ESP32 Final Code (No Encoder Version)

```
#include <ESP32Servo.h>

// --- PIN DEFINITION ---
const int R_PWM = 25;       // BTS7960 Forward
const int L_PWM = 26;       // BTS7960 Backward
const int SERVO_PIN = 18;   // สายสีเหลือง Servo CS001

// --- OBJECTS ---
Servo steeringServo;
int currentSpeed = 0;
int currentAngle = 90;

void setup() {
  Serial.begin(115200);

  // ตั้งค่ามอเตอร์ BTS7960
  pinMode(R_PWM, OUTPUT);
  pinMode(L_PWM, OUTPUT);

  // ตั้งค่า Servo
  steeringServo.attach(SERVO_PIN);
  steeringServo.write(currentAngle); // ล้อตรง

  // ตั้งค่า Safety Sensor
  pinMode(IR_PIN, INPUT);

  Serial.println(">>> ESP32 CONTROL SYSTEM READY (Final Ver.) <<<");
}

void loop() {
  // 1. ตรวจสอบความปลอดภัย (IR Sensor)
  // LOW = มีวัตถุประชิด / HIGH = ทางสะดวก
  bool isBlocked = (digitalRead(IR_PIN) == LOW);

  // 2. รับคำสั่งจาก Pi 5 (รูปแบบ D,speed,angle\n)
  if (Serial.available()) {
    String data = Serial.readStringUntil('\n');

    if (data.startsWith("D,")) {
      int firstComma = data.indexOf(',');
      int secondComma = data.indexOf(',', firstComma + 1);

      currentSpeed = data.substring(firstComma + 1, secondComma).toInt();
      currentAngle = data.substring(secondComma + 1).toInt();

      // Logic ความปลอดภัย
```

```
      if (isBlocked && currentSpeed > 0) {
        // มีสิ่งกีดขวาง: หยุดเดินหน้า แต่ยอมให้ถอยหลังได้
        executeDrive(0, currentAngle);
      } else {
        executeDrive(currentSpeed, currentAngle);
      }
    }
  }
}

void executeDrive(int speed, int angle) {
  // จำกัดองศาเซอร์โวป้องกันกลไกหัก (45-135 องศา)
  int safeAngle = constrain(angle, 45, 135);
  steeringServo.write(safeAngle);

  // สั่งงาน BTS7960
  if (speed > 0) {
    analogWrite(R_PWM, speed);
    analogWrite(L_PWM, 0);
  } else if (speed < 0) {
    analogWrite(R_PWM, 0);
    analogWrite(L_PWM, abs(speed));
  } else {
    analogWrite(R_PWM, 0);
    analogWrite(L_PWM, 0);
  }
}
```

พิน ESP32	**ต่อเข้ากับ**	**สีสาย / ข้อมูล**
GPIO 25	**R_PWM** (BTS7960)	คุมความเร็วเดินหน้า
GPIO 26	**L_PWM** (BTS7960)	คุมความเร็วถอยหลัง
GPIO 18	**Signal** (Servo)	**สายสีเหลือง**

GPIO 13	**Out** (IR Sensor)	ระบบเบรกฉุกเฉิน
Vin (5V)	**VCC** (Servo + BTS)	**สายสีแดง** (ระวังไฟตก)
GND	**GND Shared**	**สายสีน้ำตาล/ดำ** (ต้องรวมกันทุกอุปกรณ์)

```
// ขาต่อ BTS7960 (ตัวอย่างสำหรับ 1 ล้อ หรือต่อขนาน 2 ล้อ)
int RPWM = 12;
int LPWM = 13;

void setup() {
  Serial.begin(115200);
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
}

void loop() {
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    if (data.startsWith("S,")) {
      int speed = data.substring(2).toInt(); // รับค่า -255 ถึง 255

      if (speed > 0) { // เดินหน้า
        analogWrite(RPWM, speed);
        analogWrite(LPWM, 0);
      } else if (speed < 0) { // ถอยหลัง
        analogWrite(LPWM, abs(speed));
        analogWrite(RPWM, 0);
      } else { // หยุด
        analogWrite(RPWM, 0);
        analogWrite(LPWM, 0);
      }
    }
  }
}
```

วิธีเข้าเทอมินอล ssh
ssh ceo@10.74.247.146
007

ros2 run auto_tracker tracker_node

ros2 run ydlidar_ros2_driver ydlidar_ros2_driver_node --ros-args --params-file ~/ros2_ws/src/ydlidar_ros2_driver/params/ydlidar.yaml

rm ~/ros2_ws/src/auto_tracker/auto_tracker/tracker_node.py
nano ~/ros2_ws/src/auto_tracker/auto_tracker/tracker_node.py

colcon build --packages-select auto_tracker
source install/setup.bash

http://10.74.247.146:5000

```
sudo shutdown -h now
```

1. เข้าใช้งาน venv (ถ้ายังไม่ได้เข้า)
source venv/bin/activate

2. ตั้งค่าการแสดงผลหน้าต่างไปยังคอมพิวเตอร์ของคุณ
export DISPLAY=localhost:10.0

3. รันโปรแกรม
python3 main_gui.py

Ctrl+C # หยุดโปรแกรม
fuser -k 5000/tcp # ล้างพอร์ตค้าง
nano local_gui_server.py # แก้ไข/วางโค้ด
python3 local_gui_server.py # รันใหม่

โค้ด Esp32 สมบูรณ์

ESP32 Motor Control

```
#include <ESP32Servo.h>

// --- กำหนดขาใช้งาน (PIN DEFINITION) ---
const int R_PWM = 25;        // BTS7960 ขาเดินหน้า (Forward)
const int L_PWM = 26;        // BTS7960 ขาถอยหลัง (Backward)
const int SERVO_PIN = 18;    // ขาสัญญาณ Servo

// --- ประกาศตัวแปร (OBJECTS & VARIABLES) ---
Servo steeringServo;
int currentSpeed = 0;
int currentAngle = 90;

// ระบบ Safety Timeout
unsigned long lastCommandTime = 0;
const unsigned long TIMEOUT_MS = 500; // ลดเวลาเหลือ 0.5 วินาที เพื่อความไวในการตัดไฟ
bool hasReceivedFirstCommand = false;  // ตัวแปรเช็คว่าได้รับคำสั่งแรกจริงๆ หรือยัง

void setup() {
  Serial.begin(115200);

  // ตั้งค่ามอเตอร์ BTS7960
  pinMode(R_PWM, OUTPUT);
  pinMode(L_PWM, OUTPUT);

  // หยุดมอเตอร์ทันที
  stopMotors();

  // ตั้งค่า Servo
  steeringServo.attach(SERVO_PIN);
  steeringServo.write(90);

  // ล้างขยะใน Serial Buffer ที่อาจเกิดตอนบูตเครื่อง
  while(Serial.available() > 0) {
    Serial.read();
  }

  Serial.println(">>> ESP32 SYSTEM READY: STANDBY MODE <<<");
```

```
  // กำหนดให้เริ่มต้นในสถานะ Timeout ทันที
  lastCommandTime = 0;
  hasReceivedFirstCommand = false;
}

void loop() {
  // 1. รับคำสั่งจาก Pi 5
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');

    if (data.startsWith("D,")) {
      int firstComma = data.indexOf(',');
      int secondComma = data.indexOf(',', firstComma + 1);

      if (firstComma != -1 && secondComma != -1) {
        currentSpeed = data.substring(firstComma + 1,
secondComma).toInt();
        currentAngle = data.substring(secondComma + 1).toInt();

        // ยืนยันว่าได้รับคำสั่งที่ถูกต้องจาก ROS2 แล้ว
        lastCommandTime = millis();
        hasReceivedFirstCommand = true;
        executeDrive(currentSpeed, currentAngle);
      }
    }
  }

  // 2. ระบบความปลอดภัย (Failsafe)
  // ถ้ายังไม่เคยได้รับคำสั่งเลย หรือ คำสั่งขาดหายไปเกิน 0.5 วินาที -> ให้หยุดรถ
  if (!hasReceivedFirstCommand || (millis() - lastCommandTime >
TIMEOUT_MS)) {
    stopMotors();
    // ค่อยๆ คืนค่าความเร็วเป็น 0 เพื่อความปลอดภัย
    currentSpeed = 0;
  }
}

void executeDrive(int speed, int angle) {
  // ควบคุมการเลี้ยว
  int safeAngle = constrain(angle, 45, 135);
  steeringServo.write(safeAngle);
```

```
  // ควบคุมมอเตอร์ (ทำงานเฉพาะเมื่อได้รับคำสั่งที่ถูกต้อง)
  if (speed > 0) {
    analogWrite(R_PWM, constrain(speed, 0, 255));
    analogWrite(L_PWM, 0);
  } else if (speed < 0) {
    analogWrite(R_PWM, 0);
    analogWrite(L_PWM, constrain(abs(speed), 0, 255));
  } else {
    stopMotors();
  }
}

void stopMotors() {
  analogWrite(R_PWM, 0);
  analogWrite(L_PWM, 0);
}
```

Tracker Node

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from rclpy.qos import QoSProfile, ReliabilityPolicy, HistoryPolicy
import serial
import time
import numpy as np

class TrackerNode(Node):
    def __init__(self):
        super().__init__('tracker_node')

        # ล็อคพอร์ตที่ USB1
        self.target_port = '/dev/ttyUSB1'
        self.ser = None
        self.current_speed = 0.0
        self.last_send_time = time.time()
        self.send_interval = 0.1  # 10Hz

        self.connect_serial()

        qos_profile = QoSProfile(
            reliability=ReliabilityPolicy.BEST_EFFORT,
            history=HistoryPolicy.KEEP_LAST,
            depth=1
        )

        self.subscription = self.create_subscription(
            LaserScan,
            '/scan',
            self.lidar_callback,
            qos_profile)

        self.get_logger().info(f'🚀 Tracker Node Start (Increased Speed
& Corrected Logic)')

    def connect_serial(self):
        """เชื่อมต่อ USB1"""
        try:
```

```
            self.ser = serial.Serial(self.target_port, 115200,
timeout=0.1, write_timeout=0.1)
            if self.ser.is_open:
                self.get_logger().info(f'✅ USB1 Connected!')
                self.ser.reset_input_buffer()
                self.ser.reset_output_buffer()
                return True
        except Exception as e:
            self.get_logger().error(f'❌ Cannot open
{self.target_port}: {e}')
            return False

    def lidar_callback(self, msg):
        # ตรวจสอบวัตถุช่วงหน้า (ซ้าย 40 องศา ถึง ขวา 40 องศา)
        indices = list(range(0, 40)) + list(range(len(msg.ranges)-40,
len(msg.ranges)))
        valid_ranges = [msg.ranges[i] for i in indices if 0.15 <
msg.ranges[i] < 3.0]

        if len(valid_ranges) > 5:
            dist = np.nanmedian(valid_ranges) * 100
            target_dist = 60.0  # ระยะเป้าหมาย 60 ซม.
            error = dist - target_dist

            # ปรับจูน Logic และเพิ่มความเร็ว:
            # หาก dist > 60 (error > 0) -> ต้องเดินหน้า
            # หาก dist < 60 (error < 0) -> ต้องถอยหลัง
            if abs(error) < 12.0:  # Deadzone 12 ซม.
                target_raw = 0
            else:
                # เพิ่มความเร็วเป็น 105 เพื่อให้มอเตอร์มีกำลังพอจะเดินหน้า
                # หากสลับทิศทาง ให้เปลี่ยนเครื่องหมายตรงนี้
                target_raw = 105 if error > 0 else -105

            # Ultra Soft-Start: ค่อยๆ เร่งความเร็วเพื่อป้องกัน USB หลุด
            ramp_rate = 12
            if self.current_speed < target_raw:
                self.current_speed = min(self.current_speed +
ramp_rate, target_raw)
            elif self.current_speed > target_raw:
                self.current_speed = max(self.current_speed -
ramp_rate, target_raw)
```

```
            # คำนวณการเลี้ยว (Steering)
            left_side = [r for r in msg.ranges[0:40] if 0.15 < r < 3.0]
            right_side = [r for r in msg.ranges[-40:] if 0.15 < r <
3.0]
            l_avg = np.mean(left_side) if left_side else 2.5
            r_avg = np.mean(right_side) if right_side else 2.5

            steer = (l_avg - r_avg) * 45
            angle = 90 - int(steer)
            angle = max(min(angle, 135), 45)

            # ส่งคำสั่ง
            now = time.time()
            if now - self.last_send_time >= self.send_interval:
                self.send_command(int(self.current_speed), angle, dist)
                self.last_send_time = now
        else:
            self.stop_robot()

    def send_command(self, speed, angle, dist):
        if self.ser and self.ser.is_open:
            try:
                cmd = f"D,{speed},{angle}\n"
                self.ser.write(cmd.encode())
                self.ser.flush()
                if speed != 0:
                    # แสดงสถานะทิศทางใน Log เพื่อเช็คความถูกต้อง
                    direction = "FORWARD" if speed > 0 else "BACKWARD"
                    self.get_logger().info(f'📡 {direction} | Spd:
{speed} | Dist: {dist:.1f}cm')
            except:
                self.connect_serial()

    def stop_robot(self):
        if self.current_speed != 0:
            self.current_speed = 0
            if self.ser and self.ser.is_open:
                try:
                    self.ser.write(b"D,0,90\n")
                    self.ser.flush()
                except:
                    pass
```

```
def main(args=None):
    rclpy.init(args=args)
    node = TrackerNode()
    try:
        rclpy.spin(node)
    except KeyboardInterrupt:
        pass
    finally:
        node.stop_robot()
        if rclpy.ok():
            node.destroy_node()
            rclpy.shutdown()

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

รันคำสั่งนี้ทีละบรรทัดในโฟลเดอร์ `~/my_robot_gui` :

Bash
1. สร้างโฟลเดอร์ venv (ทำครั้งเดียว)
python3 -m venv venv

2. เปิดใช้งาน (ต้องทำทุกครั้งที่เปิด Terminal ใหม่เพื่อรันโปรแกรม)
source venv/bin/activate

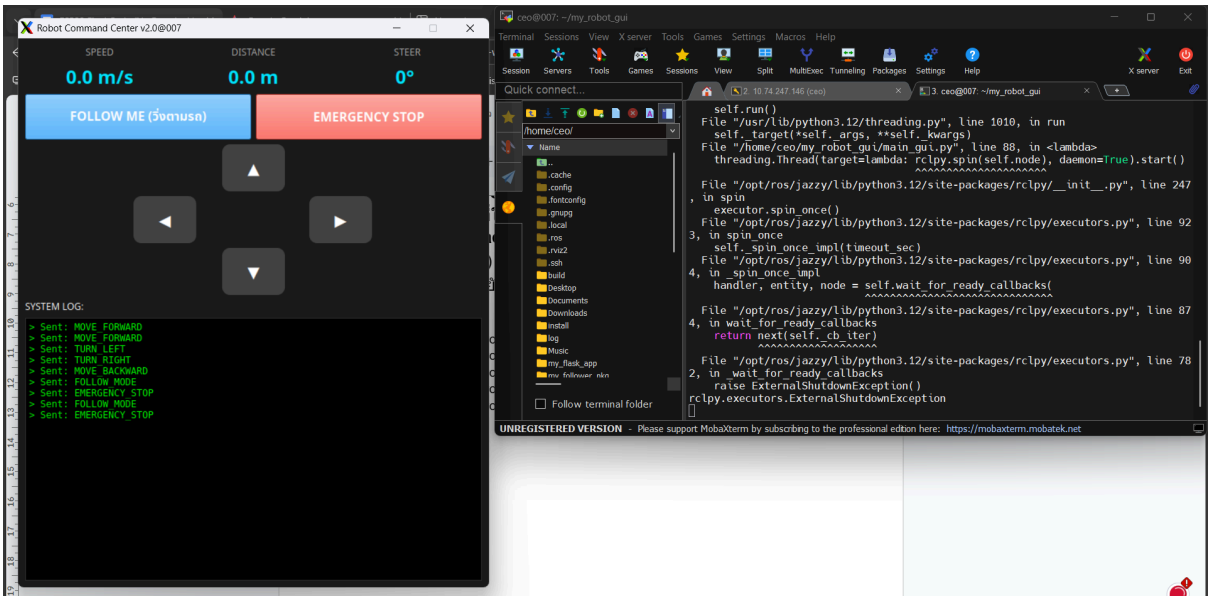
GUI

สิ่งที่ได้เพิ่มมาในเวอร์ชันนี้:

1. **Telemetry Boxes:** ด้านบนสุดจะมีตัวเลขโชว์ค่า **Speed**, **Distance**, และ **Steer Angle** แบบ Real-time
2. **Follow Mode:** เพิ่มปุ่มสีฟ้า "FOLLOW ME" สำหรับส่งคำสั่งวิ่งตามรถ
3. **D-Pad Control:** ปุ่มทิศทางถูกจัดวางให้เล็กลงแต่เป็นระเบียบ เพื่อเหลือพื้นที่ให้ส่วนแสดงผลค่าตัวเลข
4. **Data Feedback:** โค้ดเตรียมส่วนรับค่าจากหุ่นยนต์ไว้แล้ว (Subscribers) ถ้าหุ่นยนต์ส่งค่ามาที่ Topic `robot_speed` เลขในหน้าจอจะขยับทันทีครับ

(venv) ceo@007:~/my_robot_gui$ rm ~/my_robot_gui/main_gui.py //ปิดไฟล์
(venv) ceo@007:~/my_robot_gui$ nano ~/my_robot_gui/main_gui.py //สร้างใหม่
(venv) ceo@007:~/my_robot_gui$ source venv/bin/activate

(venv) ceo@007:~/my_robot_gui$ export DISPLAY=localhost:10.0
(venv) ceo@007:~/my_robot_gui$ python3 main_gui.py //คำสั่งรัน



ใช้โปรแกรมเสริม MobaXterm_Portable_v26.0

```
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import LaserScan
from std_msgs.msg import String, Int32
from rclpy.qos import QoSProfile, ReliabilityPolicy, HistoryPolicy
from flask import Flask, render_template_string, jsonify, request
from flask_cors import CORS
import threading
import serial
import numpy as np
import logging
import sys
import time

# --- 1. Flask Web Server Setup ---
app = Flask(__name__)
CORS(app)

# ปรับ Logging ของ Flask ให้แสดงเฉพาะ Error
log = logging.getLogger('werkzeug')
```

```
log.setLevel(logging.ERROR)

# สถานะหุ่นยนต์
robot_data = {
    "speed": 0,
    "angle": 90,
    "distance": 0.0,
    "mode": "Standby",
    "scan_angle": 40,
    "is_connected": False
}

# เก็บค่าระยะทางย้อนหลังเพื่อทำ Moving Average (ให้นิ่งขึ้น)
dist_history = []

HTML_UI = """
<!DOCTYPE html>
<html>
<head>
    <title>Robot Control Panel</title>
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <script src="https://cdn.tailwindcss.com"></script>
</head>
<body class="bg-slate-950 text-white font-sans min-h-screen">
    <div class="max-w-md mx-auto p-6 space-y-6">
        <div class="text-center">
            <h1 class="text-3xl font-black text-blue-500 italic
tracking-tighter">ROBOT CONTROL</h1>
            <div id="status-dot" class="text-[10px] font-mono mt-1
text-slate-500">● SYSTEM WAITING</div>
        </div>

        <div class="grid grid-cols-2 gap-4">
            <div class="bg-slate-900 border border-slate-800 p-5
rounded-3xl">
                <p class="text-[10px] text-slate-500 uppercase
font-bold">Distance</p>
                <div class="flex items-baseline gap-1">
                    <span id="dist" class="text-4xl font-black
text-emerald-400 font-mono">0.0</span>
                    <span class="text-xs text-slate-600">cm</span>
                </div>
```

```
            </div>
            <div class="bg-slate-900 border border-slate-800 p-5
rounded-3xl">
                <p class="text-[10px] text-slate-500 uppercase
font-bold">Current Speed</p>
                <div class="flex items-baseline gap-1">
                    <span id="spd" class="text-4xl font-black
text-blue-400 font-mono">0</span>
                    <span class="text-xs text-slate-600">PWM</span>
                </div>
            </div>
        </div>

        <div class="bg-slate-900 border border-slate-800 p-6
rounded-3xl space-y-4">
            <div class="flex justify-between items-center">
                <span class="font-bold text-slate-300 text-sm">SCAN
WINDOW</span>
                <span id="angle-display" class="bg-blue-600 px-3 py-1
rounded-full text-xs font-mono font-bold">±40°</span>
            </div>
            <input type="range" min="10" max="90" value="40"
                   oninput="updateScanAngle(this.value)"
                   class="w-full h-2 bg-slate-800 rounded-lg
appearance-none cursor-pointer accent-blue-500">
        </div>

        <div class="space-y-3">
            <button onclick="setMode('Auto')" class="w-full bg-blue-600
hover:bg-blue-500 py-5 rounded-2xl font-black text-xl transition-all
active:scale-95 shadow-lg shadow-blue-900/20">
                START AUTO TRACKING
            </button>
            <button onclick="setMode('Standby')" class="w-full
bg-slate-800 hover:bg-slate-700 py-4 rounded-2xl font-bold
transition-all active:scale-95">
                STOP / STANDBY
            </button>
        </div>

        <button onclick="setMode('E-STOP')" class="w-full border-2
border-red-600/30 text-red-500 py-4 rounded-2xl font-black
```

```
hover:bg-red-600/10 active:bg-red-600 active:text-white
transition-all">
            EMERGENCY STOP
        </button>
    </div>

    <script>
        function update() {
            fetch('/status').then(r => r.json()).then(data => {
                document.getElementById('spd').innerText = data.speed;
                document.getElementById('dist').innerText =
data.distance.toFixed(1);
                const dot = document.getElementById('status-dot');
                dot.innerText = data.is_connected ? "● ESP32 CONNECTED"
: "○ ESP32 DISCONNECTED";
                dot.className = data.is_connected ? "text-[10px]
font-mono mt-1 text-emerald-500" : "text-[10px] font-mono mt-1
text-red-500";
            }).catch(() => {
                document.getElementById('status-dot').innerText = "○
SERVER DOWN";
            });
        }

        function setMode(m) {
            fetch('/command', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify({mode: m})
            });
        }

        function updateScanAngle(val) {
            document.getElementById('angle-display').innerText = "±" +
val + "°";
            fetch('/config', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify({scan_angle: parseInt(val)})
            });
        }

        setInterval(update, 200);
```

```
    </script>
</body>
</html>
"""

@app.route('/')
def home(): return render_template_string(HTML_UI)

@app.route('/status')
def get_status(): return jsonify(robot_data)

@app.route('/command', methods=['POST'])
def post_cmd():
    robot_data['mode'] = request.json.get('mode')
    return jsonify({"status": "ok"})

@app.route('/config', methods=['POST'])
def post_config():
    robot_data['scan_angle'] = request.json.get('scan_angle')
    return jsonify({"status": "ok"})

# --- 2. ROS2 Node Setup ---
class GuiBridgeNode(Node):
    def __init__(self):
        super().__init__('gui_bridge_node')

        qos_profile = QoSProfile(
            reliability=ReliabilityPolicy.BEST_EFFORT,
            history=HistoryPolicy.KEEP_LAST,
            depth=1
        )

        try:
            self.ser = serial.Serial('/dev/ttyUSB1', 115200,
timeout=0.1)
            robot_data['is_connected'] = True
            self.get_logger().info("✅ Serial USB1 Connected")
        except:
            self.ser = None
            robot_data['is_connected'] = False
            self.get_logger().error("❌ Serial USB1 Fail")
```

```
        self.create_subscription(LaserScan, '/scan',
self.lidar_callback, qos_profile)

    def lidar_callback(self, msg):
        window = robot_data.get('scan_angle', 40)
        max_idx = len(msg.ranges)
        indices = list(range(0, min(window, max_idx))) +
list(range(max(0, max_idx - window), max_idx))
        valid_points = [msg.ranges[i] for i in indices if 0.15 <
msg.ranges[i] < 3.0]

        if valid_points:
            raw_dist = np.nanmedian(valid_points) * 100
            dist_history.append(raw_dist)
            if len(dist_history) > 3: dist_history.pop(0)
            dist_cm = sum(dist_history) / len(dist_history)
        else:
            dist_cm = 0.0

        robot_data['distance'] = dist_cm

        if robot_data['mode'] == "Auto":
            if dist_cm < 45: speed = 0
            elif dist_cm > 150: speed = 130
            else:
                speed = int(75 + (dist_cm - 45) * 0.6)
                speed = min(speed, 140)
            self.send_command(speed, 90)
        else:
            self.send_command(0, 90)

    def send_command(self, speed, angle):
        robot_data['speed'] = speed
        if self.ser and self.ser.is_open:
            cmd = f"D,{speed},{angle}\n"
            self.ser.write(cmd.encode())

def ros_thread():
    print("[ROS2] Initializing...")
    if not rclpy.ok():
        rclpy.init()
    node = GuiBridgeNode()
    print("[ROS2] Node Spinning...")
```

```
    rclpy.spin(node)

if __name__ == '__main__':
    print("\n--- Starting Robot Control System ---")

    # 1. เริ่ม ROS ใน Thread แยก
    t = threading.Thread(target=ros_thread, daemon=True)
    t.start()

    # 2. หน่วงเวลาเล็กน้อยให้ ROS เริ่มต้น
    time.sleep(1)

    print("[Web] Starting Server at http://0.0.0.0:5000")
    # 3. รัน Flask (ห้ามใช้ Thread เพื่อให้มันบล็อก Terminal ไว้)
    try:
        app.run(host='0.0.0.0', port=5000, debug=False, threaded=True)
    except KeyboardInterrupt:
        print("\n[System] Exiting...")
        sys.exit(0)
```