**PROGRAM CODE - 1**

**INITIAL UDP NODE PROGRAM BY DR.ROLANDO**

#!/usr/bin/env python3

import rclpy

from rclpy.node import Node

from std\_msgs.msg import Float32, String  # Import message types

import socket

import struct

Library Import

class UDPServerNode(Node):

    def \_\_init\_\_(self):

        super().\_\_init\_\_('udp\_server')

Defining the UDP node

        self.udp\_ip = "172.16.11.12"

        self.udp\_port = 7000

        self.destination\_ip = "172.16.11.70"

        self.destination\_port = 7000

Configuring UDP node IPs

  self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

        self.sock.bind((self.udp\_ip, self.udp\_port))

        self.get\_logger().info(f"UDP Server listening on {self.udp\_ip}:{self.udp\_port}")

Initialize UDP socket

Send initial message

 self.sock.sendto("hola erick".encode(), (self.destination\_ip, self.destination\_port))

self.data\_publisher = self.create\_publisher(String, '/ros2mqtt/position', 10)

        #self.string\_publisher = self.create\_publisher(String, '/ros2mqtt/position', 10)

ROS 2 Publisher

string\_msg = String()

            #string\_msg.data = f"From {addr}: {data.decode(errors='ignore')}"

self.string\_publisher.publish(string\_msg)

Publish raw message as a String

self.timer = self.create\_timer(.05, self.receive\_message)  # Check every 50s

    def receive\_message(self):

        try:

            data, addr = self.sock.recvfrom(4)# here it was 1024, it think acts as buffer when we receive smaller packeges

            self.get\_logger().info(f"Received raw data from {addr}: {data}")

Start receiving messages

            try:

                received\_float = struct.unpack('f', data)[0]

                float\_msg = Float32()

                string\_msg = String()

                float\_msg = received\_float

                string\_msg.data = ("UDP received: " + str(received\_float))

                self.data\_publisher.publish(string\_msg)

                self.get\_logger().info(f"msg Published to ros2mqtt/position: {received\_float}")

            except struct.error:

                self.get\_logger().warn(f"Invalid float data from {addr}: {data}")

        except Exception as e:

            self.get\_logger().error(f"Error receiving data: {e}")

Interpret data as a float

def destroy\_node(self):

        self.get\_logger().info("Shutting down UDP server...")

        self.sock.close()

        super().destroy\_node()

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

    main()

Logging the data and Main function sequence