Simulation-Based Autonomous Drone Using Python and Computer Vision

[1]Prof.Londhe Dhanashri, [2]Mr. Kamble Kiran Ram
[1]*Associate Professor, Zeal College of Engineering, Narhe Pune-411041*
[2]*PG Scholar, Zeal College of Engineering & Research Narhe Pune-411041*

Abstract- Unmanned Aerial Vehicles (UAVs) commonly known as drones, have gained significant importance in applications such as surveillance, agriculture, disaster management, and delivery systems. This paper presents the design and implementation of a simulation-based autonomous drone using Python programming and computer vision techniques. The proposed system utilizes the pysimverse simulation environment to control a virtual drone capable of autonomous navigation, waypoint tracking, and object detection. OpenCV is employed for real-time image processing and detection tasks. The system demonstrates efficient performance in a controlled simulation environment, reducing the need for expensive hardware testing. The results validate that the integration of artificial intelligence with UAV systems can significantly enhance autonomy and operational efficiency.

I.INTRODUCTION

Unmanned Aerial Vehicles (UAVs) are increasingly used in modern engineering applications due to their flexibility, mobility, and efficiency. Traditional drone systems require manual control, which limits their usability in complex and dynamic environments. Autonomous drones, powered by artificial intelligence and computer vision, provide a promising solution to overcome these limitations.

The aim of this research is to develop a simulation-based autonomous drone system using Python. Simulation platforms enable safe testing and validation without physical risks or high costs. This work focuses on integrating drone control with computer vision to achieve intelligent navigation and decision-making.

Unmanned Aerial Vehicles (UAVs), commonly known as drones, are aircraft systems that operate without an onboard human pilot. These systems can

be controlled remotely by a human operator or can function autonomously using pre-programmed flight plans, onboard sensors, and advanced algorithms. UAV technology has rapidly evolved over the past decade due to advancements in electronics, communication systems, and artificial intelligence.

Initially developed for military applications such as surveillance, reconnaissance, and combat missions, UAVs are now widely used in various civilian domains. These include agriculture (crop monitoring and spraying), disaster management (search and rescue operations), environmental monitoring, infrastructure inspection, and delivery services. Their ability to access hard-to-reach areas, capture real-time data, and operate under hazardous conditions makes them highly valuable in modern applications.

Modern UAV systems typically consist of multiple components, including flight controllers, GPS modules, cameras, sensors, and communication systems. Integration with technologies such as computer vision, machine learning, and the Internet of Things (IoT) has further enhanced their capabilities, enabling intelligent decision-making and autonomous navigation.

Despite their advantages, UAVs also present several challenges, including regulatory restrictions, safety concerns, limited battery life, and cybersecurity risks. Ongoing research focuses on improving efficiency,

reliability, and autonomy to expand their applications further.

This project explores the development and simulation of UAV systems using Python-based tools, focusing on autonomous navigation, real-time data processing, and intelligent control mechanisms.

II.LITERATURE REVIEW

1 UAV-Based Surveillance Systems
2 Autonomous Navigation Techniques
3 Computer Vision for UAV Applications
4 YOLO-Based Object Detection
5 Traffic Monitoring Using Drones

RESEARCH GAP
1 High cost of physical drone testing
2 Limited AI integration in simulation-based UAVs
3 Lack of real-time traffic monitoring in existing systems
4 Need for autonomous decision-making capability

III.PROBLEM STATEMENT

Manual drone operation is inefficient for large-scale and complex tasks. Hardware-based testing is expensive and risky. There is a need for a simulation-based system that can:

- Perform autonomous navigation
- Detect objects in real-time
- Make intelligent decisions
- Unmanned Aerial Vehicles (UAVs) have gained significant importance in various fields such as surveillance, agriculture, disaster management, and infrastructure inspection. However, the effective deployment of UAVs in real-world environments is still limited by several technical challenges related to autonomy, real-time processing, and navigation accuracy.
- Traditional UAV systems often rely on manual control or basic pre-programmed paths, which restrict their ability to operate efficiently in dynamic and unpredictable environments. In scenarios such as obstacle avoidance, line following, or target tracking, UAVs must process visual data in real time and make intelligent decisions without human intervention. Achieving this level of autonomy requires the integration of computer vision and artificial intelligence techniques.
- Although tools like OpenCV and advanced object detection models such as YOLO have shown promising results, their implementation in UAV systems introduces challenges such as high computational requirements, latency in decision-making, and reduced accuracy under varying environmental conditions (e.g., lighting, noise, and obstacles).
- Additionally, testing UAV algorithms directly on physical drones can be risky, costly, and time-consuming. Simulation environments like PySimVerse provide a safer alternative, but there exists a gap between simulation and real-world performance, particularly in sensor modeling and environmental realism.

IV.PROPOSED SYSTEM

The proposed system is a simulation-based autonomous drone system developed using Python programming and computer vision techniques. The system is designed to perform autonomous flight operations such as takeoff, waypoint navigation, image capturing, and object detection in a virtual environment using AirSim.

The drone operates without continuous human control and makes decisions based on real-time image processing. The proposed system reduces hardware cost and testing risks by using a software simulation platform.

Python Controller
↓
pysimverse Simulator
↓
Autonomous Drone
↓
Camera & Sensors
↓
OpenCV Vision Module
↓
Decision Making

V.METHODOLOGY

System architecture:-
The system consists of the following components:

- Pysimverse Simulator
- Python Control Script
- Computer Vision Module (OpenCV)
- Drone API Interface

System Overview
The proposed system consists of a UAV operating in a simulated environment, where it captures real-time visual data, processes it using computer vision techniques, and makes autonomous navigation decisions.

Key tools used:

- PySimVerse for simulation
- OpenCV for image processing
- YOLO for object detection

Methodology Flow
Start
↓
Initialize System
↓
Connect Drone (Simulation)
↓
Start Camera Stream
↓
Capture Frame
↓
Image Processing (OpenCV)
↓
Feature Detection (Line/Object)
↓
Decision Algorithm
↓
Send Control Commands
↓
Drone Movement
↓
Performance Evaluation
↓
End

Functions

1. Flight and Navigation

The primary function of a drone is to fly and move in different directions.

Functions:

- Takeoff
- Landing
- Hovering
- Forward/backward movement
- GPS navigation

2. Surveillance and Monitoring

Drones are widely used for monitoring large areas.

Applications:

- Security surveillance
- Border monitoring
- Traffic observation
- Forest monitoring

3. Aerial Photography and Videography

Drones capture images and videos from the air.

Uses:

- Photography
- Cinematography
- Event coverage
- Real estate imaging

4. Object Detection and Tracking

Using AI and computer vision, drones can detect and follow objects.

Examples:

- Human detection
- Vehicle tracking
- Face recognition

Implementation:

Simulation Software Drone Connection:-
Install Required Library in pycharm software

Numpy,pysimverse,cv2,opencv-python,cv zone , djitellopy etc.

```
from pysimverse import Drone
import time
drone = Drone()
drone.connect()
```

Takeoff and sleep

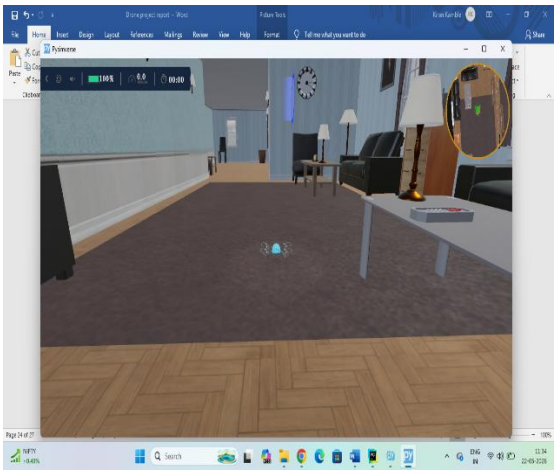
```
import time
drone = Drone()
drone.connect()
drone.take_off()

drone.land
time.sleep(1)
```

Image Capture

```
from pysimverse import Drone
import time
import cv2

drone = Drone()
drone.connect()
time.sleep(1)
drone.streamon()
time.sleep(2)
drone.take_off()

while True:
    frame, is_success = drone.get_frame()

    if not is_success or frame is None:
        print("Frame not received")
        continue

    cv2.imshow("Drone Feed", frame)

    # Press 'q' to exit loop
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
drone.land()
time.sleep(1)
```

Drone move forward backward position

```
drone.move_forward(80)
time.sleep(2)
drone.move_backward(130)
time.sleep(2)
```



Line follower drone code :-

```
import cvzone
from cvzone.ColorModule import ColorFinder
import cv2
from pysimverse import Drone
import time

# Initialize Color Finder
myColorFinder = ColorFinder(trackBar=True)

# Initialize Drone
drone = Drone()
drone.connect()
time.sleep(1)

drone.streamon()
drone.take_off(takeoff_height=30)

# Custom color values for detecting orange.
# 'hmin', 'smin', 'vmin' are the minimum values for Hue, Saturation, and Value.
# 'hmax', 'smax', 'vmax' are the maximum values for Hue, Saturation, and Value.
hsvVals = {'hmin': 10, 'smin': 55, 'vmin': 215, 'hmax': 42, 'smax': 255, 'vmax': 255}

def move_drone(drone, cx):
```

```
    center = 320   # center of frame (640/2)

    if cx is None:
        drone.hover()
        return

    if cx < center - 50:
        drone.move_left()
    elif cx > center + 50:
        drone.move_right()
    else:
        drone.move_forward()

try:
    while True:
        #  Correct method call
        frame, is_success = drone.get_frame()

        if not is_success or frame is None:
            continue

        # Resize frame
        img = cv2.resize(frame, (400, 300))

        # Detect color
        imgOrange, mask = myColorFinder.update(img,
hsvVals)

        # Stack images
        imgStack = cvzone.stackImages([img,
imgOrange, mask], 3, 0.5)

        # Show output
        cv2.imshow("Drone Color Detection",
imgStack)

        # Exit key
        if cv2.waitKey(1) & 0xFF == 27:
            break

finally:
    drone.land()
    time.sleep(1)
    cv2.destroyAllWindows()
```

Traffic Monitoring Drone Using AI and Python

Objective

- Monitor road traffic using a drone camera.
- Detect and count vehicles automatically.
- Calculate traffic density.
- Identify congestion and possible accidents.
- Generate real-time traffic reports.

System Architecture

Drone Camera
↓
Video Stream
↓
OpenCV Processing
↓
YOLO Vehicle Detection
↓
Vehicle Tracking
↓
Traffic Density Calculation
↓
Congestion/Accident Detection
↓
Dashboard & Alerts

Step 1: Drone Captures Video

The drone flies above a road and continuously captures video.

Step 2: Vehicle Detection

AI detects:

- Cars

- Buses
- Trucks
- Motorcycles

Step 3: Vehicle Counting

Each detected vehicle is counted.

Step 4: Traffic Density Calculation

Traffic Density (%) =

$$\frac{\text{Number of Vehicles}}{\text{Road Capacity}} \times 100$$

Example:

1.Vehicles = 40
2. Capacity = 50

Density = 80%

Step 5: Congestion Detection

Density	Status
0–30%	Low Traffic
31–60%	Medium Traffic
61–80%	Heavy Traffic
>80%	Congestion

Step 6: Accident Detection

Possible accident indicators:

1.Vehicle stopped for a long time.

2. Sudden speed drop.

3.Vehicle collision detected.

Python Code for Vehicle Detection

```
from ultralytics import YOLO
import cv2

model = YOLO("yolov8n.pt")

cap = cv2.VideoCapture("traffic.mp4")

while True:
    ret, frame = cap.read()

    if not ret:
        break

    results = model(frame)

    annotated_frame = results[0].plot()

    cv2.imshow("Traffic Monitoring",
annotated_frame)

    if cv2.waitKey(1) == 27:
        break

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

Vehicle Counting Code

```
from ultralytics import YOLO
import cv2

model = YOLO("yolov8n.pt")

cap = cv2.VideoCapture("traffic.mp4")

vehicle_classes = [2, 3, 5, 7]

while True:
    ret, frame = cap.read()

    if not ret:
        break

    results = model(frame)

    count = 0

    for box in results[0].boxes:
        cls = int(box.cls)

        if cls in vehicle_classes:
            count += 1

    cv2.putText(frame,
```

```
            f"Vehicles: {count}",
            (20,50),
            cv2.FONT_HERSHEY_SIMPLEX,
            1,
            (0,255,0),
            2)

    cv2.imshow("Traffic Count", frame)

    if cv2.waitKey(1) == 27:
        break

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

Traffic Density Calculation

```
road_capacity = 50
vehicles = count

density = (vehicles / road_capacity) * 100

print("Traffic Density =", density,"%")
```

Accident Detection Logic

```
if vehicle_speed < 5:
    stopped_frames += 1

if stopped_frames > 300:
    print("Possible Accident Detected"
```

Expected Output

Vehicles Detected : 38
Traffic Density : 76%
Status : Heavy Traffic
Accident Alert : No

Traffic image by drone

VI.OBSERVATIONS

The integrated system performed all major operations successfully:

1) Navigation
2) Image capture
3) Object detection
4) Decision making
5) Traffic monitoring

Drone software simulation (pysimverse) movement

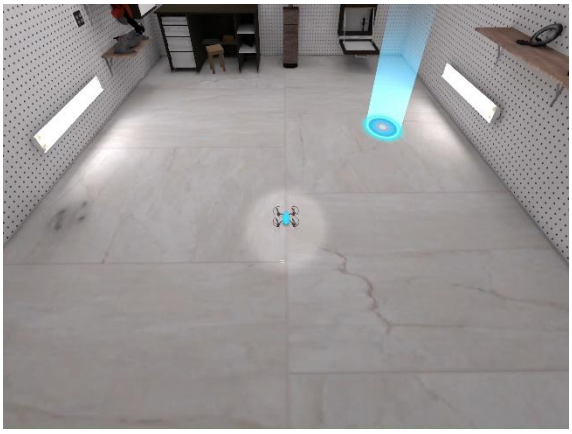
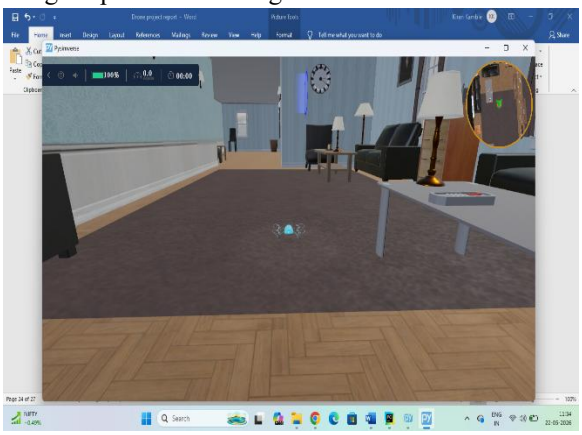


Image capture drone image



Drone agriculture uses

Line follower Drone image in drone using pysimverse simulation software by pycharm

VII.RESULT

The system was tested in the pysimverse simulation environment. The drone successfully performed:

- Stable takeoff and landing
- Accurate waypoint navigation
- Real-time image capture
- Basic object detection

The object detection module showed reliable performance under controlled conditions. The integration of navigation and vision enabled semi-autonomous behavior.

VIII.FUTURE SCOPE

1 Integration of deep learning models (YOLO, CNN)
2 Multi-drone coordination
3 Real-world implementation using hardware
4 Obstacle avoidance using sensors
5 Real drone implementation
6 GPS-based navigation
7 Multi-drone coordination
8 Deep learning optimization
9 Smart city integration 10 Traffic monitoring drone

IX.CONCLUSIONS

This paper presented a simulation-based autonomous drone system using Python and computer vision. The system successfully demonstrated autonomous navigation and object detection in a virtual environment. The results indicate that simulation platforms can significantly accelerate UAV research and development. Future improvements can enhance the system's intelligence and real-world applicability.

REFERENCES

[1] Federal Aviation Administration, *"Unmanned Aircraft Systems (UAS) Overview,"* Available: https://www.faa.gov/uas
[2] International Civil Aviation Organization, *"Manual on Remotely Piloted Aircraft Systems (RPAS),"* 2020.
[3] OpenCV Documentation, *"Open Source Computer Vision Library,"* Available: https://opencv.org
[4] Joseph Redmon et al., *"You Only Look Once: Unified, Real-Time Object Detection,"* in Proc. IEEE CVPR, 2016.
[5] PySimVerse Documentation, *"Drone Simulation Environment for Python,"* 2023.
[6] Microsoft, *"AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,"* Available: https://github.com/microsoft/AirSim
[7] Introduction to UAV Systems, 4th ed., Wiley, 2012.
[8] Computer Vision: Algorithms and Applications, Springer, 2010.
[9] TensorFlow Documentation, Available: https://www.tensorflow.org
[10] PyTorch Documentation, Available: https://pytorch.org