**SRI VENKATESWARA INTERNSHIP PROGRAM**
**FOR RESEARCH IN ACADEMICS**
**(SRI-VIPRA )**

SRI-VIPRA

# Project Report of 2025: SVPG-2507

"Design and Development of an Intelligent Quadcopter Drone for Surveillance Applications"


**Under the Guidance**
**of**
**Dr. Rahul**
**Assistant Professor**
**Department of Electronics**

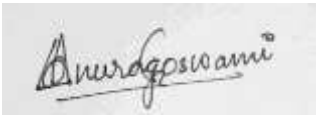


**IQAC**
**Sri Venkateswara College**
**University of Delhi**
**Benito Juarez Road, Dhaula Kuan, New Delhi**
**New Delhi -110021**

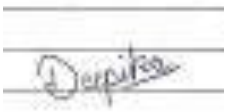

SRI-VIPRA

# SRIVIPRA PROJECT 2025

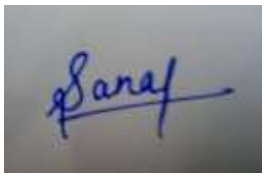
*List of students under the SRIVIPRA Project*

| S.No | Photo | Name of the student | Roll number | Course | Signature |
|---|---|---|---|---|---|
| 1 |  | Akshara Bhatt | 1823055 | BSc (H) Physics |  |
| 2 |  | Anurag Goswami | 1724090 | BSc (H) Mathematics |  |

SRI-VIPRA

| | | | | | |
|---|---|---|---|---|---|
| 3 |  | **Deepika** | **1623029** | **BSc (H) Electronics** |  |
| 4 |  | **Devansh Sawhney** | **1623058** | **BSc (H) Electronics** |  |
| 5 |  | **Diya Sharma** | **1723008** | **BSc. (H) Mathematics** |  |

SRI-VIPRA

| | | | | | |
|---|---|---|---|---|---|
| 6 |  | **Samarth** | **1623061** | **BSc (H) Electronics** |  |
| 7 |  | **Samuel Matthews** | | **BSc (H) Electronics** |  |
| 8 |  | **Sana Singh** | **1623016** | **BSc (H) Electronics** |  |

RI-VIPRA

## TABLE OF CONTENTS


| S. No | Topic | Page No. |
|---|---|---|
| 1 | Introduction | 7-8 |
| 2 | Types of Drones | 9-12 |
| 3 | Quadcopter Drone | 12-13 |
| 4 | Components of the Quadcopter | 14-19 |
| 5 | Assembly of the Quadcopter | 19-24 |
| 6 | Flight controller and Mission Planner | 24-30 |
| 7 | Final Software Implementation | 31-51 |
| 8 | Individual Software Contribution | 52-83 |
| 9 | Advantages of smart surveillance drone | 84-85 |
| 10 | Conclusion | 86 |
| 11 | Billing | 87-88 |
| 12 | References | 89 |


SR-VIRA

# 1. INTRODUCTION

Drones, or UAVs (Unmanned Aerial Vehicles) as they're technically called, have completely changed how we approach tasks like surveillance, security, environmental monitoring, and disaster management. What started as basic remote-controlled flying machines has evolved into sophisticated autonomous systems that can handle complex operations with minimal human input. This transformation has been driven by rapid progress in electronics, embedded systems, wireless communication, and AI.

Among the different types of drones out there, quadcopters have become incredibly popular and for good reason. They're mechanically simpler than other designs, can take off and land vertically, move with precision, and hover steadily even in tight spaces. These characteristics make them perfect for surveillance work, where you need stable flight, accurate positioning, and the ability to capture data in real time. Today's surveillance drones pack sensors, cameras, flight controllers, and onboard computers that enable autonomous flight and smart decision-making.

Traditional surveillance systems depend on fixed cameras and ground personnel, which comes with obvious limitations restricted coverage, line-of-sight issues, and expensive infrastructure. Drones solve these problems by offering mobility, broader coverage, and quick deployment. When you combine this with AI and computer vision, drones can automatically detect, track, and analyse targets without constant human supervision, dramatically improving response times.

Integrating AI-powered object detection with drones marks a significant leap toward truly intelligent surveillance systems. Deep learning models like YOLO (You Only Look Once) can detect objects in real time with impressive accuracy and minimal delay, making them ideal for the limited computing resources available onboard. By processing video feeds directly on the drone, these systems can identify and classify objects mid-flight, boosting both situational awareness and operational efficiency.

This project was completed as part of the Sri Venkateswara Internship Program for Research in Academics (SRI-VIPRA). The goal was to build a smart quadcopter drone for surveillance applications. The work covered everything from selecting and sourcing hardware components to assembling the mechanical and electrical systems, configuring the flight controller, planning autonomous missions using Mission Planner, and implementing an AI-based surveillance system. Essentially, we wanted to show how a standard quadcopter platform can be upgraded into an intelligent surveillance tool by combining flight control technology with computer vision.

SRI-VIPRA

Through this project, we gained hands-on experience with drone hardware integration, flight controller setup, mission planning software, and deploying AI algorithms on embedded systems. This report walks through the entire development process, discusses the challenges we faced along the way, and explores the potential applications and future possibilities for smart surveillance drones.



IMAGE FROM OFFICIAL DJI SITE

Figure 1: Related drone component or system illustration.



AI IMAGE MADE USING MOONSHOT

Figure 2: Related drone component or system illustration.

# 2. Types of Drones

Unmanned Aerial Vehicles (UAVs) are classified based on their structural design, how they generate lift, and their rotor configuration. Each specific architecture offers a different trade-off between payload capacity, stability, redundancy, and operational efficiency. The primary categories are detailed below.

## 2.1. Fixed-Wing Drones

Fixed-wing drones operate on the same aerodynamic principles as conventional aircraft, generating lift through rigid wings. This design is exceptionally energy-efficient, allowing for long-endurance flights over vast distances. However, their reliance on runways or launch systems, combined with an inability to hover in place, limits their utility for close-range surveillance or stationary monitoring.



Figure 3: Fixed Winged Drones

## 2.2. Single-Rotor Drones

Similar to helicopters, single-rotor drones utilize one large main rotor for lift and a smaller tail rotor to counteract torque. While they generally offer superior payload capacity and flight times compared to multirotor systems, they are mechanically complex. The higher cost and requirement for advanced control systems often make them less practical for general-purpose applications.

SRVIPRA



Figure 4: Single Rotor Drones

## 2.3. Multirotor Drones

Multirotor drones rely on multiple vertically oriented propellers to generate lift and control movement. Their ability to perform vertical take-off and landing (VTOL), hover steadily, and manoeuvre with precision makes them the industry standard for surveillance, photography, inspection, and research. These drones are further categorized by the number of rotors they employ:

## 2.4. Tri-copter

Tri-copters utilize three rotors for lift. To manage yaw (rotation), the rear rotor is typically mounted on a servo mechanism. Although they are lightweight and relatively efficient, Tricopters generally lack the stability and mechanical robustness found in configurations with more rotors, making them less common in professional settings.



Figure 5: Tri-Copter Drones

## 2.5. Quadcopter

The quadcopter is the most prevalent multirotor configuration, featuring four rotors arranged symmetrically. This design strikes an optimal balance between mechanical simplicity, flight stability, and cost. Because they are easy to control and maintain while offering decent payload capacity, quadcopters are the preferred choice for academic projects and general surveillance.

SV-VIPRA



Figure 6: Quadcopter Drones

## 2.6. Hexacopter

Hexacopters employ six rotors, which significantly improves payload capacity and flight stability. A key advantage of this configuration is redundancy; if a single motor fails, a hexacopter can often remain airborne and land safely. This fault tolerance makes them particularly suitable for professional surveillance and industrial tasks where equipment safety is paramount.



Images by X-botics.com

Figure 7: Hexacopter

## 2.7. Octocopter

Designed for heavy lifting and maximum reliability, octocopters utilize eight rotors. They offer superior stability and redundancy, making them for carrying expensive equipment or performing critical missions. However, these benefits come at the cost of increased weight, higher power consumption, and greater expense in both construction and maintenance.

SR-VIDRA



Images by robu.in

Figure 8: Octocopter

## 2.8. Hybrid Drones

Hybrid drones attempt to merge the best features of fixed-wing and multirotor systems, offering both vertical take-off capabilities and efficient, long-range forward flight. While this hybrid approach provides significant operational flexibility, the resulting complexity in both mechanics and control algorithms tends to limit their use to specialized defence and industrial applications.



Images by Netjunk.com

Figure 9: Hybrid Drones

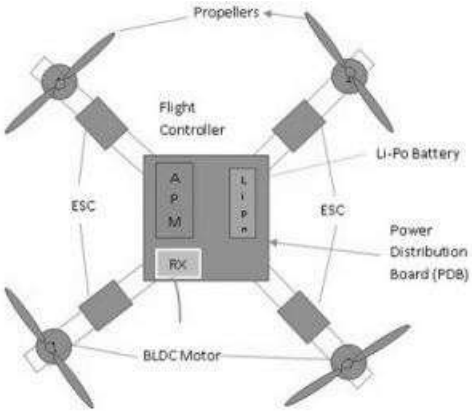
# 3. Quadcopter Drone

A quadcopter is a type of drone that uses four rotors arranged symmetrically in a cross pattern to fly. Unlike regular airplanes with wings, quadcopters have propellers pointing upward that generate thrust, which lets them take off and land vertically, hover in one spot, and move with impressive precision. Their compact size, straightforward design, and stable flight make them popular for everything from aerial photography and surveillance to inspections, mapping, and research projects.

Each rotor has its own electric motor, and the drone moves by adjusting how fast each motor spins. Two motors diagonal from each other spin clockwise, while the other two spin counterclockwise. This opposite rotation cancels out the twisting force (torque) that spinning propellers create, which prevents the drone from spinning out of control and keeps it flying smoothly.

Compared to other multi-rotor designs, quadcopters are good for small to medium applications because they need fewer parts, are simpler to put together and maintain, and work efficiently with modern flight controllers. Their ability to hover steadily makes them especially useful for surveillance, where you need stable footage and precise positioning.




valbotx.in

Figure 10: Components of a quadcopter

## 3.1. Quadcopter Mechanics

Quadcopters operate based on some fundamental physics of aerodynamics, thrust generation, and rotational dynamics. The main force that gets them airborne is lift, created when the spinning propellers push air downward. According to Newton's Third Law (every action has an equal and opposite reaction), this downward push of air creates an upward force that lifts the drone.

Lift and Thrust Generation

Each propeller is attached to a brushless DC motor that spins at high speed, forcing air downward. This downward movement of air produces upward thrust. When the total thrust from all four propellers is greater than the drone's weight, it climbs. Reduce the thrust, and it descends.

Keeping Things Balanced

Since spinning propellers naturally create torque, the quadcopter uses two motors spinning clockwise and two spinning counterclockwise to cancel out that rotational force. This balance stops the drone from spinning wildly around its vertical axis and allows it to hover steadily.

Controlling Movement

The quadcopter moves and changes direction by adjusting individual motor speeds:

- Roll (tilting side to side): Speed up one pair of motors while slowing down the opposite pair
- Pitch (tilting forward or backward): Adjust the speed difference between front and rear motors - Yaw (rotating left or right): Increase speed on one diagonal pair while decreasing the other, creating controlled rotation
- Altitude (going up or down): Change all four motor speeds equally

The Flight Controller’s task

The flight controller is basically the drone's brain. It constantly reads data from onboard sensors gyroscopes, accelerometers, and sometimes magnetometers and barometers. Using this feedback, it

SRI VIPR

adjusts motor speeds in real time to keep the drone stable, properly oriented, and moving as intended. This continuous adjustment loop is what enables precise control and even autonomous flight.

## 4. Components of a Quadcopter Drone

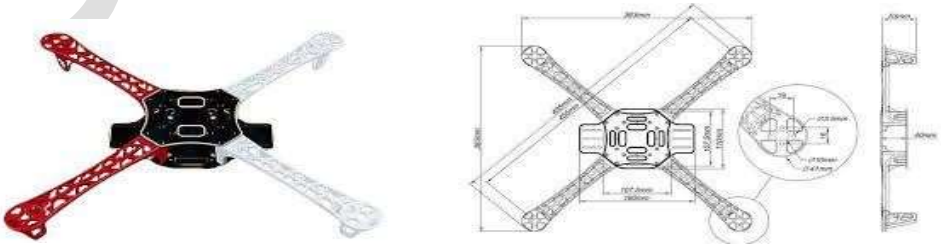
The performance, stability, and overall reliability of a quadcopter are fundamentally tied to the selection and integration of its hardware. Every component plays a distinct role in ensuring efficient flight, precise control, and the ability to handle payloads effectively. The key components utilized in the development of the smart quadcopter for this project are detailed below.



Figure 11: All components of our Quadcopter ready for assembly

### 4.1. Frame

Acting as the structural backbone, the frame provides mechanical support for all major subsystems, including the motors, flight controller, battery, and payload. A high-quality frame must be rigid enough to withstand vibration and minor impacts while remaining lightweight to maximize flight efficiency. For this project, the F450 quadcopter frame was selected. This frame is an industry standard for custom builds due to its robust symmetrical design and ease of assembly. Constructed from durable composite materials, it offers an excellent strength-to-weight ratio. The frame also includes integrated landing gear, which provides stability during vertical operations and protects the sensitive onboard electronics from ground contact.



Images from ResearchGate.com

Figure 12: Frame

## 4.2. Battery

The battery acts as the central power source, supplying electrical energy to the motors, flight controller, Electronic Speed Controllers (ESCs), and other peripherals. The choice of battery capacity is a critical trade-off between flight duration and the added weight of the power cell.

A 2200 mAh rechargeable battery was chosen for this system. This capacity ensures sufficient power delivery for stable flight while maintaining a manageable weight profile. Proper charging protocols were strictly followed to ensure safety and maximize the operational lifespan of the power unit.




Figure 13 : Lithium Polymer Battery

Robu.in

## 4.3. Electronic Speed Controller (ESC)

The Electronic Speed Controller (ESC) is responsible for regulating the speed of the brushless DC motors by managing the electrical power supplied to them. Acting on signals from the flight controller, the ESCs adjust motor speed in real-time to achieve the desired flight dynamics.

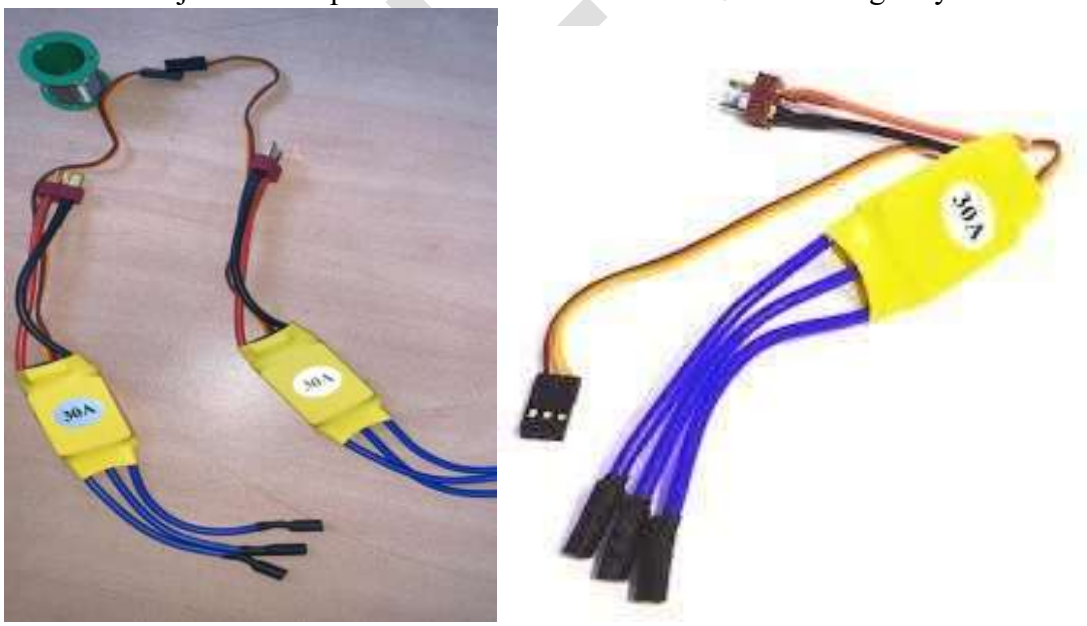



Figure 14: Electronic Speed Controller

To handle the current requirements of the propulsion system, 30-Ampere ESCs were installed for each motor. These controllers ensure smooth, responsive motor performance, which is essential for maintaining stability during complex task.

## 4.4. Transmitter and Receiver

The transmitter and receiver establish the vital communication link between the operator and the drone. The transmitter broadcasts control inputs such as throttle, pitch, roll, and yaw—which are captured by the onboard receiver and relayed to the flight controller.

A Flysky transmitter and receiver system was implemented for this project. Flysky systems are widely favoured in quadcopter development for their reliable signal range and straightforward configuration. This setup allows for precise manual control during testing phases, with the receiver ensuring consistent signal integrity.






Robu.in

Figure 15: Fly Sky Remote

## 4.5. Propellers

Propellers convert the rotational motion of the motors into aerodynamic lift by pushing air downward. The specific size and pitch of the propellers are major determinants of thrust generation and energy efficiency.

The project utilized 10-inch propeller pairs. This size is well suited for medium-scale quadcopters, providing ample thrust when paired with the selected motor and battery configuration. To minimize vibration and ensure smooth flight characteristics, the propellers were carefully balanced prior to installation.





Figure 16: Drone Propellors

## 4.6. Flight Controller with shock absorber

The flight controller functions as the "brain" of the quadcopter. It continuously processes data from onboard sensors including gyroscopes and accelerometers and autonomously adjusts motor speeds to maintain orientation and stability.

The APM 2.8 flight controller was selected for this build. As a versatile open-source platform, the APM 2.8 supports advanced autonomous flight modes and full integration with Mission Planner software. This allows for features such as automatic stabilization, altitude hold, and waypoint-based navigation.

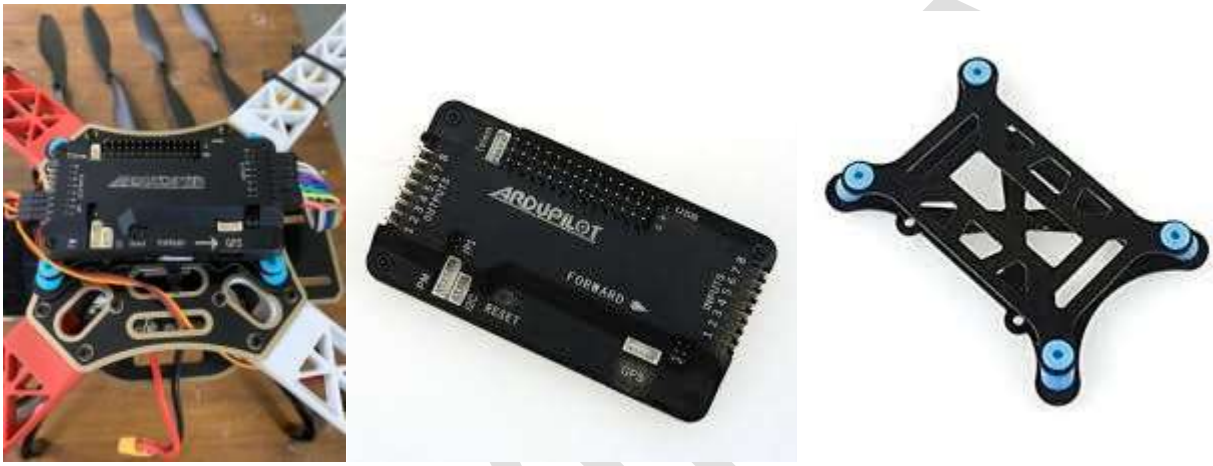
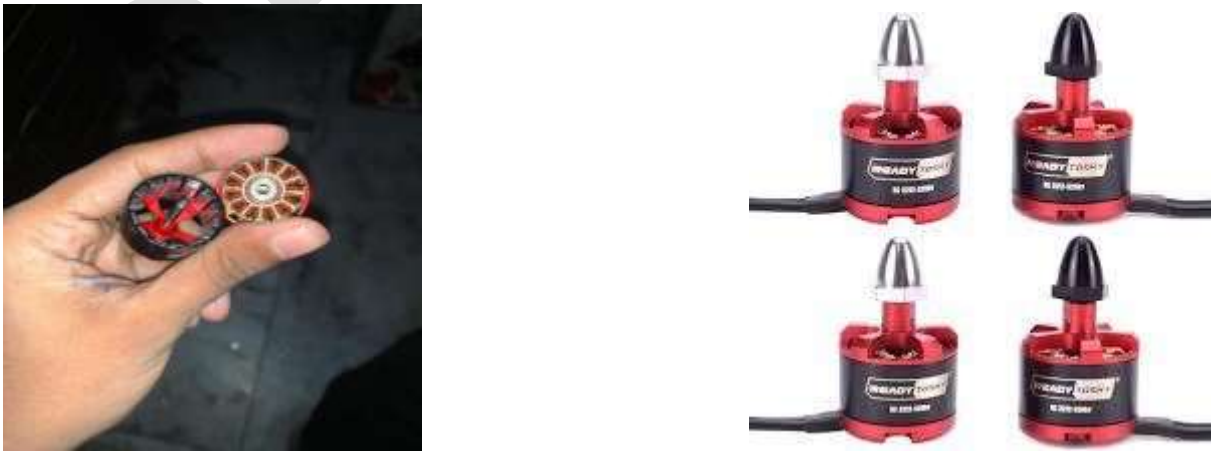



Figure 17: Flight Controller

## 4.7. Motors

Motors are the primary propulsion units, driving the propellers to generate the lift required for flight. Brushless DC motors are the standard for multirotor applications due to their durability, high efficiency, and superior power-to-weight ratio.

Ready2Sky brushless motors were used to power the quadcopter. Selected for their reliability, these motors provide consistent performance when matched with the 10-inch propellers and 30A ESCs, contributing to both stable hovering and efficient forward flight.



robu.in

Figure 18: Motors

## 4.8. Male Connector

Reliable power transmission between the high-capacity battery and the quadcopter's circuitry is essential for safety and performance. To facilitate this, a male power connector was utilized as the primary interface for the power system.

This connector was soldered directly to the power distribution board (PDB), establishing a secure yet detachable link for the battery. This setup allows for quick battery swaps while ensuring efficient current flow to all electronic components during flight. Special attention was given to the soldering quality and insulation of the connector to prevent voltage drops, short circuits, or overheating under load.



Figure 19: Male Connector

## 4.9. Camera

As the primary sensor for a surveillance drone, the camera is responsible for acquiring the visual data necessary for both monitoring and automated analysis. The effectiveness of the entire system relies heavily on the quality and stability of the video feed it produces.

For this build, a dedicated camera module was mounted to the frame to capture live footage. Beyond simple observation, this camera serves as the input source for the onboard computer vision algorithms. To ensure the footage is usable for object detection, the camera was securely mounted using vibration dampening materials to minimize the "jello effect" caused by motor vibrations.



RCdrone.com

Figure 20: Analog Camera

## 4.10. Video Transmitter (VTX) and Video Receiver (VRX)

To maintain real-time situational awareness, the drone requires a wireless link to beam visual data back to the operator. This is achieved through a video telemetry system consisting of a Video Transmitter (VTX) and a Video Receiver (VRX).
The VTX is connected directly to the onboard camera, broadcasting the video signal over a specific radio frequency. On the ground, the VRX intercepts this signal, displaying the live feed on a monitor. This setup allows the operator to see exactly what the drone sees in real-time, which is critical for manual piloting during surveillance missions and ensures the operator isn't flying "blind" if the drone goes beyond visual line of sight.




RCdrone.com

Figure 21: VTX and VRX

## 5. Assembly of the Quadcopter Drone

The assembly of a quadcopter drone is a critical phase of development, as proper mechanical alignment, electrical connections, and component placement directly affect flight stability, safety, and overall performance. In this project, the quadcopter was assembled systematically by integrating the power distribution system, propulsion units, control electronics, and communication components within the F450 frame. The complete assembly process is described in detail below.

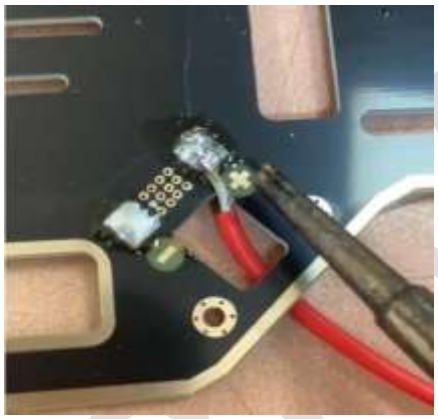
### 5.1. Integration of Power Distribution Board (PDB)

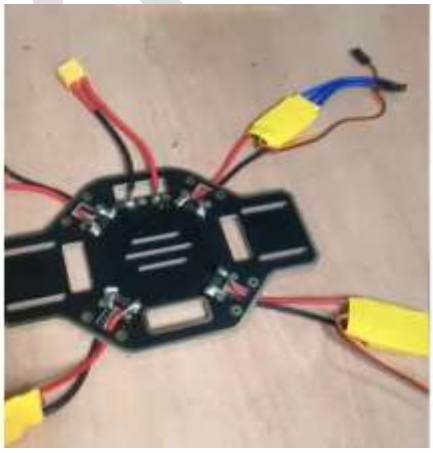
The assembly process began with the **Power Distribution Board (PDB)**, which serves as the central hub for distributing electrical power from the battery to all electronic components, particularly the ESCs and motors.
The PDB used in this project is a part of the **F450 quadcopter frame** and is designed to fit securely within the frame structure.
Initially, the PDB was positioned at the centre of the frame. A **male connector** was then soldered to the designated power input terminals of the PDB. This connector acts as the interface between the battery and the drone’s electrical system, allowing easy and secure battery connection during

operation. Following this, the **Electronic Speed Controllers (ESCs)** were soldered to the corresponding positive and negative terminals of the PDB. Care was taken to ensure correct polarity and strong solder joints to minimize electrical resistance and prevent power loss or overheating.

Figure 22: Power Distribution Board







## 5.2. Mounting of Motors on the Frame

After completing the PDB connections, the **Ready2Sky brushless motors** were mounted onto the quadcopter frame. Each motor was placed at the end of one arm of the F450 frame.
Using an **Allen key**, the motors were firmly screwed onto the motor mounts provided on the frame. Proper tightening was ensured so that the motors remained rigid during high-speed rotation while avoiding excessive force that could damage the motor mounts. Correct motor placement is essential for balanced thrust generation and stable flight.



Figure 23: Mounting of Motors on the frame

### 5.3. Frame Assembly and Enclosure of PDB

Once the motors were mounted, the **frame arms were secured to the PDB**, completing the primary structural assembly. The quadcopter frame was then further reinforced by attaching an additional **lower PDB/frame plate** beneath the main PDB.

This step enclosed the internal wiring and created a **hollow chamber** within the frame. The enclosed design protects internal components from external damage and provides a dedicated space for placing the battery and managing cables neatly.

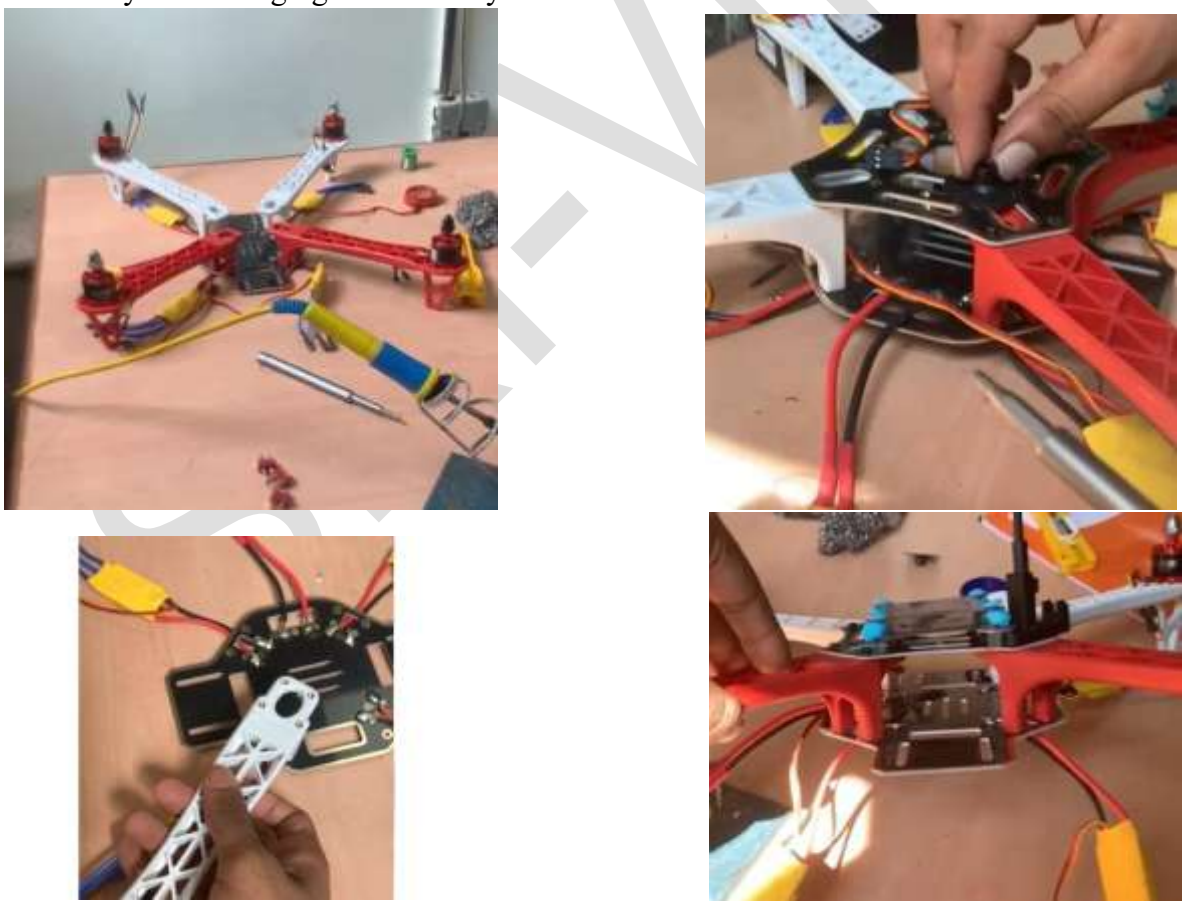


Figure 24: Frame Assembly

## 5.4. Battery Placement and Securing Mechanism

The **3300 mAh battery** was placed inside the hollow chamber formed within the frame. Proper battery placement is important to maintain the centre of gravity of the drone and ensure balanced flight.

To secure the battery:

- A **battery strap** was used to firmly hold the battery in place.
- **Double-sided tape** was additionally applied to prevent unwanted movement due to vibrations during flight.

This combination of strapping and adhesive mounting ensured that the battery remained stable throughout operation while still allowing easy removal for charging or replacement. After that fix shock absorber firmly at centre and at last strap ESCs with Zip ties too.

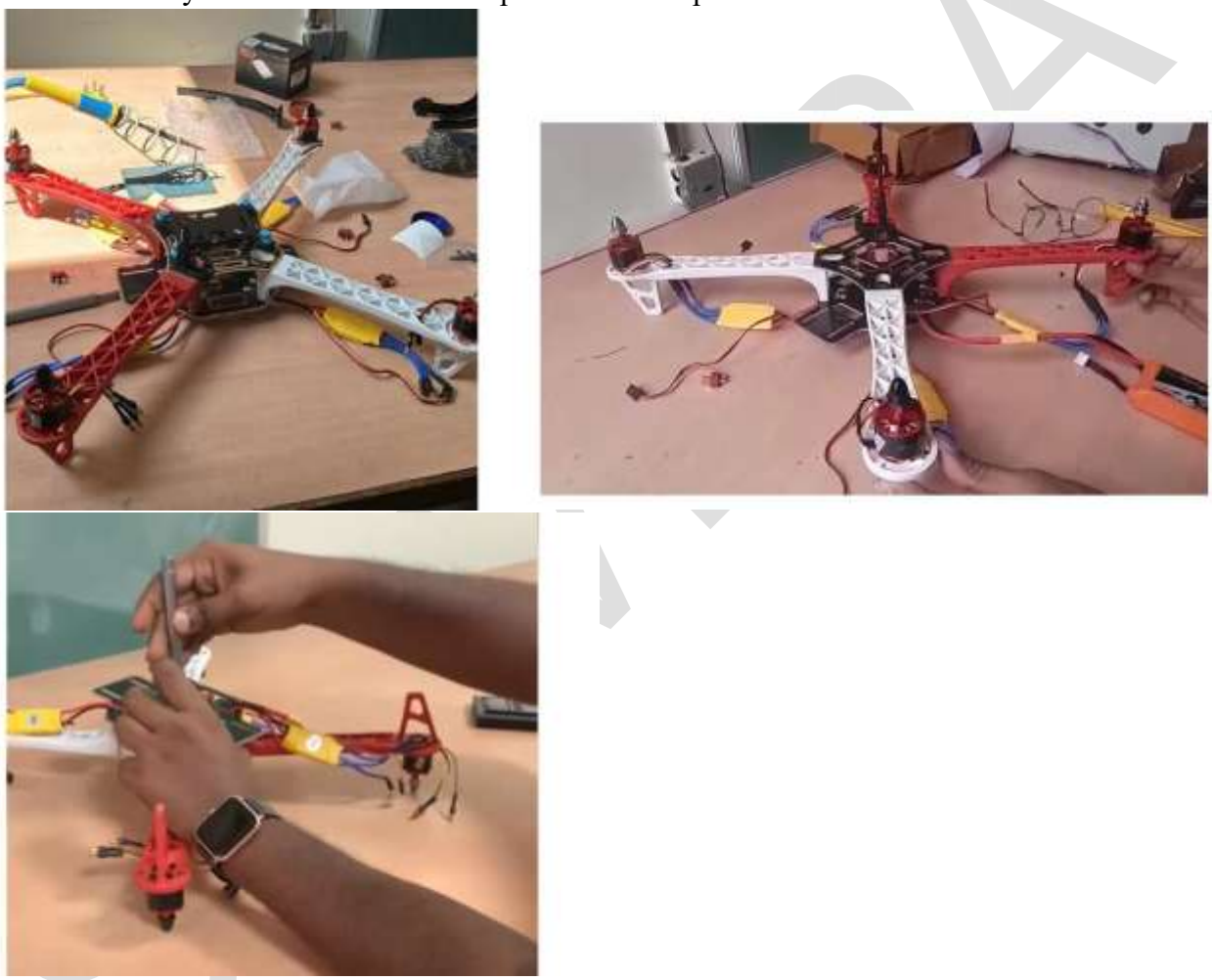


Figure 25: Battery Placement

## 5.5. Mounting of Flight Controller (APM 2.8)

The **APM 2.8 flight controller** was mounted on the **upper side of the frame**, directly above the PDB. The flight controller was placed **exactly at the centre of the drone**, which is crucial for accurate sensor readings and stable control.

Double-sided tape was used to mount the flight controller, providing vibration damping and secure attachment. Special attention was given to **orientation**, ensuring that the front of the flight controller faced the **two white arms of the F450 frame**, which were designated as the front side of the drone.

Correct orientation is essential for proper interpretation of sensor data and control commands. After mounting the flight controller, the **signal cables from each ESC** were connected to the appropriate motor output pins on the APM 2.8 flight controller.

Each ESC was connected according to the motor numbering configuration required by the flight controller firmware. These connections enable the flight controller to regulate motor speeds individually, allowing control over roll, pitch, yaw, and altitude. Care was taken to ensure secure connections and proper cable routing to avoid interference with moving parts. At last while fixing centre also place the Drone's stand, fixing them beneath the functioning drone.

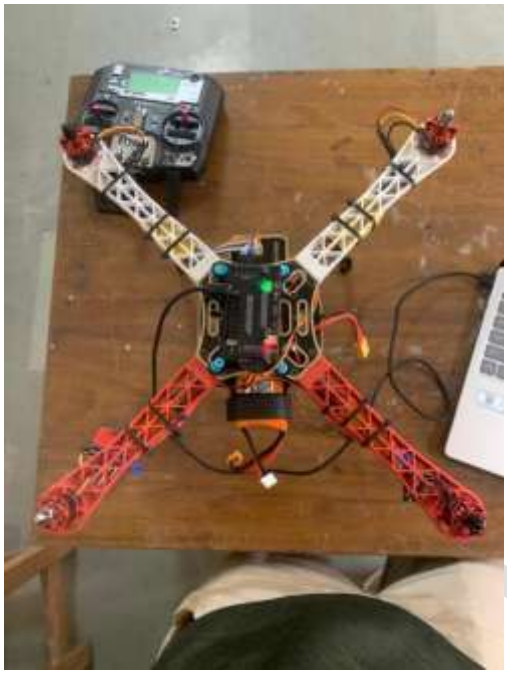




Figure 26: Mounting the FC

## 5.6. Mounting of Receiver

The **receiver**, which communicates with the Flysky transmitter held by the pilot, was mounted on the **bottom side of the drone**. The receiver is responsible for receiving control signals from the transmitter and relaying them to the flight controller.

Double-sided tape was used to mount the receiver securely to the frame. The receiver was positioned away from high-current power lines to reduce electrical interference and ensure reliable signal reception during flight.



Figure 27: Mounted Receiver

### 5.7. Completion of Mechanical and Electrical Assembly

After all components were mounted and connected, the overall assembly was inspected to verify:

- Secure mounting of all components
- Proper cable management
- Correct orientation of the flight controller
- Firm solder joints and electrical connections

At this stage, the mechanical and electrical assembly of the quadcopter drone was completed, making the system ready for software configuration, calibration, and testing.

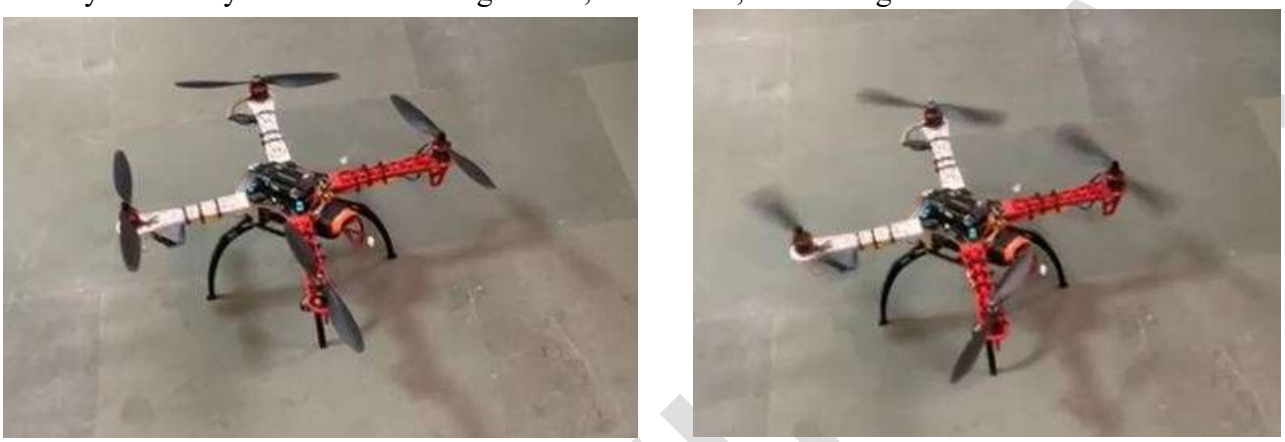


Figure 28: Finally Assembled Drone

## 6. Flight Controller and Mission Planner

The flight controller acts as the central control unit of a quadcopter, responsible for maintaining stability, interpreting pilot commands, and executing autonomous functions. In this project, the **APM 2.8 flight controller** was used in conjunction with **Mission Planner software** for configuration, calibration, and flight setup.

Mission Planner is a widely used ground control station (GCS) software for ArduPilot-based flight controllers. It provides an interface for firmware configuration, sensor calibration, parameter tuning, and flight mode assignment. Due to the legacy nature of the APM 2.8 flight controller, an older compatible version of Mission Planner was required for reliable operation.

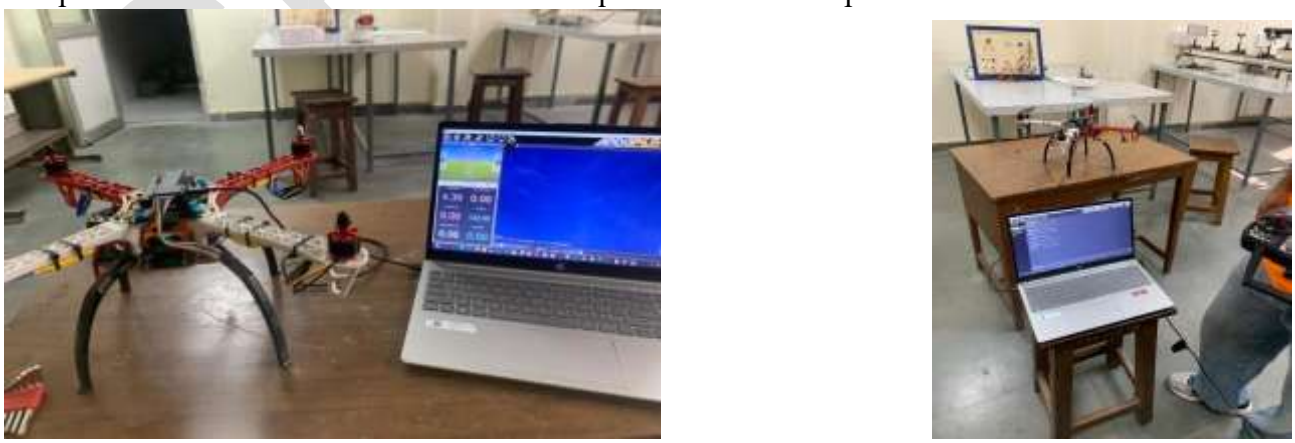


Figure 29: Setting up the Mission Planner

## 6.1. Mission Planner Software

For this project, **Mission Planner version 1.3.62** was used. The latest versions of Mission Planner were not utilized because **APM 2.8 is a retired flight controller**, and newer software versions may not fully support its firmware and hardware architecture. Version 1.3.62 provides stable compatibility with APM based systems and allows access to essential configuration and calibration features.

The flight controller was connected to the computer via USB, and Mission Planner was used to establish communication, verify connection status, and access configuration menus.

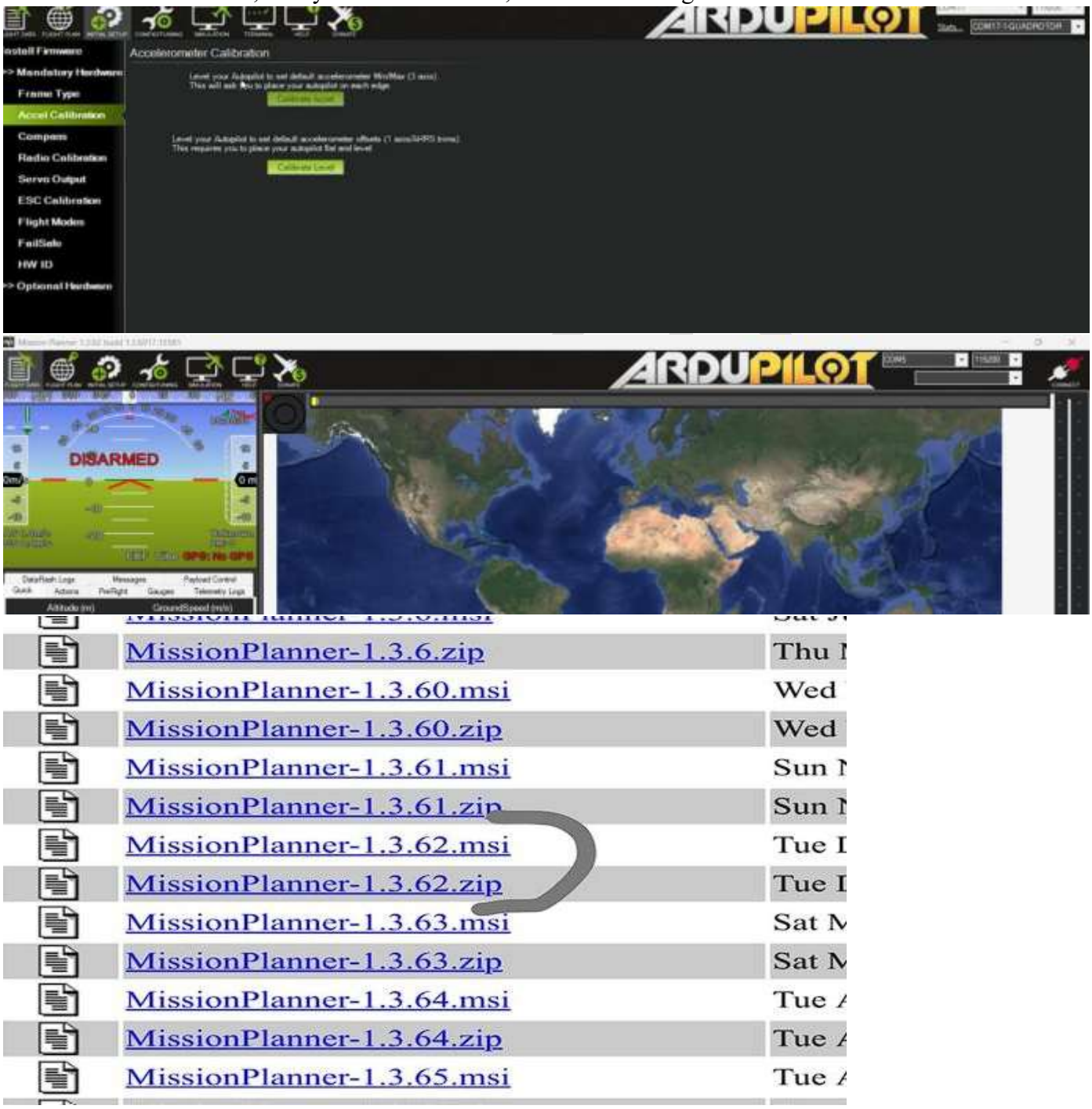



Figure 30: Mission Planner

## 6.2. Initial Setup and Frame Configuration

After establishing a successful connection between the flight controller and Mission Planner, the quadcopter frame type was selected according to the physical configuration of the drone. Selecting the correct frame type ensures that motor outputs and control algorithms correspond accurately to the quadcopter layout.

Basic parameters related to the quadcopter configuration were verified before proceeding to sensor calibration. This step ensures that the flight controller interprets control inputs and sensor data correctly.

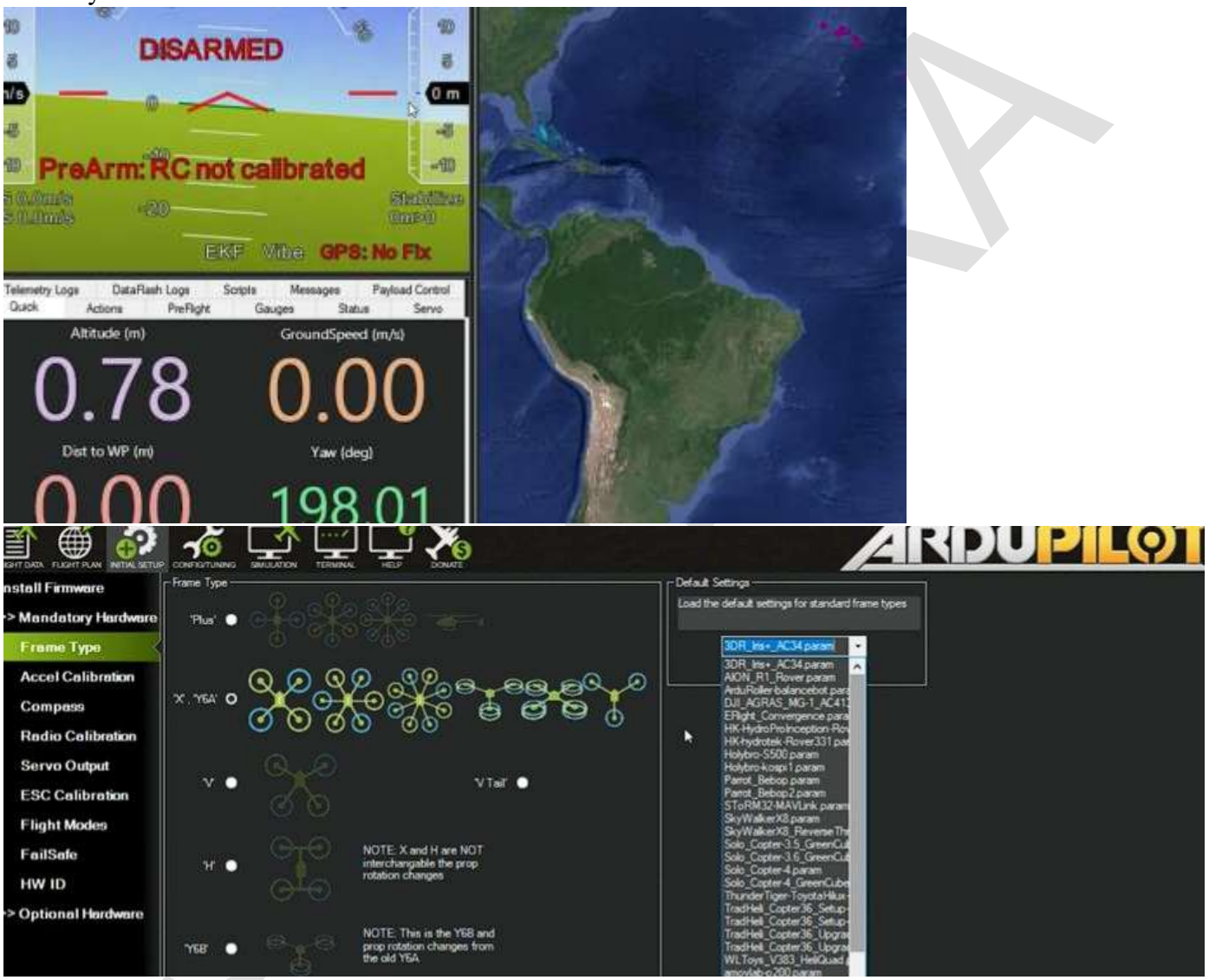



Figure 31: Initial setup on Ardupilot

## 6.3. Mandatory Hardware Calibration

Calibration of onboard sensors and control systems is essential to ensure stable and safe flight. The following mandatory calibrations were performed using Mission Planner:

### 6.3.1. Accelerometer Calibration

Accelerometer calibration was performed to allow the flight controller to correctly identify the drone's orientation in three-dimensional space. During this process, the quadcopter was placed in multiple

orientations as instructed by Mission Planner, including level, left, right, nose up, nose down, and upside down positions.

This calibration enables accurate roll, pitch, and yaw estimation and is critical for maintaining stability during flight.




Figure 32: Calibration of Drone

### 6.3.2. Radio Calibration

Radio calibration was carried out to ensure proper communication between the **Flysky transmitter** and the receiver mounted on the drone. This process involved moving all control sticks and switches on the transmitter to their full range of motion so that Mission Planner could record the minimum and maximum signal values.

Radio calibration ensures that pilot inputs such as throttle, roll, pitch, and yaw are correctly interpreted by the flight controller.

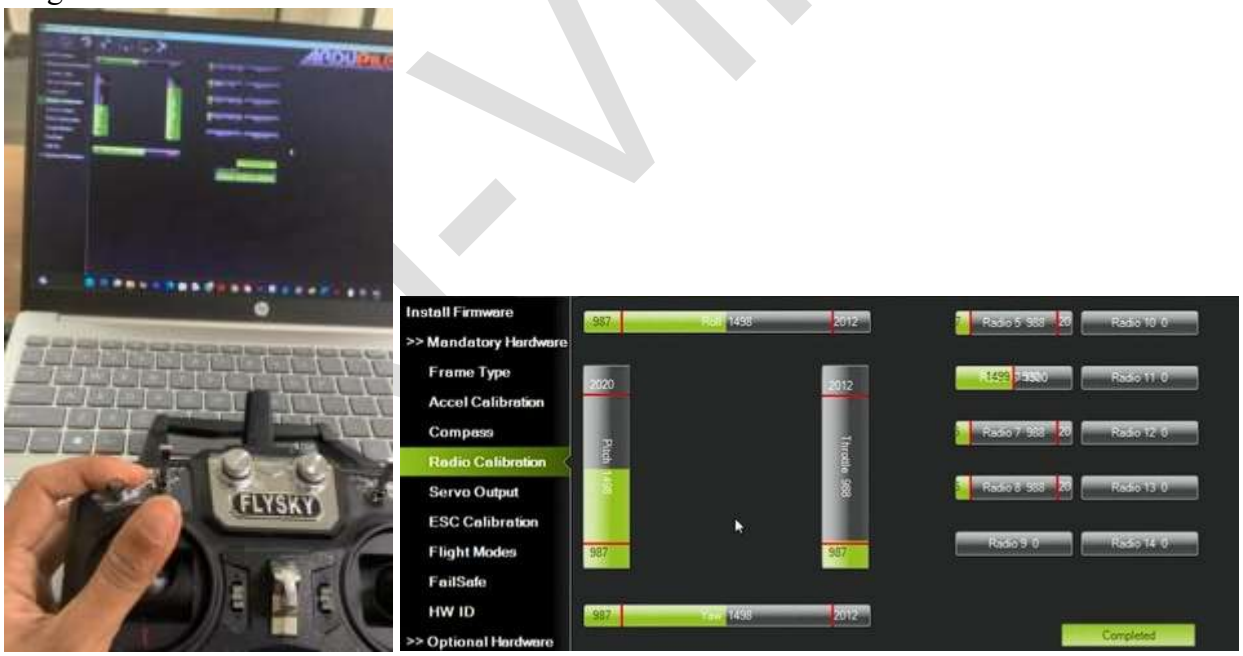



Figure 33: Radio Calibration

### 6.3.3. ESC Calibration

Electronic Speed Controller (ESC) calibration was necessary to synchronize all motors and ensure uniform throttle response. However, due to the retired status of the **APM 2.8 flight controller**, ESC calibration could not be performed directly through Mission Planner.

Instead, **traditional manual ESC calibration** was used. This involved disconnecting and reconnecting the battery while following audible beep signals from the ESCs, which confirmed successful calibration. This method ensures that all ESCs recognize the same throttle range, resulting in balanced motor performance.






Figure 34: Calibrated ESC

## 6.4. Flight Mode Configuration

After completing mandatory calibrations, various **flight modes** were configured using Mission Planner.

Flight modes define how the drone behaves under different control conditions and levels of autonomy. Using the flight mode setup menu, multiple modes were assigned to the transmitter switches. These modes enable the pilot to switch between stabilized flight, assisted flight, and other operational states during flight testing and operation.

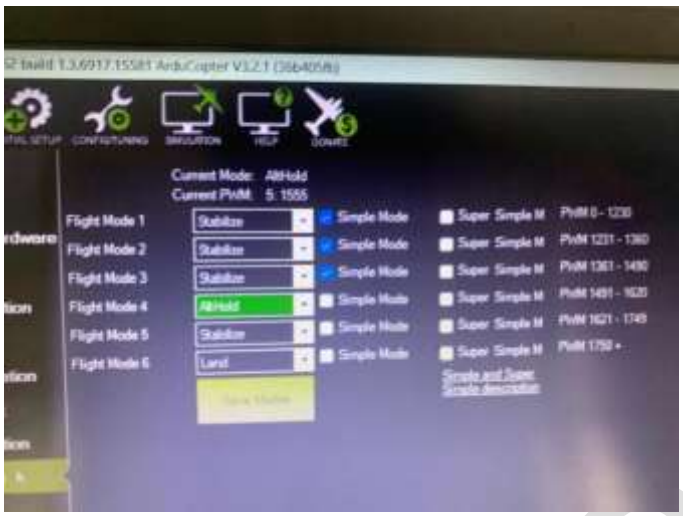



Figure 35: Flight Mode Config

## 6.5. Optional Hardware Configuration and Motor Testing

In the **Optional Hardware** section of Mission Planner, additional checks and verifications were performed:

- **Motor rotation direction** was verified to ensure that each motor rotated in the correct direction as required for quadcopter operation.
- Motor response was tested individually to confirm correct ESC-to-motor mapping.
- Minor **fine-tuning adjustments** were carried out by slightly shifting **roll and yaw parameters** to improve stability and responsiveness.

These adjustments helped achieve smoother control and reduced drift during flight tests.




Figure 36: Motor Testing

T-VIPRA

## 6.6. Compass Calibration

Compass calibration is an essential step in configuring a quadcopter drone, as it allows the flight controller to accurately determine the drone's heading relative to the Earth's magnetic field. Proper compass calibration is particularly important for navigation, waypoint missions, and maintaining correct orientation during flight.

In this project, compass calibration was performed using **Mission Planner** after completing the mandatory hardware setup. The calibration process involved rotating the drone in multiple orientations, allowing the onboard magnetometer to record magnetic field data from all axes. Mission Planner uses this data to compensate for magnetic interference caused by onboard electronics and the surrounding environment.

Additionally, the **geographical location and coordinates** were set within Mission Planner to ensure accurate alignment with the Earth's magnetic reference. Providing correct location information helps the flight controller calculate heading and navigation parameters more accurately, which is especially important for autonomous and assisted flight modes.

Care was taken to perform the compass calibration in an open area, away from large metal objects and electrical interference, to minimize errors. Successful compass calibration ensures reliable yaw control, accurate directional movement, and stable navigation during flight operations.

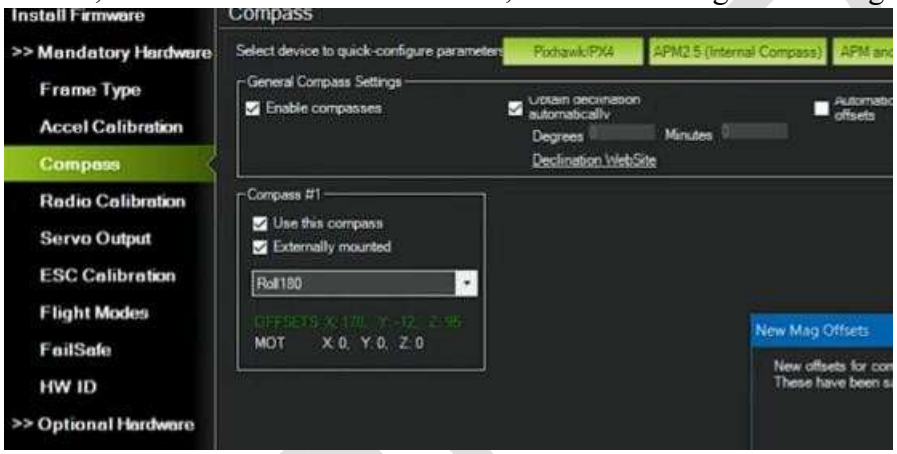



Figure 37: Final Compass Calibration

## 6.7. Final Verification and Readiness for Flight

After completing all calibrations and configurations, the system was thoroughly reviewed to ensure:

- Correct sensor calibration
- Proper motor operation
- Accurate radio response
- Correct flight controller orientation

At this stage, the quadcopter was fully configured through Mission Planner and ready for controlled flight testing and further software integration.

# 7. FINAL SOFTWARE IMPLEMENTATION

The software development phase of this project was focused on transforming the quadcopter drone into a smart surveillance system capable of real-time visual analysis and intelligent interpretation. This software was designed and implemented, integrating a state-of-the-art object detection framework with custom-built surveillance features and an agentic AI layer.

The system combines the detection capabilities of the YOLO framework with additional logic for monitoring, recording, security enforcement, and natural language interpretation of events.

## 7.1. Study and Selection of YOLO Framework

The development process began with a detailed study of the **Ultralytics YOLO framework**, which is widely used for real-time object detection and tracking. After exploring the available versions and model variants, **YOLO11** was selected due to its efficiency and suitability for real-time applications. For implementation, the **YOLO11 Nano (YOLO11n)** model was chosen. The nano variant is optimized for speed and low computational overhead, making it appropriate for live surveillance tasks while still maintaining acceptable detection accuracy.

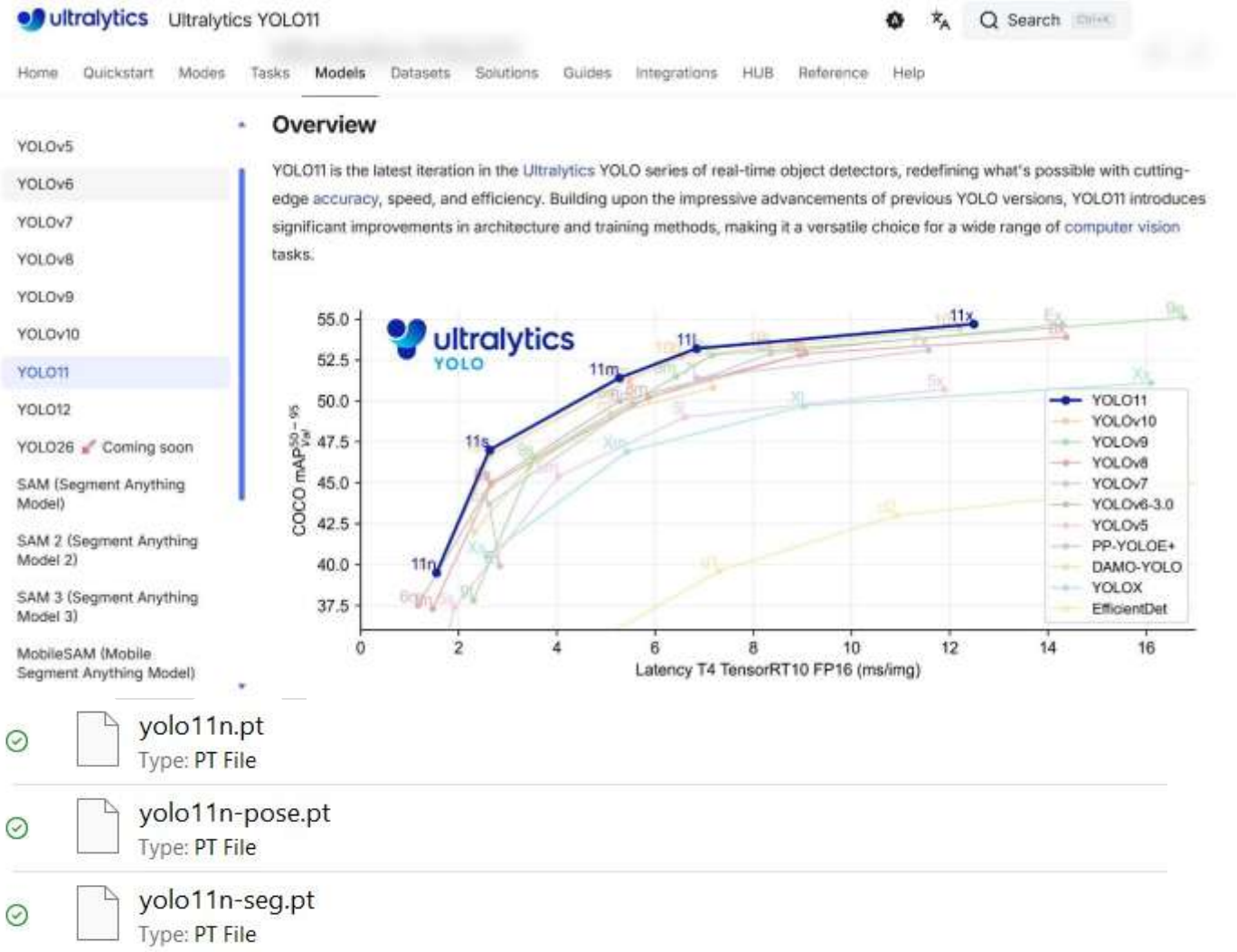



Figure 38: YOLO software

## 7.2. YOLO Capabilities Utilized in the Project

The YOLO11 framework provides multiple built-in computer vision capabilities. The following functionalities were **achieved and utilized** during development:

### 7.2.1. Object and People Detection

Using YOLO11n in detection mode, the system is capable of identifying various objects such as humans, animals, and common everyday items. The model produces bounding boxes, class labels, and confidence scores for each detected object in real time.

Detection mode was chosen as the core operating mode because it allows further extension into tracking and higher-level surveillance logic.

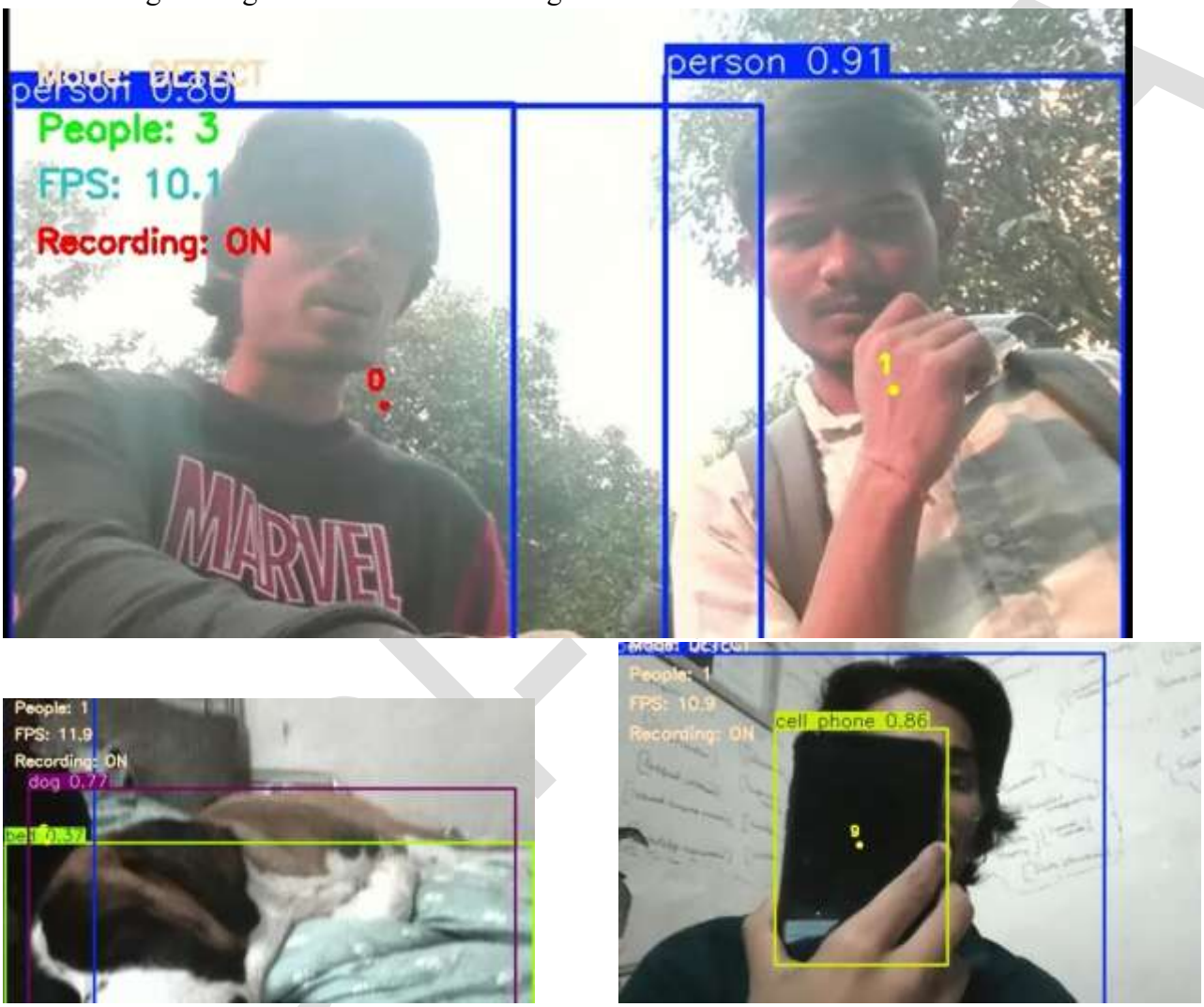



Figure 39: Object and People Detection

### 7.2.2. Segmentation

YOLO11 also supports segmentation, where objects are separated at the pixel level instead of being enclosed only by bounding boxes. This capability was explored and tested as part of understanding the framework, although it was not used as the primary mode due to higher computational requirements.




Figure 40: Segmentation

### 7.2.3. Positioning

The positioning capability of YOLO provides spatial information regarding detected objects within the frame. This feature helps in understanding object location and relative movement, which is useful for surveillance and navigation-aware applications.




Figure 41: Positioning

### 7.2.4. Object Tracking with ID Assignment

The YOLO framework supports **ID-based object tracking**, where each detected object is assigned a unique identifier. This allows the system to track the same object across multiple frames, even when temporary occlusions occur. This tracking capability forms the foundation for advanced monitoring and behavior analysis.




Figure 42: Object Tracking and ID Assignment

## 7.3. Custom Video Recording and Data Storage System

Beyond YOLO's built-in capabilities, a **custom video recording and data storage system** was developed as part of this project.

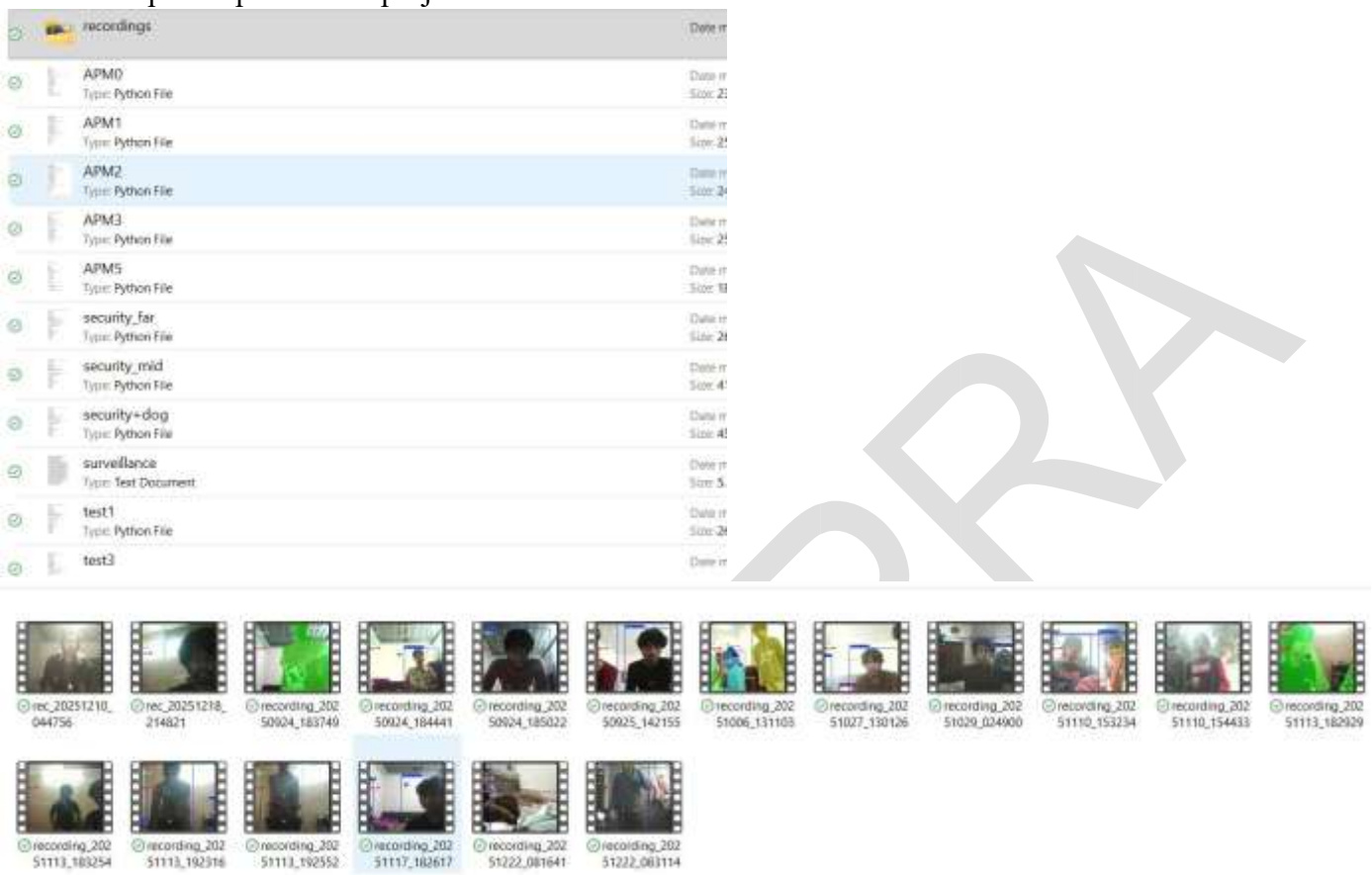



Figure 43: Custom Video Recording

The software records the live video feed during surveillance operations and automatically stores the footage in a dedicated recording directory within the project folder. Each recording session is saved systematically, enabling easy retrieval and review of surveillance data. This feature is essential for postevent analysis and documentation.

## 7.4. Polygon-Based Security and Breach Detection System

A significant contribution of this project is the implementation of a **region-based security system using polygon boundaries**.

In this system:

- The operator can draw **red polygonal lines or regions** on the live video feed.
- These polygons represent virtual security barriers or restricted zones.
- When a detected object crosses or enters the defined region, the system identifies it as a **barrier or boundary breach**.

A key feature of this system is its **dynamic nature**. The polygon boundaries can be:

- Added or removed at any time
- Repositioned anywhere on the screen

- Modified even while live detection is running

This functionality makes the system highly flexible and suitable for real-time surveillance and perimeter security applications.




Figure 44: Polygon Based Security and Breach Detection

## 7.5. Integration of LLM-Based Agentic AI using Ollama

To enhance the intelligence of the surveillance system, an **agentic AI layer** was integrated using a **Large Language Model (LLM)** through **Ollama**.

In this implementation:

- The YOLO detection and tracking outputs are converted into structured prompts.
- These prompts are passed to the LLM via an agentic pipeline.
- The LLM interprets the visual data and generates **natural language descriptions** of what is happening in the live scene.

This allows the system to explain detected activities in human-readable form, improving situational awareness and enabling more intuitive monitoring. The integration of computer vision with an LLM represents an advanced and modern approach to intelligent surveillance systems.

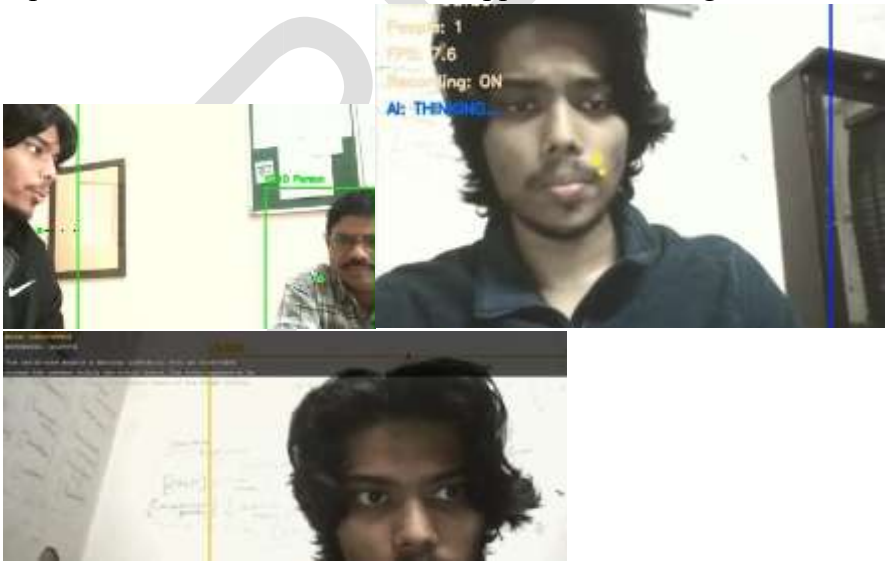


Figure 45: Integration of LLM-Based Agentic AI using Ollama

## 7.6. Overall Software Architecture and Workflow

The complete software workflow developed in this project can be summarized as follows:

1. Live video input from the onboard camera
2. Real-time object detection and ID-based tracking using YOLO11n
3. Optional use of segmentation and positioning capabilities
4. Custom video recording and organized data storage
5. Polygon-based region monitoring and breach detection
6. Event interpretation using LLM-based agentic AI

This modular architecture allows the surveillance system to be extended further for additional features such as alerts, autonomous response, or cloud-based monitoring.

SRI-VIPRA

CODE :

```
# smart_live_ai.= by samarhtkratos
# Complete version with idle detection, enhanced breach notification, and segmentation

import time, math, os
from collections import deque
import cv2
import numpy as np
from ultralytics import YOLO
import subprocess
import threading

# -------- Config (samarth) --------
DETECT_MODEL = "yolo11n.pt"
POSE_MODEL = "yolo11n-pose.pt"
SEGMENT_MODEL = "yolo11n-seg.pt"  # Added segmentation model
SOURCE_INDEX = 0
IMG_SIZE = (640, 480)
MAX_DISTANCE = 80
RECORDINGS_FOLDER = "recordings"  # Folder for recorded videos
MAX_RED_LINES = 20  # Maximum number of red lines
OLLAMA_INTERVAL = 5  # Seconds between AI analysis
ZONE_ALERT = 3  # Alert when this many people in zone
# ----------------------------------------

# Define zone polygon :
ZONE_POLY = np.array([[100, 100], [500, 100], [500, 400], [100, 400]], np.int32)

# Create recordings folder if it doesn't exist :
if not os.path.exists(RECORDINGS_FOLDER):
    os.makedirs(RECORDINGS_FOLDER)

STATE = {
    "mode": "detect",
    "zones": False,
    "hide_interface": False,
    "recording": False,
    "video_writer": None,
    "recording_start_time": None,
    "red_lines": [],  # List to store multiple red lines
    "breach_detected": False,
    "breach_id": None,
    "breach_line_idx": None,
    "total_count_mode": False,
    "total_count": 0,
    "seen_ids": set(),
    "drawing_line": False,
    "line_start": None,
```

```
    "drawing_line": False,
    "line_start": None,
    "line_end": None,
    "current_line_index": -1,
    "ai_analysis": False,
    "dog_count": 0,
    "ollama_enabled": False,
    "ollama_response": "",
    "last_ollama_time": 0,
    "show_masks": True,  # Toggle for showing segmentation masks
    "mask_alpha": 0.4,   # Transparency for masks
}

# Global variable for webcam
cap = None

# Person tracking system
class PersonTracker:
    def __init__(self, max_age=30):
        self.next_id = 0
        self.people = {}
        self.colors = {}
        self.max_age = max_age
        self.pose_history = {}  # Store pose history for action recognition
        self.pose_smoothing = {}  # For stabilizing pose dots

    def get_color(self, id):
        if id not in self.colors:
            hue = (id * 30) % 180
            color = cv2.cvtColor(np.uint8([[[hue, 255, 255]]]), cv2.COLOR_HSV2BGR)[0][0]
            self.colors[id] = (int(color[0]), int(color[1]), int(color[2]))
        return self.colors[id]

    def update(self, centers, keypoints=None):
        current_time = time.time()
        # Remove old entries
        to_remove = []
        for id, data in self.people.items():
            if current_time - data['last_seen'] > self.max_age:
                to_remove.append(id)
        for id in to_remove:
            del self.people[id]
            if id in self.pose_history:
                del self.pose_history[id]
            if id in self.pose_smoothing:
                del self.pose_smoothing[id]

                del self.pose_smoothing[id]

        # Update or assign IDs
        current_assignments = {}
        for i, (cx, cy) in enumerate(centers):
            best_id = None
            best_dist = float('inf')

            # Find the closest existing person
            for id, data in self.people.items():
                dist = math.hypot(cx - data['position'][0], cy - data['position'][1])
                if dist < best_dist and dist < MAX_DISTANCE:
                    best_dist = dist
                    best_id = id

            if best_id is not None:
                # Update existing person
                self.people[best_id] = {'last_seen': current_time, 'position': (cx, cy)}
                current_assignments[best_id] = (cx, cy)

                # Update pose history if keypoints are available
                if keypoints is not None and i < len(keypoints):
                    if best_id not in self.pose_history:
                        self.pose_history[best_id] = deque(maxlen=10)  # Store last 10 poses
                        self.pose_smoothing[best_id] = {}

                    # Smooth the keypoints
                    smoothed_keypoints = []
                    for j, kp in enumerate(keypoints[i]):
                        # Handle both (x, y, conf) and (x, y) formats
                        if len(kp) == 3:
                            x, y, conf = kp
                        else:
                            x, y = kp
                            conf = 1.0  # Default confidence

                        if best_id in self.pose_smoothing and j in self.pose_smoothing[best_id]:
                            # Apply simple moving average for stabilization
                            old_x, old_y = self.pose_smoothing[best_id][j]
                            smooth_x = 0.7 * old_x + 0.3 * x
                            smooth_y = 0.7 * old_y + 0.3 * y
                            smoothed_keypoints.append((smooth_x, smooth_y, conf))
                            self.pose_smoothing[best_id][j] = (smooth_x, smooth_y)
                        else:
                            if best_id not in self.pose_smoothing:
                                self.pose_smoothing[best_id] = {}
                            smoothed_keypoints.append((x, y, conf))
```

```
        del self.pose_smoothing[id]

# Update or assign IDs
current_assignments = {}
for i, (cx, cy) in enumerate(centers):
    best_id = None
    best_dist = float('inf')

    # Find the closest existing person
    for id, data in self.people.items():
        dist = math.hypot(cx - data['position'][0], cy - data['position'][1])
        if dist < best_dist and dist < MAX_DISTANCE:
            best_dist = dist
            best_id = id

    if best_id is not None:
        # Update existing person
        self.people[best_id] = {'last_seen': current_time, 'position': (cx, cy)}
        current_assignments[best_id] = (cx, cy)

        # Update pose history if keypoints are available
        if keypoints is not None and i < len(keypoints):
            if best_id not in self.pose_history:
                self.pose_history[best_id] = deque(maxlen=10)  # Store last 10 poses
                self.pose_smoothing[best_id] = {}

            # Smooth the keypoints
            smoothed_keypoints = []
            for j, kp in enumerate(keypoints[i]):
                # Handle both (x, y, conf) and (x, y) formats
                if len(kp) == 3:
                    x, y, conf = kp
                else:
                    x, y = kp
                    conf = 1.0  # Default confidence

                if best_id in self.pose_smoothing and j in self.pose_smoothing[best_id]:
                    # Apply simple moving average for stabilization
                    old_x, old_y = self.pose_smoothing[best_id][j]
                    smooth_x = 0.7 * old_x + 0.3 * x
                    smooth_y = 0.7 * old_y + 0.3 * y
                    smoothed_keypoints.append((smooth_x, smooth_y, conf))
                    self.pose_smoothing[best_id][j] = (smooth_x, smooth_y)
                else:
                    if best_id not in self.pose_smoothing:
                        self.pose_smoothing[best_id] = {}
                    smoothed_keypoints.append((x, y, conf))
                    self.pose_smoothing[best_id][j] = (x, y)

            self.pose_history[best_id].append((smoothed_keypoints, current_time))
        else:
            # New person
            new_id = self.next_id
            self.next_id += 1
            self.people[new_id] = {'last_seen': current_time, 'position': (cx, cy)}
            current_assignments[new_id] = (cx, cy)

            # Initialize pose history if keypoints are available
            if keypoints is not None and i < len(keypoints):
                self.pose_history[new_id] = deque(maxlen=10)
                self.pose_smoothing[new_id] = {}

                # Store initial keypoints
                smoothed_keypoints = []
                for j, kp in enumerate(keypoints[i]):
                    # Handle both (x, y, conf) and (x, y) formats
                    if len(kp) == 3:
                        x, y, conf = kp
                    else:
                        x, y = kp
                        conf = 1.0  # Default confidence

                    smoothed_keypoints.append((x, y, conf))
                    self.pose_smoothing[new_id][j] = (x, y)

                self.pose_history[new_id].append((smoothed_keypoints, current_time))

    return current_assignments

def reset(self):
    self.next_id = 0
    self.people = {}
    self.colors = {}
    self.pose_history = {}
    self.pose_smoothing = {}

def recognize_action(self, person_id):
    """Recognize action based on pose history"""
    if person_id not in self.pose_history or len(self.pose_history[person_id]) < 2:
        return "unknown"

    # Get current and previous pose
```

```
# Get current and previous pose
current_pose, current_time = self.pose_history[person_id][-1]
prev_pose, prev_time = self.pose_history[person_id][-2]

# Calculating movement speed
time_diff = current_time - prev_time
if time_diff == 0:
    return "unknown"

# Get keypoints for wrists (9,10), elbows (7,8), shoulders (5,6)
# COCO keypoint format: 0-nose, 1-left_eye, 2-right_eye, 3-left_ear, 4-right_ear,
# 5-left_shoulder, 6-right_shoulder, 7-left_elbow, 8-right_elbow,
# 9-left_wrist, 10-right_wrist, 11-left_hip, 12-right_hip,
# 13-left_knee, 14-right_knee, 15-left_ankle, 16-right_ankle

# Check if we have enough keypoints
if len(current_pose) < 17 or len(prev_pose) < 17:
    return "unknown"

# Calculate movement of wrists
left_wrist_move = math.hypot(current_pose[9][0] - prev_pose[9][0],
                             current_pose[9][1] - prev_pose[9][1]) / time_diff
right_wrist_move = math.hypot(current_pose[10][0] - prev_pose[10][0],
                              current_pose[10][1] - prev_pose[10][1]) / time_diff

# Calculate height of wrists relative to shoulders
left_wrist_height = current_pose[9][1] - current_pose[5][1]
right_wrist_height = current_pose[10][1] - current_pose[6][1]

# Calculate if hands are above head (negative values mean above shoulders)
hands_above_head = left_wrist_height < -20 and right_wrist_height < -20

# Calculate if person is likely walking/running (hip movement)
hip_movement = math.hypot(current_pose[11][0] - prev_pose[11][0],
                          current_pose[11][1] - prev_pose[11][1]) / time_diff

# Recognize actions based on thresholds
if hands_above_head:
    return "hands up"
elif hip_movement > 20:
    return "running"
elif hip_movement > 10:
    return "walking"
elif hip_movement < 2 and (left_wrist_move < 5 and right_wrist_move < 5):
    return "idle"
elif hip_movement < 5:
```

```
            return "idle"
        elif hip_movement < 5:
            return "standing"

        return "unknown"

# Motion tracking for pose mode
class MotionTracker:
    def __init__(self, max_points=10):
        self.history = {}
        self.max_points = max_points

    def update(self, centers):
        current_time = time.time()
        for person_id, cx, cy in centers:
            if person_id not in self.history:
                self.history[person_id] = deque(maxlen=self.max_points)
            self.history[person_id].append((cx, cy, current_time))

    def get_speed(self, person_id):
        if person_id not in self.history or len(self.history[person_id]) < 2:
            return 0.0

        points = list(self.history[person_id])
        total_dist = 0
        total_time = points[-1][2] - points[0][2]

        if total_time == 0:
            return 0.0

        for i in range(1, len(points)):
            x1, y1, t1 = points[i-1]
            x2, y2, t2 = points[i]
            total_dist += math.hypot(x2 - x1, y2 - y1)

        return total_dist / total_time

# Initialize the person tracker
person_tracker = PersonTracker()

# Helper functions for pose mode
def hands_up(keypoints):
    """Check if person has hands up"""
    if len(keypoints) < 17:
        return False

    # Keypoints: 5-left shoulder, 6-right shoulder, 9-left wrist, 10-right wrist
```

```
    # Keypoints: 5-left_shoulder, 6-right_shoulder, 9-left_wrist, 10-right_wrist
    left_wrist_above_shoulder = keypoints[9][1] < keypoints[5][1] - 20
    right_wrist_above_shoulder = keypoints[10][1] < keypoints[6][1] - 20

    return left_wrist_above_shoulder or right_wrist_above_shoulder

def in_zone(polygon, x, y):
    """Check if point is inside polygon"""
    return cv2.pointPolygonTest(polygon, (x, y), False) >= 0

# AI Analysis function
def run_ai_analysis(frame):
    """Run AI analysis on the frame using Ollama"""
    try:
        # Save frame temporarily
        cv2.imwrite("temp_frame.jpg", frame)

        # Run Ollama model
        result = subprocess.run([
            "ollama", "run", "llama3.2:1b",
            "Describe this image in one sentence."
        ], capture_output=True, text=True, timeout=5)

        if result.returncode == 0:
            return result.stdout.strip()
        else:
            return "AI analysis failed"
    except:
        return "AI not available"

# Global for AI analysis result
ai_result = ""
ai_processing = False

def query_ollama_async(query):
    """Run Ollama query in a separate thread"""
    def ollama_thread():
        global ai_result, ai_processing
        ai_processing = True
        try:
            result = subprocess.run([
                "ollama", "run", "llama3.2:1b", query
            ], capture_output=True, text=True, timeout=10)

            if result.returncode == 0:
                ai_result = result.stdout.strip()
```

```
            ai_result = "AI not available"
        ai_processing = False

    threading.Thread(target=ollama_thread).start()

def init_camera():
    """Initialize the webcam with DirectShow backend"""
    global cap
    if cap is not None:
        cap.release()

    for backend in [cv2.CAP_DSHOW, cv2.CAP_MSMF, cv2.CAP_ANY]:
        for camera_index in [0, 1, 2]:
            cap = cv2.VideoCapture(camera_index, backend)
            if cap.isOpened():
                print(f"Camera initialized with index {camera_index} and backend {backend}")
                cap.set(cv2.CAP_PROP_FRAME_WIDTH, IMG_SIZE[0])
                cap.set(cv2.CAP_PROP_FRAME_HEIGHT, IMG_SIZE[1])
                return True

    print("Error: Could not initialize any camera")
    return False

def release_camera():
    """Release the webcam resource"""
    global cap
    if cap is not None:
        cap.release()
        cap = None

def get_frame():
    """Get a frame from the webcam with error handling"""
    global cap
    if cap is None or not cap.isOpened():
        if not init_camera():
            return False, None

    ret, frame = cap.read()
    if not ret:
        print("Frame read failed, reinitializing camera...")
        release_camera()
        if init_camera():
            ret, frame = cap.read()

    return ret, frame

def toggle_recording():
```

```
def toggle_recording():
    """Toggle recording state and create/release video writer"""
    if not STATE["recording"]:
        timestamp = time.strftime("%Y%m%d_%H%M%S")
        filename = os.path.join(RECORDINGS_FOLDER, f"recording_{timestamp}.mp4")

        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        fps = 20.0
        frame_size = IMG_SIZE

        STATE["video_writer"] = cv2.VideoWriter(filename, fourcc, fps, frame_size)
        STATE["recording"] = True
        STATE["recording_start_time"] = time.time()
        print(f"Started recording: {filename}")
    else:
        if STATE["video_writer"] is not None:
            STATE["video_writer"].release()
            STATE["video_writer"] = None
        STATE["recording"] = False
        duration = time.time() - STATE["recording_start_time"]
        print(f"Stopped recording. Duration: {duration:.2f} seconds")

def draw_hud(frame, fps, people_count, mode):
    h, w = frame.shape[:2]

    # Draw dog count on left
    cv2.putText(frame, f"Dogs: {STATE['dog_count']:03d}", (16, 28),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 200, 200), 2)

    if STATE["hide_interface"]:
        if not STATE["total_count_mode"]:
            cv2.putText(frame, f"People: {people_count}", (w - 120, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        return

    mode_text = "TOTAL" if STATE["total_count_mode"] else mode.upper()
    cv2.putText(frame, f"Mode: {mode_text}",
                (16, 92), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (180, 220, 255), 2)
    cv2.putText(frame, f"People: {people_count}",
                (16, 124), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)

    if STATE["total_count_mode"]:
        cv2.putText(frame, f"Total: {STATE['total_count']:03d}",
                    (16, 156), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

    cv2.putText(frame, f"FPS: {fps:.1f}", (16, 180 if STATE["total_count_mode"] else 156),
```

```
    cv2.putText(frame, recording_status, (16, 220 if STATE["total_count_mode"] else 188),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    # Display AI analysis if active
    if STATE["ai_analysis"]:
        cv2.putText(frame, "AI: ON", (16, 252 if STATE["total_count_mode"] else 220),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 100, 0), 2)

        # Show AI result if available
        global ai_result
        if ai_result and not ai_processing:
            # Wrap text if too long
            if len(ai_result) > 40:
                words = ai_result.split()
                lines = []
                current_line = ""
                for word in words:
                    if len(current_line + word) < 40:
                        current_line += word + " "
                    else:
                        lines.append(current_line)
                        current_line = word + " "
                if current_line:
                    lines.append(current_line)

                for i, line in enumerate(lines):
                    cv2.putText(frame, line, (w//2 - 200, 30 + i*25),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)
            else:
                cv2.putText(frame, ai_result, (w//2 - 200, 30),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

    # Display breach alert if detected with bigger font
    if STATE["breach_detected"]:
        breach_text = f"BREACH ALERT: Person {STATE['breach_id']:03d} at Line {STATE['breach_line_idx']+1}"
        text_size = cv2.getTextSize(breach_text, cv2.FONT_HERSHEY_SIMPLEX, 1.2, 3)[0]
        text_x = (w - text_size[0]) // 2
        text_y = 50

        # Draw background for better visibility
        cv2.rectangle(frame, (text_x - 10, text_y - text_size[1] - 10),
                      (text_x + text_size[0] + 10, text_y + 10), (0, 0, 0), -1)

        # Draw text with big font
        cv2.putText(frame, breach_text, (text_x, text_y),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255), 3)
```

```
                cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 0, 255), 3)

    help_text = "[D]etect  [P]ose  [S]egment  [T]otal  [O]Draw Line  [L]Remove Line  [Q]Reset  [R]ecord  [M]ide  [A]AI  [ESC] quit"
    cv2.putText(frame, help_text,
                (16, h - 16), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (180, 180, 180), 1)

def check_line_breach(centers):
    """Check if any person crosses any red line"""
    if not STATE["red_lines"]:
        return False, None, None

    for line_idx, (line_start, line_end) in enumerate(STATE["red_lines"]):
        for person_id, (cx, cy) in centers.items():
            dist = abs((line_end[1] - line_start[1]) * cx - (line_end[0] - line_start[0]) * cy +
                       line_end[0] * line_start[1] - line_end[1] * line_start[0]) / \
                   math.sqrt((line_end[1] - line_start[1])**2 + (line_end[0] - line_start[0])**2)

            if dist < 10:  # Threshold for line crossing
                return True, person_id, line_idx

    return False, None, None

def update_total_count(person_assignments):
    """Update the total count of unique people detected"""
    for person_id in person_assignments.keys():
        if person_id not in STATE["seen_ids"] and person_id < 1000:
            STATE["seen_ids"].add(person_id)
            STATE["total_count"] = min(999, len(STATE["seen_ids"]))

def update_dog_count(dog_count):
    """Update the dog count"""
    STATE["dog_count"] = dog_count

def mouse_callback(event, x, y, flags, param):
    """Mouse callback for drawing lines manually"""
    if STATE["drawing_line"]:
        if event == cv2.EVENT_LBUTTONDOWN:
            if STATE["line_start"] is None:
                STATE["line_start"] = (x, y)
            else:
                STATE["line_end"] = (x, y)
                if len(STATE["red_lines"]) < MAX_RED_LINES:
                    STATE["red_lines"].append((STATE["line_start"], STATE["line_end"]))
                    print(f"Line {len(STATE['red_lines'])} set from {STATE['line_start']} to {STATE['line_end']}")
                else:
                    print(f"Maximum number of lines ({MAX_RED_LINES}) reached")
                STATE["drawing_line"] = False
```

```
            STATE["line_end"] = (x, y)
            if len(STATE["red_lines"]) < MAX_RED_LINES:
                STATE["red_lines"].append((STATE["line_start"], STATE["line_end"]))
                print(f"Line {len(STATE['red_lines'])} set from {STATE['line_start']} to {STATE['line_end']}")
            else:
                print(f"Maximum number of lines ({MAX_RED_LINES}) reached")
            STATE["drawing_line"] = False
            STATE["line_start"] = None
            STATE["line_end"] = None
    elif event == cv2.EVENT_MOUSEMOVE and STATE["line_start"] is not None:
        STATE["line_end"] = (x, y)

def detect_loop(model):
    win = "Smart Line AI (Detect) - Click to set line start and end points"
    cv2.namedWindow(win, cv2.WINDOW_NORMAL)
    cv2.setMouseCallback(win, mouse_callback)
    fps, frames, t0 = 0.0, 0, time.time()

    while True:
        ret, frame = get_frame()
        if not ret:
            print("Failed to get frame from camera")
            time.sleep(0.1)
            continue

        results = model(frame, imgsz=IMG_SIZE, verbose=False)
        vis = results[0].plot()

        people_count = 0
        dog_count = 0
        centers = []

        if results[0].boxes is not None:
            cls_ids = results[0].boxes.cls.cpu().numpy().astype(int) if results[0].boxes.cls is not None else []
            boxes = results[0].boxes.xyxy.cpu().numpy().astype(int)

            for i, (x1, y1, x2, y2) in enumerate(boxes):
                if i < len(cls_ids):
                    if cls_ids[i] == 0:  # Person
                        people_count += 1
                        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
                        centers.append((cx, cy))
                    elif cls_ids[i] == 16:  # Dog
                        dog_count += 1

        update_dog_count(dog_count)
        person_assignments = person_tracker.update(centers)
        update_dog_count(dog_count)
        person_assignments = person_tracker.update(centers)

        if STATE["total_count_mode"]:
            update_total_count(person_assignments)

        for person_id, (cx, cy) in person_assignments.items():
            color = person_tracker.get_color(person_id)
            dot_size = 2 if STATE["total_count_mode"] else 3
            cv2.circle(vis, (cx, cy - 15), dot_size, color, -1)

            if STATE["total_count_mode"]:
                id_text = f"{person_id:03d}"
                font_scale = 0.4
            else:
                id_text = str(person_id)
                font_scale = 0.5

            cv2.putText(vis, id_text, (cx - 10, cy - 25),
                        cv2.FONT_HERSHEY_SIMPLEX, font_scale, color, 1 if STATE["total_count_mode"] else 2)

        # Draw all red lines
        for i, (line_start, line_end) in enumerate(STATE["red_lines"]):
            cv2.line(vis, line_start, line_end, (0, 0, 255), 1)
            # Add line number
            mid_x = (line_start[0] + line_end[0]) // 2
            mid_y = (line_start[1] + line_end[1]) // 2
            cv2.putText(vis, str(i+1), (mid_x, mid_y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.4, (0, 0, 255), 1)

        # Draw temporary line while drawing
        if STATE["drawing_line"] and STATE["line_start"] is not None and STATE["line_end"] is not None:
            cv2.line(vis, STATE["line_start"], STATE["line_end"], (0, 0, 255), 1)
            cv2.circle(vis, STATE["line_start"], 3, (0, 0, 255), -1)
            cv2.putText(vis, "Click to set end point", (10, 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        breach, breach_id, line_idx = check_line_breach(person_assignments)
        if breach:
            STATE["breach_detected"] = True
            STATE["breach_id"] = breach_id
            STATE["breach_line_idx"] = line_idx
        else:
            STATE["breach_detected"] = False

        frames += 1
        if frames % 10 == 0:
```

```
        frames += 1
        if frames % 10 == 0:
            now = time.time()
            fps = 10.0 / (now - t0)
            t0 = now

        draw_hud(vis, fps, people_count, "detect")

        if STATE["recording"] and STATE["video_writer"] is not None:
            STATE["video_writer"].write(vis)

        cv2.imshow(win, vis)
        k = cv2.waitKey(1) & 0xFF
        if k == 27:
            return "quit"
        elif k in (ord('p'), ord('P')):
            return "pose"
        elif k in (ord('s'), ord('S')):
            return "segment"
        elif k in (ord('t'), ord('T')):
            STATE["total_count_mode"] = not STATE["total_count_mode"]
            if not STATE["total_count_mode"]:
                STATE["total_count"] = 0
                STATE["seen_ids"] = set()
        elif k in (ord('o'), ord('O')):
            STATE["drawing_line"] = True
            STATE["line_start"] = None
            STATE["line_end"] = None
            print("Click to set line start point")
        elif k in (ord('l'), ord('L')):
            if STATE["red_lines"]:
                STATE["red_lines"].pop()
                print(f"Removed line. Total lines: {len(STATE['red_lines'])}")
            STATE["breach_detected"] = False
            STATE["drawing_line"] = False
        elif k in (ord('q'), ord('Q')):
            person_tracker.reset()
            if STATE["total_count_mode"]:
                STATE["total_count"] = 0
                STATE["seen_ids"] = set()
        elif k in (ord('h'), ord('H')):
            STATE["hide_interface"] = not STATE["hide_interface"]
        elif k in (ord('r'), ord('R')):
            toggle_recording()
        elif k in (ord('a'), ord('A')):
            STATE["ai_analysis"] = not STATE["ai_analysis"]
            if STATE["ai_analysis"]:
```

```
        STATE["ai_analysis"] = not STATE["ai_analysis"]
        if STATE["ai_analysis"]:
            global ai_processing, ai_result
            ai_processing = True
            # Run AI analysis in a separate thread to avoid latency
            def ai_thread():
                global ai_result, ai_processing
                ai_result = run_ai_analysis(frame)
                ai_processing = False

            threading.Thread(target=ai_thread).start()
    elif k in (ord('s'), ord('S')):
        STATE["show_masks"] = not STATE["show_masks"]
    elif k in (ord(' '),):
        print("Keys: D detect, P pose, S segment, T total count, 0 draw line, 1 remove line, 2 reset, R record, H hide, A AI, M masks, ESC quit")

return "quit"

def pose_loop(model):
    win = "Smart Live AI (Pose)"
    cv2.namedWindow(win, cv2.WINDOW_NORMAL)
    fps, frames, t0 = 0.0, 0, time.time()
    motion = MotionTracker()
    id_counter = 0
    prev_centers = []

    while True:
        ret, frame = get_frame()
        if not ret:
            print("Failed to get frame from camera")
            time.sleep(0.1)
            continue

        # Run YOLO inference
        results = model(frame, imgsz=IMG_SIZE, verbose=False)
        vis = results[0].plot()

        people_count, dogs_count = 0, 0

        boxes = results[0].boxes.xyxy.cpu().numpy().astype(int) if results[0].boxes is not None else np.zeros((0, 4), int)
        cls_ids = results[0].boxes.cls.cpu().numpy().astype(int) if results[0].boxes is not None and results[0].boxes.cls is not None else np.array([])
        kpts = results[0].keypoints.data.cpu().numpy() if results[0].keypoints is not None else np.zeros((0, 17, 3))

        if len(cls_ids):
            people_count = int((cls_ids == 0).sum())
            dogs_count = int((cls_ids == 16).sum())

            current_time = STATE["last_ollama_time"] > OLLAMA_INTERVAL:
            description = f"{people_count} people detected"
            if people_count > 0:
                actions = []
                for i in range(min(people_count, len(kpts))):
                    if hands_up(kpts[i]):
                        actions.append("hands up")

                if actions:
                    description += f" with {', '.join(actions)}"

            query_ollama_async(description)
            STATE["last_ollama_time"] = current_time

        centers = []
        for i, (x1, y1, x2, y2) in enumerate(boxes):
            cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
            if i < len(cls_ids) and cls_ids[i] != 0:
                continue

            best_id = None
            best_dist = MAX_DISTANCE ** 2
            for pid, pcx, pcy in prev_centers:
                dist = (pcx - cx) ** 2 + (pcy - cy) ** 2
                if dist < best_dist:
                    best_dist = dist
                    best_id = pid

            if best_id is None or best_dist > MAX_DISTANCE ** 2:
                id_counter += 1
                best_id = id_counter

            centers.append((best_id, cx, cy))

        motion.update(centers)
        prev_centers = centers

        for i, (x1, y1, x2, y2) in enumerate(boxes):
            if i >= len(cls_ids) or cls_ids[i] != 0:
                continue

            cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
            tid = None
            for pid, px, py in centers:
                if abs(px - cx) < 5 and abs(py - cy) < 5:
                    tid = pid
                    break
```

```
                tid = pid
                break

        spd = motion.get_speed(tid) if tid is not None else 0.0
        move_label = "running" if spd > 120 else ("walking" if spd > 35 else "idle")
        lbl = move_label

        if i < len(kpts) and hands_up(kpts[i]):
            lbl += " + hands_up"

        cv2.putText(vis, lbl, (x1, max(0, y1 - 6)),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 255), 2)

    if len(centers) >= 2:
        for a in range(len(centers)):
            for b in range(a + 1, len(centers)):
                ida, ax, ay = centers[a]
                idb, bx, by = centers[b]
                va = motion.get_speed(ida)
                vb = motion.get_speed(idb)
                d = math.hypot(ax - bx, ay - by)
                if d < 120 and (va + vb) > 200:
                    midx, midy = (ax + bx) // 2, (ay + by) // 2
                    cv2.putText(vis, "pushing?", (midx - 40, midy),
                                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    if STATE["zones"]:
        cv2.polylines(vis, [ZONE_POLY], True, (0, 255, 255), 2)
        in_count = 0
        for i, (x1, y1, x2, y2) in enumerate(boxes):
            if i < len(cls_ids) and cls_ids[i] == 0:
                cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
                if in_zone(ZONE_POLY, cx, cy):
                    in_count += 1
                    cv2.circle(vis, (cx, cy), 3, (0, 255, 255), -1)
        cv2.putText(vis, f"Zone persons: {in_count}", (16, 180),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 255, 255), 2)
        if in_count >= ZONE_ALERT:
            cv2.putText(vis, "ALERT", (16, 208),
                        cv2.FONT_HERSHEY_SIMPLEX, 1.0, (0, 0, 255), 3)

    frames += 1
    if frames % 10 == 0:
        now = time.time()
        fps = 10.0 / (now - t0)
        t0 = now
```

```
        t0 = now

    draw_hud(vis, fps, people_count, "pose")

    if STATE["recording"] and STATE["video_writer"] is not None:
        STATE["video_writer"].write(vis)

    cv2.imshow(win, vis)
    k = cv2.waitKey(1) & 0xFF
    if k == 27:
        return "quit"
    elif k in (ord('d'), ord('D')):
        return "detect"
    elif k in (ord('s'), ord('S')):
        return "segment"
    elif k in (ord('z'), ord('Z')):
        STATE["zones"] = not STATE["zones"]
    elif k in (ord('o'), ord('O')):
        STATE["ollama_enabled"] = not STATE["ollama_enabled"]
        if STATE["ollama_enabled"]:
            STATE["ollama_response"] = "Analyzing..."
        else:
            STATE["ollama_response"] = "AI analysis disabled"
    elif k in (ord('h'), ord('H')):
        STATE["hide_interface"] = not STATE["hide_interface"]
    elif k in (ord('r'), ord('R')):
        toggle_recording()
    elif k in (ord('a'), ord('A')):
        STATE["ai_analysis"] = not STATE["ai_analysis"]
    elif k in (ord('m'), ord('M')):
        STATE["show_masks"] = not STATE["show_masks"]
    elif k in (ord(' '),):
        print("Keys: D detect, P pose, S segment, Z zones, O ollama on/off, H hide, R record, A AI, M masks, ESC quit")

return "quit"

def segment_loop(model):
    """Segmentation mode loop"""
    win = "Smart Live AI (Segmentation)"
    cv2.namedWindow(win, cv2.WINDOW_NORMAL)
    fps, frames, t0 = 0.0, 0, time.time()

    while True:
        ret, frame = get_frame()
        if not ret:
            print("Failed to get frame from camera")
            time.sleep(0.1)
```

```
            print("Failed to get frame from camera")
            time.sleep(0.1)
            continue

        # Run YOLO inference for segmentation
        results = model(frame, imgsz=IMG_SIZE, verbose=False)

        # Create visualization
        vis = frame.copy()

        # Extract segmentation masks if available
        people_count = 0
        dog_count = 0
        centers = []

        if results[0].masks is not None:
            # Get masks and classes
            masks = results[0].masks.data.cpu().numpy()
            boxes = results[0].boxes.xyxy.cpu().numpy().astype(int) if results[0].boxes is not None else []
            cls_ids = results[0].boxes.cls.cpu().numpy().astype(int) if results[0].boxes is not None and results[0].boxes.cls is not None else []

            # Process each detection
            for i, mask in enumerate(masks):
                if i >= len(cls_ids):
                    continue

                class_id = cls_ids[i]

                # Only process people and dogs
                if class_id == 0:  # Person
                    people_count += 1
                    # Calculate center
                    if i < len(boxes):
                        x1, y1, x2, y2 = boxes[i]
                        cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
                        centers.append((cx, cy))

                    # Draw mask if enabled
                    if STATE["show_masks"]:
                        # Resize mask to frame size
                        mask = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
                        # Create colored mask
                        color_mask = np.zeros_like(vis)
                        color = person_tracker.get_color(people_count) if people_count > 0 else (0, 255, 0)
                        color_mask[:] = color

                        # Apply mask with transparency
```

```
                elif class_id == 16:  # Dog
                    dog_count += 1

                    # Draw mask if enabled
                    if STATE["show_masks"]:
                        # Resize mask to frame size
                        mask = cv2.resize(mask, (frame.shape[1], frame.shape[0]))
                        # Create colored mask (red for dogs)
                        color_mask = np.zeros_like(vis)
                        color_mask[:] = (0, 0, 255)  # Red for dogs

                        # Apply mask with transparency
                        mask_bool = mask > 0.5
                        vis[mask_bool] = cv2.addWeighted(vis[mask_bool], 1 - STATE["mask_alpha"],
                                                         color_mask[mask_bool], STATE["mask_alpha"], 0)

        update_dog_count(dog_count)
        person_assignments = person_tracker.update(centers)

        if STATE["total_count_mode"]:
            update_total_count(person_assignments)

        # Draw bounding boxes and IDs
        if results[0].boxes is not None:
            for i, (x1, y1, x2, y2) in enumerate(boxes):
                if i < len(cls_ids):
                    if cls_ids[i] == 0:  # Person
                        color = person_tracker.get_color(i)
                        cv2.rectangle(vis, (x1, y1), (x2, y2), color, 2)

                        # Draw ID
                        id_text = f"{i:03d}" if STATE["total_count_mode"] else str(i)
                        cv2.putText(vis, id_text, (x1, max(0, y1 - 10)),
                                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

        # Draw all red lines
        for i, (line_start, line_end) in enumerate(STATE["red_lines"]):
            cv2.line(vis, line_start, line_end, (0, 0, 255), 2)
            # Add line number
            mid_x = (line_start[0] + line_end[0]) // 2
            mid_y = (line_start[1] + line_end[1]) // 2
            cv2.putText(vis, str(i+1), (mid_x, mid_y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        # Check for line breaches
        breach, breach_id, line_idx = check_line_breach(person_assignments)
```

```
# Check for line breaches
breach, breach_id, line_idx = check_line_breach(person_assignments)
if breach:
    STATE["breach_detected"] = True
    STATE["breach_id"] = breach_id
    STATE["breach_line_idx"] = line_idx
else:
    STATE["breach_detected"] = False

frames += 1
if frames % 10 == 0:
    now = time.time()
    fps = 10.0 / (now - t0)
    t0 = now

draw_hud(vis, fps, people_count, "segment")

if STATE["recording"] and STATE["video_writer"] is not None:
    STATE["video_writer"].write(vis)

cv2.imshow(win, vis)
k = cv2.waitKey(1) & 0xFF
if k == 27:
    return "quit"
elif k in (ord('d'), ord('D')):
    return "detect"
elif k in (ord('p'), ord('P')):
    return "pose"
elif k in (ord('t'), ord('T')):
    STATE["total_count_mode"] = not STATE["total_count_mode"]
    if not STATE["total_count_mode"]:
        STATE["total_count"] = 0
        STATE["seen_ids"] = set()
elif k in (ord('h'), ord('H')):
    STATE["hide_interface"] = not STATE["hide_interface"]
elif k in (ord('r'), ord('R')):
    toggle_recording()
elif k in (ord('a'), ord('A')):
    STATE["ai_analysis"] = not STATE["ai_analysis"]
    if STATE["ai_analysis"]:
        global ai_processing, ai_result
        ai_processing = True
        # Run AI analysis in a separate thread to avoid latency
        def ai_thread():
            global ai_result, ai_processing
            ai_result = run_ai_analysis(frame)
            ai_processing = False
```

```
        det_model = YOLO(DETECT_MODEL)
        pose_model = YOLO(POSE_MODEL)
        seg_model = YOLO(SEGMENT_MODEL)  # Load segmentation model
    except Exception as e:
        print(f"Error loading models: {e}")
        return

    if not init_camera():
        print("Failed to initialize camera. Exiting.")
        return

    try:
        mode = "detect"
        while True:
            try:
                if mode == "detect":
                    nxt = detect_loop(det_model)
                elif mode == "pose":
                    nxt = pose_loop(pose_model)
                elif mode == "segment":
                    nxt = segment_loop(seg_model)  # Added segmentation mode
                else:
                    nxt = "detect"  # Fallback to detect mode

                if nxt == "quit" or nxt is None:
                    break
                mode = nxt
            except KeyboardInterrupt:
                print("Interrupted by user")
                break
            except Exception as e:
                print(f"Error in {mode} loop: {e}")
                import traceback
                traceback.print_exc()
                release_camera()
                if not init_camera():
                    print("Failed to reinitialize camera. Exiting.")
                    break
    finally:
        if STATE["recording"] and STATE["video_writer"] is not None:
            STATE["video_writer"].release()
        release_camera()
        cv2.destroyAllWindows()

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

# 8. INDIVIDUAL SOFTWARE CONTRIBUTION

## 8.1. AI Model for Pothole Detection, Unique Counting & Size Estimation - Akshra

**This model worked successfully, However, the Final Application of the drone was unanimously decided to be human behavioural analysis**

**Model Architecture**

- **YOLO11n Detect model (Ultralytics framework)** was selected.
- Custom override was applied to train a **single class: pothole** (nc=1).
- The model performs **bounding-box regression and confidence scoring**, not segmentation.
- Nano backbone architecture was used for real-time inference efficiency.

### 8.1.1. Dataset Preparation

- Dataset was obtained from Kaggle, structured into train/valid/test folders.
- Labels were prepared in YOLO .txt annotation format.
- The data.yaml file was validated to point to the correct dataset directories.
- One corrupt GIF sample was automatically ignored, ensuring dataset compliance.
- Corrupt JPEG images were restored automatically during training, ensuring no frame-level instability.

Figure 8: Related drone component or system illustration.

### 8.1.2. Training Process

Training was executed using:

```
from ultralytics import YOLO
model = YOLO("yolo11n.pt")
model.train(
    data="/content/drive/MyDrive/Potholes/data.yaml",
    epochs=100,
    imgsz=640,
    batch=16,
    device=0,
    workers=2,
    augment=True,
    degrees=10,
    perspective=0.001,
    flipud=0.2,
    fliplr=0.5,
    hsv_s=0.7,
    hsv_v=0.5
)
```

### 8.1.3. Code Breakdown

- The model weights file is loaded from local storage.
- The primary camera (webcam) is selected as the input source.
- Frames are standardized to a fixed resolution (640×480) for consistent boundingbox area computation.
- Confidence threshold (0.35) ensures that only detections with ≥35% model certainty are accepted.
- IoU threshold (0.45) is used for detection filtering in NMS (Non-Maximum Suppression) inside YOLO.
- CPU is chosen for deployment to ensure compatibility across low-power systems.
- Recording directory is initialized for storing inference video logs.

```
MODEL_PATH = r"C:\Users\akshr\Downloads\Pothole_detection\best_pothle .pt"
CAMERA_ID = 0
IMG_SIZE = (640, 480)
CONF_THRES = 0.35
IOU_THRES = 0.45
DEVICE = "cpu"
FPS_WINDOW_SIZE = 10
RECORDING_ENABLED = True
OUTPUT_DIR = "recordings"
```

The camera stream object cap is created.
Resolution is forced to 640×480 so that:

- IoU tracking works on fixed coordinate space
- Pothole pixel area remains comparable across frames
- On-screen overlays render without distortion

```
cap = cv2.VideoCapture(CAMERA_ID)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, IMG_SIZE[0])
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, IMG_SIZE[1])
```

deque(maxlen=30) stores last 30 FPS values to compute smooth average FPS.
tracked[] is a manual tracking list that stores:

- track_id → unique pothole instance number
- bbox → last known bounding box of that pothole
- last_seen_frame → frame index where it was last detected

This enables **unique pothole counting**, preventing duplicates when the same defect appears again.

```
from collections import deque
tracked = []   # stores [track_id, bbox, last_seen_frame]
fps_history = deque(maxlen=30)
```

- intersection area is computed from overlapping box region.
- union area is computed from both box areas minus intersection.
- Small epsilon (1e-6) avoids division-by-zero when boxes do not overlap.
- The function returns IoU in range **0 to 1**:

○ 1 → perfect overlap (same object) ○ 0 → no overlap
(different object)

- Used later to match detections referring to the same pothole.

```
def iou(a, b):
    x1 = max(a[0], b[0]); y1 = max(a[1], b[1])
    x2 = min(a[2], b[2]); y2 = min(a[3], b[3])
    inter = max(0, x2-x1) * max(0, y2-y1)
    A = (a[2]-a[0])*(a[3]-a[1])
    B = (b[2]-b[0])*(b[3]-b[1])
    return inter/(A+B-inter+1e-6)
```

- The model processes each incoming frame (after resize).
- predict() outputs a list of detections; here only the first result is used (results[0]).
- imgsz=(640,480) ensures the model internally scales detection grid accordingly.
- conf=0.35 filters weak detections.
- iou=0.45 ensures YOLO applies NMS to suppress overlapping duplicate boxes in the same frame.

```
results = model.predict(
    display_frame,
    imgsz=IMG_SIZE,
    conf=CONF_THRES,
    iou=IOU_THRES,
    device=DEVICE,
    verbose=False
)
```

- boxes.xyxy gives bounding box corner coordinates.

- .cpu().numpy() converts tensor outputs into numerical arrays for OpenCV rendering and area computation.
- Total detections in the frame are counted.

```
if results[0].boxes is not None:
    boxes = results[0].boxes.xyxy.cpu().numpy()
    confidences = results[0].boxes.conf.cpu().numpy()
    detections = len(boxes)
```


```
if results[0].boxes is not None:
    boxes = results[0].boxes.xyxy.cpu().numpy()
    confidences = results[0].boxes.conf.cpu().numpy()
    detections = len(boxes)
```


**Pothole Size Estimation**

- Bounding-box pixel area is converted into **relative frame percentage**, making it altitude-agnostic.
- This was chosen to avoid false interpretation caused by absolute pixel area in different camera resolutions.
- Severity classes were assigned based on bbox-area percentage:

```
area = (box[2] - box[0]) * (box[3] - box[1])
area_percentage = (area / frame_area) * 100
```

- Each new bbox detection is compared against stored previous boxes.
- If IoU > 0.5 → considered the same physical pothole.
- If not matched → assigned a new unique ID and stored.
- Stale tracks are removed using max_age=15 so old defects do not persist incorrectly.

Ensures **unique defect counting across time**, which is essential for UAV surveys where hover-frames may repeat the same object.

```
for t in tracked:
    if iou(det, t[1]) > 0.5:
        t[1] = det
        t[2] = frame_no
        matched = True
        break
if not matched:
    next_id += 1
    tracked.append([next_id, det, frame_no])
```

- Webcam feed was resized to 640×480 for consistent inference throughput.
- Annotated frames were displayed using OpenCV overlays.
- A timestamp-based recording system was implemented to document inference video.

Advantages of recording:

- Provides **non-duplicated defect evidence**
- Enables later inclusion of **authentic images and videos for report validation**

Figure 9: Related drone component or system illustration.

- Ensures compliance with IQAC documentation standards

```
recorder = VideoRecorder()
if RECORDING_ENABLED:
    recorder.write_frame(display_frame)
```

These OpenCV tools were used to render:

- Detection box
- Confidence %
- Severity class
- Live recording indicator
- FPS panel
- Inference latency in ms

OpenCV was chosen instead of matplotlib because:

- No axis warping occurs.
- Real-time video overlay rendering is faster and non-blocking.
-

```
cv2.rectangle(), cv2.putText(), cv2.circle()
```

- Camera resource is released safely.
- OpenCV windows are closed.
- No background processes remain active.

```
cap.release()
cv2.destroyAllWindows()
```

```python
import cv2
import time
import numpy as np
from ultralytics import YOLO
from collections import deque
import os

# Configuration
MODEL_PATH = r"C:\Users\akshr\Downloads\Pothole_detection\best_pothole .pt"
CAMERA_ID = 0
IMG_SIZE = (640, 480)
CONF_THRES = 0.35
IOU_THRES = 0.45
DEVICE = "cpu"
FONT_SCALE = 0.6
FONT_THICKNESS = 2
FPS_WINDOW_SIZE = 10
RECORDING_ENABLED = True  # Set to True to enable recording
OUTPUT_DIR = "recordings"  # Directory to save recordings

# Create output directory if it doesn't exist
if RECORDING_ENABLED and not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)

# Load model
try:
    model = YOLO(MODEL_PATH)
    print(f"✓ Model loaded: {MODEL_PATH}")
except Exception as e:
    print(f"X Error loading model: {e}")
    exit()

# Initialize camera
cap = cv2.VideoCapture(CAMERA_ID)
if not cap.isOpened():
    print(f"X Cannot open camera {CAMERA_ID}")
    exit()

# Set camera properties
cap.set(cv2.CAP_PROP_FRAME_WIDTH, IMG_SIZE[0])
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, IMG_SIZE[1])

# Create window
WINDOW_NAME = "Pothole Detection"
cv2.namedWindow(WINDOW_NAME, cv2.WINDOW_NORMAL)
cv2.resizeWindow(WINDOW_NAME, IMG_SIZE[0], IMG_SIZE[1])
```

```python
# VIDEO RECORDING SETUP
class VideoRecorder:
    def __init__(self):
        self.video_writer = None
        self.is_recording = False
        self.output_path = ""
        self.frame_size = IMG_SIZE
        self.fps = 20  # Target FPS for recording
        self.fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # MP4 codec

    def start_recording(self):
        """Start a new recording session."""
        if self.is_recording:
            print("⚠️ Already recording!")
            return

        # Generate timestamped filename
        timestamp = time.strftime("%Y%m%d_%H%M%S")
        self.output_path = os.path.join(OUTPUT_DIR, f"pothole_detection_{timestamp}.mp4")

        # Create VideoWriter object
        self.video_writer = cv2.VideoWriter(
            self.output_path,
            self.fourcc,
            self.fps,
            self.frame_size
        )

        if not self.video_writer.isOpened():
            print("❌ Failed to initialize video writer")
            self.video_writer = None
            return False

        self.is_recording = True
        print(f"🎥 Started recording: {self.output_path}")
        return True

    def stop_recording(self):
        """Stop the current recording session."""
        if not self.is_recording or self.video_writer is None:
            print("⚠️ Not recording!")
            return

        self.video_writer.release()
        self.video_writer = None
        self.is_recording = False
        print(f"📁 Recording saved: {self.output_path}")
```

```python
    def write_frame(self, frame):
        """Write a frame to the video file if recording."""
        if self.is_recording and self.video_writer is not None:
            # Ensure frame matches the expected size
            if frame.shape[:2] != self.frame_size[::-1]:
                frame = cv2.resize(frame, self.frame_size)
            self.video_writer.write(frame)

    def get_recording_status(self):
        """Return current recording status."""
        return self.is_recording, self.output_path if self.is_recording else ""

# Initialize recorder
recorder = VideoRecorder()

def get_color_for_area(area, frame_area):
    """Assign color based on pothole area relative to frame size."""
    area_percentage = (area / frame_area) * 100

    if area_percentage < 0.1:
        return (200, 200, 200), "Tiny"
    elif area_percentage < 0.5:
        return (0, 255, 0), "Small"
    elif area_percentage < 2.0:
        return (0, 255, 255), "Medium"
    elif area_percentage < 5.0:
        return (0, 165, 255), "Large"
    else:
        return (0, 0, 255), "Critical"

def draw_detection(frame, box, confidence, color, label_text=""):
    """Draw bounding box and label for a detection."""
    x1, y1, x2, y2 = map(int, box)

    cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)

    label = f"{label_text} {confidence:.0%}"
    (text_width, text_height), _ = cv2.getTextSize(
        label, cv2.FONT_HERSHEY_SIMPLEX, FONT_SCALE, FONT_THICKNESS
    )
```

```
    cv2.rectangle(
        frame,
        (x1, y1 - text_height - 10),
        (x1 + text_width + 10, y1),
        color,
        -1
    )

    cv2.putText(
        frame,
        label,
        (x1 + 5, y1 - 5),
        cv2.FONT_HERSHEY_SIMPLEX,
        FONT_SCALE,
        (255, 255, 255),
        FONT_THICKNESS
    )

def draw_info_panel(frame, pothole_count, fps, processing_time):
    """Draw information panel on frame."""
    panel_height = 120
    overlay = frame.copy()
    cv2.rectangle(overlay, (0, 0), (300, panel_height), (0, 0, 0), -1)
    cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)

    # Add recording status
    status_lines = [
        f"Potholes: {pothole_count}",
        f"FPS: {fps:.1f}",
        f"Inference: {processing_time:.0f}ms",
        f"Confidence: {CONF_THRES:.2f}",
    ]

    # Add recording info if enabled
    if RECORDING_ENABLED:
        is_recording, _ = recorder.get_recording_status()
        status = "RECORDING ●" if is_recording else "NOT RECORDING"
        color = (0, 0, 255) if is_recording else (200, 200, 200)
        status_lines.append(f"Status: {status}")

    for i, line in enumerate(status_lines):
        color = (255, 255, 255)
        if "RECORDING" in line:
            color = (0, 0, 255) if is_recording else (200, 200, 200)
```

```
        cv2.putText(
            frame,
            line,
            (10, 25 + (i * 20)),
            cv2.FONT_HERSHEY_SIMPLEX,
            0.5,
            color,
            1
        )

def draw_legend(frame):
    """Draw color legend for pothole sizes."""
    legend_items = [
        ("Tiny (<0.1%)", (200, 200, 200)),
        ("Small (<0.5%)", (0, 255, 0)),
        ("Medium (<2%)", (0, 255, 255)),
        ("Large (<5%)", (0, 165, 255)),
        ("Critical (5%+)", (0, 0, 255))
    ]

    legend_y = frame.shape[0] - 140

    overlay = frame.copy()
    cv2.rectangle(overlay, (10, legend_y - 10), (250, frame.shape[0] - 10), (0, 0, 0), -1)
    cv2.addWeighted(overlay, 0.7, frame, 0.3, 0, frame)

    for i, (text, color) in enumerate(legend_items):
        y_pos = legend_y + (i * 25)
        cv2.circle(frame, (20, y_pos - 5), 6, color, -1)
        cv2.putText(frame, text, (35, y_pos), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 1)

print("="*50)
print("POTHOLE DETECTION SYSTEM WITH RECORDING")
print("="*50)
print("Controls:")
print("  'q' or ESC - Quit")
print("  's' - Save current frame")
print("  'r' - Start/Stop recording")
print("  '+' - Increase confidence threshold")
print("  '-' - Decrease confidence threshold")
print("  'p' - Pause/Resume detection")
print("="*50)
print(f"Recording enabled: {RECORDING_ENABLED}")
if RECORDING_ENABLED:
    print(f"Videos will be saved to: {os.path.abspath(OUTPUT_DIR)}")
print("="*50)
```

```python
# Main loop variables
fps_history = deque(maxlen=30)
frame_count = 0
start_time = time.time()
paused = False

try:
    while True:
        if not paused:
            # Read frame
            ret, frame = cap.read()
            if not ret:
                print("Failed to read frame")
                break

            frame = cv2.resize(frame, IMG_SIZE)
            display_frame = frame.copy()

            # Run inference
            inference_start = time.time()
            results = model.predict(
                display_frame,
                imgsz=IMG_SIZE,
                conf=CONF_THRES,
                iou=IOU_THRES,
                device=DEVICE,
                verbose=False
            )
            inference_time = (time.time() - inference_start) * 1000

            # Process detections
            detections = 0
            if results[0].boxes is not None:
                boxes = results[0].boxes.xyxy.cpu().numpy()
                confidences = results[0].boxes.conf.cpu().numpy()
                detections = len(boxes)

                frame_area = IMG_SIZE[0] * IMG_SIZE[1]
                for box, confidence in zip(boxes, confidences):
                    area = (box[2] - box[0]) * (box[3] - box[1])
                    color, severity = get_color_for_area(area, frame_area)
                    draw_detection(display_frame, box, confidence, color, severity)
```

```
        # Calculate FPS
        frame_count += 1
        if frame_count >= 10:
            fps = 10 / (time.time() - start_time)
            fps_history.append(fps)
            frame_count = 0
            start_time = time.time()

        avg_fps = np.mean(fps_history) if fps_history else 0

        # Draw UI elements
        draw_info_panel(display_frame, detections, avg_fps, inference_time)
        draw_legend(display_frame)

        # Draw recording indicator
        if RECORDING_ENABLED and recorder.is_recording:
            # Red circle in top-right corner
            cv2.circle(display_frame, (IMG_SIZE[0] - 20, 20), 8, (0, 0, 255), -1)
            cv2.putText(display_frame, "REC", (IMG_SIZE[0] - 50, 25),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

        # Write frame to video if recording
        if RECORDING_ENABLED:
            recorder.write_frame(display_frame)

        # Show the frame
        cv2.imshow(WINDOW_NAME, display_frame)

    # Handle keyboard input
    wait_time = 30 if not paused else 100
    key = cv2.waitKey(wait_time)

    if key != -1:
        # Quit
        if key == 27 or key == ord('q') or key == ord('Q'):
            print("\nQuitting...")
            break

        # Save frame
        elif key == ord('s') or key == ord('S'):
            timestamp = time.strftime("%Y%m%d_%H%M%S")
            filename = f"pothole_frame_{timestamp}.jpg"
            cv2.imwrite(filename, display_frame)
            print(f"✓ Saved frame: {filename}")
```

```
            # Toggle recording (only if enabled)
            elif key == ord('r') or key == ord('R'):
                if RECORDING_ENABLED:
                    if recorder.is_recording:
                        recorder.stop_recording()
                    else:
                        recorder.start_recording()
                else:
                    print("⚠️ Recording is disabled in settings")

            # Adjust confidence
            elif key == ord('+') or key == ord('='):
                CONF_THRES = min(0.95, CONF_THRES + 0.05)
                print(f"↑ Confidence threshold: {CONF_THRES:.2f}")

            elif key == ord('-') or key == ord('_'):
                CONF_THRES = max(0.05, CONF_THRES - 0.05)
                print(f"↓ Confidence threshold: {CONF_THRES:.2f}")

            # Pause/Resume
            elif key == ord('p') or key == ord('P'):
                paused = not paused
                status = "PAUSED" if paused else "RESUMED"
                print(f"⏸  {status}")

except KeyboardInterrupt:
    print("\nInterrupted by user")
except Exception as e:
    print(f"\nError: {e}")

# CLEANUP
print("\n" + "="*50)
print("CLEANING UP...")

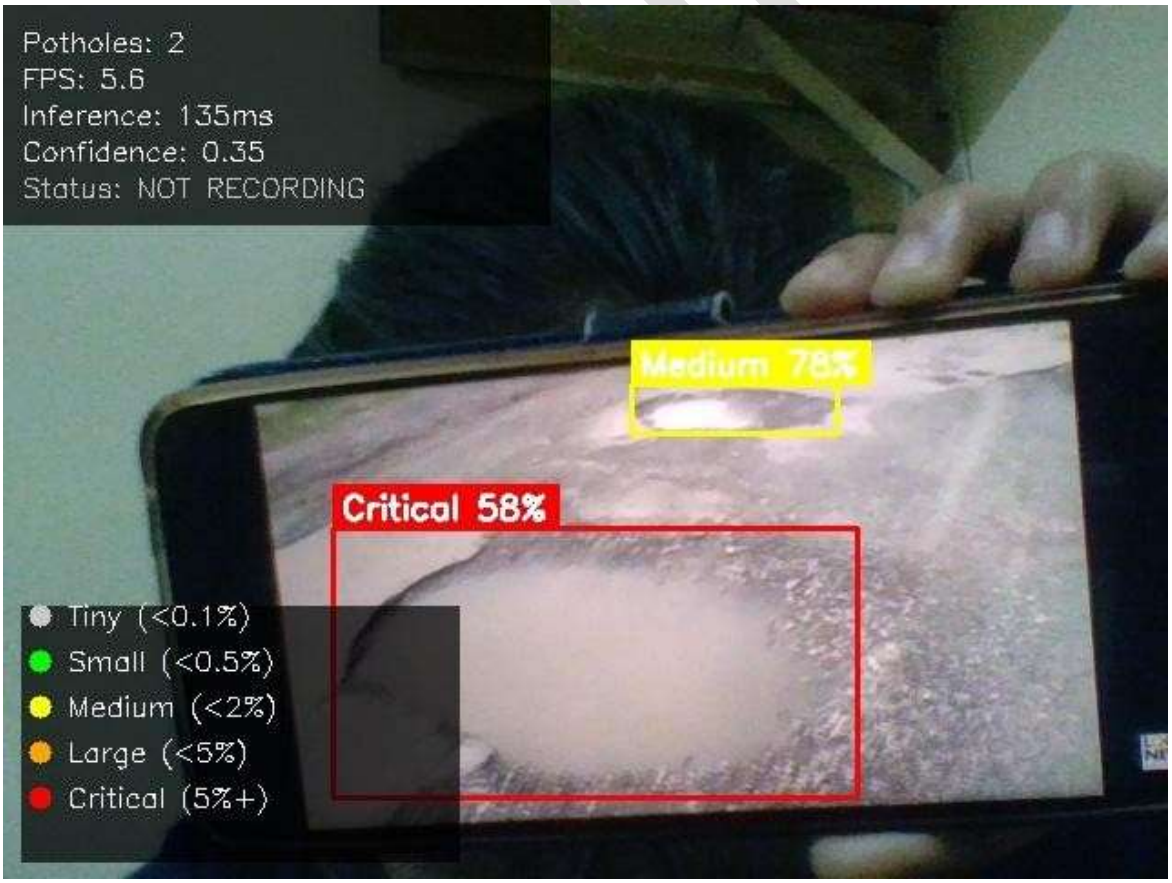
# Stop recording if still active
if RECORDING_ENABLED and recorder.is_recording:
    print("Stopping active recording...")
    recorder.stop_recording()

# Release camera
cap.release()

# Close all OpenCV windows
cv2.destroyAllWindows()

print("\nDETECTION SESSION ENDED")
print("="*50)
```


Potholes: 4
FPS: 5.6
Inference: 127ms
Confidence: 0.35
Status: NOT RECORDING
Medium 84%
Large 91%
Large 75%
Medium 77%
Tiny (<0.1%)
Small (<0.5%)
Medium (<2%)
Large (<5%)
Critical (5%+)


Potholes: 2
FPS: 5.6
Inference: 135ms
Confidence: 0.35
Status: NOT RECORDING
Medium 78%
Critical 58%
Tiny (<0.1%)
Small (<0.5%)
Medium (<2%)
Large (<5%)
Critical (5%+)

A single-class object-detection model for pothole inspection was developed using the Ultralytics YOLO11n Detect architecture, selected for its lightweight backbone and realtime inference capability. The dataset, consisting of annotated road-surface images and test video, was sourced from Kaggle and validated using a YAML schema to ensure correct training directory mapping. The model was trained for 100 epochs at 640-pixel resolution with a batch size of 16 on a Tesla T4 GPU, enabling stable gradient convergence through automatic optimizer selection and data-augmentation techniques
Figure 10: Related drone component or system illustration. (geometric transforms, flips, HSV variation, and perspective jitter) for field generalization. Detection reliability was quantified using mean Average Precision at IoU 0.5:0.95, the most recommended metric for UAV terrain-defect inspection, as it evaluates localization accuracy across multiple overlap thresholds. Unique pothole instances were counted across frames using a manual IoU-matching buffer, preventing duplicate defect registration during repeated camera views or hovering surveys. Bounding-box pixel area was converted into relative frame-coverage percentages to classify defect size (Tiny, Small, Medium, Large, Critical), ensuring scale-agnostic sizing prior to altitude-based real-world conversion. Live inference was deployed on webcam feed using OpenCV overlay rendering, paired with MP4 video logging using a timestamp-based recording subsystem to generate original, non-overwritten inspection evidence. The model was verified to be fully functional and extendable for future aerial surveillance terraininspection tasks, including real-world road-defect mapping and AI-assisted UAV diagnostics.

```
(yolo_env) PS C:\Users\akshr\Downloads\Pothole_detection> python live_pothole_drone.py
✓ Model loaded: C:\Users\akshr\Downloads\Pothole_detection\best_pothole .pt
================================================
POTHOLE DETECTION SYSTEM WITH RECORDING
================================================
Controls:
  'q' or ESC - Quit
  's' - Save current frame
  'r' - Start/Stop recording
  '+' - Increase confidence threshold
  '-' - Decrease confidence threshold
  'p' - Pause/Resume detection
================================================
Recording enabled: True
Videos will be saved to: C:\Users\akshr\Downloads\Pothole_detection\recordings
================================================
```

SKIPPA

## 8.2. Real time human Running, Standing and Walking Detection using CV and Python- Diya Sharma

.

### 8.2.1. Methodology

The working of the project can be divided into the following steps:

*Step 1: Input video acquisition*

The input is taken either from:

- a video file (e.g., running.mp4), or
- webcam stream

The program reads the frames one by one using OpenCV.

*Step 2: Pose detection*

MediaPipe Pose is applied to every frame.

It detects 33 body keypoints such as:

- shoulders
- elbows
- knees
- hips
- ankles

and gives their *(x, y, z)* coordinates.

*Step 3: Feature extraction*

From each keypoint, the coordinates are extracted and stored as a feature vector.

Example:

- Landmark 1 → (x1, y1)
- Landmark 2 → (x2, y2)
- …
- Landmark 33 → (x33, y33)

These features numerically represent human posture.

*Step 4: Classification*

A machine learning model is trained / used to classify activities like:

- Running
- Walking
- Standing

The classifier tested in this project was K-Nearest Neighbour (KNN) / Random Forest, applied to the extracted pose data.

*Step 5: Output display*

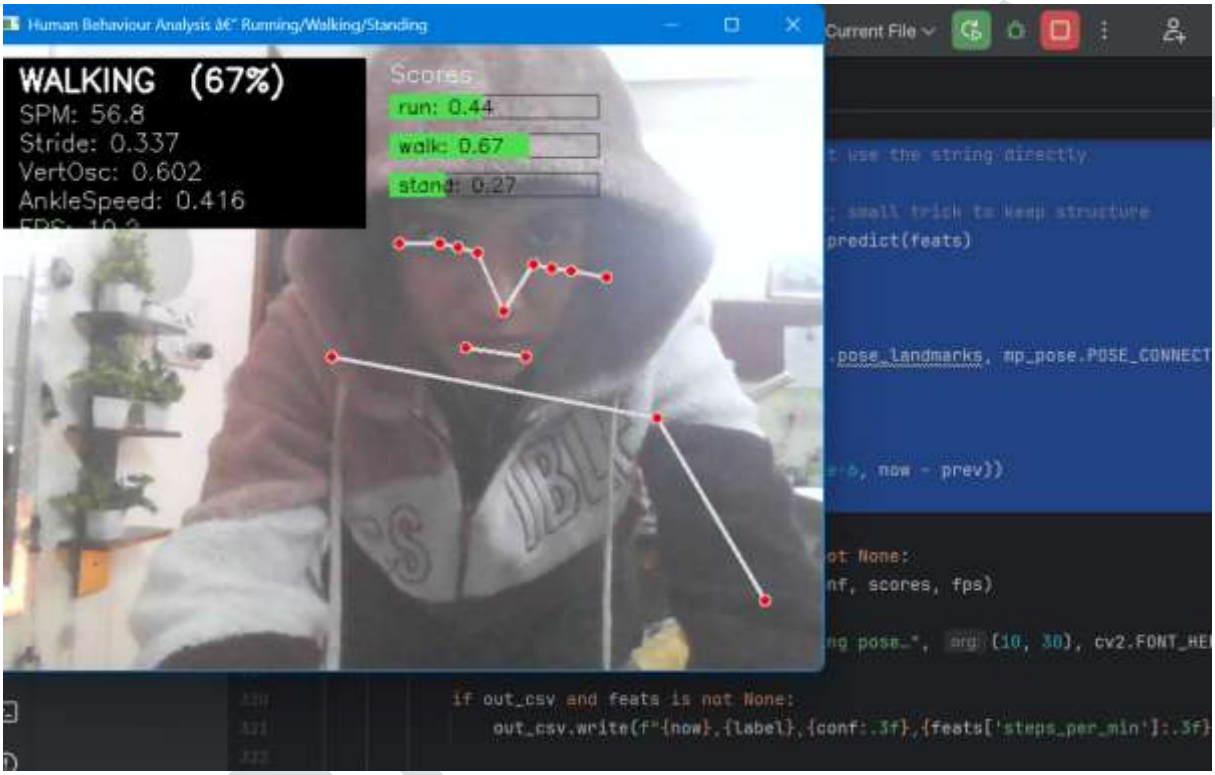
On each processed frame:

- pose skeleton is drawn
- activity label is written on screen
- confidence score is optionally displayed

SRI-VIPRA

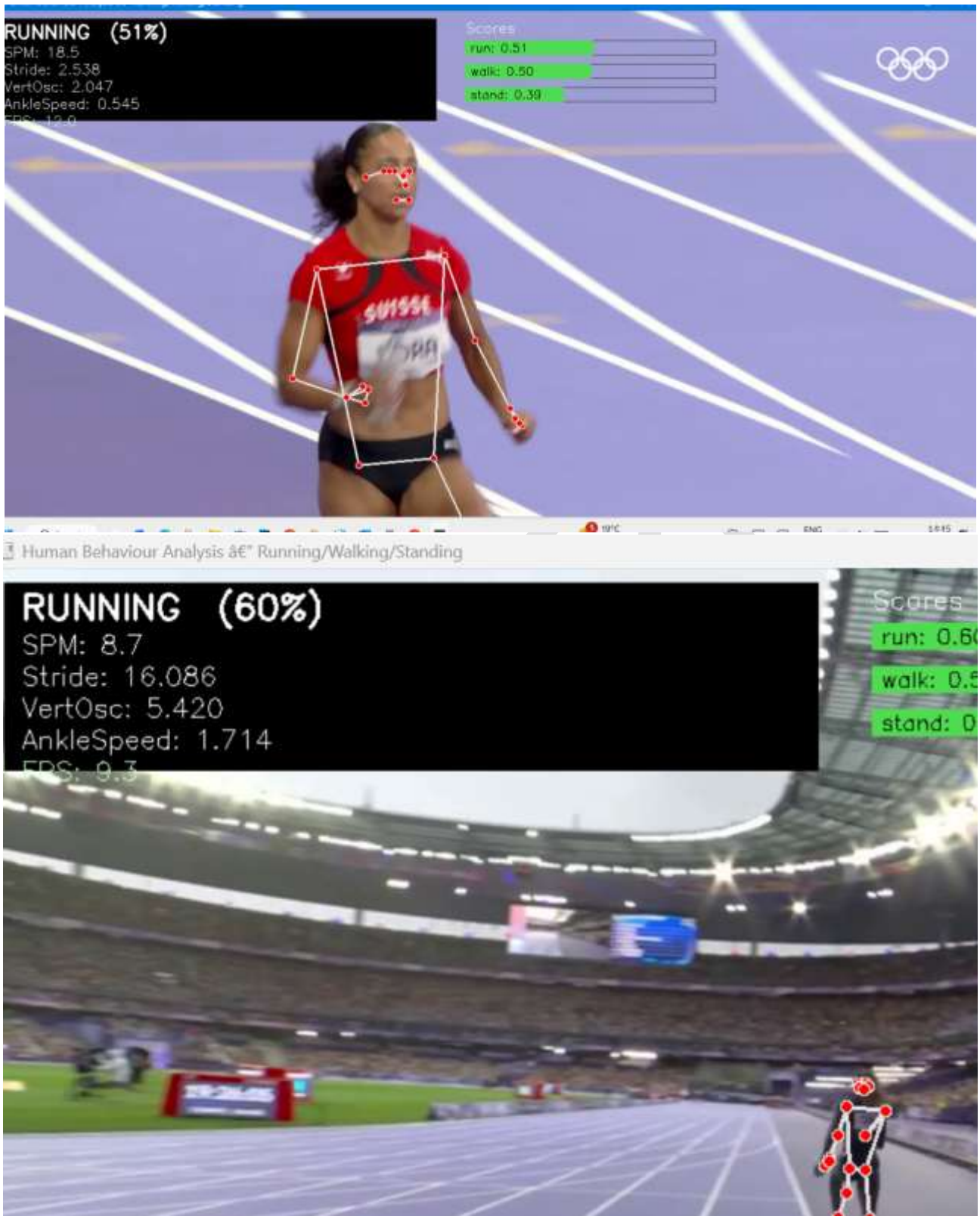
The final output is a real-time annotated video.

### 8.2.2. Images of work done and applications



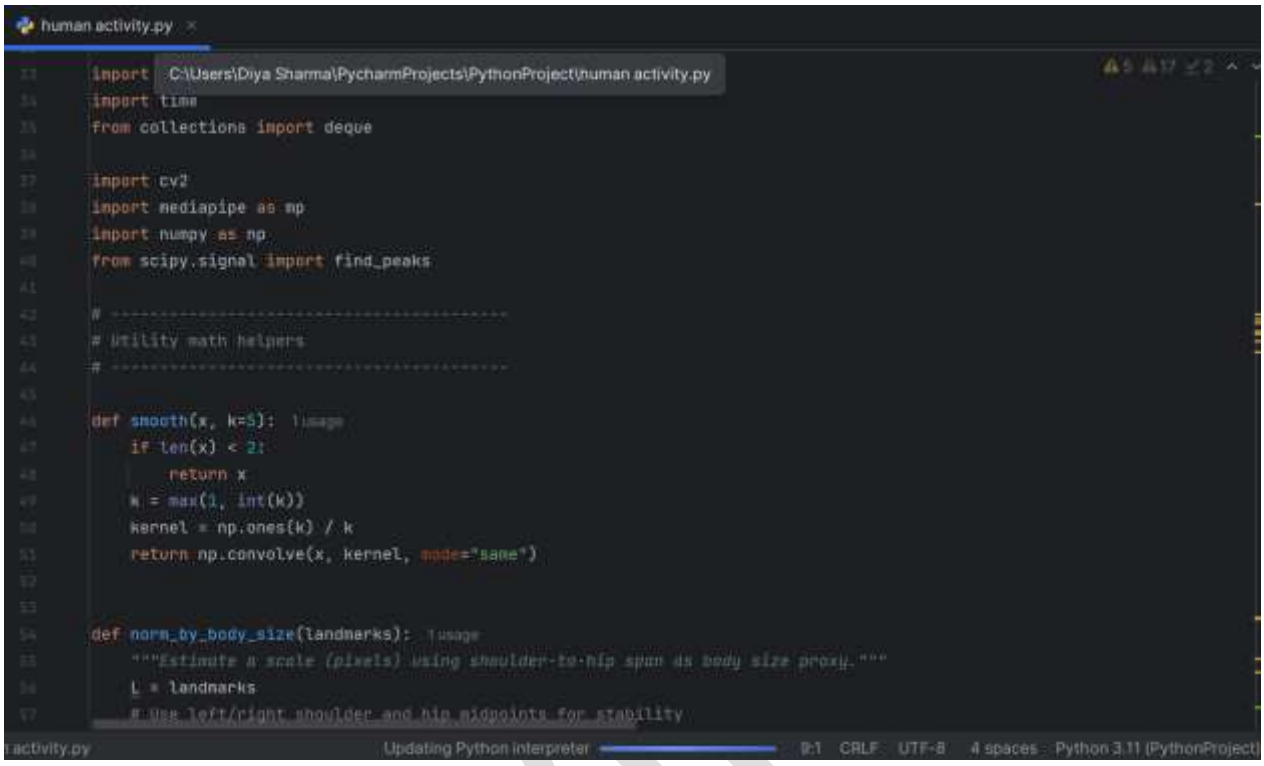

Screenshot taken right after walking into view

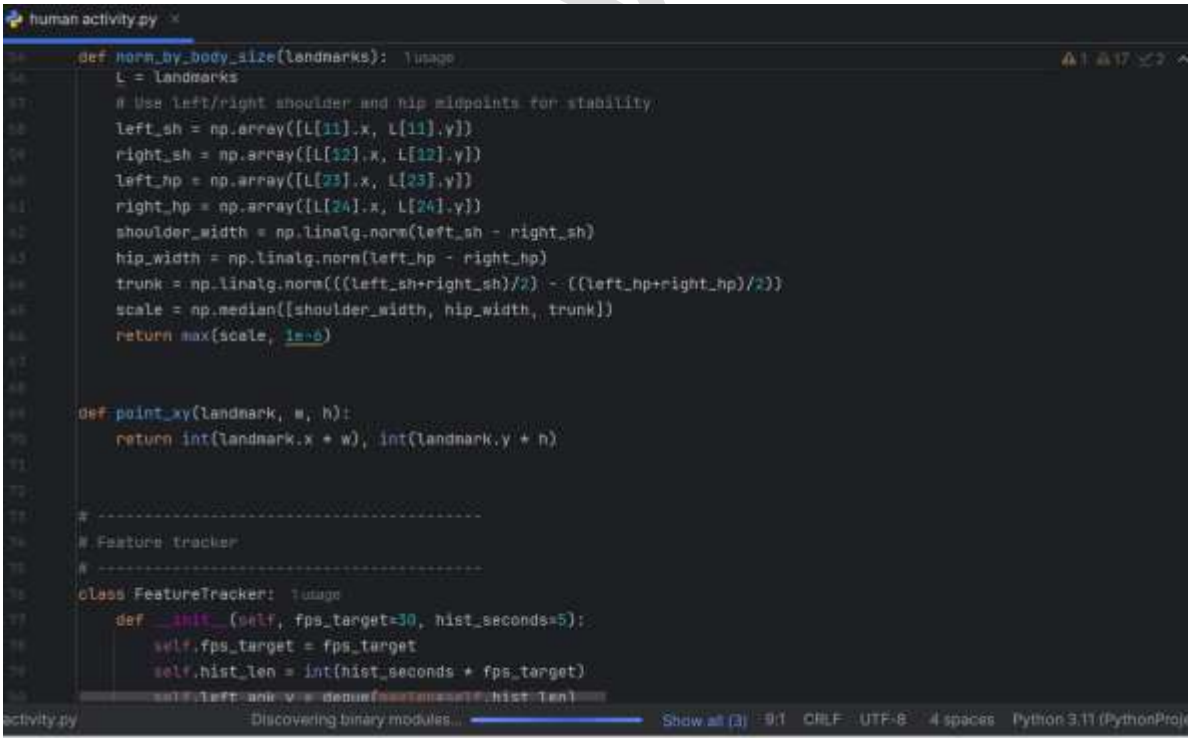
SRI




Testing the model on a clip from the Tokyo Olympics Running Championship. The model successfully detected running.

# Code:


human activity.py
C:\Users\Diya Sharma\PycharmProjects\PythonProject\human activity.py
import time
from collections import deque
import cv2
import mediapipe as mp
import numpy as np
from scipy.signal import find_peaks
# ----------------------------------------
# Utility math helpers
# ----------------------------------------
def smooth(x, k=5):
if len(x) < 2:
return x
k = max(1, int(k))
kernel = np.ones(k) / k
return np.convolve(x, kernel, mode="same")
def norm_by_body_size(landmarks):
"""Estimate a scale (pixels) using shoulder-to-hip span as body size proxy."""
L = landmarks
# Use left/right shoulder and hip midpoints for stability
Updating Python interpreter
9:1 CRLF UTF-8 4 spaces Python 3.11 (PythonProject)


human activity.py
def norm_by_body_size(landmarks):
L = landmarks
# Use left/right shoulder and hip midpoints for stability
left_sh = np.array([L[11].x, L[11].y])
right_sh = np.array([L[12].x, L[12].y])
left_hp = np.array([L[23].x, L[23].y])
right_hp = np.array([L[24].x, L[24].y])
shoulder_width = np.linalg.norm(left_sh - right_sh)
hip_width = np.linalg.norm(left_hp - right_hp)
trunk = np.linalg.norm(((left_sh+right_sh)/2) - ((left_hp+right_hp)/2))
scale = np.median([shoulder_width, hip_width, trunk])
return max(scale, 1e-6)
def point_xy(landmark, w, h):
return int(landmark.x * w), int(landmark.y * h)
# ----------------------------------------
# Feature tracker
# ----------------------------------------
class FeatureTracker:
def __init__(self, fps_target=30, hist_seconds=5):
self.fps_target = fps_target
self.hist_len = int(hist_seconds * fps_target)
Discovering binary modules...
Show all (3)
9:1 CRLF UTF-8 4 spaces Python 3.11

human activity.py

```
    class FeatureTracker:  1 usage
        def __init__(self, fps_target=30, hist_seconds=5):
            self.fps_target = fps_target
            self.hist_len = int(hist_seconds * fps_target)
            self.left_ank_y = deque(maxlen=self.hist_len)
            self.right_ank_y = deque(maxlen=self.hist_len)
            self.left_ank_x = deque(maxlen=self.hist_len)
            self.right_ank_x = deque(maxlen=self.hist_len)
            self.hip_y = deque(maxlen=self.hist_len)
            self.time_stamps = deque(maxlen=self.hist_len)
            self.body_scale = deque(maxlen=self.hist_len)

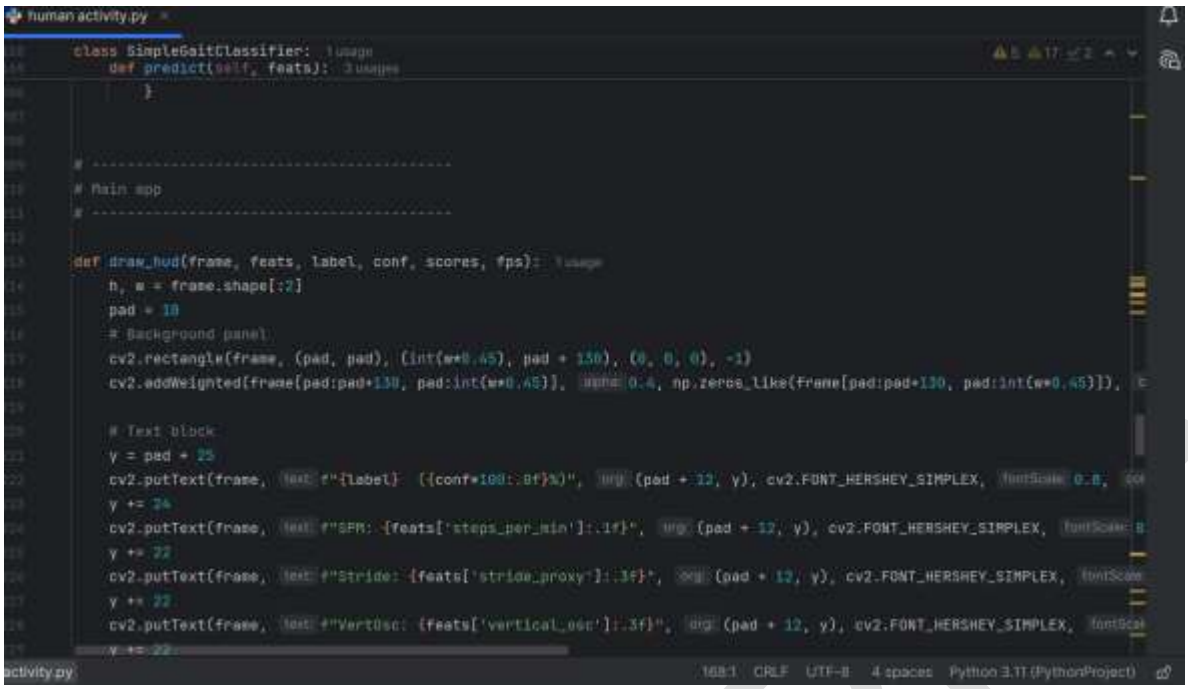
        def push(self, t, lm):  1 usage
            # MediaPipe indices
            L_ANK, R_ANK = 27, 28
            L_HIP, R_HIP = 23, 24
            scale = norm_by_body_size(lm)
            self.body_scale.append(scale)
            self.left_ank_y.append(lm[L_ANK].y)
            self.right_ank_y.append(lm[R_ANK].y)
            self.left_ank_x.append(lm[L_ANK].x)
            self.right_ank_x.append(lm[R_ANK].x)
            hip_mid_y = (lm[L_HIP].y + lm[R_HIP].y) / 2
            self.hip_y.append(hip_mid_y)
            self.time_stamps.append(t)
```

human activity.py

```
    class FeatureTracker:  1 usage
        def compute_features(self):  1 usage
            t = np.array(self.time_stamps)
            dt = np.diff(t)
            dt[dt == 0] = 1e-3

            # Normalize by median body scale to be camera-distance invariant
            scale = np.median(self.body_scale)
            L_y = np.array(self.left_ank_y)
            R_y = np.array(self.right_ank_y)
            L_x = np.array(self.left_ank_x)
            R_x = np.array(self.right_ank_x)
            hip_y = np.array(self.hip_y)

            # Cadence via vertical oscillations of ankles (y increases downward in image)
            # Find peaks on -ankle_y (so up swings are peaks) for both ankles
            signal = smooth(-(L_y - L_y.mean()) + -(R_y - R_y.mean()), k=5)
            # Dynamic prominence threshold
            prom = max(0.02, 0.5 * np.std(signal))
            peaks, _ = find_peaks(signal, prominence=prom, distance=max(3, int(0.2 * self.fps_target)))
            if len(peaks) > 1:
                period_s = np.median(np.diff(t[peaks]))
                spm = 60.0 / max(period_s, 1e-3)  # steps per minute (combined)
            else:
                spm = 0.0
```



```
human activity.py
class SimpleGaitClassifier:  1 usage
    def predict(self, feats):  3 usages
        }


# ----------------------------------------
# Main app
# ----------------------------------------

def draw_hud(frame, feats, label, conf, scores, fps):  1 usage
    h, w = frame.shape[:2]
    pad = 10
    # Background panel
    cv2.rectangle(frame, (pad, pad), (int(w*0.45), pad + 150), (0, 0, 0), -1)
    cv2.addWeighted(frame[pad:pad+130, pad:int(w*0.45)], alpha 0.4, np.zeros_like(frame[pad:pad+130, pad:int(w*0.45)]),

    # Text block
    y = pad + 25
    cv2.putText(frame, text f"{label}  ({conf*100:.0f}%)", org (pad + 12, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale 0.8,
    y += 24
    cv2.putText(frame, text f"SPM: {feats['steps_per_min']:.1f}", org (pad + 12, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale
    y += 22
    cv2.putText(frame, text f"Stride: {feats['stride_proxy']:.3f}", org (pad + 12, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale
    y += 22
    cv2.putText(frame, text f"VertOsc: {feats['vertical_osc']:.3f}", org (pad + 12, y), cv2.FONT_HERSHEY_SIMPLEX, fontScale
    y += 22
activity.py                                                    168:1  CRLF  UTF-8  4 spaces  Python 3.11 (PythonProject)
```

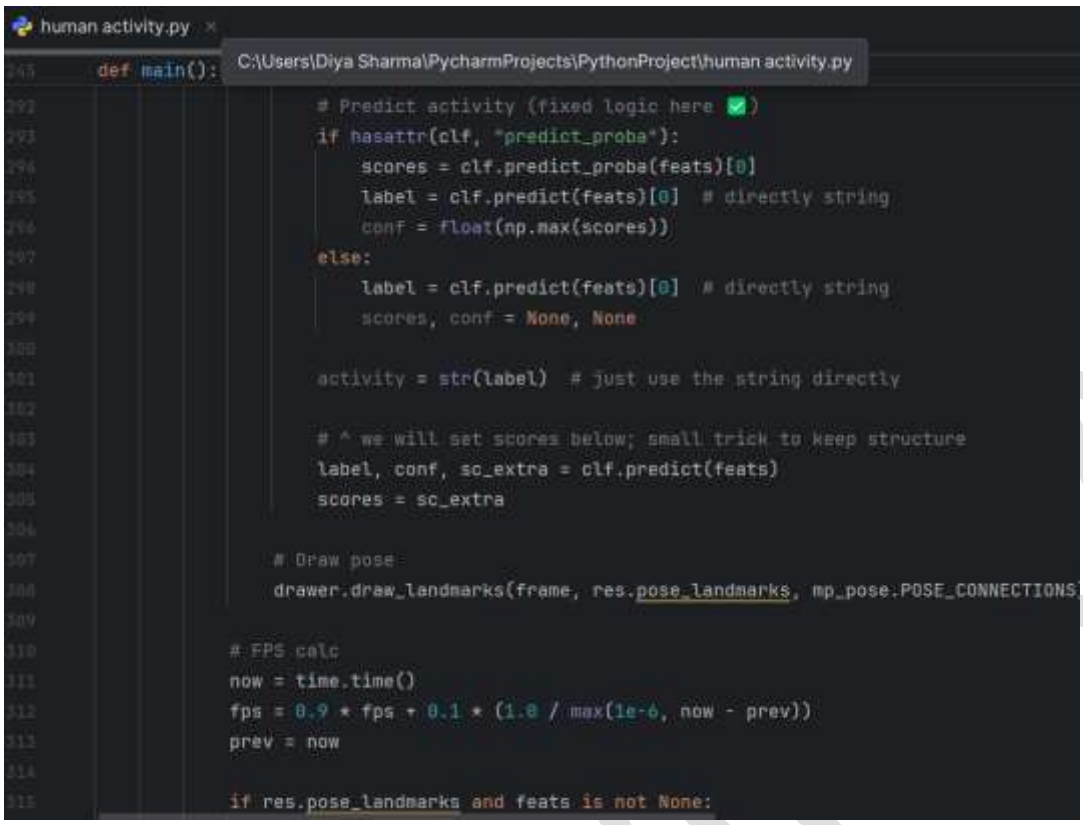


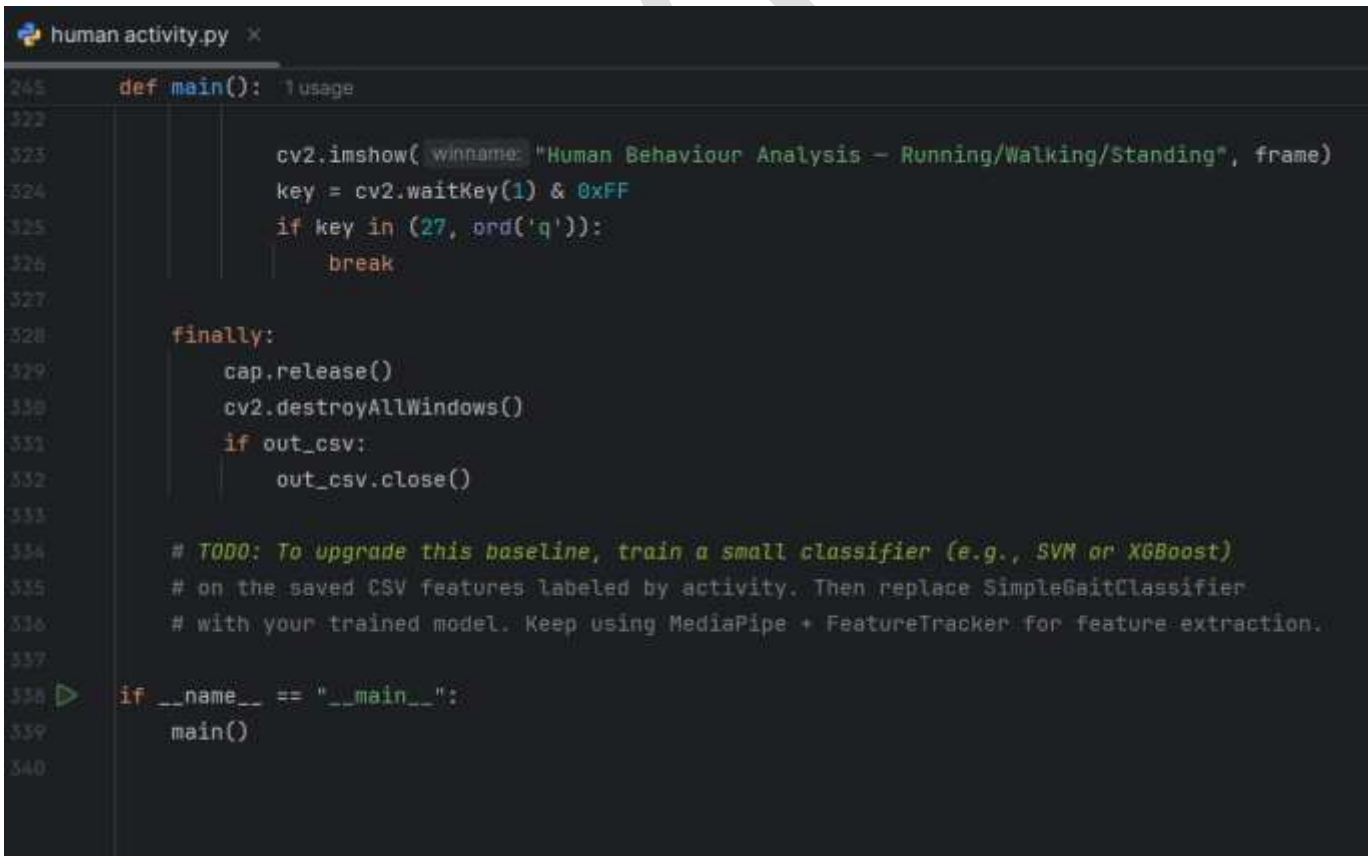
```
def draw_hud(frame, feats, label, conf, scores, fps):  1 usage
    # Score bar
    bar_w = int(w * 0.25)
    bx, by = int(w*0.48), pad
    cv2.putText(frame, text "Scores", org (bx, by + 18), cv2.FONT_HERSHEY_SIMPLEX, fontScale 0.6, color (255,255,255),
    for i, (name, sc) in enumerate([("run", scores["run_score"]), ("walk", scores["walk_score"]), ("stand", scores["stand
        y0 = by + 28 + i*30
        cv2.rectangle(frame, (bx, y0), (bx + bar_w, y0 + 18), (60,60,60), 1)
        cv2.rectangle(frame, (bx, y0), (bx + int(bar_w * max(0.0, min(1.0, sc))), y0 + 18), (80,220,80), -1)
        cv2.putText(frame, text f"{name}: {sc:.2f}", org (bx + 6, y0 + 14), cv2.FONT_HERSHEY_SIMPLEX, fontScale 0.5,


def main():  1 usage
    ap = argparse.ArgumentParser()
    ap.add_argument( name_or_flags "--video", type=str, default=None, help="Path to a video file (defaults to webcam)")
    ap.add_argument( name_or_flags "--save_csv", type=str, default=None, help="Optional: save per-frame features to CSV")
    ap.add_argument( name_or_flags "--fps", type=float, default=30.0, help="Assumed FPS for texture windows")
    ap.add_argument( name_or_flags "--mirror", action="store_true", help="Mirror webcam preview left-right")
    args = ap.parse_args()

    cap = cv2.VideoCapture(0 if args.video is None else args.video)
    if not cap.isOpened():
        raise SystemExit("Error: cannot open webcam/video.")

    mp_pose = mp.solutions.pose
```


human activity.py
C:\Users\Diya Sharma\PycharmProjects\PythonProject\human activity.py
def main():
# Predict activity (fixed logic here ✅)
if hasattr(clf, "predict_proba"):
scores = clf.predict_proba(feats)[0]
label = clf.predict(feats)[0] # directly string
conf = float(np.max(scores))
else:
label = clf.predict(feats)[0] # directly string
scores, conf = None, None
activity = str(label) # just use the string directly
# ^ we will set scores below; small trick to keep structure
label, conf, sc_extra = clf.predict(feats)
scores = sc_extra
# Draw pose
drawer.draw_landmarks(frame, res.pose_landmarks, mp_pose.POSE_CONNECTIONS
# FPS calc
now = time.time()
fps = 0.9 * fps + 0.1 * (1.0 / max(1e-6, now - prev))
prev = now
if res.pose_landmarks and feats is not None:


human activity.py
def main(): 1 usage
cv2.imshow( winname: "Human Behaviour Analysis – Running/Walking/Standing", frame)
key = cv2.waitKey(1) & 0xFF
if key in (27, ord('q')):
break
finally:
cap.release()
cv2.destroyAllWindows()
if out_csv:
out_csv.close()
# TODO: To upgrade this baseline, train a small classifier (e.g., SVM or XGBoost)
# on the saved CSV features labeled by activity. Then replace SimpleGaitClassifier
# with your trained model. Keep using MediaPipe + FeatureTracker for feature extraction.
if __name__ == "__main__":
main()

## 8.3. Detection of people and animals, counting them, and track their activities in real time using computer vision and deep learning – Devansh Sawhney

### 8.3.1. Tools and Technologies Used

- **Python**
- YOLO (Ultralytics) – object and pose detection
- OpenCV – video processing and visualization
- NumPy – numerical computation
- SciPy (Hungarian algorithm) – stable tracking and ID matching

### 8.3.2. Methodology

#### 8.3.2.1. Object Detection and Counting

The proposed surveillance system employs a deep learning–based object detection approach using the YOLO (You Only Look Once) algorithm to identify and classify objects present in each video frame captured by the drone camera. YOLO is a single-stage detection model that performs object localization and classification simultaneously, making it highly suitable for real-time aerial surveillance applications. In this system, the YOLO model is configured to detect specific object classes relevant to a college campus environment. Class ID 0 corresponds to persons (students), while Class ID 16 corresponds to dogs, enabling the system to differentiate between human and animal presence within the monitored area. For every processed video frame, the total number of detected people is calculated, and dogs are counted separately. In addition to per-frame counts, a global counter is maintained to track the number of unique individuals observed over time. This prevents repeated counting of the same person as the drone continues to patrol the campus. Such object detection and counting capabilities allow the quadcopter to continuously monitor student strength in open spaces, corridors, and lawns, while also identifying the presence of animals in restricted or sensitive areas. The ability to perform this task in real time significantly enhances situational awareness and reduces the reliance on manual monitoring.

#### 8.3.2.2. Person Tracking and Unique ID Assignment

While object detection identifies individuals in individual frames, continuous surveillance requires maintaining the identity of each person across multiple frames. To address this requirement, a custom multi-person tracking system has been implemented. Each detected person is assigned a unique identification number (ID) when first observed, which remains associated with that individual as long as they remain within the camera's field of view. This tracking mechanism ensures that the same person is not counted multiple times across consecutive frames.

The tracking algorithm utilizes a combination of spatial information and appearance-based features to maintain identity consistency. The center point of each person's bounding box is tracked across frames to estimate motion continuity. In addition, color histogram–based appearance features are extracted from each person's region of interest, allowing the system to re-identify individuals after short-term occlusions or temporary disappearance from the camera view. To optimally match new detections with existing tracks, the Hungarian assignment algorithm is employed, which minimizes

SRIVIRA

the overall matching cost between detected objects and tracked identities. This approach ensures stable tracking even in crowded scenes, prevents ID switching, and maintains an accurate count of unique individuals on campus.

**Activity Tracking: Idle, Walking, and Running**

Beyond detection and tracking, the system performs basic human activity recognition to analyze movement patterns. Activity classification is achieved through motion-based speed estimation, which tracks the displacement of each person's center point across consecutive frames. The displacement per unit time is calculated using Euclidean distance, providing an estimate of the person's instantaneous movement speed. To reduce noise caused by camera motion or detection jitter, an exponential moving average is applied to smooth the speed values over time.

Based on predefined speed thresholds, each tracked individual is classified into one of three activity states: idle, walking, or running. Very low speeds indicate idle or stationary behavior, moderate speeds correspond to walking, and high speeds are classified as running. This activity tracking enables the drone to detect unusual or potentially suspicious behavior, such as running in restricted or sensitive areas. The use of rule-based speed thresholds ensures fast and interpretable activity classification without requiring complex training datasets, making the approach practical for real-time aerial surveillance.

### 8.3.2.3. Visual Output and Real-Time Monitoring

To facilitate effective monitoring, the system provides a comprehensive real-time visual output by overlaying relevant information directly on the video feed. Bounding boxes are drawn around detected people and dogs, and each person is labeled with their unique ID for easy identification. The current activity state of each individual—idle, walking, or running—is displayed near the bounding box, allowing operators to quickly assess movement behavior.

In addition to per-person information, the system displays global statistics such as the total number of people detected, the current frame count, and the frames-per-second (FPS) rate. FPS monitoring is particularly important for evaluating real-time performance during live drone operation. This visual feedback makes the system intuitive to use and allows security personnel or administrators to make immediate decisions based on live surveillance data.

### 8.3.2.4. Testing and Validation

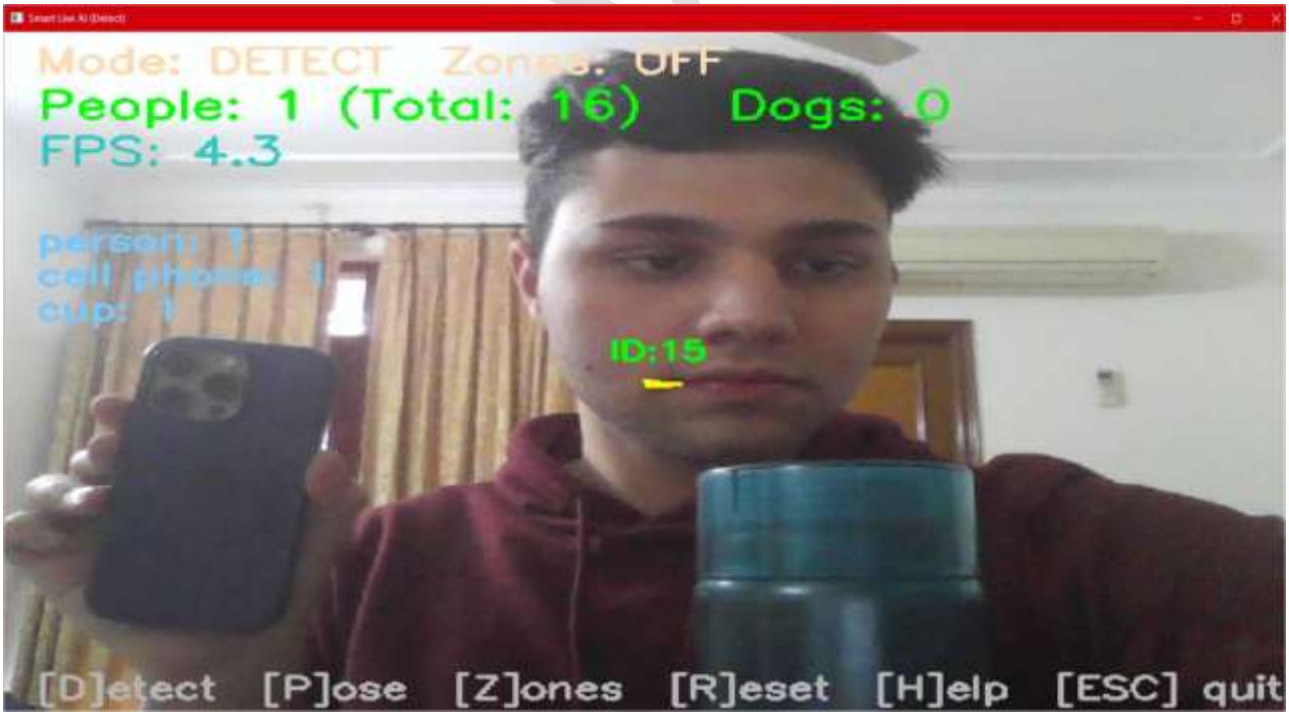
The developed software was tested under controlled conditions using both a live webcam feed and pre-recorded video footage simulating drone-captured scenes. These testing environments allowed validation of the system's performance in terms of detection accuracy, tracking stability, and activity classification reliability. During testing, people and dogs were detected consistently, and unique tracking IDs remained stable even during moderate movement and short occlusions. Activity labels dynamically updated in response to changes in motion speed, confirming the effectiveness of the motion-based activity analysis.

Furthermore, the system maintained real-time performance throughout testing, demonstrating its suitability for deployment in live surveillance scenarios. The successful validation under these conditions confirms that the software is robust and capable of handling practical surveillance requirements in a campus environment.

SRIVIPRA

## 8.4. Images of work done and applications




Mode: POSE Zones: OFF
People: 1 (Total: 20) Dogs: 0
FPS: 3.5
idle
ID:19
[D]etect [P]ose [Z]ones [R]eset [H]elp [ESC] quit

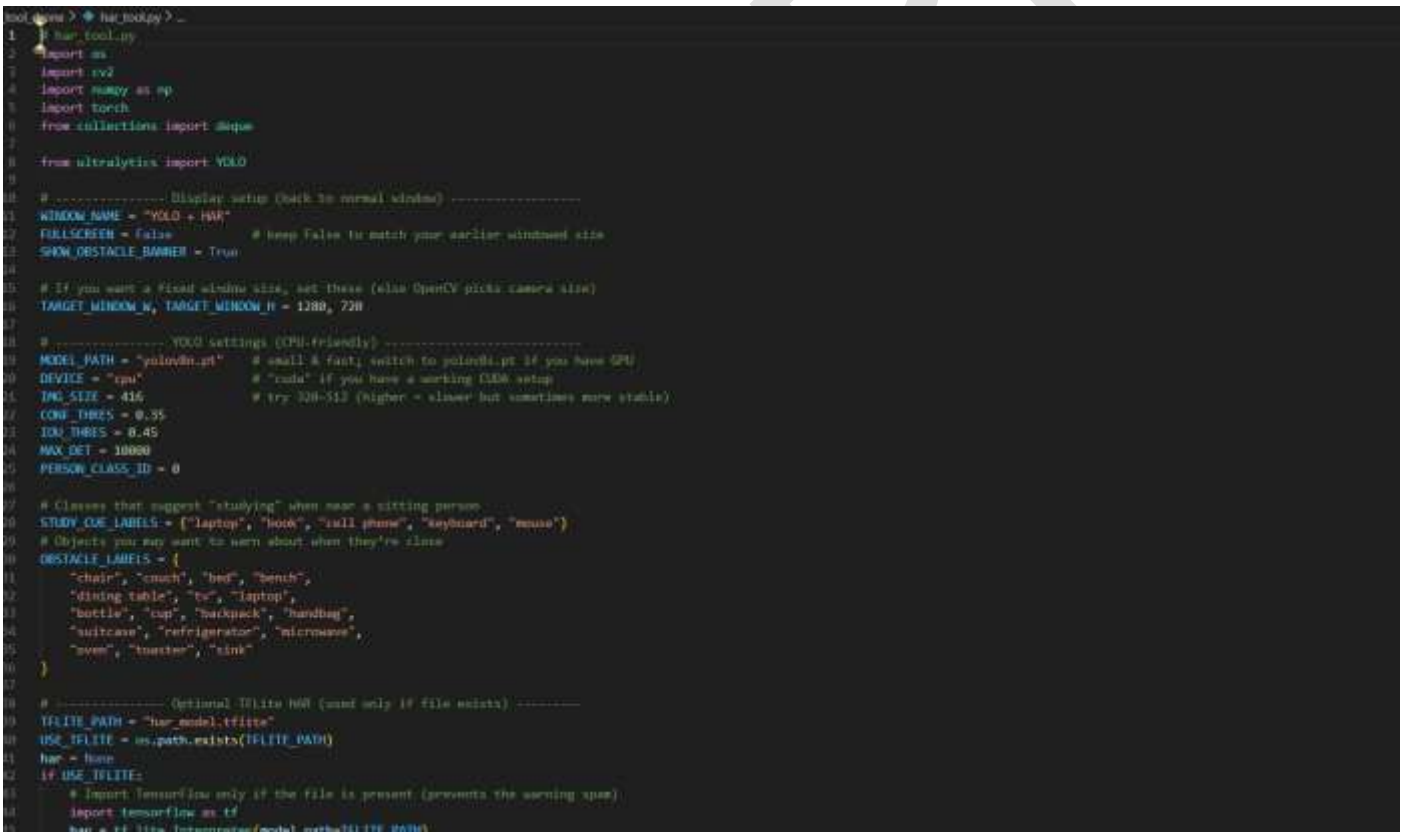

Mode: POSE Zones: OFF
People: 1 (Total: 20) Dogs: 0
FPS: 3.3
walking
ID:19
[D]etect [P]ose [Z]ones [R]eset [H]elp [ESC] quit

## 8.5. Object Identification and Activity Detection Software- Deepika

The software developed for this project combines both YOLOv8 and Haar Cascade models to perform basic surveillance-related tasks. Haar Cascade is used for fast human detection, while YOLOv8 handles detailed object identification.

In addition to identifying objects, the software can also recognize basic human activities such as sitting, standing, and walking. This is done by observing movement patterns and changes in posture across consecutive video frames. Each detected person is given a unique identity so that their movement and activity can be followed consistently.

The software does **not focus on counting the total number of objects**. Instead, its main purpose is to identify objects, recognize human presence, and determine the activity being performed. The results are shown directly on the video screen using labels and bounding boxes, making the output easy to understand.



```python
HAR_LABELS = ["Walking", "Running", "Sitting", "Standing", "Unknown"]

# Heuristic thresholds used when TFLite is absent
MOTION_THR = 18.0     # mean abs diff (0-255) inside person box
ASPECT_STAND = 1.6    # h/w > this → tall → standing-ish

# ---------- Helpers ----------
def put_text(img, text, org, color=(0, 255, 0), scale=0.9, thickness=2):
    cv2.putText(img, text, org, cv2.FONT_HERSHEY_SIMPLEX, scale, color, thickness, cv2.LINE_AA)

def iou(boxA, boxB):
    xA = max(boxA[0], boxB[0]); yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2]); yB = min(boxA[3], boxB[3])
    inter = max(0, xB - xA) * max(0, yB - yA)
    if inter == 0: return 0.0
    areaA = (boxA[2] - boxA[0]) * (boxA[3] - boxA[1])
    areaB = (boxB[2] - boxB[0]) * (boxB[3] - boxB[1])
    return inter / float(areaA + areaB - inter + 1e-6)

def close_obstacle(box, frame_shape, area_ratio=0.08):
    x1, y1, x2, y2 = box
    H, W = frame_shape[:2]
    barea = max(0, (x2 - x1)) * max(0, (y2 - y1))
    return (barea / float(W * H + 1e-6)) > area_ratio

def har_tflite_on_crop(bgr):
    if bgr.size == 0: return "Unknown"
    rgb = cv2.cvtColor(bgr, cv2.COLOR_BGR2RGB)
    rgb = cv2.resize(rgb, (224, 224)).astype(np.float32) / 255.0
    rgb = np.expand_dims(rgb, 0)
    har.set_tensor(har_in, rgb)
    har.invoke()
    probs = har.get_tensor(har_out)[0]
    idx = int(np.argmax(probs))
    return HAR_LABELS[idx] if 0 <= idx < len(HAR_LABELS) else "Unknown"

def har_heuristic(bgr, prev_gray_crop, motion_thr=MOTION_THR):
    if bgr.size == 0:
        return "Unknown", None, 0.0
    gray = cv2.cvtColor(bgr, cv2.COLOR_BGR2GRAY)
    motion = 0.0
    if prev_gray_crop is not None:
        # same size for diff
```

```python
class Track:
    def __init__(self, tid, box):
        self.age = 0
        self.misses = 0
        self.activity_hist = deque(maxlen=8)
        self.prev_gray = None

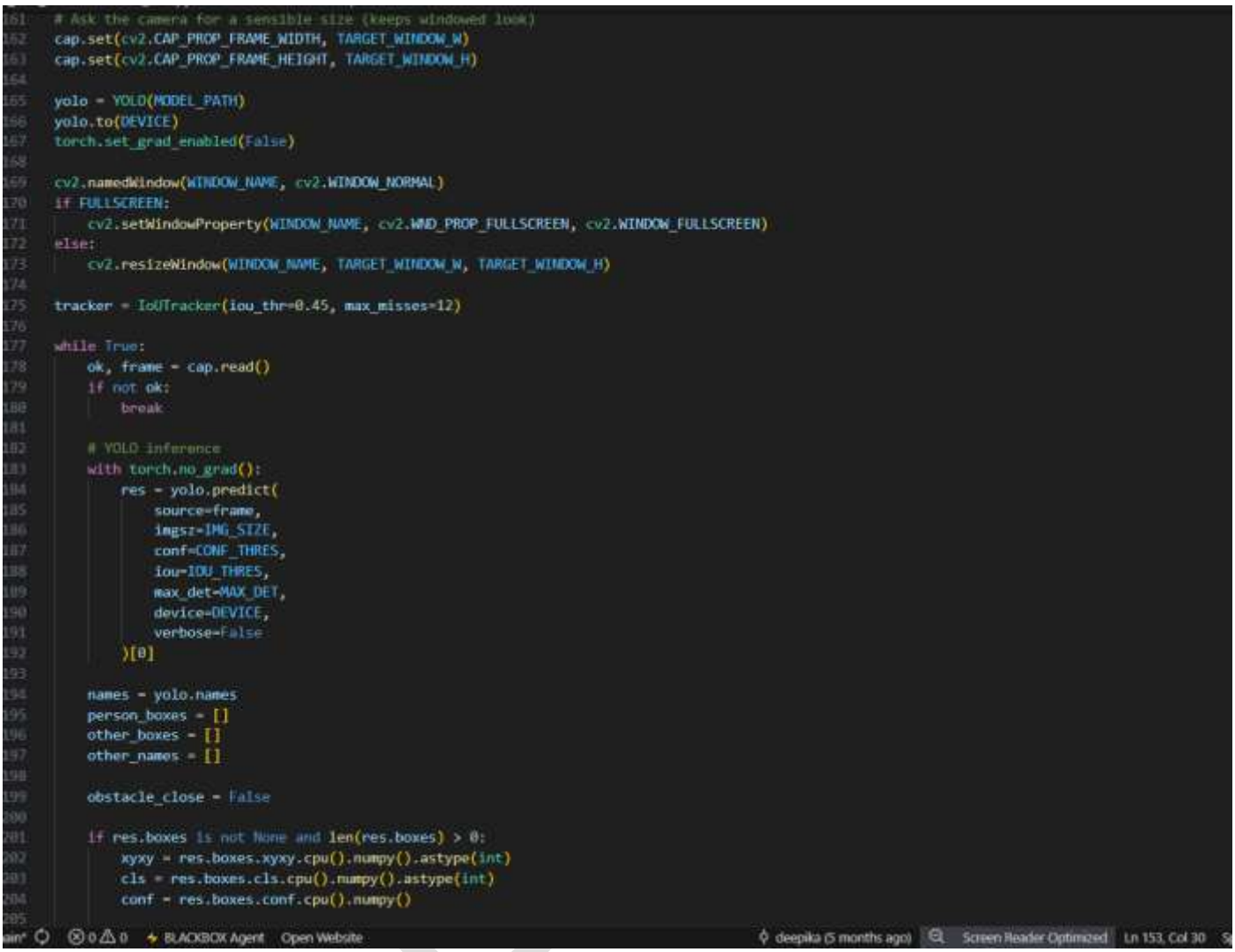
    def update(self, box):
        self.box = box
        self.age += 1
        self.misses = 0

    def vote(self, default="Unknown"):
        if not self.activity_hist: return default
        # majority vote with last item bias
        vals, counts = np.unique(self.activity_hist, return_counts=True)
        return vals[np.argmax(counts)]

class IoUTracker:
    def __init__(self, iou_thr=0.5, max_misses=10):
        self.iou_thr = iou_thr
        self.max_misses = max_misses
        self.tracks = []
        self.next_id = 1

    def step(self, dets):
        # dets: list of (x1,y1,x2,y2)
        unmatched = set(range(len(dets)))
        # greedy match
        for tr in self.tracks:
            best_j, best_iou = -1, 0.0
            for j in list(unmatched):
                i = iou(tr.box, dets[j])
                if i > best_iou:
                    best_iou, best_j = i, j
            if best_j >= 0 and best_iou >= self.iou_thr:
                tr.update(dets[best_j])
                unmatched.discard(best_j)
            else:
                tr.misses += 1

        # new tracks
        for j in unmatched:
            self.tracks.append(Track(self.next_id, dets[j]))
```



```
# Ask the camera for a sensible size (keeps windowed look)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, TARGET_WINDOW_W)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, TARGET_WINDOW_H)

yolo = YOLO(MODEL_PATH)
yolo.to(DEVICE)
torch.set_grad_enabled(False)

cv2.namedWindow(WINDOW_NAME, cv2.WINDOW_NORMAL)
if FULLSCREEN:
    cv2.setWindowProperty(WINDOW_NAME, cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
else:
    cv2.resizeWindow(WINDOW_NAME, TARGET_WINDOW_W, TARGET_WINDOW_H)

tracker = IoUTracker(iou_thr=0.45, max_misses=12)

while True:
    ok, frame = cap.read()
    if not ok:
        break

    # YOLO inference
    with torch.no_grad():
        res = yolo.predict(
            source=frame,
            imgsz=IMG_SIZE,
            conf=CONF_THRES,
            iou=IOU_THRES,
            max_det=MAX_DET,
            device=DEVICE,
            verbose=False
        )[0]

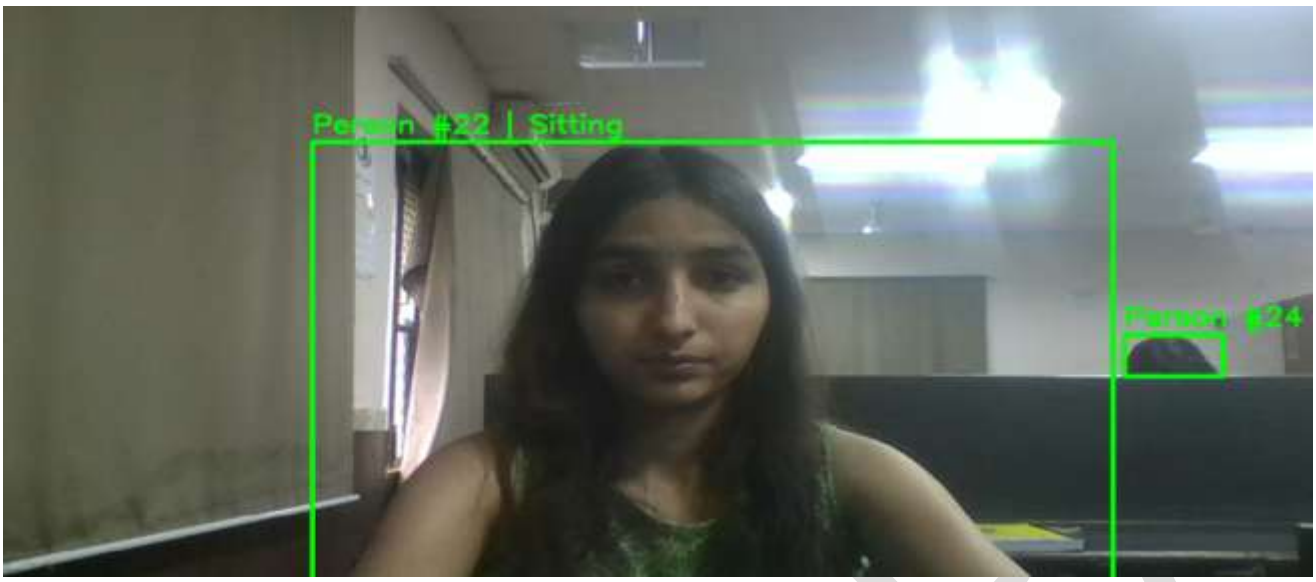
    names = yolo.names
    person_boxes = []
    other_boxes = []
    other_names = []

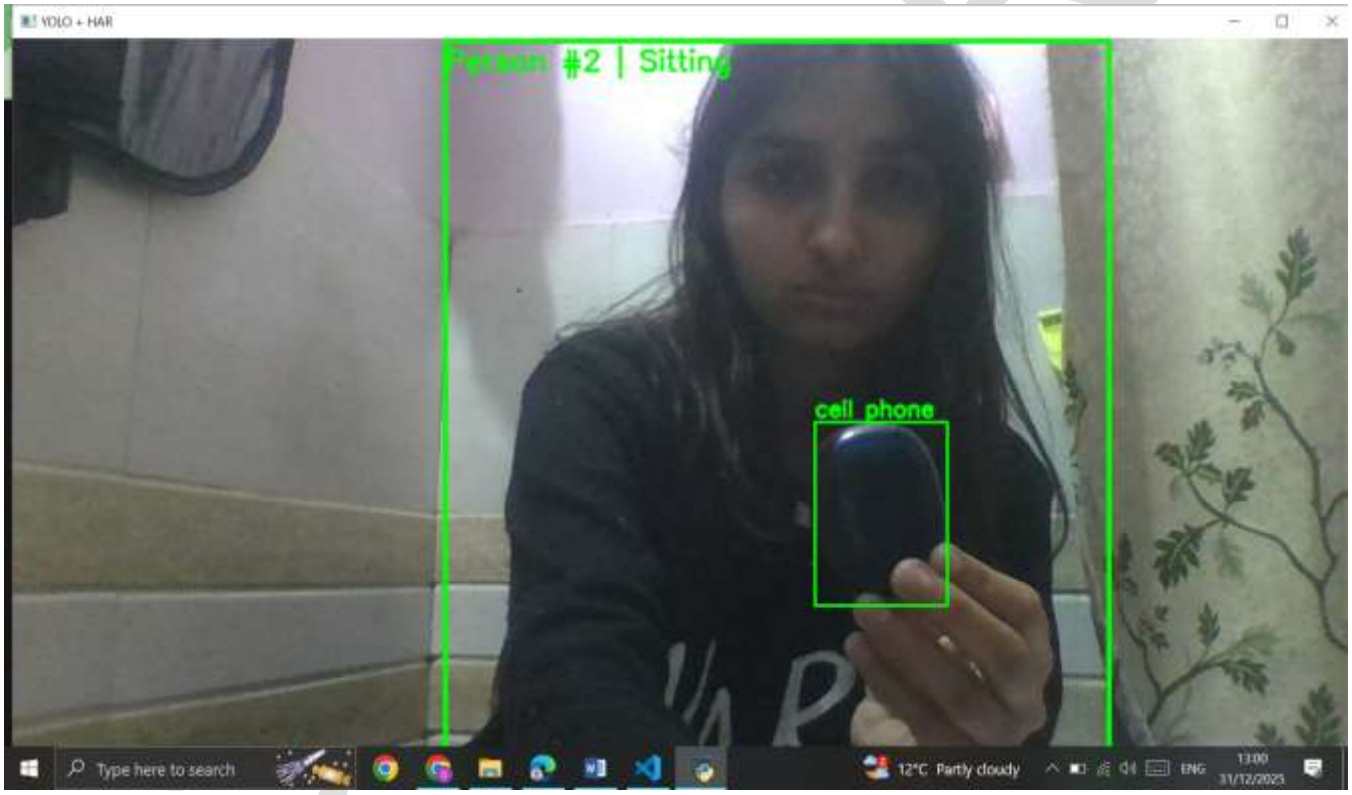
    obstacle_close = False

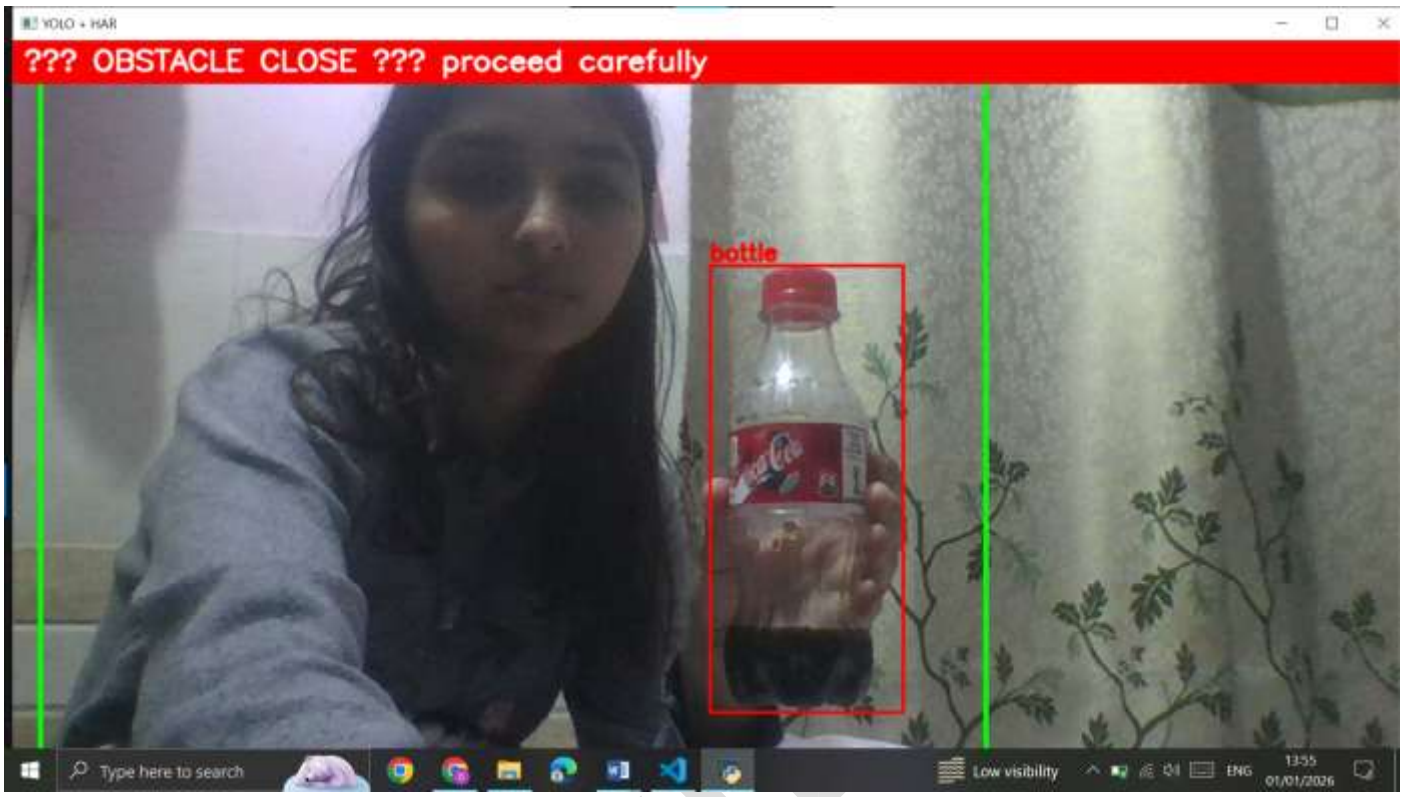
    if res.boxes is not None and len(res.boxes) > 0:
        xyxy = res.boxes.xyxy.cpu().numpy().astype(int)
        cls = res.boxes.cls.cpu().numpy().astype(int)
        conf = res.boxes.conf.cpu().numpy()
```



SRI


Person #22 | Sitting
Person #24


YOLO + HAR
Person #2 | Sitting
cell phone
Type here to search
12°C Partly cloudy
ENG
13:00
31/12/2025




Minor challenges such as lighting variations were observed, but overall system performance remained accurate and consistent. The results confirm that the project objectives were successfully achieved.

SRI-VI

## 8.6. Detecting and counting the number of people and common objects- Sana Singh

```python
import cv2
from ultralytics import YOLO
from collections import Counter

# Load YOLO model (nano version = fastest)
model = YOLO("yolo11n.pt")  # auto-downloads first time

# Open laptop webcam (0 = default cam)
cap = cv2.VideoCapture(0)

if not cap.isOpened():
    print("Error: Could not open webcam")
    exit()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Run YOLO detection
    results = model(frame, imgsz=480, verbose=False)
    detected_species = []

    # Process results
    for r in results:
        for box in r.boxes:
            cls = int(box.cls[0])
            label = model.names[cls]   # e.g., "person", "cat", "dog"
```

SRI

```
            # Track humans, animals, birds etc.
            if cls in [0,14,15,16,17,18,19,20,21,22,23]:
                detected_species.append(label)

                # Draw bounding box
                x1, y1, x2, y2 = map(int, box.xyxy[0])
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, label, (x1, y1 - 5),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

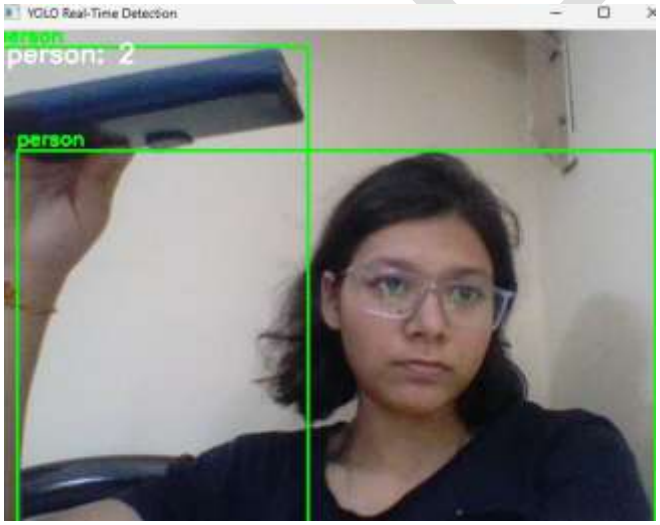
    # Show counts on screen
    counts = Counter(detected_species)
    y = 30
    for species, count in counts.items():
        cv2.putText(frame, f"{species}: {count}", (10, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 255, 255), 2)
        y += 30

    # Display frame
    cv2.imshow("YOLO Real-Time Detection", frame)

    # Quit when 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

# OBSERVED VIDEO:



# 9. Advantages of Smart Surveillance Drones

Smart surveillance drones offer significant advantages over traditional ground-based monitoring systems by combining aerial mobility, real-time data acquisition, and artificial intelligence. The integration of advanced sensors, autonomous flight control, and AI-based analysis enables drones to perform surveillance tasks more efficiently, accurately, and safely across various domains.

## 9.1. Enhanced Surveillance and Monitoring

One of the primary advantages of smart surveillance drones is their ability to provide **real-time aerial monitoring** over large areas. Unlike fixed cameras, drones can move dynamically and cover regions that are otherwise difficult to access. This makes them highly effective for continuous surveillance of urban areas, industrial sites, forests, and open terrain.



Figure 46: Drone in function

## 9.2. Security and Intrusion Detection

Smart drones equipped with AI-based object detection and region-based security mechanisms can identify unauthorized access or suspicious activity. Features such as polygon-based boundary monitoring and automated breach detection make drones suitable for **security applications** in restricted areas such as campuses, warehouses, power plants, and sensitive installations.



Figure 47: Drone flying on Campus

## 9.3. Search and Rescue Operations

In emergency situations such as natural disasters, accidents, or missing-person cases, smart drones can be rapidly deployed to locate individuals and assess conditions. Their ability to cover large areas quickly and provide live video feeds significantly reduces response time. AI-based detection further enhances their effectiveness by identifying humans or objects of interest in challenging environments.



## 9.4. Border Surveillance and National Security

Smart surveillance drones have strong potential applications in **border monitoring and national security**, particularly in countries with extensive and diverse borders such as India. Drones can patrol border regions, monitor movement, and provide real-time intelligence without risking human lives. AIdriven analysis improves detection accuracy and reduces operator workload, making border surveillance more efficient and reliable.



## 9.5. Military and Defence Applications

In military contexts, smart drones can be used for reconnaissance, intelligence gathering, and situational awareness. Their ability to operate in hostile or inaccessible areas makes them valuable assets for surveillance missions. The integration of AI enables automated threat detection and continuous monitoring, supporting strategic decision-making while minimizing human exposure to danger.

SRISHTI



## 9.6. Reduced Human Risk and Cost Efficiency

By replacing or assisting human patrols, smart drones reduce physical risk to personnel in dangerous environments. Additionally, once deployed, drones offer a cost-effective solution for long-term surveillance compared to installing and maintaining extensive ground-based infrastructure.



## 9.7. Future Scope: Fully AI-Controlled Surveillance Drones

With further advancements in artificial intelligence, smart surveillance drones have the potential to evolve into **fully autonomous AI-controlled systems**. Such drones could independently plan missions, detect and analyse events, make decisions, and respond to situations with minimal human intervention. This future scope highlights the scalability and long-term relevance of smart surveillance drone projects.

# 10.Conclusion

This project successfully demonstrated the design, assembly, and implementation of a working F450 quadcopter drone with a smart surveillance application. The quadcopter was assembled using carefully selected hardware components and configured using the APM 2.8 flight controller with Mission Planner. The drone was tested and achieved stable flight performance, successfully flying at an altitude of approximately 10 to 12 feet, demonstrating reliable control and balance.

In addition to the hardware development, a custom AI-based surveillance software was developed and implemented. By integrating a camera with the drone and using Ultralytics YOLO11 pre-trained models, the system is capable of performing real-time AI-based object detection. The software further extends basic detection by incorporating advanced AI features, making the surveillance system intelligent and adaptable.

Overall, the project achieved its intended objectives by combining drone technology with artificial intelligence to create a functional and scalable smart surveillance platform. The work provided valuable hands-on experience in drone assembly, flight control systems, software development, and AI integration, and it lays a strong foundation for future enhancements such as higher autonomy, improved flight altitude, and fully AI-controlled drone operations.

SRI-VIPRA

# 11. BILLING

**Budget Allocation**

The project was executed under a strict financial framework sanctioned by the college administration.

- **Initial Sanctioned Budget:** ₹25,000.00
- **Additional Grant:** ₹2,500.00
- **Total Available Budget: ₹27,500.00**
- **Total Expenses: ₹ 28,777.86.00**
- **Over budget by ₹1,277.86.**

| Part Name/Description | Quantity | Unit Price (₹) | Total Price (₹) | Tax | final cost per shipment |
|---|---|---|---|---|---|
| [30219] Pro-Range Propellers 1045(10X4.5) ABS DJI Black 1CW+1CCW-1pair- Premium Quality | 2 | ₹ 69.62 | ₹ 139.24 | | |
| [std_shipping] STANDARD SHIPPING | 1 | ₹ 41.53 | ₹ 41.53 | ₹ 14.43 | ₹ 194.96 |
| Duracell Ultra AA Alkaline Battery(Pack) | 1 | 160.17 | 189 | 28.84 | |
| Link Round 30 mm Mini Travel Single Pad Lock with 3 | 1 | 48.31 | 57 | 8.7 | |
| Wingz Cable Organizer (Black) (HSN-39269099) \| 100 pieces | 1 | 110.17 | 130 | 19.84 | ₹ 376.00 |
| DJI 2212 920KV Brushless Motor | 1 | ₹ 582.68 | ₹ 611.81 | ₹ 29.13 | |
| [30219] Pro-Range Propellers 1045(10X4.5) ABS DJI Black 1CW+1CCW-1pair- Premium Quality | 2 | ₹ 62.95 | ₹ 132.19 | ₹ 6.29 | |
| [std_shipping] STANDARD SHIPPING | 1 | ₹ 41.53 | ₹ 7.47 | ₹ 49.00 | ₹ 793.00 |
| Orange 11.1V 3300mAh 35C 3S Lithium Polymer Battery Pack | 2 | ₹ 4,018.64 | ₹ 4,742.00 | ₹ 723.36 | |

| | | | | | |
|---|---|---|---|---|---|
| [7364] IMAX B3 AC Pro Compact Balance Charger for 2S-3S LiPo | 1 | ₹ 285.59 | ₹ 337.00 | ₹ 51.41 | ₹ 2,708.00 |
| Caddx Ratel 2 Analog FPV Camera | 1 | ₹ 2,875.00 | ₹ 2,998.90 | - | ₹ 2,998.90 |
| [15907] Standard 30A BLDC ESC Electronic Speed Controller with Connector | ₹ 5.00 | ₹ 315.24 | ₹ 1,655.00 | ₹ 78.81 | |
| [88756] Plastic GPS Antenna Foldable Mount Holder | 1 | ₹ 158.10 | ₹ 166.00 | ₹ 7.90 | |
| [57286] NEO-M8N GPS with Compass for Pixhawk with extra connector for APM | 1 | ₹ 1,969.52 | ₹ 2,068.00 | ₹ 98.48 | ₹ 3,889.00 |
| [33428] Flysky FS-i6X 2.4GHz 6CH AFHDS 2A RC Transmitter With FS-iA10B 2.4GHz 10CH Receiver | 1 | ₹ 4,380.00 | ₹ 4,599.00 | ₹ 219.00 | ₹ 4,599.00 |
| Flight controller board | 1 | 4046.67 | ₹ 4,249.00 | ₹ 202.33 | |
| Glass fiber shock absorber | 1 | 164.41 | ₹ 194.00 | ₹ 29.59 | |
| propellers | 8 | 31.36 | ₹ 296.00 | ₹ 45.15 | |
| BDLC moter CW | 2 | 618.1 | ₹ 1,298.00 | ₹ 61.81 | |
| BDLDC motor CCW | 2 | 608.57 | ₹ 1,278.00 | ₹ 60.69 | |
| Frame and landing gear | 1 | 760 | ₹ 798.00 | ₹ 38.00 | |
| Lipo battery connector | 1 | 49.15 | ₹ 58.00 | ₹ 8.85 | |
| Nylon strap | 1 | 49.15 | ₹ 58.00 | ₹ 8.85 | ₹ 8,229.00 |
| Duracell Ultra A | 1 | 161.02 | ₹ 190.00 | ₹ 28.98 | ₹ 190.00 |
| 5.8G UVC OTG Android AV Phone Receiver | 1 | ₹ 2,313.33 | ₹ 115.67 | ₹ 2,429.00 | ₹ 2,429.00 |
| grand total | | | | | ₹ 28,777.86 |

## 12. REFERENCES

[1] Ultralytics YOLO – Official Website. https://www.ultralytics.com
[2] Ultralytics YOLO Documentation. https://docs.ultralytics.com
[3] ArduPilot – Official Website. https://ardupilot.org
[4] ArduPilot Mission Planner Documentation. https://ardupilot.org/dev/docs
[5] DJI – Official Drone Technology Website. https://www.dji.com
[6] OpenCV – Open Source Computer Vision Library. https://opencv.org
[7] Google MediaPipe Documentation. https://developers.google.com/mediapipe
[8] Ollama – Local Large Language Model Platform. https://ollama.com

SRI-VIPRA