Lab6: ROS2 Perception – Line Follower

* How to use OpenCV in ROS2
* How to follow a line
* How to find different elements based on color
* Track multiple paths and decide
* Create a basic PID for the line following

OpenCV is the most extensive and complete library for image recognition. With it, you can work with images like never before: applying filters, post-processing, and working with images in any way you want. OpenCV is not a ROS2 library, but it's been integrated nicely into ROS with http://wiki.ros.org/cv\_bridge. Thispackage allows the ROS imaging topics to use the OpenCV image variable format.

For example, OpenCV images come in BGR image format, while regular ROS images are in the more standard RGB encoding. OpenCV\_bridge provides a nice feature to convert between them. Also, there are many other functions to transfer images to OpenCV variables transparently.

If gazebo is stuck use the following command

killall -9 gzserver

sudo apt install libopencv-dev python3-opencv

cd ros2\_ws/src

ros2 pkg create lab6 --build-type ament\_python --dependencies rclpy std\_msgs

cd ..

colcon build

open lab6 with visual studio application

create urdf folder

create a file car.urdf inside the urdf folder

<?xml version="1.0" ?>

<robot name = "car">

<link name="base">

<visual>

<geometry>

<box size="0.75 0.4 0.1"/>

</geometry>

<material name="pink">

<color rgba="1 0 1 1" />

</material>

</visual>

<inertial>

<mass value="1" />

<inertia ixx="0.01" ixy="0.0" ixz="0" iyy="0.01" iyz="0" izz="0.01" />

</inertial>

<collision>

<geometry>

<box size="0.75 0.4 0.1"/>

</geometry>

</collision>

</link>

<link name="wheel\_right\_link">

<inertial>

<mass value="2" />

<inertia ixx="0.01" ixy="0.0" ixz="0"

iyy="0.01" iyz="0" izz="0.01" />

</inertial>

<visual>

<geometry>

<cylinder radius="0.15" length="0.1"/>

</geometry>

<material name="blue">

<color rgba="0 0 1 1"/>

</material>

</visual>

<collision>

<geometry>

<cylinder radius="0.15" length="0.1"/>

</geometry>

<contact\_coefficients mu="1" kp="1e+13" kd="1.0"/>

</collision>

</link>

<joint name="wheel\_right\_joint" type="continuous">

<origin xyz="0.2 0.25 0.0" rpy="1.57 0.0 0.0"/>

<parent link="base"/>

<child link="wheel\_right\_link"/>

<axis xyz="0.0 0.0 1.0"/>

</joint>

<link name="wheel\_left\_link">

<inertial>

<mass value="2" />

<inertia ixx="0.01" ixy="0.0" ixz="0"

iyy="0.01" iyz="0" izz="0.01" />

</inertial>

<visual>

<geometry>

<cylinder radius="0.15" length="0.1"/>

</geometry>

<material name="blue">

<color rgba="0 0 1 1"/>

</material>

</visual>

<collision>

<geometry>

<cylinder radius="0.15" length="0.1"/>

</geometry>

<contact\_coefficients mu="1" kp="1e+13" kd="1.0"/>

</collision>

</link>

<joint name="wheel\_left\_joint" type="continuous">

<origin xyz="0.2 -0.25 0.0" rpy="1.57 0.0 0.0"/>

<parent link="base"/>

<child link="wheel\_left\_link"/>

<axis xyz="0.0 0.0 1.0"/>

</joint>

<link name="caster">

<inertial>

<mass value="1" />

<inertia ixx="0.01" ixy="0.0" ixz="0"

iyy="0.01" iyz="0" izz="0.01" />

</inertial>

<visual>

<geometry>

<sphere radius=".08" />

</geometry>

<material name="white" />

</visual>

<collision>

<origin/>

<geometry>

<sphere radius=".08" />

</geometry>

</collision>

</link>

<joint name="caster\_joint" type="continuous">

<origin xyz="-0.3 0.0 -0.07" rpy="0.0 0.0 0.0"/>

<axis xyz="0 0 1" />

<parent link="base"/>

<child link="caster"/>

</joint>

<link name="camera">

<inertial>

<mass value="0.5" />

<inertia ixx="0.01" ixy="0.0" ixz="0"

iyy="0.01" iyz="0" izz="0.01" />

</inertial>

<visual>

<geometry>

<box size="0.1 0.1 0.1"/>

</geometry>

<material name="red">

<color rgba="1 0 0 1"/>

</material>

</visual>

<collision>

<geometry>

<box size="0.1 0.1 0.1"/>

</geometry>

</collision>

</link>

<joint name="camera\_joint" type="fixed">

<origin xyz="-0.32 0 0.1" rpy="0 0.0 3.14"/>

<parent link="base"/>

<child link="camera"/>

<axis xyz="0.0 0.0 1.0"/>

</joint>

<!--http://wiki.ros.org/simulator\_gazebo/Tutorials/ListOfMaterials-->

<gazebo reference="base">

<material>Gazebo/WhiteGlow</material>

</gazebo>

<gazebo reference="wheel\_left\_link">

<material>Gazebo/SkyBlue</material>

</gazebo>

<gazebo reference="wheel\_right\_link">

<material>Gazebo/SkyBlue </material>

</gazebo>

<gazebo reference="caster">

<material>Gazebo/Grey</material>

</gazebo>

<gazebo reference="camera">

<material>Gazebo/Blue</material>

</gazebo>

<!-- differential robot-->

<gazebo>

<plugin filename="libgazebo\_ros\_diff\_drive.so" name="gazebo\_base\_controller">

<odometry\_frame>odom</odometry\_frame>

<commandTopic>cmd\_vel</commandTopic>

<publish\_odom>true</publish\_odom>

<publish\_odom\_tf>true</publish\_odom\_tf>

<update\_rate>15.0</update\_rate>

<left\_joint>wheel\_left\_joint</left\_joint>

<right\_joint>wheel\_right\_joint</right\_joint>

<wheel\_separation>0.5</wheel\_separation>

<wheel\_diameter>0.3</wheel\_diameter>

<max\_wheel\_acceleration>0.7</max\_wheel\_acceleration>

<max\_wheel\_torque>8</max\_wheel\_torque>

<robotBaseFrame>base</robotBaseFrame>

</plugin>

</gazebo>

<!-- camera plugin-->

<gazebo reference="camera">

<sensor type="camera" name="camera1">

<visualize>true</visualize>

<update\_rate>30.0</update\_rate>

<camera name="head">

<horizontal\_fov>1.3962634</horizontal\_fov>

<image>

<width>512</width>

<height>512</height>

<format>R8G8B8</format>

</image>

<clip>

<near>0.02</near>

<far>500</far>

</clip>

</camera>

<plugin name="camera\_controller" filename="libgazebo\_ros\_camera.so">

<alwaysOn>true</alwaysOn>

<updateRate>0.0</updateRate>

<cameraName>/camera</cameraName>

<imageTopicName>image\_raw</imageTopicName>

<cameraInfoTopicName>camera\_info</cameraInfoTopicName>

<frameName>camera\_link</frameName>

<hackBaseline>0.07</hackBaseline>

</plugin>

</sensor>

<material>Gazebo/Blue</material>

</gazebo>

</robot>

Create a launch folder

Create a launch file rviz.launch.py

import os

from ament\_index\_python.packages import get\_package\_share\_directory

from launch import LaunchDescription

from launch\_ros.actions import Node

def generate\_launch\_description():

package\_dir = '/home/asha/ros2\_ws/src/lab6/urdf'

urdf = os.path.join(package\_dir,'car.urdf')

return LaunchDescription([

Node(

package='robot\_state\_publisher',

executable='robot\_state\_publisher',

name='robot\_state\_publisher',

output='screen',

arguments=[urdf]),

Node(

package='joint\_state\_publisher\_gui',

executable='joint\_state\_publisher\_gui',

name='joint\_state\_publisher\_gui',

arguments=[urdf]),

Node(

package='rviz2',

executable='rviz2',

name='rviz2',

output='screen'),

])

Create a launch folder

Create a launch file gazebo.launch.py inside the launch folder

import os

from ament\_index\_python.packages import get\_package\_share\_directory

from launch import LaunchDescription

from launch.actions import DeclareLaunchArgument, ExecuteProcess

from launch.substitutions import LaunchConfiguration

from launch\_ros.actions import Node

from launch.launch\_description\_sources import PythonLaunchDescriptionSource

def generate\_launch\_description():

urdf = '/home/asha/ros2\_ws/src/lab6/urdf/car.urdf'

return LaunchDescription([

# publishes TF for links of the robot without joints

Node(

package='robot\_state\_publisher',

executable='robot\_state\_publisher',

name='robot\_state\_publisher',

output='screen',

arguments=[urdf]),

# publish TF for Joints only links

Node(

package='joint\_state\_publisher',

executable='joint\_state\_publisher',

name='joint\_state\_publisher',

output='screen',

),

# open gazebo

ExecuteProcess(

cmd=['gazebo', '--verbose', '-s', 'libgazebo\_ros\_factory.so'],

output='screen'),

Node(

package='gazebo\_ros',

executable='spawn\_entity.py',

name='urdf\_spawner',

output='screen',

arguments=["-topic", "/robot\_description", "-entity", "lab6"])

])

* **Height and width**: These are the dimensions in camera pixels. In this case, it's 512 **x 512**.
* **Encoding**: How these pixels are encoded. This means what each value in the data array will mean. In this case, it's **rgb8**. This means that the data values will be a color value represented as red/green/blue in 8-bit integers.
* **Data**: The image data.

Terminal 1:

ros2 launch lab6 rviz.launch.py

ros2 launch lab6 gazebo.launch.py

ctrl + c to kill the rviz and gazebo window

create a file capture\_image.py inside the lab 6 folder

import rclpy

import cv2

from rclpy.node import Node

from cv\_bridge import CvBridge

from sensor\_msgs.msg import Image

class Capture(Node):

def \_\_init\_\_(self):

super().\_\_init\_\_('video\_subscriber')

self.subscriber = self.create\_subscription(Image,'/camera1/image\_raw',self.process\_data,10)

self.out = #cv2.VideoWriter('/home/asha/output.avi',cv2.VideoWriter\_fourcc('M','J','P','G'), 10, (512,512))

self.bridge = CvBridge()

def process\_data(self, data):

frame = self.bridge.imgmsg\_to\_cv2(data)

self.out.write(frame)

self.img = cv2.imwrite('/home/asha/shot.png', frame)

cv2.imshow("output", frame)

cv2.waitKey()

cv2.destroyAllWindows()

def main(args=None):

rclpy.init(args=args)

node = Capture()

rclpy.spin(node)

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

Terminal 1:

ros2 launch lab6 gazebo.launch.py

draw a line and place the robot on the line (insertyellow line)

Terminal 2:

ros2 run lab6 capture

Press ctrl + c to capture the image of the road

Graphical user interface, application

Description automatically generated

create a file extract\_road.py inside the lab 6 folder

import cv2

import numpy

image = cv2.imread('/home/asha/shot.png')

def mouse(event,x,y,flags,param):

if event==cv2.EVENT\_LBUTTONDOWN:

h=image[y,x,0]

s=image[y,x,1]

v=image[y,x,2]

print("H:",h)

print("S:",s)

print("V:",v)

cv2.namedWindow('mouse')

cv2.setMouseCallback('mouse',mouse)

cv2.imshow("original image", image)

cv2.imshow("mouse", image)

cv2.waitKey(0)

cv2.destroyAllWindows()

light\_line = numpy.array([250,0,0])

dark\_line = numpy.array([255,10,10])

mask = cv2.inRange(image, light\_line,dark\_line)

cv2.imshow('mask', mask)

cv2.waitKey(0)

cv2.destroyAllWindows()

canny= cv2.Canny(mask,30,5)

cv2.imshow('edge', canny)

cv2.waitKey(0)

cv2.destroyAllWindows()

print(canny.shape)

r1=200;c1=0

img = canny[r1:r1+200,c1:c1+512]

cv2.imshow('crop', img)

cv2.waitKey(0)

cv2.destroyAllWindows()

edge=[]

row =150

for i in range (512):

if(img[row,i]==255):

edge.append(i)

print(edge)

if(len(edge)==4):

left\_edge=edge[0]

right\_edge=edge[2]

print(edge)

if(len(edge)==3):

if(edge[1]-edge[0] > 5):

left\_edge=edge[0]

right\_edge=edge[1]

else:

left\_edge=edge[0]

right\_edge=edge[2]

road\_width=(right\_edge-left\_edge)

frame\_mid = left\_edge + (road\_width/2)

mid\_point = 512/2

img[row,int(mid\_point)]=255

print(mid\_point)

error=mid\_point-frame\_mid

if(error < 0):

action="Go Right"

else :

action="Go Left"

print("error", error)

img[row,int(frame\_mid)]=255

print("mid point of the frame", frame\_mid)

f\_image = cv2.putText(img, action, (50,50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,0), 1, cv2.LINE\_AA)

cv2.imshow('final image',f\_image)

cv2.waitKey(0)

cv2.destroyAllWindows()

Graphical user interface, text, application

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Create a line\_follow.py inside the folder lab 6

#!/usr/bin/env python3

import sys

import cv2

import numpy

import rclpy

from rclpy.node import Node

from cv\_bridge import CvBridge

from sensor\_msgs.msg import Image

from geometry\_msgs.msg import Twist

class LineFollower(Node):

def \_\_init\_\_(self):

super().\_\_init\_\_('line\_follower')

self.bridge = CvBridge()

self.subscriber = self.create\_subscription(Image,'/camera1/image\_raw',self.process\_data, 10)

self.publisher = self.create\_publisher(Twist, '/cmd\_vel', 40)

timer\_period = 0.2

self.timer = self.create\_timer(timer\_period, self.send\_cmd\_vel)

self.velocity=Twist()

self.empty = False

self.error = 0

self.action=""

self.get\_logger().info("Node Started!")

def send\_cmd\_vel(self):

if(self.empty):

self.velocity.linear.x=0.0

self.velocity.angular.z= 0.0

self.action="Stop"

else:

if(self.error > 0):

self.velocity.linear.x=0.1

self.velocity.angular.z=0.1

self.action="Go Left"

elif(self.error < 0):

self.velocity.linear.x=0.1

self.velocity.angular.z=-0.1

self.action="Go Right"

elif(self.error==0):

self.velocity.linear.x=0.1

self.velocity.angular.z= 0.0

self.action="Go Straight"

self.publisher.publish(self.velocity)

## Subscriber Call Back

def process\_data(self, data):

self.get\_logger().info("Image Received!")

frame = self.bridge.imgmsg\_to\_cv2(data)

light\_line = numpy.array([250,0,0])

dark\_line = numpy.array([255,10,10])

mask = cv2.inRange(frame, light\_line,dark\_line)

cv2.imshow('mask', mask)

canny= cv2.Canny(mask,30,5)

cv2.imshow('edge', canny)

r1=200;c1=0

img = canny[r1:r1+200,c1:c1+512]

cv2.imshow('crop', img)

edge=[]

row =150

for i in range(512):

if(img[row,i]==255):

edge.append(i)

print(edge)

if(len(edge)==0):

left\_edge=512//2

right\_edge=512//2

self.empty = True

if(len(edge)==1):

if edge[0]>512//2:

left\_edge=0

right\_edge=edge[0]

self.empty = False

else:

left\_edge=edge[0]

right\_edge=512

self.empty = False

if(len(edge)==2):

left\_edge=edge[0]

right\_edge=edge[1]

self.empty = False

if(len(edge)==3):

if(edge[1]-edge[0]>5):

left\_edge=edge[0]

right\_edge=edge[1]

self.empty = False

else:

left\_edge=edge[0]

right\_edge=edge[2]

self.empty = False

if(len(edge)==4):

left\_edge=edge[0]

right\_edge=edge[2]

self.empty = False

if(len(edge)>=5):

left\_edge=edge[0]

right\_edge=edge[len(edge)-1]

self.empty = False

road\_width=(right\_edge-left\_edge)

frame\_mid = left\_edge + (road\_width/2)

mid\_point = 512/2

img[row,int(mid\_point)]=255

print(mid\_point)

self.error=mid\_point-frame\_mid

img[row,int(frame\_mid)]=255

print(self.action)

f\_image = cv2.putText(img, self.action, (100,100), cv2.FONT\_HERSHEY\_SIMPLEX, 1, (255,0,), 2, cv2.LINE\_AA)

def main(args=None):

rclpy.init(args=args)

node = LineFollower()

rclpy.spin(node)

rclpy.shutdown()

if \_\_name\_\_ == '\_\_main\_\_':

main()

edit setup.py file

from setuptools import setup

import os

from glob import glob

package\_name = 'lab6'

setup(

name=package\_name,

version='0.0.0',

packages=[package\_name],

data\_files=[

('share/ament\_index/resource\_index/packages',

['resource/' + package\_name]),

('share/' + package\_name, ['package.xml']),

(os.path.join('share', package\_name), glob('launch/\*')),

(os.path.join('share', package\_name), glob('urdf/\*'))

],

install\_requires=['setuptools'],

zip\_safe=True,

maintainer='asha',

maintainer\_email='asha.cs12@gmail.com',

description='TODO: Package description',

license='TODO: License declaration',

tests\_require=['pytest'],

entry\_points={

'console<?xml version="1.0"?>

<sdf version="1.6">

<model name="yellow line">

<static>true</static>

<link name="link\_ground">

<collision name="collision">

<geometry>

<plane>

<normal>0 0 1</normal>

<size>0.1 3.2</size>

</plane>

</geometry>

<surface>

<friction>

<ode>

<mu>100</mu>

<mu2>50</mu2>

</ode>

</friction>

</surface>

</collision>

<visual name="visual\_ground">

<cast\_shadows>false</cast\_shadows>

<geometry>

<plane>

<normal>0 0 1</normal>

<size>0.5 10</size>

</plane>

</geometry>

<material>

<script>

<uri>file://media/materials/scripts/gazebo.material</uri>

<name>Gazebo/Red</name>

</script>

</material>

</visual>

</link>

</model>

</sdf>\_scripts': [

'capture = lab6.capture\_image:main',

'line = lab6.line\_follow:main'

],

},

)

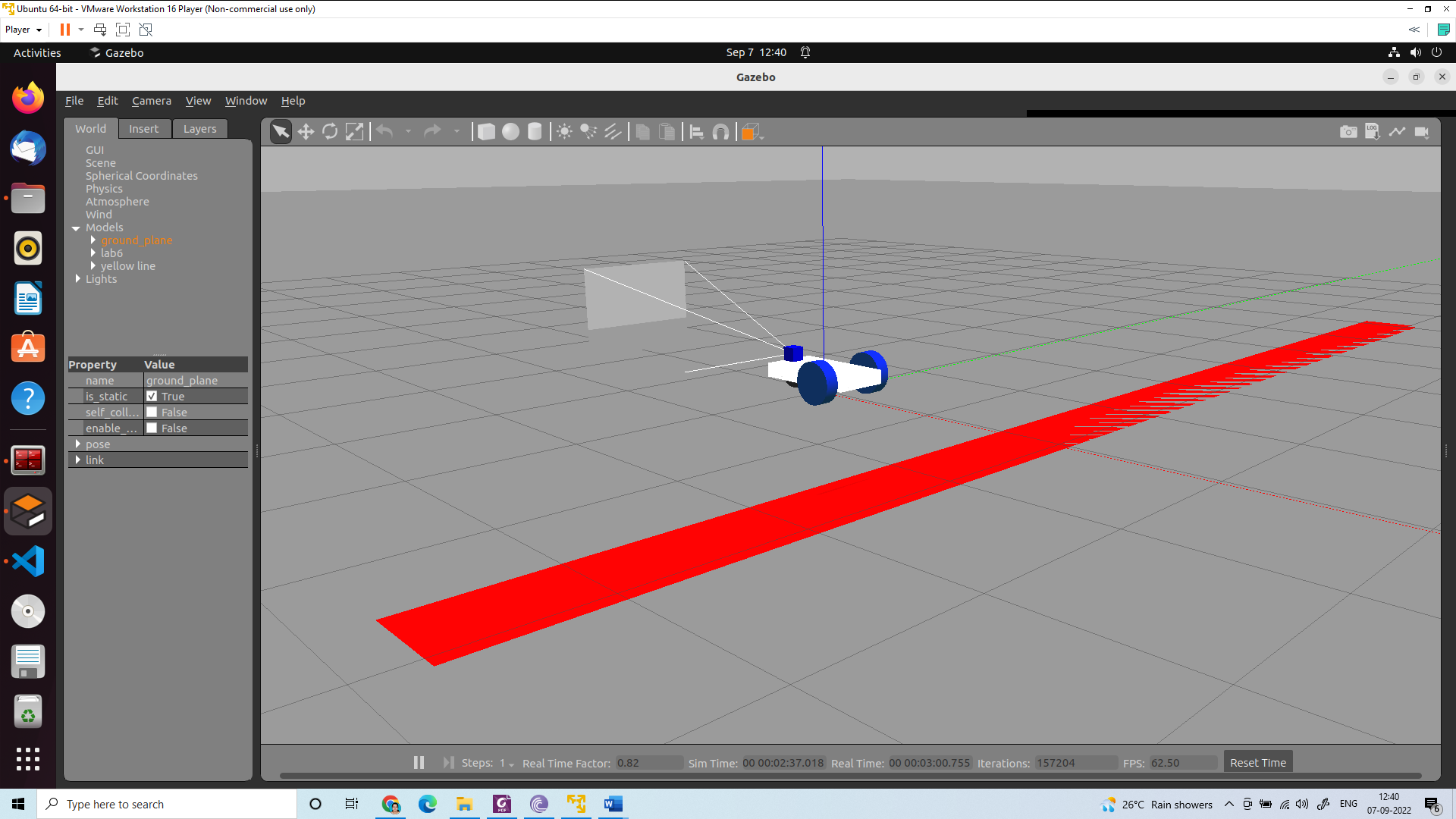
A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

select yellow line



Use these to change the location and orientation of the robot

A screenshot of a computer

Description automatically generated with medium confidence

Terminal 1:

ros2 launch lab6 gazebo.launch.py

Terminal 2:

ros2 topic list

Terminal 3:

ros2 run lab6 line

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated with medium confidence

Create a red line

Create a world folder

Create a yellow line folder

Supplementary material:

Create a model.config

<?xml version="1.0" ?>

<model>

<name>yellowline</name>

<version>1.0</version>

<sdf version="1.6">model.sdf</sdf>

<author>

<name></name>

<email></email>

</author>

<description></description>

</model>

Create a model.sdf

<?xml version="1.0"?>

<sdf version="1.6">

<model name="yellow line">

<static>true</static>

<link name="link\_ground">

<collision name="collision">

<geometry>

<plane>

<normal>0 0 1</normal>

<size>0.1 3.2</size>

</plane>

</geometry>

<surface>

<friction>

<ode>

<mu>100</mu>

<mu2>50</mu2>

</ode>

</friction>

</surface>

</collision>

<visual name="visual\_ground">

<cast\_shadows>false</cast\_shadows>

<geometry>

<plane>

<normal>0 0 1</normal>

<size>0.5 10</size>

</plane>

</geometry>

<material>

<script>

<uri>file://media/materials/scripts/gazebo.material</uri>

<name>Gazebo/Red</name>

</script>

</material>

</visual>

</link>

</model>

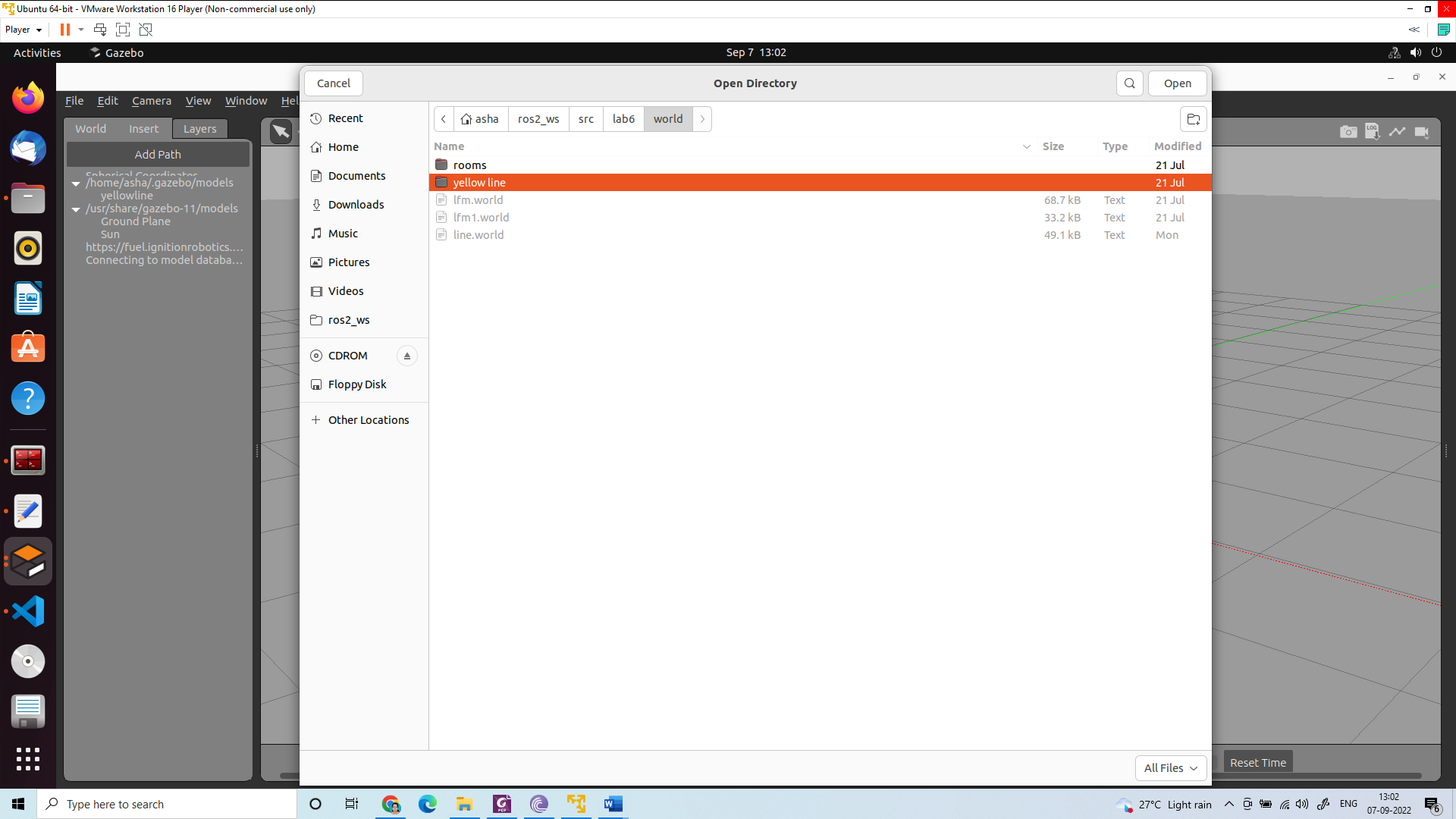
</sdf>

Copy the folder into /home/asha/.gazebo/models/

In the gazebo window (gazebo)

Insert/home/asha/.gazebo/models/yellow lineall files

close gazebo (killall 9 gzserver) and open again (gazebo)



Insert/home/asha/.gazebo/models/yellow line

Exercise: Write a code to track the red ball in the gazebo simulation.