Room Navigator Robot Using ROS 2 and Machine Learning

By:
Sambhav Jain

Abstract

A smart room navigation system is developed using ROS 2 and a trained machine learning model. It predicts the target room based on environmental task inputs (time of day, task type, dirt level) and simulates movement. The system is modular, dynamic, and scalable for both simulation and real-robot deployment.

Introduction

Problem Statement

Cleaning/navigation robots often operate inefficiently due to fixed rule-based logic. A smarter approach is needed to dynamically decide which room to go to based on the situation.

Objective

To design and implement a robot navigation system that uses machine learning and ROS 2 communication to predict and simulate room navigation based on task inputs.

Technologies Used

Component	Description
ROS 2 (Jazzy)	Robotics framework (publish/subscribe)
Python 3.12	Main programming language
scikit-learn	ML model training (Decision Tree)
joblib	Model serialization
pandas	Data loading and manipulation
rclpy	Python client library for ROS 2

Dataset and Model

Dataset: 200+ rows

Fields: Time of Day, Task Type, Dirt Level, Target Room

Link to Dataset: https://docs.google.com/spreadsheets/d/1GWw2Ie7FlO5spMtEjTkoVTBc-WpgwOJ7BYcMv_7_we8/edit?pli=1&gid=193057329#gid=193057329

ML

Preprocessing

Used LabelEncoder to convert categorical data into numbers
Split into 80% training and 20% testing

Model

Algorithm: DecisionTreeClassifier
Accuracy: ~47% (can improve with more data or model tuning)

Output

Saved using joblib:
room_decision_tree.pkl
label_encoders.pkl

System Architecture

ROS 2 Nodes

Node	Role
input_node	Publishes random task input every 5 seconds
decision_node	Predicts the room using ML and publishes to /target_room
navigator_node	Subscribes to room and simulates movement

Communication Flow

- input_node --> /task_conditions --> decision_node
- decision_node --> /target_room --> navigator_node

Node Descriptions

1 input_node.py

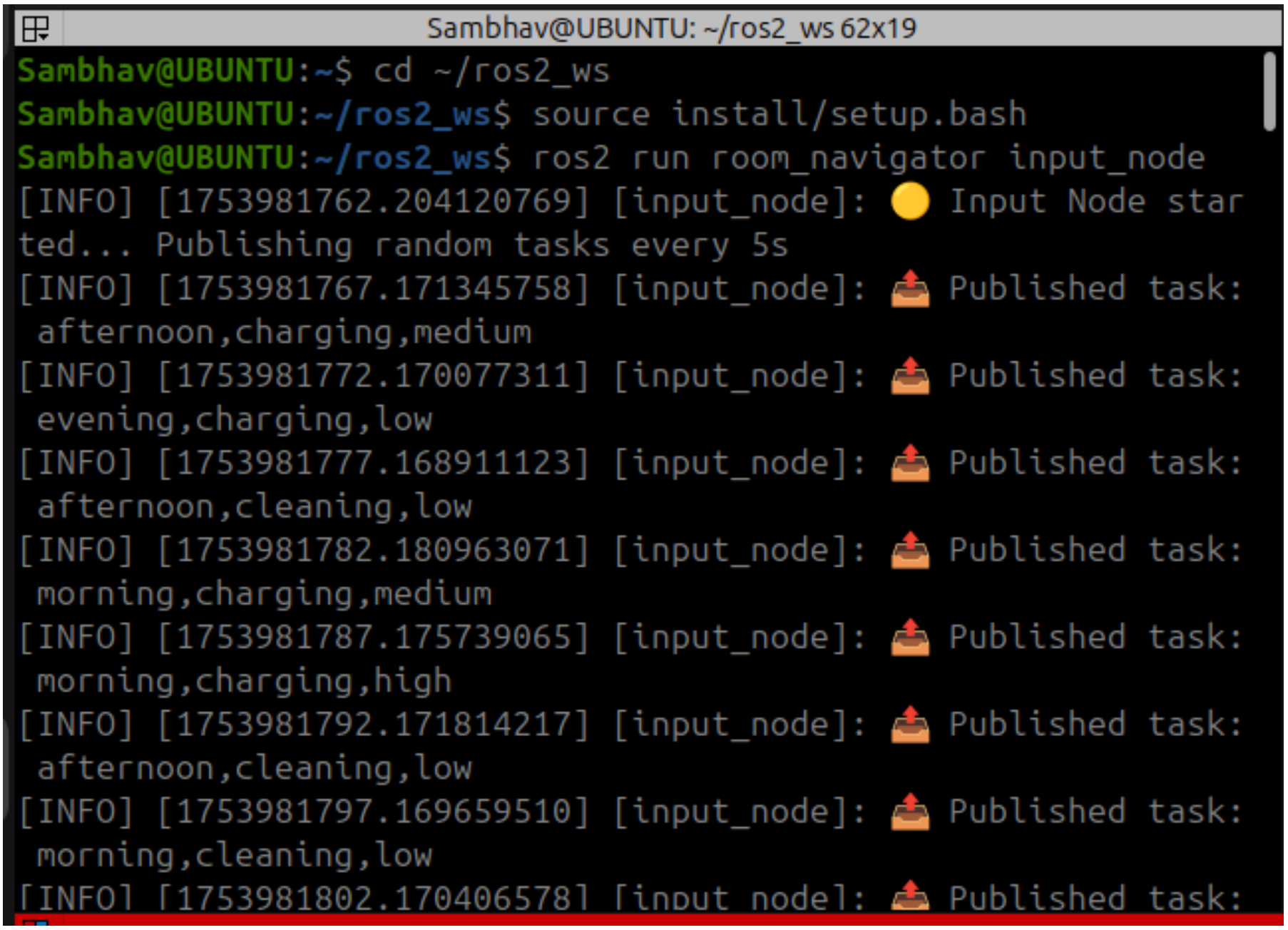
- Uses random.choice() to simulate real-time input
- Publishes morning,delivery,high like messages

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import random

class InputNode(Node):
    def __init__(self):
        super().__init__('input_node')
        self.publisher_ = self.create_publisher(String, '/task_conditions', 10)
        self.timer = self.create_timer(5.0, self.publish_task)
        self.times = ['morning', 'afternoon', 'evening', 'night']
        self.tasks = ['delivery', 'cleaning', 'charging']
        self.dirt_levels = ['low', 'medium', 'high']
        self.get_logger().info('🟡 Input Node started... Publishing random tasks every 5s')

    def publish_task(self):
        time = random.choice(self.times)
        task = random.choice(self.tasks)
        dirt = random.choice(self.dirt_levels)
        msg = String()
        msg.data = f"{time},{task},{dirt}"
        self.publisher_.publish(msg)
        self.get_logger().info(f"📤 Published task: {msg.data}")

def main(args=None):
    rclpy.init(args=args)
    node = InputNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Run input_node (publishes task conditions):

“ ros2 run room_navigator input_node “

Output:


Sambhav@UBUNTU: ~/ros2_ws 62x19
Sambhav@UBUNTU:~$ cd ~/ros2_ws
Sambhav@UBUNTU:~/ros2_ws$ source install/setup.bash
Sambhav@UBUNTU:~/ros2_ws$ ros2 run room_navigator input_node
[INFO] [1753981762.204120769] [input_node]: 🟡 Input Node started... Publishing random tasks every 5s
[INFO] [1753981767.171345758] [input_node]: 📤 Published task: afternoon,charging,medium
[INFO] [1753981772.170077311] [input_node]: 📤 Published task: evening,charging,low
[INFO] [1753981777.168911123] [input_node]: 📤 Published task: afternoon,cleaning,low
[INFO] [1753981782.180963071] [input_node]: 📤 Published task: morning,charging,medium
[INFO] [1753981787.175739065] [input_node]: 📤 Published task: morning,charging,high
[INFO] [1753981792.171814217] [input_node]: 📤 Published task: afternoon,cleaning,low
[INFO] [1753981797.169659510] [input_node]: 📤 Published task: morning,cleaning,low
[INFO] [1753981802.170406578] [input_node]: 📤 Published task:

2 decision_node.py

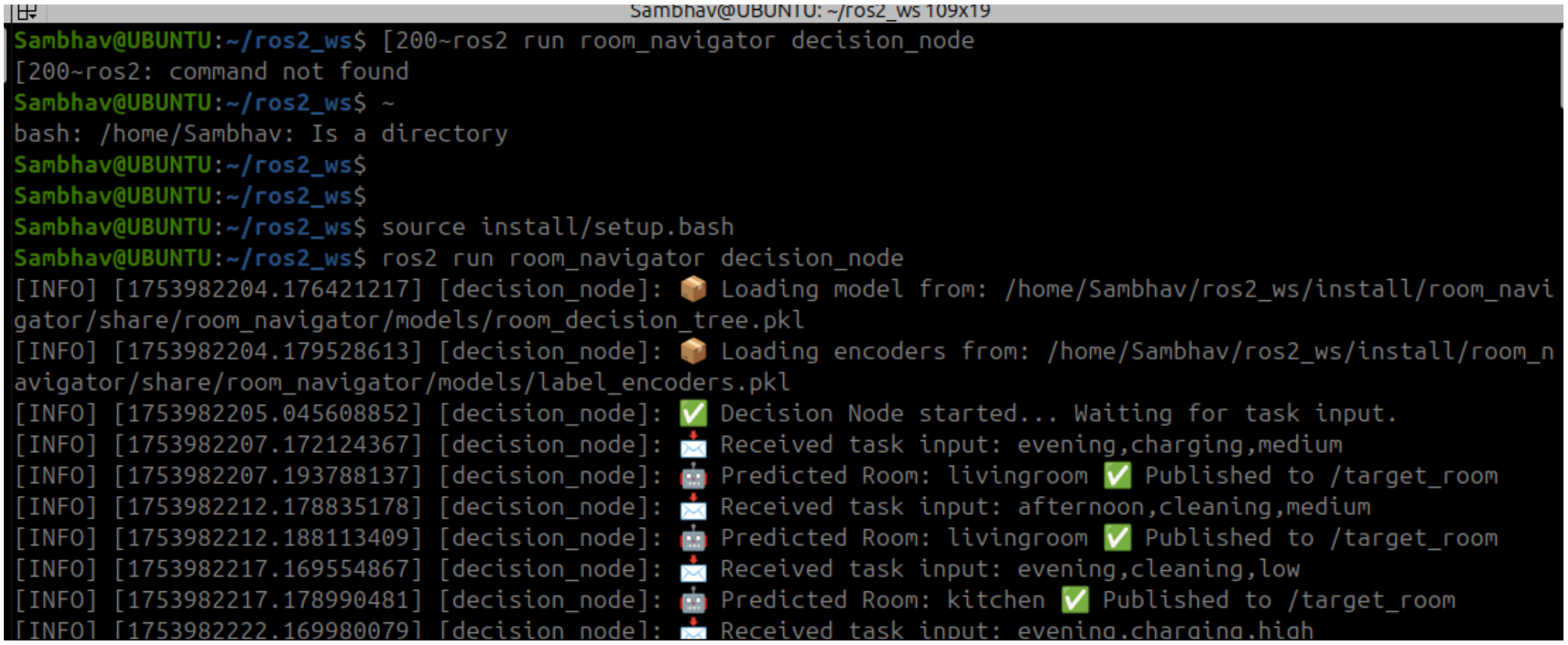
- Loads ML model and encoders
- Subscribes to /task_conditions
- Publishes predicted room to /target_room

```python
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import joblib
import os
import pandas as pd  # ✅ Used for clean prediction input

class DecisionNode(Node):
    def __init__(self):
        super().__init__('decision_node')

        # Locate installed model directory
        install_base = os.environ['AMENT_PREFIX_PATH'].split(':')[0]
        model_dir = os.path.join(install_base, 'share', 'room_navigator', 'models')
        model_path = os.path.join(model_dir, 'room_decision_tree.pkl')
        encoder_path = os.path.join(model_dir, 'label_encoders.pkl')

        # Logging model paths
        self.get_logger().info(f"📦 Loading model from: {model_path}")
        self.get_logger().info(f"📦 Loading encoders from: {encoder_path}")

        # Load model and encoders
        self.model = joblib.load(model_path)
        self.le_time, self.le_task, self.le_dirt, self.le_target = joblib.load(encoder_path)

        # ROS2 Subscriber and Publisher
        self.subscription = self.create_subscription(
            String,
            '/task_conditions',
            self.listener_callback,
            10
        )
        self.publisher_ = self.create_publisher(String, '/target_room', 10)

        self.get_logger().info('✅ Decision Node started... Waiting for task input.')

    def listener_callback(self, msg):
        self.get_logger().info(f"📩 Received task input: {msg.data}")
        try:
            time_str, task_str, dirt_str = msg.data.strip().lower().split(',')

            # ✅ Use DataFrame to avoid warning
            X_input = pd.DataFrame([{
                'time_encoded': self.le_time.transform([time_str])[0],
                'task_encoded': self.le_task.transform([task_str])[0],
                'dirt_encoded': self.le_dirt.transform([dirt_str])[0]
            }])

            pred_class = self.model.predict(X_input)[0]
            predicted_room = self.le_target.inverse_transform([pred_class])[0]

            # Publish predicted room
            out_msg = String()
            out_msg.data = predicted_room
            self.publisher_.publish(out_msg)

            self.get_logger().info(f"🖨️ Predicted Room: {predicted_room} ✅ Published to /target_room")

        except Exception as e:
            self.get_logger().error(f"❌ Prediction failed: {str(e)}")

def main(args=None):
    rclpy.init(args=args)
    node = DecisionNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Run decision_node (predicts room using ML:

“ ros2 run room_navigator decision_node “

Output:



3 navigator_node.py

- Listens to /target_room
- Simulates movement with print logs
- Adds a 3-second delay to simulate travel time

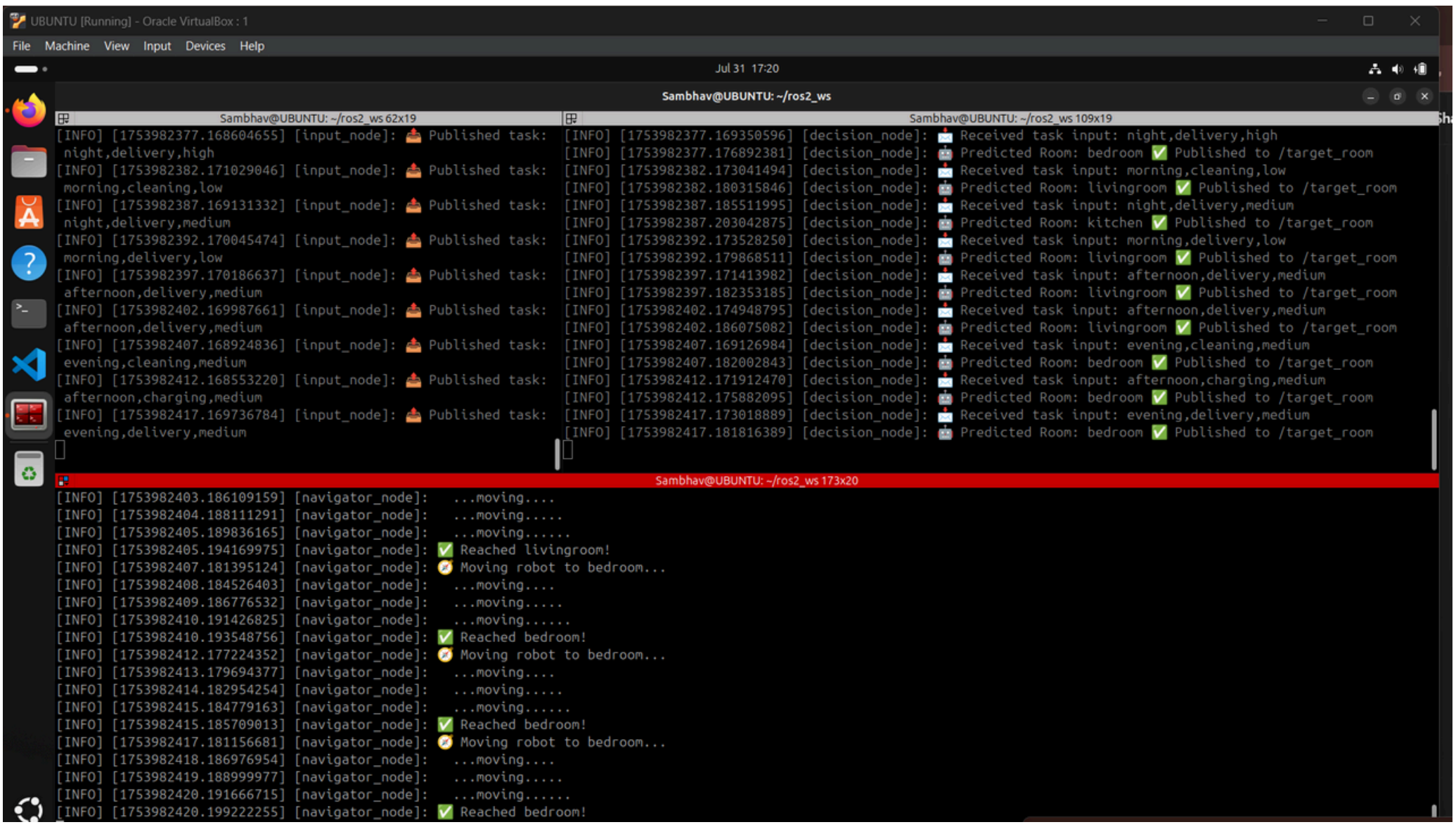
```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
import time

class NavigatorNode(Node):
    def __init__(self):
        super().__init__('navigator_node')
        self.subscription = self.create_subscription(String, '/target_room', self.listener_callback, 10)
        self.get_logger().info('🚀 Navigator Node started. Waiting for room target...')

    def listener_callback(self, msg):
        room = msg.data
        self.get_logger().info(f"🧭 Moving robot to {room}...")
        for i in range(3):
            time.sleep(1)
            self.get_logger().info(f"  ...moving{'.' * (i + 1)}")
        self.get_logger().info(f"✅ Reached {room}!")

def main(args=None):
    rclpy.init(args=args)
    node = NavigatorNode()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Sample Output

Console:

📤 Published task: morning,delivery,high
🤖 Predicted Room: kitchen ✅ Published to /target_room
🧭 Moving robot to kitchen...
✅ Reached kitchen!

Terminal



Simulation Limitation

Due to system resource constraints, full simulation using Gazebo or RViz could not be performed on my laptop. However, logical simulation was implemented through ROS 2 nodes and console output.

Conclusion

The project successfully demonstrates a smart, modular, ML-powered robot navigation system. Despite not running full 3D simulation, the system:
Processes real-time input
Makes predictions via a trained model
Simulates robot behavior through ROS 2 messaging
The architecture is ready for extension to real robots or integration into simulation environments.

Drive Link for all code and set up: Link

---------- Thank You ----------