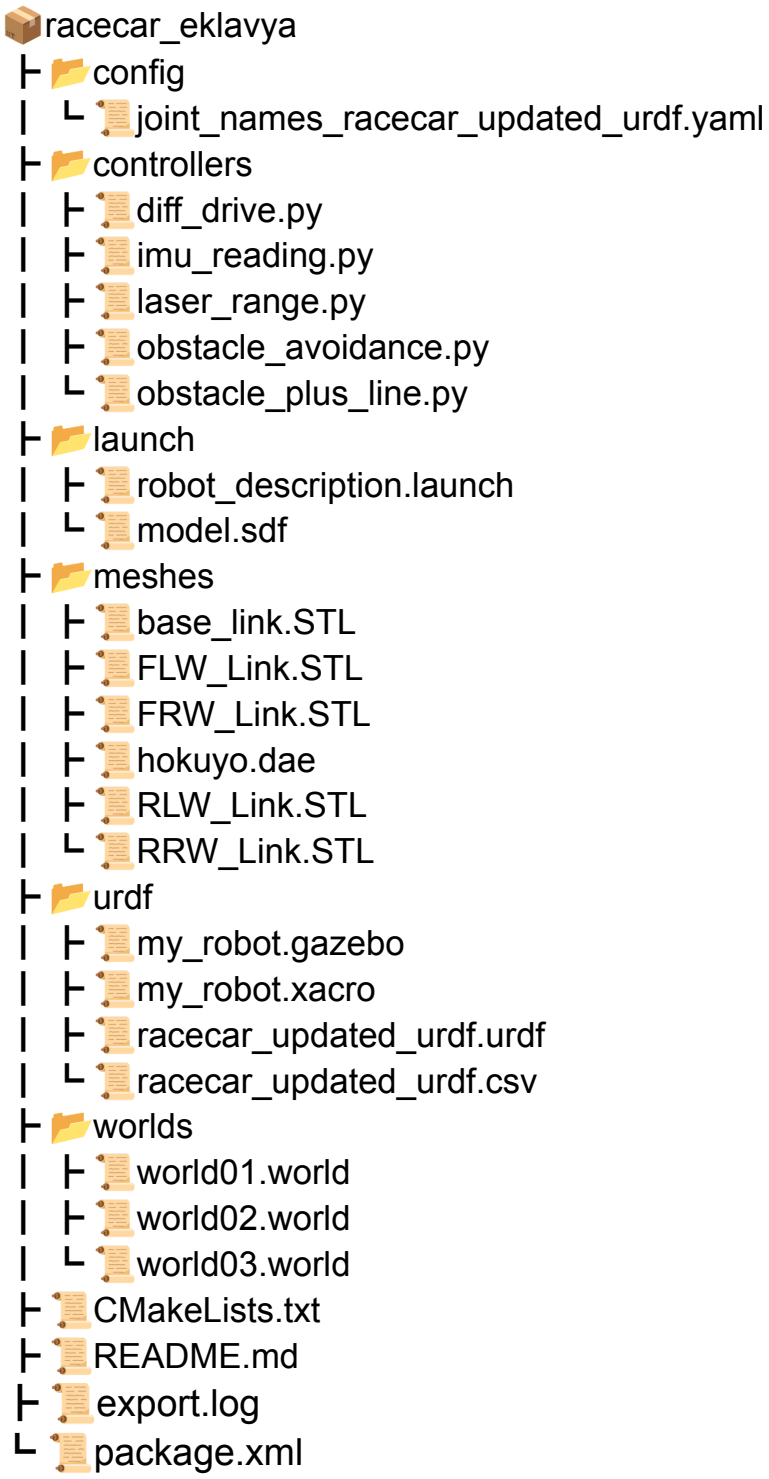
    - **State**- output produce by a robotic system is known as a state. Normally we denote it by **x**, the state depends on its previous states, stimulus (signals) applied to the actuators, and the physics of the environment. The state can be anything pose, speed, velocity, angular velocity, force and etc.
    - **Estimate**- Robots cannot determine the exact state **x**, but they can **estimate** it using the sensors attached to them. These estimations are denoted with **y**. It is the responsibility of the robotic engineer to select good enough sensors or to calibrate the sensors well, such that they can produce **y ~ x.**
    - **Reference**- the goal state we wish to achieve, it is denoted using **r.**
    - **Error**- the difference between the **reference and estimate** is known as error.
    - **Control Signal**- the stimulus produces/output by the controller is known as the control signal, it is denoted using **u.**
    - **Dynamics**- it is also called as the system plant/system model, it denotes how the system will behave under non-static conditions. Dynamics are affected by the environment that may change or not always linear. For example, floor type (concrete/wood), the air drag, slope and etc.

- Line by Line code explanation:

```
# BEGIN CONTROL
global prev_err, err_diff,error_area,k_p, k_i , k_d
err = cx - w/2
error_area += err
err_diff = err - prev_err
prev_err = err
correction = k_p*err + k_d*err_diff + k_i*error_area
print(f"Error : {err}    Error_Area : {error_area} Error_Diff : {err_diff}")
print(correction)
```

3. Implementations

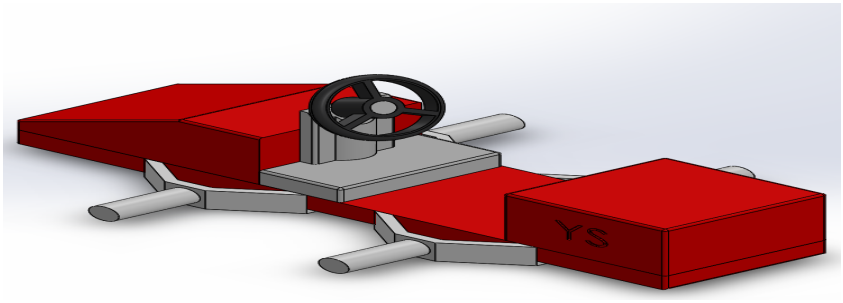
3.1 racecar_eklavya package overview



3.2 .xacro, .gazebo and .launch files

3.2.1 my_robot.xacro:

a. base_link/chassis connected to gazebo ground plane.




```
<link name="robot_footprint"></link>

<joint name="robot_footprint_joint" type="fixed">
  <origin xyz="0 0 0" rpy="0 0 0" />
  <parent link="robot_footprint"/>
  <child link="chassis" />
</joint>

<link
  name="chassis">
  <inertial>
    <origin
      xyz="-0.0958133923403369 -2.27563018667665E-06 0.0145539997290763"
      rpy="0 0 0" />
    <mass
      value="6.78003193648517" />
    <inertia
      ixx="0.0213363092460489"
      ixy="5.40950247339377E-08"
      ixz="-0.000249956847087839"
      iyy="0.113723385248228"
      iyz="-5.58676125268779E-08"
      izz="0.128537247306062" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://racecar_eklavya/meshes/base_link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
    </material>
  </visual>
  <collision>

    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://racecar_eklavya/meshes/base_link.STL" />
    </geometry>
  </collision>
</link>
```

b. Wheel link connected to base_link/chassis:

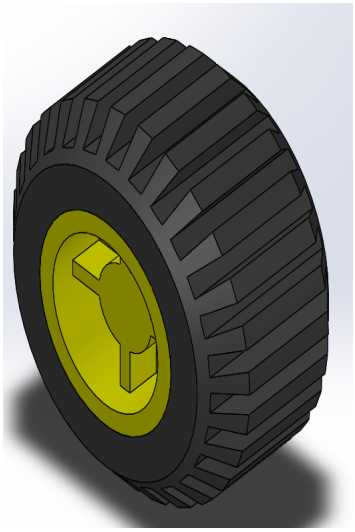
```
<joint type="continuous" name="left_wheel_hinge">
  <origin
    xyz="0.12 0.2 -0.02"
    rpy="0 0 0" />
  <parent
    link="chassis" />
  <child
    link="left_wheel" />
  <axis
    xyz="0 1 0" rpy="0 0 0"/>

</joint>

<link name="left_wheel">

  <inertial>
    <origin
      xyz="-1.89376292425436E-12 -0.0310103769760117 4.59632332194815E-13"
      rpy="0 0 0" />
    <mass
      value="1.70383251159279" />
    <inertia
      ixx="0.00449158610281422"
      ixy="-1.34110727409117E-13"
      ixz="-3.31900178997306E-13"
      iyy="0.00645305207636973"
      iyz="3.57698040849964E-14"
      izz="0.00449158610263334" />
  </inertial>
  <visual>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://racecar_eklavya/meshes/FLW_Link.STL" />
    </geometry>
    <material
      name="">
      <color
        rgba="0.792156862745098 0.819607843137255 0.933333333333333 1" />
    </material>
  </visual>
  <collision>
    <surface>
      <friction>
        <ode>
          <mu>1000.0</mu>
          <mu2>1000.0</mu2>
        </ode>
      </friction>
    </surface>
    <origin
      xyz="0 0 0"
      rpy="0 0 0" />
    <geometry>
      <mesh
        filename="package://racecar_eklavya/meshes/FLW_Link.STL" />
    </geometry>
  </collision>

</link>
```



c. Camera link connected to base_link/chassis:

```
<joint type="fixed" name="camera_joint">
  <origin xyz="0.19 0 0.1" rpy="0 0 0"/>
  <child link="camera"/>
  <parent link="chassis"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
</joint>

<link name="camera">

  <inertial>
    <mass value="0.1"/>
    <origin xyz="0.0 0 0" rpy="0 0 0"/>
    <inertia
        ixx="1e-6" ixy="0" ixz="0"
        iyy="1e-6" iyz="0"
        izz="1e-6"
    />
    <box_inertia m="0.1" x="0.05" y="0.05" z="0.05"/>
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.05 0.05"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.05 0.05"/>
    </geometry>
  </collision>

</link>
```

d. Hokuyo/Lidar link connected to base_link/chassis:

```
<joint type="fixed" name="hokuyo_joint_1">
  <origin xyz="0.12 -0.07 0.12" rpy="0 0 0"/>
  <child link="hokuyo_1"/>
  <parent link="chassis"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
</joint>

<link name="hokuyo_1">

  <inertial>
    <mass value="1e-5"/>
    <origin xyz="0.0 0 0" rpy="0 0 0"/>
    <inertia
        ixx="1e-6" ixy="0" ixz="0"
        iyy="1e-6" iyz="0"
        izz="1e-6"
    />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <mesh filename="package://racecar_eklavya/meshes/hokuyo.dae"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

</link>
```

e. IMU Sensor link connected to base_link/chassis:

```
<!-- IMU SENSOR -->
<joint type="fixed" name="imu_joint">
  <origin xyz="0.12 0 0.12" rpy="0 0 0"/>
  <child link="imu_link"/>
  <parent link="chassis"/>
  <axis xyz="0 1 0" rpy="0 0 0"/>
</joint>
<link name="imu_link">

  <inertial>
    <mass value="1e-5"/>
    <origin xyz="0.0 0 0" rpy="0 0 0"/>
    <inertia
        ixx="1e-6" ixy="0" ixz="0"
        iyy="1e-6" iyz="0"
        izz="1e-6"
    />
  </inertial>

  <visual>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.05 0.05"/>
    </geometry>
  </visual>

  <collision>
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <geometry>
      <box size="0.1 0.1 0.1"/>
    </geometry>
  </collision>

</link>
```

3.2.2 my_robot.gazebo

a. Differential Drive plugin:

```
  <gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <legacyMode>false</legacyMode>
    <updateRate>40</updateRate>
    <leftJoint>RLW_Joint</leftJoint>
    <rightJoint>RRW_Joint</rightJoint>
    <wheelSeparation>0.3</wheelSeparation>
    <wheelDiameter>0.18</wheelDiameter>
    <torque>10000</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>chassis</robotBaseFrame>
  </plugin>
</gazebo>

<gazebo>
  <plugin name="differential_drive_controller" filename="libgazebo_ros_diff_drive.so">
    <alwaysOn>true</alwaysOn>
    <legacyMode>false</legacyMode>
    <updateRate>40</updateRate>
    <leftJoint>left_wheel_hinge</leftJoint>
    <rightJoint>right_wheel_hinge</rightJoint>
    <wheelSeparation>0.3</wheelSeparation>
    <wheelDiameter>0.18</wheelDiameter>
    <torque>10000</torque>
    <commandTopic>cmd_vel</commandTopic>
    <odometryTopic>odom</odometryTopic>
    <odometryFrame>odom</odometryFrame>
    <robotBaseFrame>chassis</robotBaseFrame>
  </plugin>
</gazebo>
```

b. Camera Sensor plugin:

```
<!-- camera -->
<gazebo reference="camera">
  <sensor type="camera" name="camera1">
    <update_rate>30.0</update_rate>
    <camera name="head">
      <horizontal_fov>1.3962634</horizontal_fov>
      <image>
        <width>800</width>
        <height>800</height>
        <format>R8G8B8</format>
      </image>
      <clip>
        <near>0.02</near>
        <far>300</far>
      </clip>
    </camera>
    <plugin name="camera_controller" filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>0.0</updateRate>
      <cameraName>camera</cameraName>
      <imageTopicName>rgb/image_raw</imageTopicName>
      <cameraInfoTopicName>rgb/camera_info</cameraInfoTopicName>
      <frameName>camera</frameName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

c. Hokuyo sensor plugin:

```
<!-- hokuyo -->
<gazebo reference="hokuyo_1">
  <sensor type="ray" name="head_hokuyo_sensor_1">
    <pose>0 0 0 0 0 0</pose>
    <visualize>true</visualize>
    <update_rate>40</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>720</samples>
          <resolution>2</resolution>
          <min_angle>-1.57079632679</min_angle>
          <max_angle>1.57079632679</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.3</min>
        <max>10.0</max>
        <resolution>0.01</resolution>
      </range>
      <noise>
        <type>gaussian</type>
        <!-- Noise parameters based on published spec for Hokuyo laser
             achieving "+-30mm" accuracy at range < 10m.  A mean of 0.0m and
             stddev of 0.01m will put 99.7% of samples within 0.03m of the true
             reading. -->
        <mean>0.0</mean>
        <stddev>0.01</stddev>
      </noise>
    </ray>
    <plugin name="gazebo_ros_head_hokuyo_controller" filename="libgazebo_ros_laser.so">
      <topicName>/scan_1</topicName>
      <frameName>hokuyo_1</frameName>
    </plugin>
  </sensor>
</gazebo>
```

d. IMU Sensor Plugin:

```
<!-- imu -->
<gazebo reference="imu_link">
  <gravity>true</gravity>
  <sensor name="imu_sensor" type="imu">
    <always_on>true</always_on>
    <update_rate>100</update_rate>
    <visualize>true</visualize>
    <topic>__default_topic__</topic>
    <plugin filename="libgazebo_ros_imu_sensor.so" name="imu_plugin">
      <topicName>imu</topicName>
      <bodyName>imu_link</bodyName>
      <updateRateHZ>10.0</updateRateHZ>
      <gaussianNoise>0.0</gaussianNoise>
      <xyzOffset>0 0 0</xyzOffset>
      <rpyOffset>0 0 0</rpyOffset>
      <frameName>imu_link</frameName>
      <initialOrientationAsReference>false</initialOrientationAsReference>
    </plugin>
    <pose>0 0 0 0 0 0</pose>
  </sensor>
</gazebo>
```

3.2.3 robot_description.launch:

```
<?xml version="1.0"?>
<launch>

  <!-- send urdf to param server -->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find racecar_eklavya)/urdf/my_robot.xacro'" />

  <!-- Send fake joint values-->
  <node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
    <param name="use_gui" value="false"/>
  </node>

  <!-- Send robot states to tf -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen"/>

</launch>
```

3.2.4 world.launch:

```
<?xml version="1.0" encoding="UTF-8"?>

<launch>
  <!-- Robot pose -->
  <arg name="x" default="0"/>
  <arg name="y" default="0"/>
  <arg name="z" default="0"/>
  <arg name="roll" default="0"/>
  <arg name="pitch" default="0"/>
  <arg name="yaw" default="0"/>


  <!-- Launch other relevant files-->
  <include file="$(find racecar_eklavya)/launch/robot_description.launch"/>

  <!-- World File -->
  <arg name="world_file" default="$(find racecar_eklavya)/worlds/world01.world"/>

  <!-- Launch Gazebo World -->
  <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="use_sim_time" value="true"/>
    <arg name="debug" value="false"/>
    <arg name="gui" value="true" />
    <arg name="world_name" value="$(arg world_file)"/>
  </include>

  <!-- Find my robot Description-->
  <param name="robot_description" command="$(find xacro)/xacro --inorder '$(find racecar_eklavya)/urdf/my_robot.xacro'"/>

  <node
    name="tf_footprint_base"
    pkg="tf"
    type="static_transform_publisher"
    args="0 0 0 0 0 0 chassis robot_footprint 40" />


  <!-- Spawn My Robot -->
  <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false" output="screen"
        args="-urdf -param robot_description -model my_robot
              -x $(arg x) -y $(arg y) -z $(arg z)
              -R $(arg roll) -P $(arg pitch) -Y $(arg yaw)"/>



  <!--launch rviz-->
  <node name="rviz" pkg="rviz" type="rviz" respawn="false"/>



</launch>
```

● Conclusion

- So finally in this project we got to learn how to develop your own dynamic robot which solves a real world problem, of automating driving precisely.

- Writing various python scripts to control a particular operations, Like simple actuations or movements and also the complex ones Like avoiding obstacles and following lines.

- In this paper, we introduce a novel method for obstacle avoidance, ODG-PF, which avoids obstacles very efficiently and safely. Although AFPFM is an improvement that avoids the local minima problem, its attractive field coefficient should be adjusted when the environment is changed. As with conventional PFM, when the vehicle is in a narrow environment, the repulsive field becomes strong; thus value needs to be large. Otherwise, the robot will be stuck at a local minimum or go toward the goal very inefficiently.

- Simulations and experiments showed that vehicle movements with ODG-PF were very stable. It seems that this stable movement is because of two reasons. One reason is that it detects and defines obstacles first rather than directly calculating the repulsive field from the distance data, and the other is that it finds the angle with minimum values from the total field function. Thus, ODG-PF does not have the local minima problem that causes a vehicle to become stuck at certain points.

- One more merit of ODG-PF is that it avoids both static and moving obstacles without any adjustments. In the performed moving obstacle scenarios, the vehicle using it did not collide with any of the static or moving obstacles and instead avoided them using the same method. This is also quite a good point in that it does not need to carry out any time-consuming activities such as image processing or computer vision processing.

For Being this far with us….