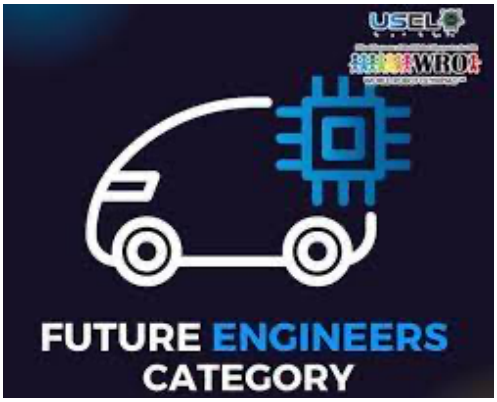
World Robot Olympiad

Ibn Roshd Max



Team members

Hassan alasmari
Yazen mohamed

Team number : 2849

Table of Contents

list of figures

List of tables

Chapter 1 Introduction

1.1 abstract

This project presents the design and development of an autonomous robot tailored for the WRO Future Engineers competition. The robot is engineered to perform complex tasks such as obstacle navigation and item delivery using a combination of sensors, controllers, and intelligent algorithms. The system integrates hardware and software components to achieve real-time decision-making, environmental awareness, and task execution. The report outlines the robot's architecture, coding logic, challenges faced, and future improvements, aiming to contribute to the advancement of robotics in competitive and practical applications.

1.2 introduction

Robotics has become a cornerstone of modern engineering, offering solutions to real-world problems through automation, intelligence, and adaptability. In the context of the World Robot Olympiad (WRO), the Future Engineers category challenges participants to build robots that simulate autonomous vehicles and smart systems. This project focuses on developing a robot capable of navigating both open and obstacle-filled environments, performing tasks with minimal human intervention. The robot combines mechanical design, sensor integration, and algorithmic control to meet competition requirements and explore broader applications in autonomous systems.

1.3 Problem Statement

Traditional robots often struggle with dynamic environments, especially when faced with unpredictable obstacles or complex delivery tasks. In competitive settings like WRO, the challenge lies in building a robot that can:

- Navigate autonomously through a maze or open field
- Detect and avoid obstacles in real time
- Identify and reach target locations accurately
- Operate reliably under varying conditions

This project addresses these challenges by designing a robot that integrates multiple sensors, intelligent path planning, and modular hardware to ensure robust performance.

1.4 Objectives

- **Design and Build a Modular Robot Chassis** Create a stable and adaptable physical structure that supports all required components and allows for future upgrades.

- **Integrate Sensor Systems for Environmental Awareness** Use ultrasonic, infrared, IMU, and GPS sensors to detect obstacles, track movement, and determine location.

- **Implement Real-Time Navigation Algorithms** Develop code that enables the robot to calculate paths, avoid obstacles, and reach target destinations autonomously.

- **Enable Task Execution in Open Challenge** Program the robot to identify delivery points, drop items accurately, and return to base.

- **Achieve Reliable Maze Navigation in Obstacle Challenge** Ensure the robot can detect walls, choose optimal paths, and exit the maze without manual intervention.

- **Provide User Interaction and Feedback Mechanisms** Include displays, indicators, and control interfaces to allow users to monitor and interact with the robot effectively.

- **Test and Evaluate Performance Under Competition Conditions** Conduct trials to assess reliability, accuracy, and responsiveness in simulated challenge environments.

1.4 Advantages and Disadvantages

- **Autonomous Navigation:** The robot can operate without manual control, reducing human error and increasing efficiency.

- **Sensor Fusion:** Combines data from ultrasonic, infrared, IMU, and GPS for accurate environmental awareness.

- **Modular Design:** Allows easy upgrades and maintenance of individual components.

- **Real-Time Decision Making:** Enables quick responses to obstacles and dynamic changes in the environment.

- **Scalability:** The system can be adapted for larger platforms or more complex tasks beyond the competition.

Disadvantages:

- **Limited Processing Power:** Jetson Nano may struggle with high-resolution image processing under time constraints.

- **Battery Life Constraints:** Continuous operation drains power quickly, requiring frequent recharging or battery swaps.

- **Sensor Sensitivity:** Ultrasonic and IR sensors can be affected by lighting, surface texture, and noise.

- **Environmental Dependency:** Performance may vary based on floor material, lighting conditions, and obstacle types.

- **Complex Calibration:** Requires precise tuning of sensors and motors for optimal performance,

Chapter 2 project hardware description

2.1 Table of components

Item N0.	Item	Used Quantity
1	Jetson nano	1
2	Ultrasonic sensor	4
3	Mbu6050	1
4	On/off switch	1
5	3.7 Lithium battery	4
6	Voltage regulator	1
7	Jumper wires	3
8	Servo motor	1
9	Dc motor	1
10	3d printed parts	600 gram

Table 1 components

2.2 Main Controller Unit

- **Jetson Nano Developer Kit** The Jetson Nano serves as the robot's central processing unit, handling image recognition, path planning, and high-level decision-making. Its GPU acceleration enables real-time object detection and AI-based navigation.
- **Arduino Nano** Used for low-level control tasks such as motor driving and sensor data acquisition. It communicates with the Jetson Nano via serial connection, ensuring modularity and responsiveness.

2.3 Power System

- **Lithium Polymer Battery (11.1V, 2200mAh)** Powers the entire robot, including motors and controllers. Chosen for its high energy density and lightweight design.
- **Voltage Regulators (5V and 3.3V)** Ensure stable power delivery to sensitive components like sensors and microcontrollers.
- **Power Distribution Board** Manages current flow across modules and protects against overloads.

2.4 Locomotion System

- **DC Gear Motors (12V, 100 RPM)** Provide torque and speed suitable for both open terrain and obstacle navigation.
- **Motor Driver (L298N Dual H-Bridge)** Controls motor direction and speed via PWM signals from the Arduino Nano.
- **Differential Drive Mechanism** Enables precise turning and maneuverability, especially in tight spaces.
- **Wheels (Rubberized, 65mm Diameter)** Offer traction and stability across various surfaces.

2.5 Sensor Suite

- **Ultrasonic Sensors (HC-SR04)** Used for obstacle detection and distance measurement. Placed at the front and sides for 180° coverage.
- **Infrared Sensors (TCRT5000)** Assist in line following and edge detection during maze navigation.
- **IMU Sensor (MPU-6050)** Provides orientation data (pitch, roll, yaw) for path correction and stabilization.
- **GPS Module (Neo-6M)** Used in the Open Challenge to determine global position and assist in route planning.
- **Camera Module (Raspberry Pi HQ Camera)** Mounted on the Jetson Nano for visual recognition of delivery points and obstacles.

2.6 Feedback and Display

- **OLED Display (0.96", I2C)** Shows real-time status, sensor readings, and debug information.
- **LED Indicators** Provide visual feedback for system states such as obstacle detection, delivery confirmation, and battery status.
- **Buzzer Module** Emits sound alerts during critical events (e.g., collision warning or task completion).

2.7 Chassis and Structural Components

- **Acrylic Base Plate (3mm)** Lightweight and durable platform for mounting components.
- **Aluminum Brackets and Standoffs** Used to secure sensors and maintain structural integrity.
- **3D Printed Mounts** Custom-designed holders for camera, sensors, and battery pack to optimize layout and accessibility.

2.8 Connectivity and Expansion

- **Jumper Wires and Breadboards** Used for prototyping and flexible connections between modules.
- **Serial Communication (UART)** Enables data exchange between Jetson Nano and Arduino Nano.
- **I2C and SPI Buses** Used for connecting sensors and displays with minimal wiring complexity.

2.9 Safety and Maintenance Features

- **Circuit Protection Fuses** Prevent damage from overcurrent or short circuits.
- **Modular Design** Allows easy replacement and upgrading of individual components without affecting the entire system.

Chapter 3 – Methodology

3.1 block diagram

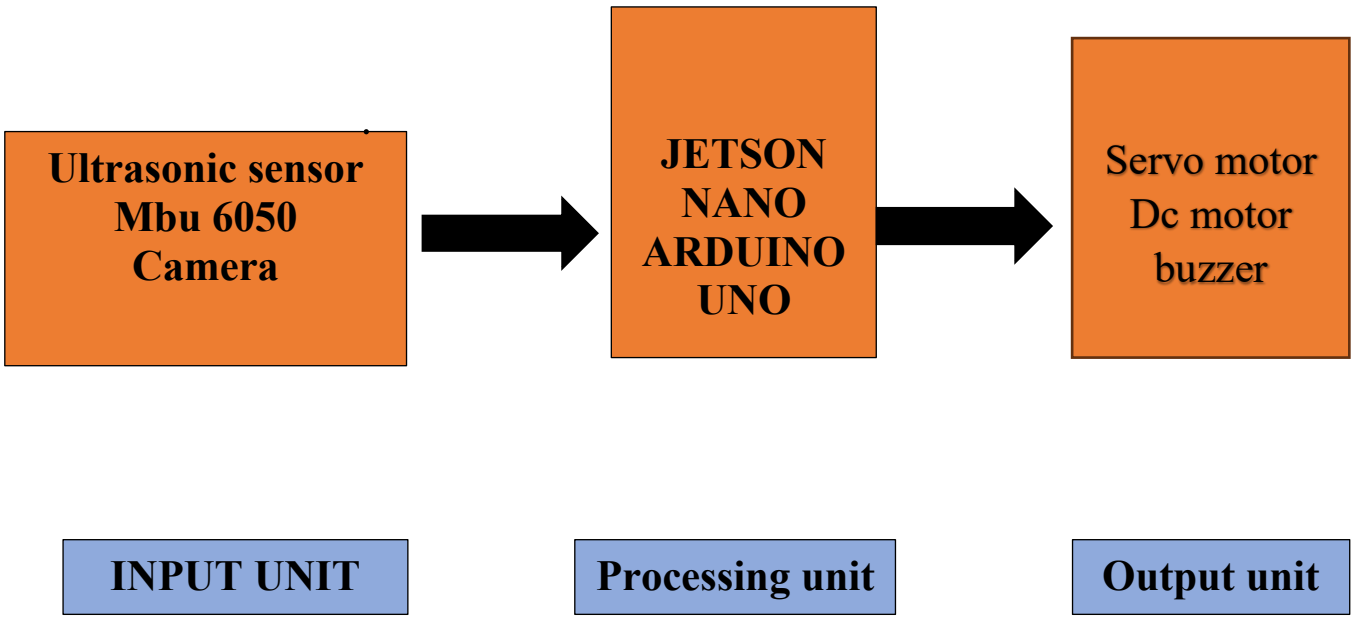


Figure 1 Block diagram of the proposed system

Flow chart

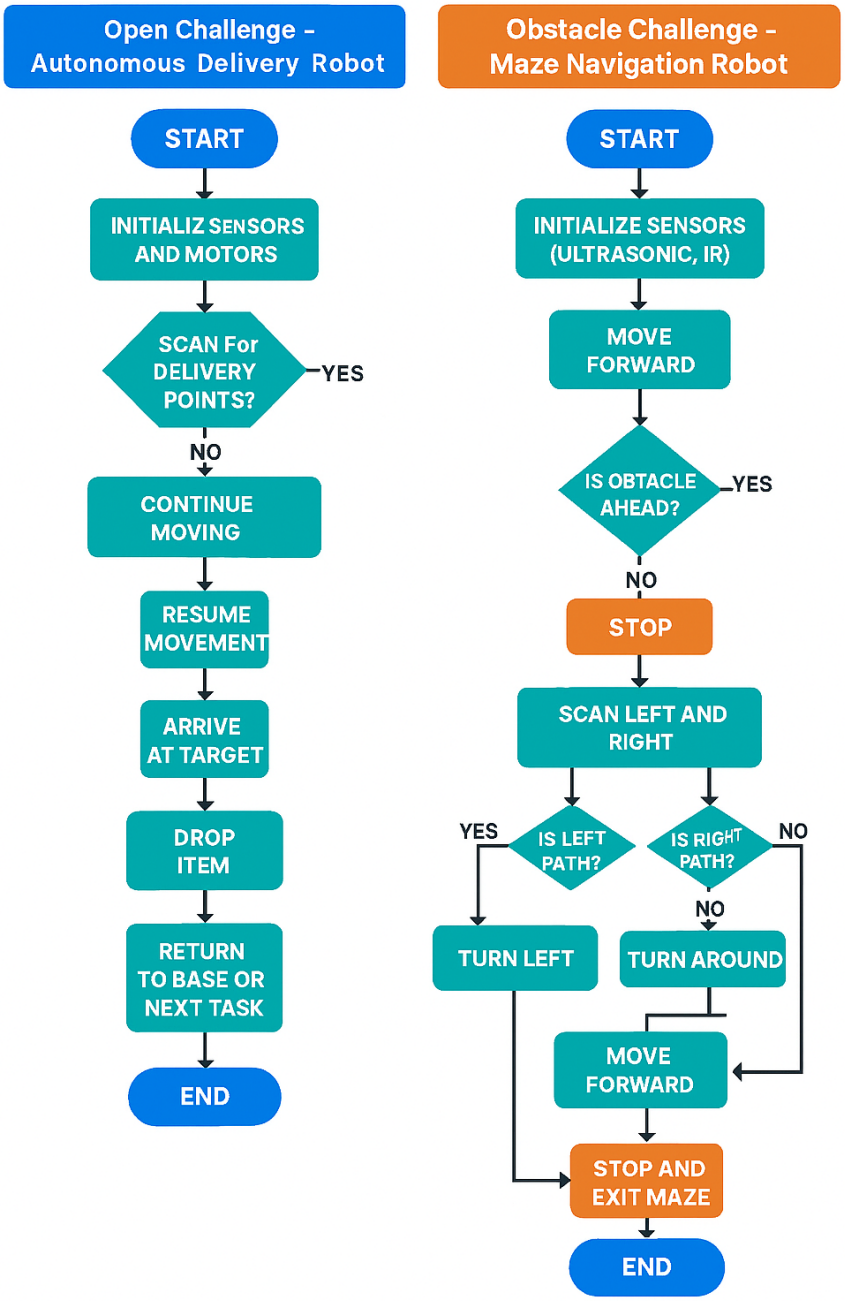


Figure 2 flow chart

3.2 weekly working progress

Week	work
Week1	Distributing the work team to achieve a common union of interests, efforts, cooperation, and using the team members' skills, talents, training, and experiences.
Week 2	Select the project and obtain approval from the responsible engineer.
Week 3	Determine the scope of the project and determine the roles of each individual in the work team.
Week 4	Search on Google Scholar for similar project ideas and collect the necessary information.
Week 5	Buy components
Week 6	Receive components and see hoe to connect it then make electric circuit diagram
Week 7	Start project programing
Week 8	Test the project
Week 9	Solve the problems then final test and make the project rebort
Week 10	Report discussion

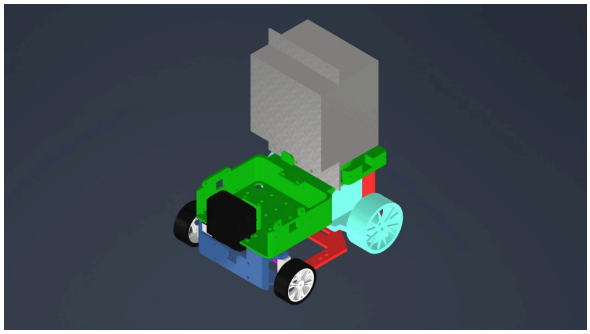
Table 2 working plan

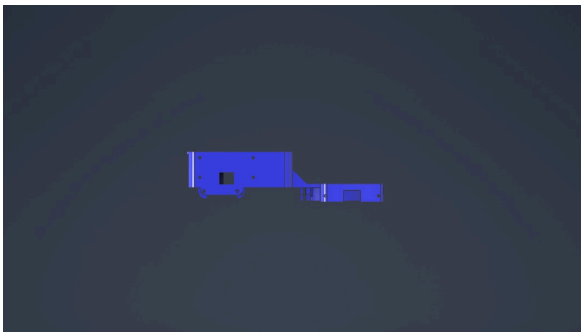
3.3 Manufacturing steps

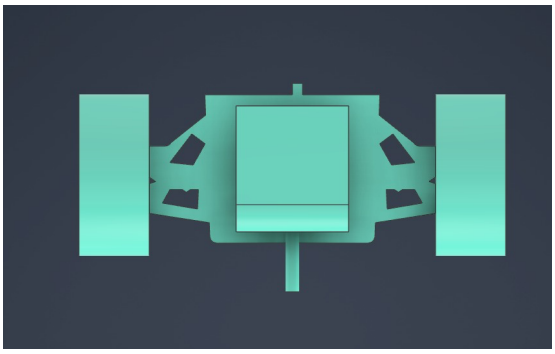
week	work
Week 1	Buy components
Week 2	Draw circuit diagram for the peoject
Week 3	Receive components and see how to connect
Week 4	3d print and wood working to make the project body
Week 5	Connect the components together
Week 6	Start programming
Week 7	Test project
Week 8	The project lost
Week 9	Buying ll parts again and connect and program it
Week 10	Test the project , solve the problems , final test and make the project rebort

Table 3 manufacturing steps

3.3.1 design

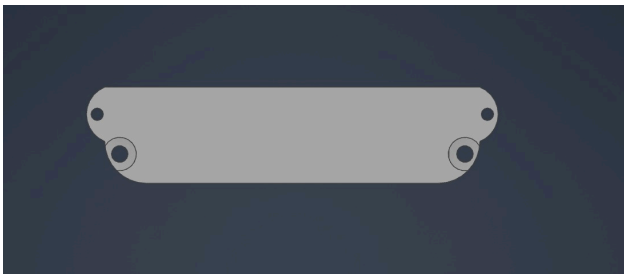






wheels design.

Back base design



3.3.2 3d printing

Figure 3 printe parts

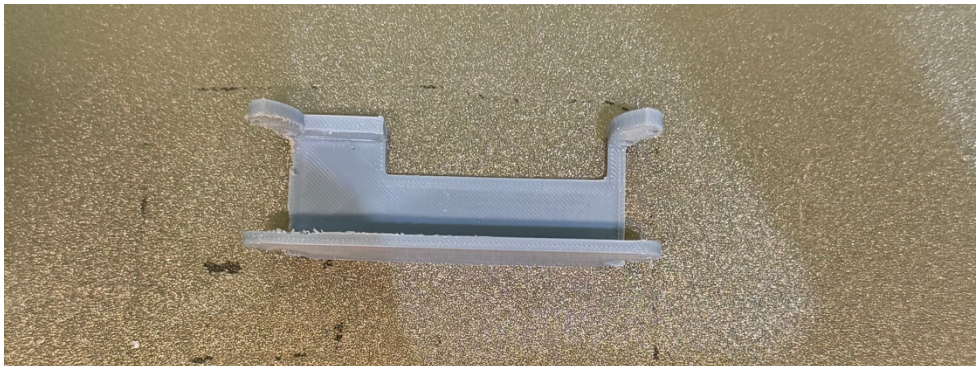


Figure 43d printing

3.3.3 Assembling parts

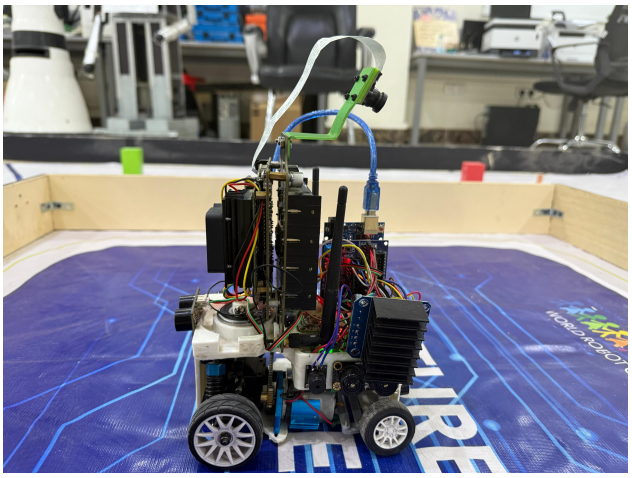


Figure 5 assembling wires

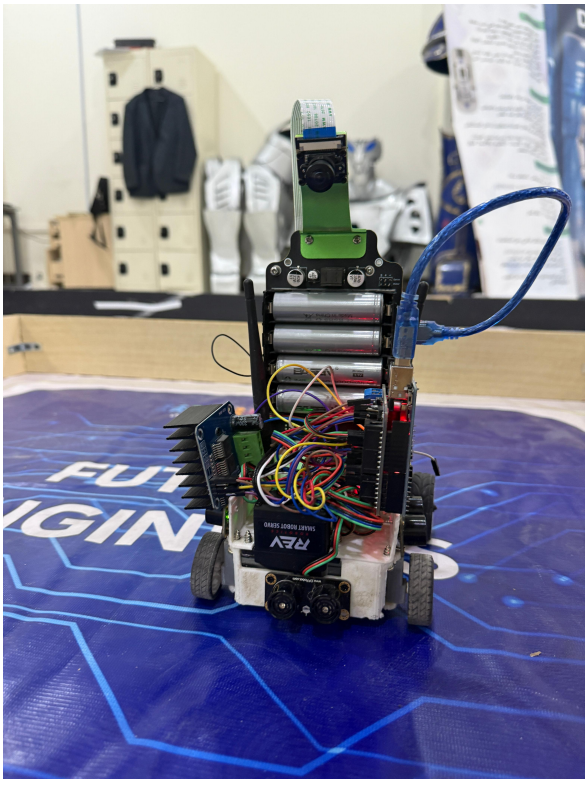


Figure 6 testing

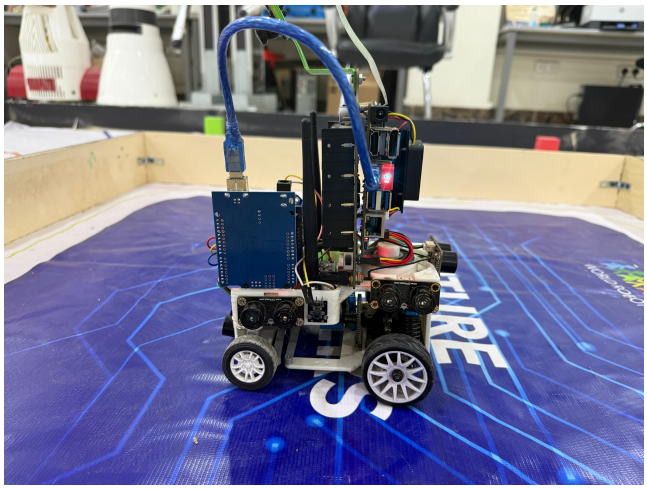


Figure 7 final shape

3.4 Circuit diagram

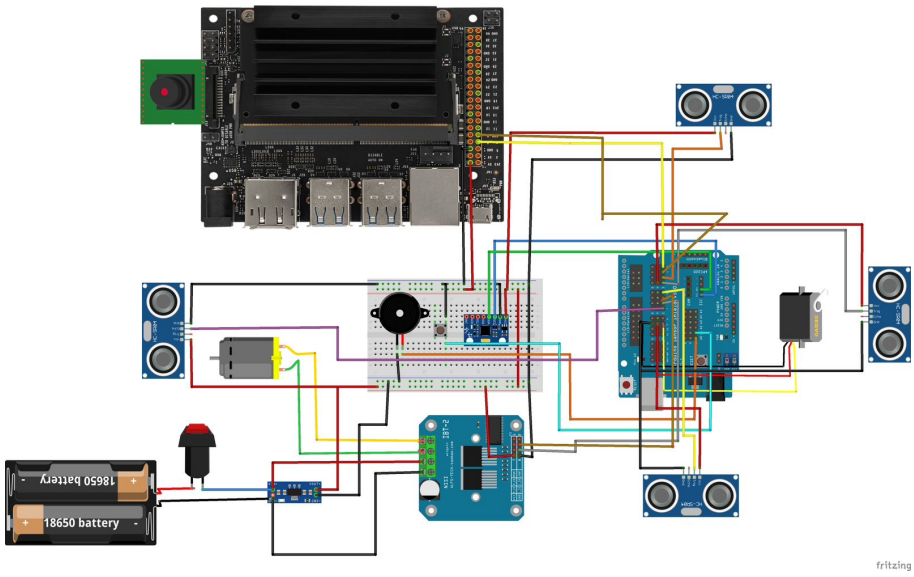


Figure 8 circuit diagram

schemes :

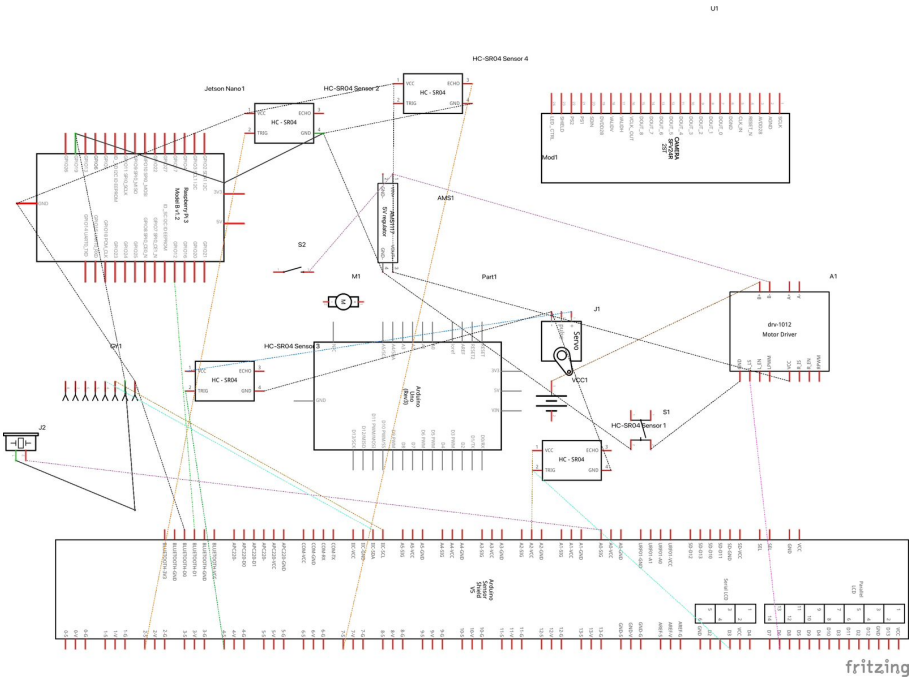


Figure 9sheme

image Processing and HSV Tuning in the Robot

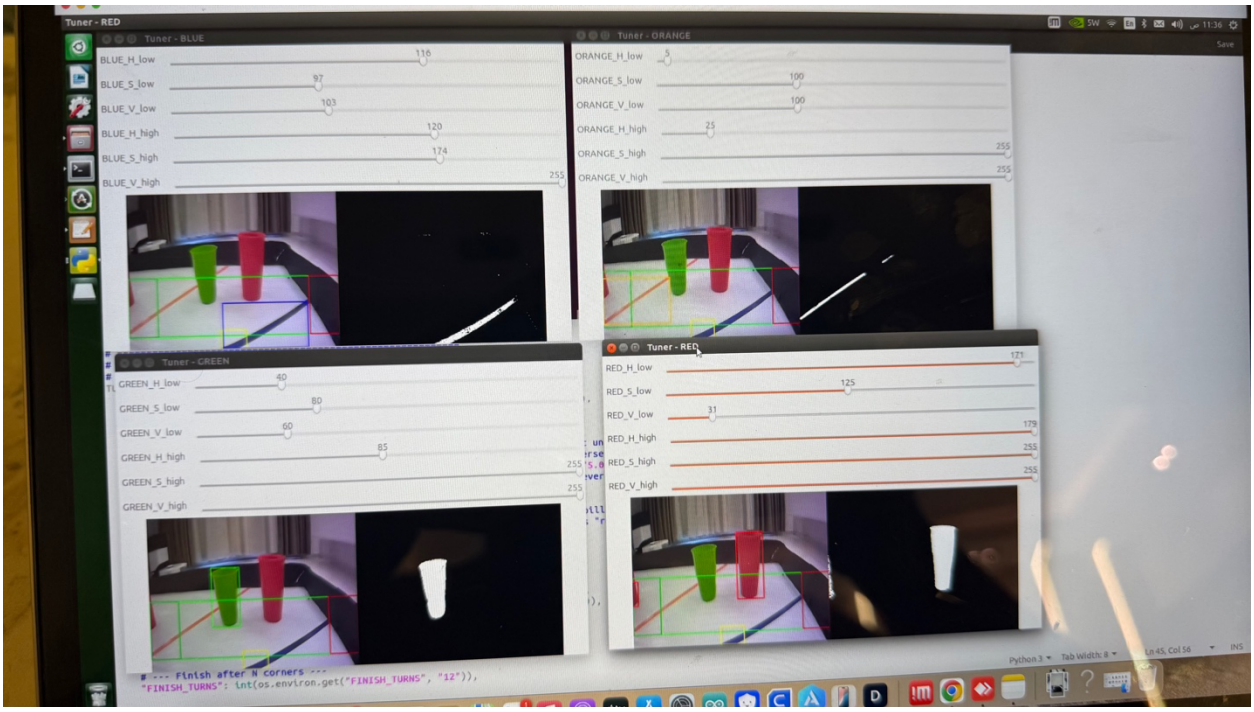


Figure 10 hsv tunning for pillars and orange and blue lines

Why HSV?

In robotics vision systems, **HSV (Hue, Saturation, Value)** color space is preferred over RGB because it separates color information (hue) from lighting intensity (value). This makes it more robust in varying lighting conditions—critical for competition environments where shadows, reflections, and brightness can fluctuate.

Pillar Detection: Red and Green

The robot uses its camera to detect **red and green pillars**, which serve as key visual markers in the environment. Here's how the detection process works:

1. **Live Camera Feed Acquisition** The Jetson Nano captures frames from the camera in real time.
2. **Conversion to HSV Color Space** Each frame is converted from BGR (default OpenCV format) to HSV using:

   python

   ```
   hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
   ```

3. **HSV Thresholding for Red and Green**

- Red is tricky because it wraps around the hue spectrum (low and high hue values).
- Green has a more stable hue range.

Example thresholds:

yaml

```
red_lower1: [0, 100, 100]
red_upper1: [10, 255, 255]
red_lower2: [160, 100, 100]
red_upper2: [180, 255, 255]

green_lower: [40, 70, 70]
green_upper: [80, 255, 255]
```

4. **Masking and Contour Detection** The robot applies masks to isolate red and green regions, then uses contour detection to locate pillar shapes and estimate their position and size.

🟧🔵 Line Detection: Orange and Blue

For path guidance, the robot relies on **orange and blue lines** on the ground. These are detected using similar HSV techniques:

1. **HSV Thresholds for Orange and Blue** These colors are tuned to avoid confusion with ambient colors or floor textures.

   Example:

   yaml

   ```
   orange_lower: [10, 100, 100]
   orange_upper: [25, 255, 255]

   blue_lower: [100, 150, 0]
   blue_upper: [140, 255, 255]
   ```

2. **Perspective Correction (Optional)** If the camera is angled, a perspective transform may be applied to flatten the view for better line detection.
3. **Line Tracking Logic** The robot uses the centroid of the detected line to adjust its heading. If the line curves or splits, the robot decides direction based on line continuity and position.

HSV Tuning Program and YAML Integration

To simplify calibration, the team developed a **custom HSV tuning GUI** that allows users to adjust HSV ranges interactively:

- **Sliders for Hue, Saturation, and Value** Users can drag sliders to fine-tune color detection while viewing live feedback from the camera.
- **Save to YAML File** Once optimal values are found, the program saves them to a structured `.yaml` file:

  yaml

  ```
  red_lower1: [0, 100, 100]
  red_upper1: [10, 255, 255]
  ...
  ```

- **Main Code Integration** The robot's main vision script loads these values at runtime:

  python

  ```
  import yaml
  with open("hsv_config.yaml", "r") as file:
      hsv_values = yaml.safe_load(file)
  ```

  This allows the robot to adapt to different lighting conditions or competition setups without modifying the core code.

Overall Image Processing Pipeline

1. **Capture Frame**
2. **Convert to HSV**
3. **Apply HSV Masks**
4. **Detect Contours or Lines**
5. **Calculate Centroids / Bounding Boxes**
6. **Make Navigation Decisions**
7. **Send Commands to Motor Controller**

Benefits of This System

- **Modularity:** HSV values can be tuned without touching the main code.
- **Adaptability:** Works across different venues and lighting setups.
- **Efficiency:** Real-time detection with minimal computational load.
- **Scalability:** Can be extended to detect more colors or shapes.

Region of Interest (ROI) in Image Processing

What Is ROI?

Region of Interest (ROI) refers to a specific portion of an image that is selected for focused analysis. Instead of processing the entire frame—which can be computationally expensive and noisy—ROI allows the robot to concentrate only on the relevant area where visual targets (like lines or pillars) are expected to appear.

How ROI Is Used in Your Robot

In your robot's vision system, ROI is applied to optimize detection of:

- **Red and Green Pillars** (used for navigation or task triggers)
- **Orange and Blue Lines** (used for path guidance)

By defining ROIs, the robot reduces false positives and speeds up processing.

ROI Implementation Workflow

1. **Capture Full Frame from Camera**

   python

   ```
   frame = camera.read()
   ```

2. **Define ROI Boundaries** For example, to focus on the lower half of the image where floor lines appear:

   python

   ```
   roi = frame[height//2 : height, 0 : width]
   ```

3. **Convert ROI to HSV**

   python

   ```
   hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
   ```

4. **Apply HSV Masking** Use the tuned HSV values (loaded from the YAML file) to isolate colors:

   python

```
mask = cv2.inRange(hsv_roi, hsv_lower, hsv_upper)
```

5. **Contour or Line Detection** Analyze the masked ROI to find shapes, centroids, or line paths.

ROI + HSV Tuning + YAML Integration

Your system smartly combines ROI with HSV tuning:

- **HSV Tuning GUI** allows users to interactively adjust color thresholds.
- **YAML File** stores these thresholds for red, green, orange, and blue.
- **Main Code** loads HSV values and applies them only within the ROI, improving speed and accuracy.

This modular approach means:

- You can tune HSV values for specific ROIs (e.g., top for pillars, bottom for lines).
- You avoid processing irrelevant parts of the image (e.g., ceiling, background).
- You reduce computational load on Jetson Nano, keeping frame rates high.

Benefits of Using ROI

- **Performance Boost:** Less data to process means faster frame rates.
- **Noise Reduction:** Eliminates distractions from irrelevant areas.
- **Targeted Detection:** Improves accuracy for color-based object recognition.
- **Modularity:** Different ROIs can be defined for different tasks (e.g., top for pillars, bottom for lines).

Example Use Case

- **Obstacle Challenge:** ROI is set to the lower center of the frame to detect blue lines guiding the robot through the maze.
- **Open Challenge:** ROI is set to the upper portion to detect red/green pillars indicating delivery zones.

Chapter 4: result and conclusion

4.1 results

- **Functional Robot Design:** The team successfully developed a robot capable of autonomous navigation and task execution in both open and obstacle-based environments.

- **Sensor Integration:** Sensors such as ultrasonic, infrared, and GPS/IMU were effectively integrated to detect obstacles, track movement, and guide decision-making.

- **Software Implementation:** The robot's logic was implemented using structured code that allowed for real-time decision-making, path recalculation, and task completion.

- **Challenge Performance:**

    - In the **Open Challenge**, the robot was able to identify delivery points, calculate paths, avoid obstacles, and complete delivery tasks.
    - In the **Obstacle Challenge**, the robot demonstrated reliable maze navigation, obstacle avoidance, and exit detection.

- **Team Collaboration:** The report highlights strong teamwork, iterative testing, and problem-solving throughout the development process.

4.2 Challenges and Future Improvement

1. **Sensor Calibration Issues**
    - Ultrasonic sensors occasionally gave inconsistent readings due to environmental noise and surface reflections.
    - IMU drift affected yaw accuracy, requiring frequent resets and filtering.
2. **Serial Communication Delays**
    - Data exchange between Jetson Nano and Arduino Nano sometimes experienced latency, impacting real-time responsiveness.
3. **Power Management**
    - Voltage fluctuations from the battery affected sensor stability and motor performance, especially during extended runs.
4. **Obstacle Detection in Tight Spaces**
    - In the Obstacle Challenge, detecting narrow gaps and sharp corners was difficult, requiring manual tuning of detection thresholds.
5. **Camera Processing Bottlenecks**
    - Real-time image processing on Jetson Nano was limited by frame rate and lighting conditions, affecting cube recognition accuracy.
6. **Mechanical Constraints**
    - The differential mechanism improved turning but introduced complexity in balancing torque across wheels.

Future Improvements

1. **Advanced Sensor Fusion**
    - Integrate LiDAR or depth cameras for more reliable obstacle detection and mapping.
2. **Improved IMU Handling**
    - Use sensor fusion algorithms (e.g., Kalman filters) to reduce drift and improve orientation tracking.
3. **Optimized Communication Protocols**
    - Implement buffered or interrupt-driven serial communication to reduce latency between controllers.
4. **AI-Based Path Planning**
    - Introduce machine learning models to predict optimal paths and adapt to dynamic environments.
5. **Power Regulation Circuitry**
    - Add dedicated voltage regulators and capacitors to stabilize power delivery to sensitive components.
6. **Modular Chassis Design**
    - Redesign the robot's body to allow easier upgrades and component swaps without full reassembly.
7. **Enhanced Vision System**

    - Upgrade to higher-resolution cameras with better low-light performance and faster frame rates.
8. **Remote Monitoring Dashboard**
    - Develop a web-based interface to monitor telemetry, sensor data, and robot status in real time.

4.3 User Interaction

- **Graphical User Interface (GUI):**
    - The robot was controlled and monitored using a touchscreen interface.
    - Real-time data such as sensor readings and movement status were displayed clearly.
- **Manual Override:**
    - In case of unexpected behavior, users could manually intervene using a joystick or keyboard input.
- **Visual Feedback:**
    - LED indicators and screen messages provided status updates (e.g., obstacle detected, delivery complete).
- **Voice Prompts (Optional Module):**
    - A prototype module allowed the robot to give verbal cues to users, enhancing accessibility.

4.4 conclusion

The WRO Future Engineers project showcases how robotics and programming can be applied to solve real-world challenges in autonomous navigation. Through careful design, sensor integration, and logical coding, the team created a robot that performs complex tasks with precision and adaptability.

This project not only met the competition requirements but also laid a foundation for future innovations in smart robotics. The experience gained in hardware-software integration, environmental sensing, and algorithmic control will serve as a valuable stepping stone for the team's continued growth in engineering and technology.

References

1. CICSA WRO 2025 – Future Engineers GitHub Repository *CICSA-NET/WRO2025-CICSA* https://github.com/CICSA-NET/WRO2025-CICSA → Contains code, schematics, and documentation used by participating teams.
2. WRO® 2025 Future Engineers – Self-Driving Cars General Rules *World Robot Olympiad Association* https://wro-association.org/wp-content/uploads/WRO-2025-Future-Engineers-Self-Driving-Cars-General-Rules.pdf → Official competition rules and technical specifications.
3. Arduino IDE Documentation *Arduino.cc* https://www.arduino.cc/en/software → Reference for programming microcontrollers and sensor integration.
4. NVIDIA Jetson Nano Developer Guide *NVIDIA Developer* https://developer.nvidia.com/embedded/jetson-nano-developer-kit → Used for image processing and AI-based navigation.
5. RemoteXY Platform Documentation https://remotexy.com/en/help/ → Used for GUI development and real-time data visualization.
6. Tuya Smart Platform https://developer.tuya.com/en → Used for IoT-based control of camera and cleaning modules.
7. Fritzing Circuit Design Tool https://fritzing.org/home/ → Used for designing and documenting electronic schematics.

appendix

Code

open challenge code (paython)

```
# !/usr/bin/env python3
# -- coding: utf-8 --

import cv2 as cv
import numpy as np
import yaml, glob, time, os, re, math, sys
from collections import deque

try:
    import serial
except Exception:
    serial = None

# ============================================================
# ================== TUNING PANEL ============================
# ============================================================
TUNE = {
    # --- Corridor turns (BLUE/ORANGE) ---
    "TURN_DEG": float(os.environ.get("TURN_DEG", "90.0")),
```

```
"TURN_TOL_DEG": float(os.environ.get("TURN_TOL_DEG", "10.0")),

# --- Pillar sidestep turns (RED/GREEN) ---
"PILLAR_TURN_DEG": float(os.environ.get("PILLAR_TURN_DEG", "90.0")),
"PILLAR_TURN_TOL_DEG": float(os.environ.get("PILLAR_TURN_TOL_DEG", "10.0")),

# --- Flip (first pillar after 7th corner) ---
"FLIP_TURN_DEG": float(os.environ.get("FLIP_TURN_DEG", "180.0")),     # ~180; tune if needed
"FLIP_FWD_HOLD_S": float(os.environ.get("FLIP_FWD_HOLD_S", "1.50")),   # forward-centering hold before flip
"FLIP_LEFT_SIGN": float(os.environ.get("FLIP_LEFT_SIGN", "1.0")),      # +1: LEFT = +angle; use -1 if your yaw is opposite

# --- Turn watchdogs (safety net only) ---
"CORNER_TURN_MAX_S": float(os.environ.get("CORNER_TURN_MAX_S", "2.3")),
"PILLAR_TURN_MAX_S": float(os.environ.get("PILLAR_TURN_MAX_S", "2.0")),

# --- Run cap ---
"MAX_TURNS": int(os.environ.get("MAX_TURNS", "12")),

# --- Servo geometry (deg) ---
# center=90, LEFT=120, RIGHT=65 (forward)
"CENTER_DEG": int(os.environ.get("CENTER_DEG", "90")),
"LEFT_LIMIT": int(os.environ.get("LEFT_LIMIT", "65")),     # numeric min (right-most)
"RIGHT_LIMIT": int(os.environ.get("RIGHT_LIMIT", "120")),  # numeric max (left-most)

# --- General speeds (PWM 0..255) ---
"DRIVE_SPEED": int(os.environ.get("DRIVE_SPEED", "26")),
"SEE_COLOR_SPEED": int(os.environ.get("SEE_COLOR_SPEED", "22")),
"TURN_SPEED": int(os.environ.get("TURN_SPEED", "28")),

# --- Pillar pass speeds ---
"PILLAR_TURN_SPEED": int(os.environ.get("PILLAR_TURN_SPEED", "25")),
"PILLAR_FWD_SPEED": int(os.environ.get("PILLAR_FWD_SPEED", "26")),
"PILLAR_BACK_SPEED": int(os.environ.get("PILLAR_BACK_SPEED", "26")),

# --- Corner (BLUE/ORANGE) backward-turn fixed servo angles ---
"BACK_LEFT_SERVO_DEG": int(os.environ.get("BACK_LEFT_SERVO_DEG", "65")),    # BLUE -> left while reversing
"BACK_RIGHT_SERVO_DEG": int(os.environ.get("BACK_RIGHT_SERVO_DEG", "120")), # ORANGE -> right while reversing

# --- Distances (cm) ---
"FRONT_TURN_CM": float(os.environ.get("FRONT_TURN_CM", "25.0")),
"BACK_STOP_CM": float(os.environ.get("BACK_STOP_CM", "30.0")),
"PILLAR_ACCEPT_CM": float(os.environ.get("PILLAR_ACCEPT_CM", "50.0")),
"PILLAR_APPROACH_FRONT_CM": float(os.environ.get("PILLAR_APPROACH_FRONT_CM", "6.0")),
"PILLAR_BACK_TO_FRONT_CM": float(os.environ.get("PILLAR_BACK_TO_FRONT_CM", "13.0")),

# --- Color detection / ROI (BLUE/ORANGE lines) ---
"COLOR_CONFIRM_FRAMES": int(os.environ.get("COLOR_CONFIRM_FRAMES", "2")),
"MIN_PIX_COLOR": int(os.environ.get("MIN_PIX_COLOR", "50")),
"BLUR_KSIZE": int(os.environ.get("BLUR_KSIZE", "3")),
"COLOR_ROI_WIDTH_FRAC": float(os.environ.get("COLOR_ROI_WIDTH_FRAC", "0.28")),
"COLOR_ROI_HEIGHT_FRAC": float(os.environ.get("COLOR_ROI_HEIGHT_FRAC", "0.20")),
"ROI_SCALE_AFTER_FIRST_TURN": float(os.environ.get("ROI_SCALE_AFTER_FIRST_TURN", "1.3")),
"SAT_BOOST": float(os.environ.get("SAT_BOOST", "1.10")),
"COLOR_SUSPEND_SEC": float(os.environ.get("COLOR_SUSPEND_SEC", "4.0")),

# --- Pillar detection (RED/GREEN) ---
"PILLAR_CONFIRM_FRAMES": int(os.environ.get("PILLAR_CONFIRM_FRAMES", "2")),
"PILLAR_MIN_PIX": int(os.environ.get("PILLAR_MIN_PIX", "80")),
"PILLAR_ROI_WIDTH_FRAC": float(os.environ.get("PILLAR_ROI_WIDTH_FRAC", "0.60")),
"PILLAR_ROI_HEIGHT_FRAC": float(os.environ.get("PILLAR_ROI_HEIGHT_FRAC", "0.55")),
"PILLAR_DIST_K": float(os.environ.get("PILLAR_DIST_K", "4000.0")),

# --- Pillar gates (full-frame) ---
"RED_LINE_X_FRAC": float(os.environ.get("RED_LINE_X_FRAC", "0.150")),
"GREEN_LINE_X_FRAC": float(os.environ.get("GREEN_LINE_X_FRAC", "0.850")),

# --- Yaw centering ---
"STEER_SIGN": float(os.environ.get("STEER_SIGN", "1.0")),             # if steering flips after flip, set -1
"STEER_KP_DEG_PER_DEG": float(os.environ.get("STEER_KP_DEG_PER_DEG", ".835")),
"STEER_MAX_DEG": float(os.environ.get("STEER_MAX_DEG", "8.0")),
```

```
    "STEER_DEAD_ERR_DEG": float(os.environ.get("STEER_DEAD_ERR_DEG", "0.01")),
    "ERR_FILTER_ALPHA": float(os.environ.get("ERR_FILTER_ALPHA", "0.75")),
    "STEER_SLEW_DEG": int(os.environ.get("STEER_SLEW_DEG", "20")),
    "CENTER_STEER_MIN": 65, "CENTER_STEER_MAX": 105,

    # --- Post-flip centering softener (helps stabilize after ±180) ---
    "POST_FLIP_KP_GAIN": float(os.environ.get("POST_FLIP_KP_GAIN", "0.95")),  # scales STEER_KP for a short time
    "POST_FLIP_MAX_DEG": float(os.environ.get("POST_FLIP_MAX_DEG", "8.0")),   # caps steering during soft phase
    "POST_FLIP_KP_HOLD_S": float(os.environ.get("POST_FLIP_KP_HOLD_S", "1.5")),

    # --- Wall slowdowns ---
    "WALL_SLOW_SIDE_ON_CM": float(os.environ.get("WALL_SLOW_SIDE_ON_CM", "7.5")),
    "WALL_SLOW_SIDE_OFF_CM": float(os.environ.get("WALL_SLOW_SIDE_OFF_CM", "8.5")),
    "SIDE_SLOWDOWN_FACTOR": float(os.environ.get("SIDE_SLOWDOWN_FACTOR", "0.55")),
    "WALL_SLOW_FRONT_ON_CM": float(os.environ.get("WALL_SLOW_FRONT_ON_CM", "45.0")),
    "FRONT_SLOWDOWN_FACTOR": float(os.environ.get("FRONT_SLOWDOWN_FACTOR", "0.55")),
    "WALL_SLOW_MIN_SPEED": int(os.environ.get("WALL_SLOW_MIN_SPEED", "21")),

    # --- Wall avoid ---
    "WALL_AVOID_ON_CM": float(os.environ.get("WALL_AVOID_ON_CM", "7.5")),
    "WALL_AVOID_OFF_CM": float(os.environ.get("WALL_AVOID_OFF_CM", "8.5")),
    "WALL_AVOID_HOLD_S": float(os.environ.get("WALL_AVOID_HOLD_S", "0.35")),
    "WALL_AVOID_KP_DEG_PER_CM": float(os.environ.get("WALL_AVOID_KP_DEG_PER_CM", "1.8")),
    "WALL_AVOID_EXP": float(os.environ.get("WALL_AVOID_EXP", "1.0")),
    "WALL_AVOID_MIN_DEG": float(os.environ.get("WALL_AVOID_MIN_DEG", "2.0")),
    "WALL_AVOID_MAX_BIAS_DEG": float(os.environ.get("WALL_AVOID_MAX_BIAS_DEG", "16.0")),
    "WALL_AVOID_GAIN": float(os.environ.get("WALL_AVOID_GAIN", "1.0")),
    "AVOID_INVERT_DIR": int(os.environ.get("AVOID_INVERT_DIR", "1")),   # keep your rig's inversion
    "WALL_AVOID_SPEED": int(os.environ.get("WALL_AVOID_SPEED", "24")),

    # --- Camera / serial ---
    "ALLOW_NO_CAMERA": int(os.environ.get("ALLOW_NO_CAMERA", "0")),
    "FRAME_W": int(os.environ.get("FRAME_W", "640")),
    "FRAME_H": int(os.environ.get("FRAME_H", "480")),
    "ROBOT_PORT": os.environ.get("ROBOT_PORT", "/dev/ttyUSB0"),
    "ROBOT_BAUD": int(os.environ.get("ROBOT_BAUD", "115200")),

    # --- Link watchdogs ---
    "STALE_TLM_SEC": float(os.environ.get("STALE_TLM_SEC", "1.2")),
    "STALE_LINK_SEC": float(os.environ.get("STALE_LINK_SEC", "2.5")),
    "PING_PERIOD_SEC": float(os.environ.get("PING_PERIOD_SEC", "0.7")),
    "RESET_BACKOFF_SEC": float(os.environ.get("RESET_BACKOFF_SEC", "4.0")),

    # --- Misc ---
    "US_MAX_CM": float(os.environ.get("US_MAX_CM", "300.0")),
}

# ================ CONSTANTS / HELPERS ================
MIN_SPEED   = 0
CENTER_DEG  = TUNE["CENTER_DEG"]
LEFT_LIMIT  = TUNE["LEFT_LIMIT"]
RIGHT_LIMIT = TUNE["RIGHT_LIMIT"]
SPAN_LEFT   = abs(CENTER_DEG - LEFT_LIMIT)
SPAN_RIGHT  = abs(RIGHT_LIMIT - CENTER_DEG)

def gstreamer_pipeline(sensor_id=0, capture_width=1280, capture_height=720,
                       display_width=640, display_height=480, framerate=30, flip_method=0):
    return (f"nvarguscamerasrc sensor-id={sensor_id} ! "
            f"video/x-raw(memory:NVMM), width={capture_width}, height={capture_height}, "
            f"format=(string)NV12, framerate={framerate}/1 ! "
            f"nvvidconv flip-method={flip_method} ! "
            f"video/x-raw, width={display_width}, height={display_height}, format=(string)BGRx ! "
            "videoconvert ! video/x-raw, format=(string)BGR ! appsink drop=1")

def open_camera():
    w, h = TUNE["FRAME_W"], TUNE["FRAME_H"]
    cap = None
    try:
        cap = cv.VideoCapture(gstreamer_pipeline(display_width=w, display_height=h), cv.CAP_GSTREAMER)
    except Exception:
```

```
        cap = None
    if not (cap and cap.isOpened()):
        try: cap = cv.VideoCapture(0, cv.CAP_V4L2)
        except Exception: cap = None
    if cap and cap.isOpened(): return cap, False
    if TUNE["ALLOW_NO_CAMERA"]:
        print("WARNING: camera open failed; using dummy frames.")
        class DummyCap:
            def isOpened(self): return True
            def read(self):
                frame = np.zeros((h, w, 3), dtype=np.uint8)
                cv.putText(frame, "NO CAMERA", (10,30),
                           cv.FONT_HERSHEY_SIMPLEX, 0.9, (0,0,255), 2)
                return True, frame
            def release(self): pass
        return DummyCap(), True
    print("ERROR: camera open failed."); return None, False

def load_latest_yaml():
    override = os.environ.get("VISION_CFG", "").strip()
    if override and os.path.isfile(override): path = override
    else:
        cands = sorted(glob.glob("config/vision_*.yaml"))
        if not cands: raise FileNotFoundError("No config in ./config — need RED/GREEN/BLUE/ORANGE ranges.")
        path = cands[-1]
    with open(path, "r") as f: return yaml.safe_load(f), path

def wrap180(a):
    while a>180: a-=360
    while a<-180: a+=360
    return a

def shortest_err(target_deg, now_deg): return wrap180(target_deg - now_deg)

def deg_to_norm(angle_deg):
    angle_deg = float(max(LEFT_LIMIT, min(RIGHT_LIMIT, angle_deg)))
    if angle_deg >= CENTER_DEG:
        return (angle_deg - CENTER_DEG) / float(max(1.0, SPAN_RIGHT))
    else:
        return (angle_deg - CENTER_DEG) / float(max(1.0, SPAN_LEFT))

def slew_limit(prev, target, max_delta):
    if prev is None: return target
    if target > prev + max_delta: return prev + max_delta
    if target < prev - max_delta: return prev - max_delta
    return target

def mask_hsv_ranges(hsv, ranges):
    if isinstance(ranges, dict) and "low" in ranges and "high" in ranges:
        low, high = np.array(ranges["low"], np.uint8), np.array(ranges["high"], np.uint8)
        return cv.inRange(hsv, low, high)
    mask = None
    for r in (ranges if isinstance(ranges, list) else []):
        if "low" in r and "high" in r:
            m = cv.inRange(hsv, np.array(r["low"], np.uint8), np.array(r["high"], np.uint8))
            mask = m if mask is None else cv.bitwise_or(mask, m)
    if mask is None: mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
    return mask

# ==================== MAIN ====================
def main():
    cfg, cfg_path = load_latest_yaml()
    print("Loaded vision config:", cfg_path)

    # ---- Serial ----
    ser = None; device = TUNE["ROBOT_PORT"]; baud = TUNE["ROBOT_BAUD"]
    last_tlm_t = 0.0; last_pong_t = 0.0; last_ping_t = 0.0; last_reset_t = -1e9
    rx_buf = b""
    tlm_re = re.compile(
        r"^TLM,.*yaw=([\-0-9\.]+),state=([0-9]),steer=([\-0-9\.]+),speed=([0-9]+),dF=([\-0-9]+),dL=([\-0-9]+),dR=([\-0-9]+),dB=([\-0-9]+)"
```

```
)

def open_serial():
    nonlocal ser
    if serial is None: return
    try:
        ser = serial.Serial(device, baudrate=baud, timeout=0.01, write_timeout=0.2, dsrdtr=False)
        time.sleep(0.30)
        ser.reset_input_buffer(); ser.reset_output_buffer()
        print("Serial OK on", device)
    except Exception as e:
        print("WARNING: serial not open ->", e); ser = None

def safe_write(line_bytes):
    nonlocal ser
    if serial is None: return False
    if ser is None:
        open_serial()
        if ser is None: return False
    try:
        ser.write(line_bytes); return True
    except Exception as e:
        print("Serial write error:", e)
        try: ser.close()
        except Exception: pass
        ser=None
        open_serial()
        if ser:
            try: ser.write(line_bytes); return True
            except Exception as e2: print("Write retry failed:", e2)
    return False

def reset_arduino():
    nonlocal ser, last_reset_t
    now = time.time()
    if now - last_reset_t < TUNE["RESET_BACKOFF_SEC"]: return
    last_reset_t = now
    try:
        if ser and ser.is_open: ser.close()
    except Exception: pass
    time.sleep(0.2)
    open_serial()
    print("[RESET] (basic reopen)")

def link_alive(now): return (now - max(last_tlm_t, last_pong_t)) < TUNE["STALE_LINK_SEC"]

def maybe_ping(now):
    nonlocal last_ping_t
    if (now - last_ping_t) >= TUNE["PING_PERIOD_SEC"]:
        safe_write(b"PING\n")
        last_ping_t = now

# --- Telemetry ---
last_yaw = 0.0
last_US = {"dF":-1, "dL":-1, "dR":-1, "dB":-1}
us_ema = {"dF":-1.0, "dL":-1.0, "dR":-1.0, "dB":-1.0}
arduino_armed = False

def sane_cm(x):
    if x is None or x < 0: return -1
    return float(min(x, TUNE["US_MAX_CM"]))

def read_tlm():
    nonlocal rx_buf, last_yaw, last_tlm_t, last_US, us_ema, last_pong_t, arduino_armed, ser
    if ser is None: return
    try:
        data = ser.read(256)
        if not data: return
        rx_buf += data
        while b"\n" in rx_buf:
```

```
            line, rx_buf = rx_buf.split(b"\n", 1)
            s = line.decode("utf-8", "ignore").strip()
            if s == "PONG": last_pong_t = time.time(); continue
            if s == "READY": last_pong_t = time.time(); continue
            if s == "ARMED": arduino_armed = True; last_pong_t = time.time(); continue
            m = tlm_re.match(s)
            if m:
                last_yaw = float(m.group(1)); last_tlm_t = time.time()
                vals = [int(m.group(i)) for i in (5,6,7,8)]
                for (k, v) in zip(("dF","dL","dR","dB"), vals):
                    v = float(v)
                    a = 0.35
                    if v < 0: last_US[k] = -1
                    else:
                        if us_ema[k] < 0: us_ema[k] = v
                        else: us_ema[k] = (1-a)*us_ema[k] + a*v
                        last_US[k] = int(us_ema[k])
        except Exception as e:
            print("Serial read error:", e)
            try: ser.close()
            except Exception: pass
            open_serial()

    # ---- Camera ----
    cap, using_dummy = open_camera()
    if cap is None: return

    # ============= State & corridor model =============
    STATE_WAIT_START, STATE_RUN, STATE_CORNER_TURNING, STATE_CORNER_BACKCENTER, STATE_FLIP_WAIT,
STATE_FLIP_TURN, STATE_DONE = range(7)
    state = STATE_WAIT_START

    start_yaw = 0.0
    corridor_idx = 0
    orientation_offset_deg = 0.0

    yaw_ref = 0.0
    def set_ref_to_corridor():
        nonlocal yaw_ref
        yaw_ref = wrap180(start_yaw + corridor_idx * TUNE["TURN_DEG"] + orientation_offset_deg)

    # Corner book-keeping
    yaw_at_corner_start = None
    intended_corner_yaw = None
    corner_turn_start_t = -1.0
    turn_count = 0
    target_idx = 0

    # NEW: latch the global corner direction from the first corner we see.
    # +1 => RIGHT (CW), -1 => LEFT (CCW)
    corner_dir_latch = None

    # Inversion + flip scheduling
    invert_last4 = False
    waiting_first_pillar_after_turn7 = False

    # Flip micro-FSM
    flip_scheduled_color = None  # RED/GREEN of the first pillar after 7th corner
    flip_wait_until = -1.0
    rotate_target = None
    rotate_start_t = -1.0
    post_flip_soft_until = -1.0   # for gentle centering immediately after flip

    # Corner detection control
    detect_enabled = True
    color_pause_until = -1.0
    color_q = deque(maxlen=TUNE["COLOR_CONFIRM_FRAMES"])
    see_color_slow = False
    corner_color = None
```

```
# Pillar detection control
pillar_q = deque(maxlen=TUNE["PILLAR_CONFIRM_FRAMES"])
pillar_active = False
pillar_color = None
pillar_state = "IDLE" # IDLE, TURN, FWD, BACK, RETURN
pillar_turn_target = None
pillar_turn_start_t = -1.0
corridor_yaw_before_pillar = None

# Pillar memory / recording
first_pillar_after_turn = {1:None, 2:None, 3:None, 4:None}
need_record_after_turn = False
last_pillar_before_turn4 = None

# Steering control
err_ema = 0.0
last_steer_cmd = None
last_tx = 0.0

# Wall avoid FSM
avoid_active = False
avoid_side = None
avoid_until = -1.0

# HUD window
cv.namedWindow("robot", cv.WINDOW_AUTOSIZE)

# --- helpers to send ---
def send_center_abs_deg(steer_deg, speed_pwm):
    steer_deg = int(max(LEFT_LIMIT, min(RIGHT_LIMIT, int(steer_deg))))
    steer_norm = deg_to_norm(steer_deg)
    spd = max(MIN_SPEED, min(255, int(speed_pwm)))
    safe_write(f"CENTER,{steer_norm:.3f},{spd}\n".encode("ascii"))

def send_center_deg(steer_deg, speed_pwm):
    send_center_abs_deg(steer_deg, speed_pwm)

def send_back(speed_pwm):
    spd = max(0, min(255, int(speed_pwm)))
```

obstacle challenge code (Arduino)

```
# #include <Wire.h>
#include "DFRobot_BNO055.h"
#include <Servo.h>

/* ================== Pins ================== */
#define BUZZER_PIN A0
#define START_BTN  A1

// BTS7960 pins
#define RPWM 5
#define LPWM 6
#define R_EN 7
#define L_EN 8
```

```
// Servo
#define SERVO_PIN 9

// Ultrasonic (single-pin trig+echo)
#define US_R_PIN  2
#define US_L_PIN  3
#define US_F_PIN  4
#define US_B_PIN  10   // rear ultrasonic

/* ===== Buzzer Master Switch ===== */
#define BUZZ_ENABLE 1   // 0=OFF, 1=ON

/* ============ Servo geometry ============
 * Servo: center=90, left max=120, right max=65
 * Forward: 120 = LEFT, 65 = RIGHT
 * Backward: driving direction flips the steering effect (handled here)
 */
int   SERVO_CENTER = 90;   // change via SET_CENTER,<deg>
#define SERVO_MIN    65
#define SERVO_MAX    120
const int SERVO_DIR = -1;  // +1=normal, -1=reversed (keep -1 for your servo)

float SPAN_LEFT  = 90 - 65;    // updated in syncCenterSpans()
float SPAN_RIGHT = 120 - 90;
float STEER_TRIM_NORM = 0.0f;  // + = steer RIGHT

static void syncCenterSpans(){
  SPAN_LEFT  = (float)(SERVO_CENTER - SERVO_MIN);
  SPAN_RIGHT = (float)(SERVO_MAX    - SERVO_CENTER);
}

/* ===== Ultrasonic timing ===== */
#define VELOCITY_TEMP(tempC) ((331.5 + 0.6 * (float)(tempC)) * 100 / 1000000.0)  // cm/us
int16_t us_tempC = 20;
unsigned long US_TIMEOUT = 120000UL;  // µs
long US_FAR_CM = 999;

/* ================== IMU ================== */
typedef DFRobot_BNO055_IIC BNO;
BNO bno(&Wire, 0x28);
unsigned long lastImuOkMs = 0;
int imuBadBurst = 0;
const unsigned long IMU_STALE_REINIT_MS = 400;
const int IMU_BAD_LIMIT = 3;
static float wrap180(float a){ while(a>180)a-=360; while(a<-180)a+=360; return a; }

/* ================ State ================ */
enum State { IDLE=0, CENTERING=1, TURNING=2 };
State state = IDLE;
bool armed = false;

/* ===== Controls & smoothing ===== */
Servo myservo;
float current_steer_norm = 0.0f;
int   current_speed      = 0;      // 0..255
int   current_dir        = 0;      // +1 forward, -1 reverse, 0 stop
float steer_f = 0.0f, speed_f = 0.0f;
const float ALPHA_STEER = 0.18f, ALPHA_SPEED = 0.35f;

/* ============ Yaw / Turn control ============ */
float yaw_deg = 0.0f;
float target_yaw_deg = 0.0f;
```

```
int   turn_drive_dir = +1;      // +1 forward, -1 reverse
float Kp = 3.0f, Ki = 0.0f, Kd = 0.26f;
float pid_i = 0, prev_err = 0, d_filt = 0;
const float D_ALPHA = 0.35f;
unsigned long lastPIDms = 0;
const float    TURN_DONE_TOL_DEG = 2.5f;
const uint32_t TURN_MAX_MS       = 2500;
unsigned long  turnStartMs       = 0;

/* ======== Speed clamps ======== */
const int SPEED_MIN_CLAMP = 0;
const int SPEED_MAX_CLAMP = 255;

/* ======== Turn speed default ======== */
int TURN_PWM_DEFAULT = 22;
int turn_pwm_override = -1;

/* ====== Buzzer helpers ====== */
inline void beep(uint16_t ms=80){
#if BUZZ_ENABLE
  tone(BUZZER_PIN, 3000, ms);
#else
  (void)ms;
#endif
}

bool buzzerRapid = false;
unsigned long buzzLastMs = 0;
const uint16_t BUZZ_PERIOD_MS = 80;
const uint16_t BUZZ_PULSE_MS  = 40;

void chirpRapidTick(){
#if BUZZ_ENABLE
  if(!buzzerRapid) return;
  unsigned long now = millis();
  if(now - buzzLastMs >= BUZZ_PERIOD_MS){
    buzzLastMs = now;
    tone(BUZZER_PIN, 3500, BUZZ_PULSE_MS);
  }
#endif
}

/* ============ Motor / Servo helpers ============ */
void bts_init(){
  pinMode(RPWM, OUTPUT); pinMode(LPWM, OUTPUT);
  pinMode(R_EN, OUTPUT); pinMode(L_EN, OUTPUT);
  digitalWrite(R_EN, HIGH); digitalWrite(L_EN, HIGH);
  analogWrite(RPWM, 0); analogWrite(LPWM, 0);
}
void setDrive(int pwm, int dir){
  pwm = constrain(pwm, SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);
  if(dir > 0){ analogWrite(RPWM, pwm); analogWrite(LPWM, 0); }
  else if(dir < 0){ analogWrite(RPWM, 0); analogWrite(LPWM, pwm); }
  else { analogWrite(RPWM, 0); analogWrite(LPWM, 0); }
}
void stopMotor(){ analogWrite(RPWM,0); analogWrite(LPWM,0); }

void setSteerNorm(float u_in){
  float u = SERVO_DIR * (u_in + STEER_TRIM_NORM);
  if (fabs(u) < 0.02f) u = 0.0f;
  u = constrain(u, -1.0f, 1.0f);
  float angle = (u >= 0.0f) ? (SERVO_CENTER + u*SPAN_RIGHT) : (SERVO_CENTER + u*SPAN_LEFT);
```

```
  angle = constrain(angle, (float)SERVO_MIN, (float)SERVO_MAX);
  myservo.write((int)angle);
}
void steerTo(int deg){ deg = constrain(deg, SERVO_MIN, SERVO_MAX); myservo.write(deg); }

/* ============ IMU ============ */
bool initIMU(){
  bno.setOprMode(BNO::eOprModeConfig); delay(25);
  if(bno.begin() != BNO::eStatusOK) return false;
  bno.setOprMode(BNO::eOprModeNdof); delay(30);
  lastImuOkMs = millis(); imuBadBurst = 0; return true;
}
bool reinitIMU(){
  bno.setOprMode(BNO::eOprModeConfig); delay(25);
  bool ok = (bno.begin() == BNO::eStatusOK);
  if(ok){
    bno.setOprMode(BNO::eOprModeNdof); delay(30);
    lastImuOkMs=millis(); imuBadBurst=0;
    Serial.println("INFO,IMU_REINIT_OK");
  } else {
    Serial.println("WARN,IMU_REINIT_FAIL");
  }
  return ok;
}
void readYaw(){
  BNO::sEulAnalog_t e = bno.getEul();
  unsigned long now = millis();
  bool ok = (bno.lastOperateStatus == BNO::eStatusOK);
  bool looksZeroFrame = (fabs(e.head) < 0.001f && fabs(e.roll) < 0.001f && fabs(e.pitch) < 0.001f);
  if(ok && !looksZeroFrame){
    float y = wrap180(e.head);
    if(!isnan(y)){
      yaw_deg = y;
      lastImuOkMs = now;
      imuBadBurst = 0;
    } else {
      imuBadBurst++;
    }
  } else {
    imuBadBurst++;
  }
  if(imuBadBurst >= IMU_BAD_LIMIT || (now - lastImuOkMs) > IMU_STALE_REINIT_MS) reinitIMU();
}

/* ========== Ultrasonic ========== */
volatile long distF=-1, distL=-1, distR=-1, distB=-1;

inline long usPingOnePin(int pin){
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);  delayMicroseconds(2);
  digitalWrite(pin, HIGH); delayMicroseconds(10);
  digitalWrite(pin, LOW);
  pinMode(pin, INPUT);
  unsigned long pw = pulseIn(pin, HIGH, US_TIMEOUT);
  if(pw == 0) return US_FAR_CM;
  float cm = pw * VELOCITY_TEMP(us_tempC) / 2.0f;
  return (long)cm;
}
inline long readFrontMin2(){
  long a = usPingOnePin(US_F_PIN);
  long b = usPingOnePin(US_F_PIN);
  if(a<0) return b; if(b<0) return a; return (a<b)?a:b;
```

```
}
uint8_t us_phase = 0;
void readUSStaggered(){
  switch(us_phase){
    case 0: distF = readFrontMin2(); break;
    case 1: distL = usPingOnePin(US_L_PIN); break;
    case 2: distR = usPingOnePin(US_R_PIN); break;
    case 3: distB = usPingOnePin(US_B_PIN); break;
  }
  us_phase = (us_phase + 1) & 3;
}

/* ============ Turn controller ============ */
static float shortestErr(float target, float now){ return wrap180(target - now); }
void runTurnController(){
  if(state != TURNING) return;
  int turn_pwm = (turn_pwm_override >= 0) ? turn_pwm_override : TURN_PWM_DEFAULT;
  float err = shortestErr(target_yaw_deg, yaw_deg);

  if (fabs(err) <= TURN_DONE_TOL_DEG || (millis()-turnStartMs) > TURN_MAX_MS){
    stopMotor(); steerTo(SERVO_CENTER);
    state = IDLE; pid_i=0; prev_err=0; d_filt=0; lastPIDms=0; turn_pwm_override=-1; turn_drive_dir=+1;
#if BUZZ_ENABLE
    buzzerRapid=false; noTone(BUZZER_PIN);
#endif
    Serial.println("TURN_DONE");
    beep(60);
    return;
  }

  if (fabs(err) < 1.0f) err = 0.0f;
  unsigned long now = millis();
  float dt = (lastPIDms==0)? 0.02f : (now - lastPIDms)/1000.0f;
  lastPIDms = now;

  pid_i += err * dt;
  float d = (err - prev_err) / (dt > 1e-3f ? dt : 1e-3f);
  d_filt = (D_ALPHA * d) + (1.0f - D_ALPHA) * d_filt;
  prev_err = err;

  float u = Kp*err + Ki*pid_i + Kd*d_filt;  // degrees
  float u_norm = constrain(u/45.0f, -1.2f, 1.2f);

  // Flip steering when reversing so nose still rotates toward target.
  float steer_cmd = (turn_drive_dir >= 0) ? u_norm : -u_norm;

  setSteerNorm(steer_cmd);
  setDrive(turn_pwm, turn_drive_dir);
#if BUZZ_ENABLE
  buzzerRapid = true;
#endif
}

/* ============ Serial protocol ============
 * START
 * CENTER,<norm>,<speed>           (forward)
 * BACK,<speed>                    (reverse; keep last steer)
 * BACKC,<norm>,<speed>            (reverse with steering)
 * TURN_ABS,<target_yaw>[,<pwm>[,REV]]
 * STEER_DEG,<deg>                 absolute servo degree (65..120)
 * TRIM_NORM,<value>
 * SET_CENTER,<deg>
```

```
 * SET_TURN_PWM,<pwm>
 * CLEAR_TURN_PWM
 * SET_US_TIMEOUT,<micros>
 * SET_US_FAR_CM,<cm>
 * STOP
 * PING -> PONG
 */
String rx;

void setArmed(bool on){
  if(on && !armed){
    armed = true;
    beep(100);
    Serial.println("ARMED");
  } else if(!on && armed){
    armed = false;
  }
}

void handleLine(const String &line){
  if(line.equalsIgnoreCase("PING")){ Serial.println("PONG"); return; }
  if(line.equalsIgnoreCase("START") || line.equalsIgnoreCase("S")){ setArmed(true); return; }
  if(!armed){
    if(line.startsWith("SET_US_TIMEOUT")){
      int c1=line.indexOf(','); if(c1>0){ US_TIMEOUT=(unsigned long)line.substring(c1+1).toInt(); beep(40); }
    } else if(line.startsWith("SET_US_FAR_CM")){
      int c1=line.indexOf(','); if(c1>0){ US_FAR_CM=line.substring(c1+1).toInt(); beep(40); }
    }
    return;
  }

  if(line.startsWith("CENTER")){
    int c1=line.indexOf(','), c2=line.indexOf(',',c1+1);
    if(c1>0 && c2>c1){
      float u=line.substring(c1+1,c2).toFloat();
      int spd=line.substring(c2+1).toInt();
      current_steer_norm = constrain(u,-1.0f,1.0f);
      current_speed = constrain(spd,SPEED_MIN_CLAMP,SPEED_MAX_CLAMP);
      current_dir = (current_speed>0)? +1:0;
      if(state!=TURNING) state=CENTERING;
#if BUZZ_ENABLE
      buzzerRapid=false; noTone(BUZZER_PIN);
#endif
    }
  }
  else if(line.startsWith("BACKC")){
    int c1=line.indexOf(','), c2=line.indexOf(',',c1+1);
    if(c1>0 && c2>c1){
      float u=line.substring(c1+1,c2).toFloat();
      int spd=line.substring(c2+1).toInt();
      current_steer_norm = constrain(u,-1.0f,1.0f);
      current_speed = constrain(spd,SPEED_MIN_CLAMP,SPEED_MAX_CLAMP);
      current_dir = (current_speed>0)? -1:0;
      if(state!=TURNING) state=CENTERING;
#if BUZZ_ENABLE
      buzzerRapid=false; noTone(BUZZER_PIN);
#endif
    }
  }
  else if(line.startsWith("BACK")){
    int c1=line.indexOf(','); int spd=0;
    if(c1>0){ spd=line.substring(c1+1).toInt(); spd=constrain(spd,SPEED_MIN_CLAMP,SPEED_MAX_CLAMP); }
```

```
    current_speed = spd; current_dir = (spd>0)? -1:0;
    if(state!=TURNING) state=CENTERING;
#if BUZZ_ENABLE
    buzzerRapid=false; noTone(BUZZER_PIN);
#endif
  }
  else if(line.startsWith("TURN_ABS")){
    int c1=line.indexOf(','); if(c1>0){
      int c2=line.indexOf(',',c1+1);
      target_yaw_deg = wrap180(line.substring(c1+1,(c2>0?c2:line.length())).toFloat());
      turn_pwm_override=-1; turn_drive_dir=+1;
      if(c2>0){
        String tail=line.substring(c2+1); tail.trim();
        int ct=tail.indexOf(',');
        if(ct<0){
          if(tail.equalsIgnoreCase("REV")) turn_drive_dir=-1;
          else turn_pwm_override=tail.toInt();
        }else{
          String p1=tail.substring(0,ct), p2=tail.substring(ct+1);
          turn_pwm_override=p1.toInt();
          if(p2.equalsIgnoreCase("REV")) turn_drive_dir=-1;
        }
      }
      state=TURNING; pid_i=0; prev_err=0; d_filt=0; lastPIDms=0; turnStartMs=millis();
#if BUZZ_ENABLE
      buzzerRapid=true; buzzLastMs=0; beep(120);
#endif
    }
  }
  else if(line.startsWith("STEER_DEG")){   // absolute servo set (65..120)
    int c1=line.indexOf(',');
    if(c1>0){
      int d=line.substring(c1+1).toInt();
      steerTo(d);
    }
  }
  else if(line.startsWith("TRIM_NORM")){
    int c1=line.indexOf(','); if(c1>0){
      STEER_TRIM_NORM=line.substring(c1+1).toFloat();
      STEER_TRIM_NORM=constrain(STEER_TRIM_NORM,-0.3f,0.3f);
      beep(40);
    }
  }
  else if(line.startsWith("SET_CENTER")){
    int c1=line.indexOf(','); if(c1>0){
      SERVO_CENTER=constrain(line.substring(c1+1).toInt(),SERVO_MIN,SERVO_MAX);
      syncCenterSpans(); steerTo(SERVO_CENTER); beep(80);
    }
  }
  else if(line.startsWith("SET_TURN_PWM")){
    int c1=line.indexOf(','); if(c1>0){
      TURN_PWM_DEFAULT=constrain(line.substring(c1+1).toInt(),SPEED_MIN_CLAMP,SPEED_MAX_CLAMP);
      beep(60);
    }
  }
  else if(line.startsWith("CLEAR_TURN_PWM")){
    turn_pwm_override=-1; beep(40);
  }
  else if(line.startsWith("SET_US_TIMEOUT")){
    int c1=line.indexOf(','); if(c1>0){ US_TIMEOUT=(unsigned long)line.substring(c1+1).toInt(); beep(40); }
  }
  else if(line.startsWith("SET_US_FAR_CM")){
```

```
    int c1=line.indexOf(','); if(c1>0){ US_FAR_CM=line.substring(c1+1).toInt(); beep(40); }
  }
  else if(line.startsWith("STOP")){
    state=IDLE; current_speed=0; current_dir=0; setDrive(0,0); setSteerNorm(0.0f);
#if BUZZ_ENABLE
    buzzerRapid=false; noTone(BUZZER_PIN);
#endif
  }
}

/* ============ Telemetry (~60 Hz) ============ */
void sendTelemetry(){
  static unsigned long last = 0;
  unsigned long now = millis();
  if(now - last >= 16){
    last = now;
    Serial.print("TLM,yaw=");   Serial.print(yaw_deg, 1);
    Serial.print(",state=");    Serial.print((int)state);
    Serial.print(",steer=");    Serial.print(current_steer_norm, 2);
    Serial.print(",speed=");    Serial.print(current_speed);
    Serial.print(",dF=");       Serial.print(distF);
    Serial.print(",dL=");       Serial.print(distL);
    Serial.print(",dR=");       Serial.print(distR);
    Serial.print(",dB=");       Serial.println(distB);
  }
}

/* ================== Setup / Loop ================== */
void setup(){
  Serial.begin(115200);
  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(START_BTN, INPUT_PULLUP);
  bts_init(); stopMotor();
  myservo.attach(SERVO_PIN); syncCenterSpans(); steerTo(SERVO_CENTER);
  Wire.begin(); Wire.setClock(100000);
  initIMU();
  Serial.println("READY");
}

void loop(){
  chirpRapidTick();
  readUSStaggered();

  // Serial lines
  while(Serial.available()){
    char ch = Serial.read();
    if(ch=='\n' || ch=='\r'){
      if(rx.length()>0){ handleLine(rx); rx=""; }
    } else {
      rx += ch; if(rx.length()>128) rx.remove(0);
    }
  }

  // Local button
  if(!armed && digitalRead(START_BTN) == LOW){
    delay(20);
    if(digitalRead(START_BTN) == LOW){
      setArmed(true);
      while(digitalRead(START_BTN) == LOW) { delay(5); }
    }
  }
```

```
  // IMU
  readYaw();
  if(state == TURNING && (millis() - lastImuOkMs) > IMU_STALE_REINIT_MS){
    stopMotor(); steerTo(SERVO_CENTER); state = IDLE;
#if BUZZ_ENABLE
    buzzerRapid=false; noTone(BUZZER_PIN);
#endif
    Serial.println("WARN,IMU_STALE_STOP");
  }

  switch(state){
    case IDLE:
      setDrive(0,0); setSteerNorm(0.0f);
      break;

    case CENTERING:{
      steer_f += ALPHA_STEER * (current_steer_norm - steer_f);
      speed_f += ALPHA_SPEED * ((float)current_speed - speed_f);
      float u_eff = (current_dir < 0) ? -steer_f : steer_f; // flip in reverse
      setSteerNorm(u_eff);
      setDrive((int)speed_f, current_dir);
      break;
    }

    case TURNING:
      runTurnController();
      break;
  }

  sendTelemetry();
```

open challenge code (arduino)

```
# Arduino open challenge

#include <Wire.h>
#include "DFRobot_BNO055.h"
#include <Servo.h>

/* ================== Pins ================== */
#define BUZZER_PIN A0
#define START_BTN  A1
#define START_ACTIVE_LOW 1    // 1 = button pulls pin LOW when pressed (with INPUT_PULLUP); 0 = active HIGH

// BTS7960 pins
#define RPWM 5
#define LPWM 6
#define R_EN 7
#define L_EN 8

// Servo
#define SERVO_PIN 9

// Ultrasonic (single-pin trig+echo)
#define US_R_PIN  2
#define US_L_PIN  3
#define US_F_PIN  4

/* ===== Buzzer Master Switch ===== */
#define BUZZ_ENABLE 1   // 0=OFF, 1=ON

/* ============ Servo geometry (LIVE-TRIMMABLE) ============ */
// Reversed install -> center 90, max-left 110, max-right 60
int   SERVO_CENTER = 90;   // can be changed via SET_CENTER,<deg>
#define SERVO_MIN    65
```

```
#define SERVO_MAX    120
float SPAN_LEFT  = 90 - 60;    // updated in syncCenterSpans()
float SPAN_RIGHT = 110 - 90;
float STEER_TRIM_NORM = 0.0f;  // + = steer RIGHT

static void syncCenterSpans(){
  SPAN_LEFT  = (float)(SERVO_CENTER - SERVO_MIN);
  SPAN_RIGHT = (float)(SERVO_MAX    - SERVO_CENTER);
}

/* ===== Ultrasonic speed (temp-corrected) ===== */
#define VELOCITY_TEMP(tempC) ((331.5 + 0.6 * (float)(tempC)) * 100 / 1000000.0)  // cm/us
int16_t us_tempC = 20;

/* ================= IMU ================= */
typedef DFRobot_BNO055_IIC BNO;
BNO bno(&Wire, 0x28);

/* ---- IMU watchdog / recovery ---- */
unsigned long lastImuOkMs = 0;
int imuBadBurst = 0;
const unsigned long IMU_STALE_REINIT_MS = 400; // if no good data for > this, re-init IMU
const int IMU_BAD_LIMIT = 3;                   // consecutive bad reads before re-init

/* ================ State ================ */
enum State { IDLE=0, CENTERING=1, TURNING=2 };
State state = IDLE;
bool armed = false;

/* ===== Controls & smoothing ===== */
Servo myservo;
float current_steer_norm = 0.0f;  // last commanded from Jetson
int   current_speed      = 0;     // last commanded PWM (0..255)
int   current_dir        = 0;     // +1 forward, -1 reverse, 0 stop

float steer_f = 0.0f;
float speed_f = 0.0f;

const float ALPHA_STEER = 0.18f;
const float ALPHA_SPEED = 0.35f;

/* ============= Yaw / Turn control ============= */
static float wrap180(float a){ while(a>180)a-=360; while(a<-180)a+=360; return a; }
static float shortestErr(float target, float now){ return wrap180(target - now); }
float yaw_deg = 0.0f;

/* PID for turning */
float Kp = 3.0f, Ki = 0.0f, Kd = 0.26f;
float pid_i = 0, prev_err = 0, d_filt = 0;
const float D_ALPHA = 0.35f;

unsigned long lastPIDms = 0;
const float TURN_DONE_TOL_DEG = 2.0f;
const uint16_t TURN_HOLD_MS = 100;
float target_yaw_deg = 0.0f;
bool turnDoneLatched = false;
unsigned long turnWithinTolStart = 0;

/* ======= Speed clamps ======= */
const int SPEED_MIN_CLAMP = 0;
const int SPEED_MAX_CLAMP = 255;

/* ======= Turn speed (default + per-turn override) ======= */
int TURN_PWM_DEFAULT = 120;   // default turn PWM
int turn_pwm_override = -1;   // set by TURN_ABS optional param; cleared after turn

/* ============= Buzzer helpers ============= */
inline void beep(uint16_t ms=80){
#if BUZZ_ENABLE
  tone(BUZZER_PIN, 3000, ms);
```

```
#endif
}
bool buzzerRapid = false;
unsigned long buzzLastMs = 0;
const uint16_t BUZZ_PERIOD_MS = 80;
const uint16_t BUZZ_PULSE_MS  = 40;
void chirpRapidTick(){
#if BUZZ_ENABLE
  if(!buzzerRapid) return;
  unsigned long now = millis();
  if(now - buzzLastMs >= BUZZ_PERIOD_MS){
    buzzLastMs = now;
    tone(BUZZER_PIN, 3500, BUZZ_PULSE_MS);
  }
#endif
}

/* ============ Motor / Servo helpers (BTS7960) ============ */
void bts_init(){
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  pinMode(R_EN, OUTPUT);
  pinMode(L_EN, OUTPUT);
  digitalWrite(R_EN, HIGH);
  digitalWrite(L_EN, HIGH);
  analogWrite(RPWM, 0);
  analogWrite(LPWM, 0);
}

void setDrive(int pwm, int dir){
  pwm = constrain(pwm, SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);
  if(dir > 0){        // forward
    analogWrite(RPWM, pwm);
    analogWrite(LPWM, 0);
  }else if(dir < 0){  // reverse
    analogWrite(RPWM, 0);
    analogWrite(LPWM, pwm);
  }else{              // stop (coast)
    analogWrite(RPWM, 0);
    analogWrite(LPWM, 0);
  }
}
void stopMotor(){
  analogWrite(RPWM, 0);
  analogWrite(LPWM, 0);
}

/* ---------- Servo APIs ---------- */
void steerTo(int deg){
  deg = constrain(deg, SERVO_MIN, SERVO_MAX);
  myservo.write(deg);
}

// Keep mapping the same (u>=0 -> angle increases); Jetson compensates in CENTER.
// For TURNING (PID), we invert u sign at the call site.
void setSteerNorm(float u_in){
  float u = u_in + STEER_TRIM_NORM;
  if (fabs(u) < 0.02f) u = 0.0f;
  u = constrain(u, -1.0f, 1.0f);
  float angle = (u >= 0.0f) ? (SERVO_CENTER + u*SPAN_RIGHT) : (SERVO_CENTER + u*SPAN_LEFT);
  angle = constrain(angle, (float)SERVO_MIN, (float)SERVO_MAX);
  myservo.write((int)angle);
}

/* ============ IMU ============ */
bool initIMU(){
  bno.setOprMode(BNO::eOprModeConfig);
  delay(25);
  if(bno.begin() != BNO::eStatusOK) return false;
  bno.setOprMode(BNO::eOprModeNdof);
```

```
  delay(30);
  lastImuOkMs = millis();
  imuBadBurst = 0;
  return true;
}

bool reinitIMU(){
  bno.setOprMode(BNO::eOprModeConfig);
  delay(25);
  bool ok = (bno.begin() == BNO::eStatusOK);
  if(ok){
    bno.setOprMode(BNO::eOprModeNdof);
    delay(30);
    lastImuOkMs = millis();
    imuBadBurst = 0;
    Serial.println("INFO,IMU_REINIT_OK");
  }else{
    Serial.println("WARN,IMU_REINIT_FAIL");
  }
  return ok;
}

void readYaw(){
  BNO::sEulAnalog_t e = bno.getEul();
  unsigned long now = millis();

  bool ok = (bno.lastOperateStatus == BNO::eStatusOK);
  bool looksZeroFrame = (fabs(e.head) < 0.001f && fabs(e.roll) < 0.001f && fabs(e.pitch) < 0.001f);

  if(ok && !looksZeroFrame){
    float y = wrap180(e.head);
    if(!isnan(y)){
      yaw_deg = y;
      lastImuOkMs = now;
      imuBadBurst = 0;
    }else{
      imuBadBurst++;
    }
  }else{
    imuBadBurst++;
  }

  if(imuBadBurst >= IMU_BAD_LIMIT || (now - lastImuOkMs) > IMU_STALE_REINIT_MS){
    reinitIMU();
  }
}

/* ============ Ultrasonic ============ */
// Desired maximum measurable range (cm). We will CLAMP readings to this.
#define US_MAX_RANGE_CM 300

// Forward declarations for distances (must be before readUSAllFast)
long distF = -1, distL = -1, distR = -1;

// Compute timeout (us) from desired max range & temperature (round trip) with headroom
static inline unsigned long usTimeoutFromRangeCm(int cm){
  float v = VELOCITY_TEMP(us_tempC);           // cm/us
  return (unsigned long)((2.0f * cm) / v * 1.2f);   // ~20% headroom
}

inline long usPingOnePin(int pin){
  pinMode(pin, OUTPUT);
  digitalWrite(pin, LOW);  delayMicroseconds(2);
  digitalWrite(pin, HIGH); delayMicroseconds(10);
  digitalWrite(pin, LOW);
  pinMode(pin, INPUT);

  unsigned long timeout = usTimeoutFromRangeCm(US_MAX_RANGE_CM);
  unsigned long pw = pulseIn(pin, HIGH, timeout);
  if(pw == 0) return -1;  // truly no echo within extended timeout
```

```
  long cm = (long)(pw * VELOCITY_TEMP(us_tempC) / 2.0f);
  if (cm > US_MAX_RANGE_CM) cm = US_MAX_RANGE_CM;   // clamp far values
  return cm;
}

inline void readUSAllFast(){
  distF = usPingOnePin(US_F_PIN);
  distL = usPingOnePin(US_L_PIN);
  distR = usPingOnePin(US_R_PIN);
}

/* ============ Start button helpers ============ */
static inline bool startPressedRaw(){
  int v = digitalRead(START_BTN);
  return START_ACTIVE_LOW ? (v == LOW) : (v == HIGH);
}
static bool startPressedDebounced(uint16_t ms=20){
  if(!startPressedRaw()) return false;
  delay(ms);
  return startPressedRaw();
}

/* ============ Turn controller ============ */
void runTurnController(){
  int turn_pwm = (turn_pwm_override >= 0) ? turn_pwm_override : TURN_PWM_DEFAULT;
  turn_pwm = constrain(turn_pwm, SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);

  float err = shortestErr(target_yaw_deg, yaw_deg);
  if (fabs(err) < 1.0f) err = 0.0f;

  unsigned long now = millis();
  float dt = (lastPIDms==0)? 0.02f : (now - lastPIDms)/1000.0f;
  lastPIDms = now;

  pid_i += err * dt;

  float d = (err - prev_err) / max(dt, 1e-3f);
  d_filt = (D_ALPHA * d) + (1.0f - D_ALPHA) * d_filt;
  prev_err = err;

  // Compute normalized steering command from PID
  float u_deg = Kp*err + Ki*pid_i + Kd*d_filt;   // degrees-like
  float u_norm = constrain(u_deg/45.0f, -1.2f, 1.2f);

  // SERVO REVERSED FOR TURNING: invert sign here
  setSteerNorm(-u_norm);

  setDrive(turn_pwm, +1);     // forward during turn

#if BUZZ_ENABLE
  buzzerRapid = true;         // chirp while turning
#endif

  if(fabs(err) <= TURN_DONE_TOL_DEG){
    if(!turnDoneLatched){ turnWithinTolStart = now; turnDoneLatched = true; }
    else if(now - turnWithinTolStart >= TURN_HOLD_MS){
      stopMotor();
      steerTo(SERVO_CENTER);
      state = IDLE;
      // reset PID and clear per-turn override
      pid_i = 0; prev_err = 0; d_filt = 0; lastPIDms = 0; turnDoneLatched = false;
      turn_pwm_override = -1;
#if BUZZ_ENABLE
      buzzerRapid = false; noTone(BUZZER_PIN);
#endif
      beep(60);
    }
  }else{
    turnDoneLatched = false;
```

```
  }
}

/* ============ Serial protocol ============
 * START
 * CENTER,<norm>,<speed>   (forward; -1<=norm<=1; 0..255)
 * BACK,<speed>            (reverse; keeps last steer)
 * TURN_ABS,<target_yaw>[,<turn_pwm>]
 * TRIM_NORM,<value>
 * SET_CENTER,<deg>
 * SET_TURN_PWM,<pwm>
 * CLEAR_TURN_PWM
 * STOP
 * PING  -> respond with "PONG"
 */
String rx;

void setArmed(bool on){
  if(on && !armed){
    armed = true;
    beep(100);
    Serial.println("ARMED");
  }else if(!on && armed){
    armed = false;
  }
}

void handleLine(const String &line){
  if(line.equalsIgnoreCase("PING")){
    Serial.println("PONG");
    return;
  }

  if(line.equalsIgnoreCase("START") || line.equalsIgnoreCase("S")){
    setArmed(true);
    return;
  }

  if(!armed) return;

  if(line.startsWith("CENTER")){
    int c1 = line.indexOf(',');
    int c2 = line.indexOf(',', c1+1);
    if(c1>0 && c2>c1){
      float u = line.substring(c1+1, c2).toFloat();
      int spd = line.substring(c2+1).toInt();
      current_steer_norm = constrain(u, -1.0f, 1.0f);
      current_speed = constrain(spd, SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);
      current_dir = (current_speed > 0) ? +1 : 0;
      if(state != TURNING) state = CENTERING;
#if BUZZ_ENABLE
      buzzerRapid = false; noTone(BUZZER_PIN);
#endif
    }
  }else if(line.startsWith("BACK")){
    int c1 = line.indexOf(',');
    int spd = 0;
    if(c1>0){
      spd = line.substring(c1+1).toInt();
      spd = constrain(spd, SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);
    }
    current_speed = spd;
    current_dir   = (spd > 0) ? -1 : 0;
    if(state != TURNING) state = CENTERING;
#if BUZZ_ENABLE
    buzzerRapid = false; noTone(BUZZER_PIN);
#endif
  }else if(line.startsWith("TURN_ABS")){
    int c1 = line.indexOf(',');
    if(c1>0){
```

```
      int c2 = line.indexOf(',', c1+1);
      target_yaw_deg = wrap180(line.substring(c1+1, (c2>0? c2 : line.length())).toFloat());
      if(c2>0){
        turn_pwm_override = line.substring(c2+1).toInt();
      }else{
        turn_pwm_override = -1;
      }
      state = TURNING;
      pid_i = 0; prev_err = 0; d_filt = 0; lastPIDms = 0; turnDoneLatched=false;
#if BUZZ_ENABLE
      buzzerRapid = true; buzzLastMs = 0;
      beep(120);
#endif
    }
  }else if(line.startsWith("TRIM_NORM")){
    int c1 = line.indexOf(',');
    if(c1>0){
      STEER_TRIM_NORM = line.substring(c1+1).toFloat();
      STEER_TRIM_NORM = constrain(STEER_TRIM_NORM, -0.3f, 0.3f);
      beep(40);
    }
  }else if(line.startsWith("SET_CENTER")){
    int c1 = line.indexOf(',');
    if(c1>0){
      SERVO_CENTER = constrain(line.substring(c1+1).toInt(), SERVO_MIN, SERVO_MAX);
      syncCenterSpans();
      steerTo(SERVO_CENTER);
      beep(80);
    }
  }else if(line.startsWith("SET_TURN_PWM")){
    int c1 = line.indexOf(',');
    if(c1>0){
      TURN_PWM_DEFAULT = constrain(line.substring(c1+1).toInt(), SPEED_MIN_CLAMP, SPEED_MAX_CLAMP);
      beep(60);
    }
  }else if(line.startsWith("CLEAR_TURN_PWM")){
    turn_pwm_override = -1; beep(40);
  }else if(line.startsWith("STOP")){
    state = IDLE;
    current_speed = 0;
    current_dir = 0;
    setDrive(0,0);
    setSteerNorm(0.0f);
#if BUZZ_ENABLE
    buzzerRapid = false; noTone(BUZZER_PIN);
#endif
  }
}

/* ============ Telemetry (~40 Hz) ============ */
void sendTelemetry(){
  static unsigned long last = 0;
  unsigned long now = millis();
  if(now - last >= 25){
    last = now;
    readUSAllFast();
    Serial.print("TLM,yaw=");   Serial.print(yaw_deg, 1);
    Serial.print(",state=");    Serial.print((int)state);
    Serial.print(",steer=");    Serial.print(current_steer_norm, 2);
    Serial.print(",speed=");    Serial.print(current_speed);
    Serial.print(",dF=");       Serial.print(distF);
    Serial.print(",dL=");       Serial.print(distL);
    Serial.print(",dR=");       Serial.println(distR);
  }
}

/* ============ Start latch (button OR serial) ============ */
void waitForStart(){
  stopMotor();
  steerTo(SERVO_CENTER);
```

```
  String localRx = "";
  while(!armed){
    // Debounced button
    if(startPressedDebounced()){
      setArmed(true);                       // will print "ARMED"
      while(startPressedRaw()) { delay(5); }  // wait for release
      break;
    }
    // Serial quick parser for S/START during wait
    while(Serial.available()){
      char ch = Serial.read();
      if(ch=='\n' || ch=='\r'){
        if(localRx.equalsIgnoreCase("S") || localRx.equalsIgnoreCase("START")){
          setArmed(true);
          localRx = "";
          break;
        }
        localRx = "";
      } else {
        if(localRx.length() < 32) localRx += ch;
      }
    }
    delay(2);
  }
}

/* ================== Setup / Loop ================== */
void setup(){
  Serial.begin(115200);

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(START_BTN, INPUT_PULLUP);

  bts_init();
  stopMotor();

  myservo.attach(SERVO_PIN);
  syncCenterSpans();
  steerTo(SERVO_CENTER);

  Wire.begin();
  Wire.setClock(100000);   // keep I2C at 100 kHz for robustness
  initIMU();

  Serial.println("READY");  // tell Jetson we're alive
  waitForStart();           // blocks until button or START
  state = IDLE;
}

void loop(){
  chirpRapidTick();

  // Read serial lines
  while(Serial.available()){
    char ch = Serial.read();
    if(ch=='\n' || ch=='\r'){
      if(rx.length()>0){ handleLine(rx); rx=""; }
    } else {
      rx += ch; if(rx.length()>128) rx.remove(0);
    }
  }

  // Allow button to arm anytime (e.g., after STOP)
  if(!armed && startPressedDebounced()){
    setArmed(true);
    while(startPressedRaw()) { delay(5); }  // wait for release
  }

  readYaw();
```

```
  // If we're turning but IMU hasn't produced a good update recently, stop safely.
  if(state == TURNING && (millis() - lastImuOkMs) > IMU_STALE_REINIT_MS){
    stopMotor();
    steerTo(SERVO_CENTER);
    state = IDLE;
#if BUZZ_ENABLE
    buzzerRapid = false; noTone(BUZZER_PIN);
#endif
    Serial.println("WARN,IMU_STALE_STOP");
  }

  if (state != TURNING) {
#if BUZZ_ENABLE
    buzzerRapid = false; noTone(BUZZER_PIN);
#endif
  }

  switch(state){
    case IDLE:
      setDrive(0,0);
      setSteerNorm(0.0f);
      break;

    case CENTERING: {
      // Filter commanded steer & speed and apply current direction (+1/-1)
      steer_f += ALPHA_STEER * (current_steer_norm - steer_f);
      speed_f += ALPHA_SPEED * ((float)current_speed - speed_f);
      setSteerNorm(steer_f);
      setDrive((int)speed_f, current_dir);
      break;
    }

    case TURNING:
      runTurnController();
      break;
  }

  sendTelemetry();
}
```

open challenge code (paython)

```
#  #!/usr/bin/env python3
# -- coding: utf-8 --
"""
Jetson side — starts on S key OR when Arduino push button arms (prints 'ARMED').

Hotkeys:
  S -> send START (manual start)
  X -> reset Arduino (with backoff)
  Q -> STOP and quit
"""

import cv2 as cv
import numpy as np
import yaml, glob, time, os, re, math, sys
from collections import deque

try:
    import serial
except Exception:
    serial = None

# ============================================================
# ==================== TUNING PANEL ==========================
# ============================================================
TUNE = {
```

```
# --- Turn / geometry ---
"TURN_DEG": float(os.environ.get("TURN_DEG", "85.0")),
"TURN_TRIGGER_TOL_DEG": float(os.environ.get("TURN_TRIGGER_TOL_DEG", "80.0")),
"CENTER_TURN_STEP_DEG": float(os.environ.get("CENTER_TURN_STEP_DEG", "90.0")),

# --- Run & stop caps ---
"MAX_TURNS": int(os.environ.get("MAX_TURNS", "12")),
"STOP_AT_END_CM": float(os.environ.get("STOP_AT_END_CM", "140.0")),

# --- Steering servo geometry (absolute degrees) ---
# Reversed-install servo: center=90, left=110, right=60
"CENTER_DEG": int(os.environ.get("CENTER_DEG", "90")),
"LEFT_LIMIT": int(os.environ.get("LEFT_LIMIT", "60")),
"RIGHT_LIMIT": int(os.environ.get("RIGHT_LIMIT", "110")),
# flip world->servo sign cleanly (1=reversed)
"SERVO_REVERSED": int(os.environ.get("SERVO_REVERSED", "1")),

# --- Drive speeds (PWM 0..255) ---
"DRIVE_SPEED": int(os.environ.get("DRIVE_SPEED", "43")),
"TURN_SPEED": int(os.environ.get("TURN_SPEED", "27")),
"POST_TURN_SPEED": int(os.environ.get("POST_TURN_SPEED", "25")),
"POST_TURN_SEC": float(os.environ.get("POST_TURN_SEC", "1.5")),
"SEE_COLOR_SPEED": int(os.environ.get("SEE_COLOR_SPEED", "27")),
"MIN_SPEED": int(os.environ.get("MIN_SPEED", "0")),

# --- Wall proximity slowdowns (speed management) ---
"WALL_SLOW_SIDE_ON_CM": float(os.environ.get("WALL_SLOW_SIDE_ON_CM", "8.0")),
"WALL_SLOW_SIDE_OFF_CM": float(os.environ.get("WALL_SLOW_SIDE_OFF_CM", "9.0")),
"SIDE_SLOWDOWN_FACTOR": float(os.environ.get("SIDE_SLOWDOWN_FACTOR", "0.55")),
"WALL_SLOW_FRONT_ON_CM": float(os.environ.get("WALL_SLOW_FRONT_ON_CM", "45.0")),
"FRONT_SLOWDOWN_FACTOR": float(os.environ.get("FRONT_SLOWDOWN_FACTOR", "0.550")),
"WALL_SLOW_MIN_SPEED": int(os.environ.get("WALL_SLOW_MIN_SPEED", "21")),

# --- Wall-safety steering authority (small push away near walls) ---
"WALL_STEER_GAIN": float(os.environ.get("WALL_STEER_GAIN", ".3")),
"WALL_STEER_MAX_ADD_DEG": float(os.environ.get("WALL_STEER_MAX_ADD_DEG", "0.80")),
"WALL_STEER_EXPAND_CENTER_CLAMP": int(os.environ.get("WALL_STEER_EXPAND_CENTER_CLAMP", "1")),
"WALL_STEER_CENTER_MIN": int(os.environ.get("WALL_STEER_CENTER_MIN", "80")),
"WALL_STEER_CENTER_MAX": int(os.environ.get("WALL_STEER_CENTER_MAX", "100")),

# --- Strong Wall AVOID (independent of slowdown) ---
"WALL_AVOID_ON_CM": float(os.environ.get("WALL_AVOID_ON_CM", "11.5")),
"WALL_AVOID_OFF_CM": float(os.environ.get("WALL_AVOID_OFF_CM", "12.5")),
"WALL_AVOID_MAX_BIAS_DEG": float(os.environ.get("WALL_AVOID_MAX_BIAS_DEG", "25.0")),
"WALL_AVOID_GAIN": float(os.environ.get("WALL_AVOID_GAIN", "0.65")),
"WALL_AVOID_SPEED": int(os.environ.get("WALL_AVOID_SPEED", "24")),
"WALL_AVOID_TIMEOUT_S": float(os.environ.get("WALL_AVOID_TIMEOUT_S", "1.2")),

# --- Contact failsafe (very close or sensors blind) ---
"CONTACT_ON_CM": float(os.environ.get("CONTACT_ON_CM", "12.0")),
"CONTACT_TREAT_INVALID_AS_ON": int(os.environ.get("CONTACT_TREAT_INVALID_AS_ON", "1")),
"CONTACT_STEER_DEG": float(os.environ.get("CONTACT_STEER_DEG", "14.0")),
"CONTACT_HOLD_S": float(os.environ.get("CONTACT_HOLD_S", "0.3")),
"CONTACT_SPEED_FACTOR": float(os.environ.get("CONTACT_SPEED_FACTOR", "0.6")),

# --- Turn gating (FRONT + SIDE) ---
"FRONT_TURN_THRESH_CM": float(os.environ.get("FRONT_TURN_THRESH_CM", "35.0")),
"TURN_USE_SIDE_GATE": int(os.environ.get("TURN_USE_SIDE_GATE", "1")),
"TURN_SIDE_OPEN_CM": float(os.environ.get("TURN_SIDE_OPEN_CM", "120.0")),
"TURN_SIDE_LOGIC": os.environ.get("TURN_SIDE_LOGIC", "OR").upper(),
"TURN_GATE_COMBINE": os.environ.get("TURN_GATE_COMBINE", "OR").upper(),

# --- Color detection / ROI (BLUE/ORANGE) ---
"COLOR_CONFIRM_FRAMES": int(os.environ.get("COLOR_CONFIRM_FRAMES", "1")),
"DETECT_COOLDOWN_S": float(os.environ.get("DETECT_COOLDOWN_S", "2.5")),
"MIN_PIX_COLOR": int(os.environ.get("MIN_PIX_COLOR", "1")),
"COLOR_ERODE_IT": int(os.environ.get("COLOR_ERODE_IT", "0")),
"COLOR_DILATE_IT": int(os.environ.get("COLOR_DILATE_IT", "1")),
"COLOR_ROI_WIDTH_FRAC": float(os.environ.get("COLOR_ROI_WIDTH_FRAC", "0.25")),
"COLOR_ROI_HEIGHT_FRAC": float(os.environ.get("COLOR_ROI_HEIGHT_FRAC", "0.20")),
```

```
    "ROI_SCALE_AFTER_FIRST_TURN": float(os.environ.get("ROI_SCALE_AFTER_FIRST_TURN", "1.3")),
    "COLOR_ROI_AFTER_TURNS": int(os.environ.get("COLOR_ROI_AFTER_TURNS", "0")),
    "COLOR_PICK_MODE": "maxpix",
    "COLOR_TURN_MAP": {"BLUE": "LEFT", "ORANGE": "RIGHT"},
    "SAT_BOOST": float(os.environ.get("SAT_BOOST", "1.15")),
    "BLUR_KSIZE": int(os.environ.get("BLUR_KSIZE", "3")),
    "COLOR_SUSPEND_SEC": float(os.environ.get("COLOR_SUSPEND_SEC", "2.5")),

    # --- Color re-arm (prepare for next junction) ---
    "COLOR_REARM_FRONT_CM": float(os.environ.get("COLOR_REARM_FRONT_CM", "80.0")),
    "COLOR_REARM_DELAY_S": float(os.environ.get("COLOR_REARM_DELAY_S", "0.6")),
    "DISABLE_DETECT_DURING_TURN": True,

    # --- Yaw centering controller ---
    "STEER_KP_DEG_PER_DEG": float(os.environ.get("STEER_KP_DEG_PER_DEG", "8.90")),
    "STEER_EXP": float(os.environ.get("STEER_EXP", "1.0")),
    "STEER_DEAD_ERR_DEG": float(os.environ.get("STEER_DEAD_ERR_DEG", "0.02")),
    "STEER_OUT_DB_DEG": float(os.environ.get("STEER_OUT_DB_DEG", ".2")),
    "STEER_MAX_DEG": float(os.environ.get("STEER_MAX_DEG", "5.0")),
    "ERR_FILTER_ALPHA": float(os.environ.get("ERR_FILTER_ALPHA", "1.75")),
    "STEER_SLEW_DEG": int(os.environ.get("STEER_SLEW_DEG", "20")),
    "CENTER_STEER_MIN": 70,
    "CENTER_STEER_MAX": 110,

    # --- Ultrasonic averaging & repel ---
    "US_MAX_CM": float(os.environ.get("US_MAX_CM", "300.0")),   # CHANGED (was 200.0)
    "US_EMA_ALPHA": float(os.environ.get("US_EMA_ALPHA", "0.35")),
    "US_REPEL_ON_CM": float(os.environ.get("US_REPEL_ON_CM", "18.0")),
    "US_REPEL_OFF_CM": float(os.environ.get("US_REPEL_OFF_CM", "22.0")),
    "US_REPEL_K": float(os.environ.get("US_REPEL_K", "0.06")),
    "US_REPEL_STEER_BOOST_DEG": float(os.environ.get("US_REPEL_STEER_BOOST_DEG", "6.0")),
    "US_REPEL_EXPAND_CENTER_CLAMP": int(os.environ.get("US_REPEL_EXPND_CENTER_CLAMP", "1")),
    "US_REPEL_CENTER_MIN": int(os.environ.get("US_REPEL_CENTER_MIN", "80")),
    "US_REPEL_CENTER_MAX": int(os.environ.get("US_REPEL_CENTER_MAX", "100")),

    # --- Camera ---
    "ALLOW_NO_CAMERA": int(os.environ.get("ALLOW_NO_CAMERA", "1")),
    "CAMERA_BACKEND": os.environ.get("CAMERA_BACKEND", "AUTO").upper(),
    "FRAME_W": int(os.environ.get("FRAME_W", "640")),
    "FRAME_H": int(os.environ.get("FRAME_H", "480")),

    # --- Final-run logic ---
    "FINAL_BURST_S": float(os.environ.get("FINAL_BURST_S", "1.0")),
    "FINAL_US_DELAY_S": float(os.environ.get("FINAL_US_DELAY_S", ".9")),
    "REVERSE_SUPPORTED": int(os.environ.get("REVERSE_SUPPORTED", "0")),
    "REVERSE_SPEED": int(os.environ.get("REVERSE_SPEED", "30")),
    "REVERSE_MAX_S": float(os.environ.get("REVERSE_MAX_S", "2.0")),

    # --- Serial link & watchdog ---
    "ROBOT_PORT": os.environ.get("ROBOT_PORT", "/dev/ttyUSB0"),
    "ROBOT_BAUD": int(os.environ.get("ROBOT_BAUD", "115200")),
    "STALE_TLM_SEC": float(os.environ.get("STALE_TLM_SEC", "1.2")),
    "STALE_LINK_SEC": float(os.environ.get("STALE_LINK_SEC", "2.5")),
    "PING_PERIOD_SEC": float(os.environ.get("PING_PERIOD_SEC", "0.7")),
    "RESET_BACKOFF_SEC": float(os.environ.get("RESET_BACKOFF_SEC", "4.0")),

    "ARD_RESET_METHOD": os.environ.get("ARD_RESET_METHOD", "AUTO").upper(),
    "ARD_RESET_TOUCH_DELAY_S": float(os.environ.get("ARD_RESET_TOUCH_DELAY_S", "0.8")),
    "ARD_RESET_DTR_PULSE_S": float(os.environ.get("ARD_RESET_DTR_PULSE_S", "0.12")),
}

# ================ CONSTANTS / HELPERS ================
MIN_SPEED   = TUNE["MIN_SPEED"]
CENTER_DEG  = TUNE["CENTER_DEG"]
LEFT_LIMIT  = TUNE["LEFT_LIMIT"]
RIGHT_LIMIT = TUNE["RIGHT_LIMIT"]
SPAN_LEFT   = abs(CENTER_DEG - LEFT_LIMIT)
SPAN_RIGHT  = abs(RIGHT_LIMIT - CENTER_DEG)
SERVO_DIR   = -1 if int(TUNE.get("SERVO_REVERSED", 1)) else +1
```

```
def gstreamer_pipeline(sensor_id=0, capture_width=1280, capture_height=720,
                       display_width=640, display_height=480, framerate=30, flip_method=0):
    return (f"nvarguscamerasrc sensor-id={sensor_id} ! "
            f"video/x-raw(memory:NVMM), width={capture_width}, height={capture_height}, "
            f"format=(string)NV12, framerate={framerate}/1 ! "
            f"nvvidconv flip-method={flip_method} ! "
            f"video/x-raw, width={display_width}, height={display_height}, format=(string)BGRx ! "
            "videoconvert ! video/x-raw, format=(string)BGR ! appsink drop=1")

def open_camera():
    back = TUNE["CAMERA_BACKEND"]; w, h = TUNE["FRAME_W"], TUNE["FRAME_H"]
    cap = None
    def make_gst():
        try: return cv.VideoCapture(gstreamer_pipeline(display_width=w, display_height=h), cv.CAP_GSTREAMER)
        except Exception: return None
    def make_v4l2():
        try: return cv.VideoCapture(0, cv.CAP_V4L2)
        except Exception: return None
    if back == "GST": cap = make_gst()
    elif back == "V4L2": cap = make_v4l2()
    else:
        cap = make_gst()
        if not (cap and cap.isOpened()): cap = make_v4l2()
    if cap and cap.isOpened(): return cap, False
    if TUNE["ALLOW_NO_CAMERA"]:
        print("WARNING: camera open failed; using dummy frames.")
        class DummyCap:
            def isOpened(self): return True
            def read(self):
                frame = np.zeros((h, w, 3), dtype=np.uint8)
                cv.putText(frame, "NO CAMERA (dummy)", (10,30),
                           cv.FONT_HERSHEY_SIMPLEX, 0.8, (0,0,255), 2)
                return True, frame
            def release(self): pass
        return DummyCap(), True
    print("ERROR: camera open failed."); return None, False

def load_latest_yaml():
    override = os.environ.get("VISION_CFG", "").strip()
    if override and os.path.isfile(override): path = override
    else:
        cands = sorted(glob.glob("config/vision_*.yaml"))
        if not cands: raise FileNotFoundError("No config in ./config — run tuner_multi.py and press S to save.")
        path = cands[-1]
    with open(path, "r") as f: return yaml.safe_load(f), path

def wrap180(a):
    while a>180: a-=360
    while a<-180: a+=360
    return a

def shortest_err(target_deg, now_deg): return wrap180(target_deg - now_deg)

def deg_to_norm(angle_deg):
    # Inverse of Arduino's setSteerNorm() mapping
    angle_deg = float(max(LEFT_LIMIT, min(RIGHT_LIMIT, angle_deg)))
    if angle_deg >= CENTER_DEG:
        return (angle_deg - CENTER_DEG) / float(max(1.0, SPAN_RIGHT))
    else:
        return (angle_deg - CENTER_DEG) / float(max(1.0, SPAN_LEFT))

def slew_limit(prev, target, max_delta):
    if prev is None: return target
    if target > prev + max_delta: return prev + max_delta
    if target < prev - max_delta: return prev - max_delta
    return target

def mask_hsv_single(hsv, low, high, ek=0, dk=0):
    m = cv.inRange(hsv, np.array(low, np.uint8), np.array(high, np.uint8))
    if ek > 0: m = cv.erode(m, cv.getStructuringElement(cv.MORPH_ELLIPSE, (ek, ek)))
```

```
    if dk > 0: m = cv.dilate(m, cv.getStructuringElement(cv.MORPH_ELLIPSE, (dk, dk)))
    return m

def mask_hsv_multi(hsv, ranges, ek=0, dk=0):
    if isinstance(ranges, dict) and "low" in ranges and "high" in ranges:
        return mask_hsv_single(hsv, ranges["low"], ranges["high"], ek, dk)
    mask = None
    for r in (ranges if isinstance(ranges, list) else []):
        if "low" in r and "high" in r:
            m = mask_hsv_single(hsv, r["low"], r["high"], ek, dk)
            mask = m if mask is None else cv.bitwise_or(mask, m)
    if mask is None: mask = np.zeros(hsv.shape[:2], dtype=np.uint8)
    return mask

# ================== MAIN ==================
def main():
    cfg, cfg_path = load_latest_yaml()
    print("Loaded vision config:", cfg_path)

    # ---- Serial (robust) ----
    ser = None
    device = TUNE["ROBOT_PORT"]
    baud = TUNE["ROBOT_BAUD"]

    last_tlm_t = 0.0
    last_pong_t = 0.0
    last_ping_t = 0.0
    last_reset_t = -1e9

    rx_buf = b""
    tlm_re = re.compile(r"^TLM,.*yaw=([\-0-9\.]+),state=([0-9]),steer=([\-0-9\.]+),speed=([0-9]+),dF=([\-0-9]+),dL=([\-0-9]+),dR=([\-0-9]+)")

    def open_serial():
        nonlocal ser
        if serial is None:
            print("PySerial not available."); return
        try:
            ser = serial.Serial(device, baudrate=baud, timeout=0.01, write_timeout=0.2, dsrdtr=False)
            time.sleep(0.35)
            ser.reset_input_buffer(); ser.reset_output_buffer()
            print("Serial OK on", device)
        except Exception as e:
            print("WARNING: serial not open ->", e); ser = None

    def safe_write(line_bytes):
        nonlocal ser
        if serial is None: return False
        if ser is None:
            open_serial()
            if ser is None: return False
        try:
            ser.write(line_bytes); return True
        except Exception as e:
            print("Serial write error:", e)
            try: ser.close()
            except Exception: pass
            ser=None
            open_serial()
            if ser:
                try: ser.write(line_bytes); return True
                except Exception as e2: print("Write retry failed:", e2)
        return False

    def reset_arduino(method="AUTO"):
        nonlocal ser, last_reset_t
        now = time.time()
        if now - last_reset_t < TUNE["RESET_BACKOFF_SEC"]: return False
        last_reset_t = now
        meth = (method or "AUTO").upper()
        port = device
```

```
def do_touch1200():
    try:
        print("[RESET] 1200-bps touch...")
        tmp = serial.Serial(port, 1200, timeout=0.05)
        time.sleep(0.05); tmp.setDTR(False); tmp.flush(); tmp.close()
        time.sleep(TUNE["ARD_RESET_TOUCH_DELAY_S"])
        return True
    except Exception as e:
        print("[RESET] 1200-bps touch failed:", e); return False
def do_dtr():
    try:
        print("[RESET] DTR pulse...")
                                                        s = serial.Serial(
```