## ДОДАТОК А

Код численного моделювання відстеження об’єктів

```
# FPV Computer Vision System Numerical Simulation
# ------------------------------------------------
# This script simulates the core mathematical models from Section 2.1:
# - Video discretization (time steps)
# - Stochastic detector outputs (classes, boxes, scores)
# - IoU and IoU-cost based association
# - Kalman filter tracking (constant-velocity model in image plane)
# - Inference latency compensation
# - Homography (ground plane <-> image plane) mapping
# - PID-based pointing control from pixel errors
#
# It produces figures that you can compare against experimental data:
# - Trajectories (GT vs. detections vs. Kalman tracks)
# - IoU over time
# - Effect of latency compensation on position error
# - PID command signals over time
#
# NOTE: To keep the runtime small and reproducible, we use a 2-object scene and
#       lightweight models (no neural net is run here — detections are synthesized
#       by adding noise, misses, and false positives to ground truth).

import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import List, Tuple, Optional
import os
```

```
import random
import pandas as pd

# ---------- Helper structures ----------

@dataclass
class Box:
    # Bounding box in image pixel coordinates: [x1,y1,x2,y2]
    x1: float
    y1: float
    x2: float
    y2: float

    @property
    def cx(self):
        return 0.5 * (self.x1 + self.x2)

    @property
    def cy(self):
        return 0.5 * (self.y1 + self.y2)

    @property
    def w(self):
        return max(self.x2 - self.x1, 1e-6)

    @property
    def h(self):
        return max(self.y2 - self.y1, 1e-6)

def iou(b1: Box, b2: Box) -> float:
    ix1 = max(b1.x1, b2.x1)
```

```
        iy1 = max(b1.y1, b2.y1)
        ix2 = min(b1.x2, b2.x2)
        iy2 = min(b1.y2, b2.y2)
        iw = max(ix2 - ix1, 0.0)
        ih = max(iy2 - iy1, 0.0)
        inter = iw * ih
        area1 = b1.w * b1.h
        area2 = b2.w * b2.h
        union = area1 + area2 - inter
        if union <= 0:
            return 0.0
        return inter / union

    def iou_cost(b_pred: Box, b_det: Box) -> float:
        return 1.0 - iou(b_pred, b_det)

    # ---------- Kalman Filter (constant velocity on cx, cy, w, h) ----------

    class KalmanTrack:
        def __init__(self, init_box: Box, dt: float):
            # State: [cx, cy, vx, vy, w, h]
            self.dt = dt
            self.x = np.array([init_box.cx, init_box.cy, 0.0, 0.0, init_box.w, init_box.h],
dtype=float)

            # Transition matrix F
            self.F = np.eye(6)
            self.F[0,2] = dt
            self.F[1,3] = dt

            # Observation matrix H (measures cx, cy, w, h)
```

```
        self.H = np.zeros((4,6))
        self.H[0,0] = 1.0
        self.H[1,1] = 1.0
        self.H[2,4] = 1.0
        self.H[3,5] = 1.0

        # Process noise covariance Q
        q_pos = 2.0
        q_vel = 10.0
        q_size = 1.0
        self.Q = np.diag([q_pos, q_pos, q_vel, q_vel, q_size, q_size])

        # Measurement noise covariance R
        r_pos = 4.0
        r_size = 3.0
        self.R = np.diag([r_pos, r_pos, r_size, r_size])

        # State covariance P
        self.P = np.diag([10, 10, 10, 10, 10, 10]).astype(float)

        self.age = 0
        self.time_since_update = 0

    def predict(self):
        # x = F x
        self.x = self.F.dot(self.x)
        # P = F P F^T + Q
        self.P = self.F.dot(self.P).dot(self.F.T) + self.Q
        self.age += 1
        self.time_since_update += 1
```

```
    def update(self, z: np.ndarray):
        # z = [cx, cy, w, h]
        y = z - self.H.dot(self.x)  # innovation
        S = self.H.dot(self.P).dot(self.H.T) + self.R
        K = self.P.dot(self.H.T).dot(np.linalg.inv(S))
        self.x = self.x + K.dot(y)
        I = np.eye(6)
        self.P = (I - K.dot(self.H)).dot(self.P)
        self.time_since_update = 0

    def to_box(self) -> Box:
        cx, cy, vx, vy, w, h = self.x
        return Box(cx - 0.5*w, cy - 0.5*h, cx + 0.5*w, cy + 0.5*h)

    def velocity(self) -> Tuple[float, float]:
        return float(self.x[2]), float(self.x[3])

# ---------- Simulation parameters ----------

H, W = 720, 1280        # original frame size
h_in, w_in = 320, 320   # detector input (after resizing)
alpha = H / h_in
beta = W / w_in

FPS = 30.0
dt = 1.0 / FPS
T = 12.0     # seconds
N = int(T / dt)

frame_skip = 3          # emulate running inference every m-th frame
```

```
    tau = 0.5           # detection confidence threshold (not used explicitly in synthetic gen)
    dt_inf = 0.06       # 60 ms inference latency compensation

    rng = np.random.default_rng(42)
    random.seed(42)

    # ---------- Ground Truth trajectories (two moving objects) ----------

    def gt_trajectory(t: float, obj_id: int) -> Box:
        # Object 0: moves left->right with slight vertical oscillation
        if obj_id == 0:
            cx = 150 + 700 * (t / T) + 40*np.sin(0.8*t)
            cy = 300 + 30*np.sin(0.6*t)
            w = 140 + 10*np.sin(0.2*t)
            h = 120 + 5*np.cos(0.3*t)
        else:
            # Object 1: moves right->left and downwards slowly
            cx = 1100 - 600 * (t / T) + 30*np.cos(0.7*t)
            cy = 200 + 220 * (t / T) + 25*np.sin(0.9*t)
            w = 100 + 8*np.sin(0.25*t)
            h = 100 + 8*np.cos(0.35*t)

        return Box(cx - 0.5*w, cy - 0.5*h, cx + 0.5*w, cy + 0.5*h)

    # ---------- Detection generator (noisy, with misses & FPs) ----------

    def noisy_detection_from_gt(gt_box: Box, miss_prob=0.12, pos_noise=8.0, size_noise=6.0) -> Optional[Tuple[Box, float]]:
        # miss
        if rng.random() < miss_prob:
```

```
        return None

    # add noise to corners via cx,cy,w,h
    cx = gt_box.cx + rng.normal(0, pos_noise)
    cy = gt_box.cy + rng.normal(0, pos_noise)
    w = max(10.0, gt_box.w + rng.normal(0, size_noise))
    h = max(10.0, gt_box.h + rng.normal(0, size_noise))

    det = Box(cx - 0.5*w, cy - 0.5*h, cx + 0.5*w, cy + 0.5*h)

    # synthetic score correlated with IoU to GT (clipped to [0,1])
    scr = max(0.0, min(1.0, 0.4 + 0.6*iou(det, gt_box) + rng.normal(0, 0.05)))
    return det, scr

def random_false_positive(fp_prob=0.08) -> Optional[Tuple[Box, float]]:
    if rng.random() < fp_prob:
        cx = rng.uniform(100, W-100)
        cy = rng.uniform(100, H-100)
        w = rng.uniform(40, 140)
        h = rng.uniform(40, 140)
        det = Box(cx - 0.5*w, cy - 0.5*h, cx + 0.5*w, cy + 0.5*h)
        scr = rng.uniform(0.3, 0.7)
        return det, scr
    return None

# ---------- Homography (ground plane -> image plane) ----------

H_hmg = np.array([[ 1.2,   0.05,   50.0],
                  [ 0.02,  1.15,   40.0],
                  [ 0.0008, 0.0006, 1.0]])
```

```
H_inv = np.linalg.inv(H_hmg)

def image_to_ground(u, v):
    p = np.array([u, v, 1.0])
    pg = H_inv.dot(p)
    if abs(pg[2]) < 1e-9:
        return 0.0, 0.0
    X = pg[0]/pg[2]
    Y = pg[1]/pg[2]
    return X, Y

# ---------- PID controller (for pointing to object 0) ----------

class PID:
    def __init__(self, kp, ki, kd, umin=None, umax=None):
        self.kp, self.ki, self.kd = kp, ki, kd
        self.umin, self.umax = umin, umax
        self.I = 0.0
        self.e_prev = 0.0

    def step(self, e, dt):
        self.I += e * dt
        D = (e - self.e_prev)/dt if dt > 0 else 0.0
        u = self.kp*e + self.ki*self.I + self.kd*D
        self.e_prev = e
        if self.umin is not None: u = max(u, self.umin)
        if self.umax is not None: u = min(u, self.umax)
        return u

pid_yaw = PID(2.0, 0.2, 0.35, umin=-2.5, umax=2.5)
pid_pitch = PID(2.2, 0.25, 0.4, umin=-2.0, umax=2.0)
```

```
# ---------- Simulation loop ----------

gt_boxes = []
det_boxes = []        # per-frame list of detections (list of Box)
det_scores = []
trk_boxes = []        # best track for each GT (simple two-track scenario)
trk_boxes_latency_comp = []
ious_to_gt0 = []
ious_to_gt1 = []
lat_err_px = []       # |predicted cx - gt cx| for latency vs no-latency
yaw_cmds = []
pitch_cmds = []
frames_used_for_inference = []

# Initialize two tracks with first available detections at t=0
t0 = 0.0
gt0 = gt_trajectory(t0, 0)
gt1 = gt_trajectory(t0, 1)
# create initial detections (ensure not missed)
det0, _ = noisy_detection_from_gt(gt0, miss_prob=0.0)
det1, _ = noisy_detection_from_gt(gt1, miss_prob=0.0)

kf0 = KalmanTrack(det0, dt)
kf1 = KalmanTrack(det1, dt)

N_int = int(N)  # ensure integer loop bound

for k in range(N_int):
    t = k * dt

    # Ground truth
```

```
gt0 = gt_trajectory(t, 0)
gt1 = gt_trajectory(t, 1)
gt_boxes.append([gt0, gt1])

# Synthesize detections
det_list = []
scores = []

if k % frame_skip == 0:
    # detections for object 0 and 1, possibly missed
    d0 = noisy_detection_from_gt(gt0)
    if d0 is not None and d0[1] >= tau:
        det_list.append(d0[0]); scores.append(d0[1])

    d1 = noisy_detection_from_gt(gt1)
    if d1 is not None and d1[1] >= tau:
        det_list.append(d1[0]); scores.append(d1[1])

    # random false positive
    fp = random_false_positive()
    if fp is not None and fp[1] >= tau:
        det_list.append(fp[0]); scores.append(fp[1])

    frames_used_for_inference.append(True)
else:
    frames_used_for_inference.append(False)

det_boxes.append(det_list)
det_scores.append(scores)

# Predict tracks every frame
```

```
kf0.predict()
kf1.predict()

# Associate when we have detections
if len(det_list) > 0:
    # Compute IoU-costs to both tracks
    pred0 = kf0.to_box()
    pred1 = kf1.to_box()
    costs0 = [iou_cost(pred0, d) for d in det_list]
    costs1 = [iou_cost(pred1, d) for d in det_list]

    # Greedy association: choose best for each track if cost<th
    th_cost = 0.6
    if len(costs0) > 0:
        i0 = int(np.argmin(costs0))
        if costs0[i0] < th_cost:
            d = det_list[i0]
            z = np.array([d.cx, d.cy, d.w, d.h])
            kf0.update(z)
    if len(costs1) > 0:
        i1 = int(np.argmin(costs1))
        # Avoid using the same detection twice if possible
        if len(det_list) > 1 and i1 == int(np.argmin(costs0)):
            # choose alternative if available
            sorted_idx = np.argsort(costs1)
            for idx in sorted_idx:
                if idx != i0:
                    i1 = int(idx)
                    break
        if costs1[i1] < th_cost:
            d = det_list[i1]
```

```
            z = np.array([d.cx, d.cy, d.w, d.h])
            kf1.update(z)

        # Convert tracks to boxes
        trk0 = kf0.to_box()
        trk1 = kf1.to_box()

        # Latency compensation: adjust predicted centers by dt_inf * velocity
        vx0, vy0 = kf0.velocity()
        vx1, vy1 = kf1.velocity()
        trk0_comp = Box(trk0.x1 + dt_inf*vx0, trk0.y1 + dt_inf*vy0,
                        trk0.x2 + dt_inf*vx0, trk0.y2 + dt_inf*vy0)
        trk1_comp = Box(trk1.x1 + dt_inf*vx1, trk1.y1 + dt_inf*vy1,
                        trk1.x2 + dt_inf*vx1, trk1.y2 + dt_inf*vy1)

        trk_boxes.append([trk0, trk1])
        trk_boxes_latency_comp.append([trk0_comp, trk1_comp])

        # IoU to GT (for diagnostics)
        ious_to_gt0.append((iou(trk0, gt0), iou(trk0_comp, gt0)))
        ious_to_gt1.append((iou(trk1, gt1), iou(trk1_comp, gt1)))

        # Latency compensation error (absolute cx error, uncompensated vs compensated) for object 0
        err_uncomp = abs(trk0.cx - gt0.cx)
        err_comp = abs(trk0_comp.cx - gt0.cx)
        lat_err_px.append((err_uncomp, err_comp))

        # PID pointing control to object 0 using compensated track
        # Compute pixel error relative to image center
        u0 = trk0_comp.cx
```

```
    v0 = trk0_comp.cy
    u_center = W/2.0
    v_center = H/2.0
    e_u = (u0 - u_center) / W
    e_v = (v0 - v_center) / H

    yaw = pid_yaw.step(e_u, dt)
    pitch = pid_pitch.step(e_v, dt)
    yaw_cmds.append(yaw)
    pitch_cmds.append(pitch)

# ---------- Metrics and Visualization ----------

time = np.arange(N_int) * dt

# 1) Trajectories: cx, cy over time for GT0 vs det vs track
gt0_cx = np.array([gb[0].cx for gb in gt_boxes])
gt0_cy = np.array([gb[0].cy for gb in gt_boxes])
trk0_cx = np.array([tb[0].cx for tb in trk_boxes])
trk0_cy = np.array([tb[0].cy for tb in trk_boxes])
trk0c_cx = np.array([tb[0].cx for tb in trk_boxes_latency_comp])
trk0c_cy = np.array([tb[0].cy for tb in trk_boxes_latency_comp])

# Condense detections to closest-to-gt per frame for visualization
det0_cx = np.full(N_int, np.nan)
det0_cy = np.full(N_int, np.nan)
for k in range(N_int):
    if len(det_boxes[k]) > 0:
        # pick detection with highest IoU to GT0
        ious = [iou(d, gt_boxes[k][0]) for d in det_boxes[k]]
        idx = int(np.argmax(ious))
```

```
            det0_cx[k] = det_boxes[k][idx].cx
            det0_cy[k] = det_boxes[k][idx].cy

        # 2) IoU time series
        iou0_uncomp = np.array([p[0] for p in ious_to_gt0])
        iou0_comp   = np.array([p[1] for p in ious_to_gt0])

        # 3) Latency compensation absolute error (pixels)
        err_uncomp = np.array([e[0] for e in lat_err_px])
        err_comp   = np.array([e[1] for e in lat_err_px])

        # 4) Commands
        yaw_cmds = np.array(yaw_cmds)
        pitch_cmds = np.array(pitch_cmds)

        # Ensure output directory
        os.makedirs("fpv_sim_outputs", exist_ok=True)

        # ---------- Figure 1: Trajectory (cx over time) ----------
        plt.figure()
        plt.plot(time, gt0_cx, label="GT cx")
        plt.plot(time, det0_cx, label="Det cx (noisy)", linestyle="--")
        plt.plot(time, trk0_cx, label="KF cx (uncomp)", linestyle=":")
        plt.plot(time, trk0c_cx, label="KF cx (latency-comp)", linestyle="-")
        plt.xlabel("Time [s]"); plt.ylabel("Center x [px]"); plt.title("Object 0 trajectory:
cx(t)")
        plt.legend()
        plt.tight_layout()
        plt.savefig("fpv_sim_outputs/fig1_cx.png", dpi=150)

        # ---------- Figure 2: Trajectory (cy over time) ----------
```

```
    plt.figure()
    plt.plot(time, gt0_cy, label="GT cy")
    plt.plot(time, det0_cy, label="Det cy (noisy)", linestyle="--")
    plt.plot(time, trk0_cy, label="KF cy (uncomp)", linestyle=":")
    plt.plot(time, trk0c_cy, label="KF cy (latency-comp)", linestyle="-")
    plt.xlabel("Time [s]"); plt.ylabel("Center y [px]"); plt.title("Object 0 trajectory: cy(t)")
    plt.legend()
    plt.tight_layout()
    plt.savefig("fpv_sim_outputs/fig2_cy.png", dpi=150)

    # ---------- Figure 3: IoU to GT (uncomp vs compens) ----------
    plt.figure()
    plt.plot(time, iou0_uncomp, label="IoU (KF vs GT)")
    plt.plot(time, iou0_comp, label="IoU (KF latency-comp vs GT)", linestyle="--")
    plt.xlabel("Time [s]"); plt.ylabel("IoU [-]"); plt.title("Effect of latency compensation on IoU")
    plt.legend()
    plt.tight_layout()
    plt.savefig("fpv_sim_outputs/fig3_iou.png", dpi=150)

    # ---------- Figure 4: Absolute cx error (pixels) ----------
    plt.figure()
    plt.plot(time, err_uncomp, label="|cx_err| uncomp")
    plt.plot(time, err_comp, label="|cx_err| latency-comp", linestyle="--")
    plt.xlabel("Time [s]"); plt.ylabel("|cx error| [px]"); plt.title("Latency compensation error comparison")
    plt.legend()
    plt.tight_layout()
    plt.savefig("fpv_sim_outputs/fig4_lat_err.png", dpi=150)
```

```
# ---------- Figure 5: PID commands (yaw & pitch) ----------
plt.figure()
plt.plot(time, yaw_cmds, label="yaw cmd")
plt.plot(time, pitch_cmds, label="pitch cmd", linestyle="--")
plt.xlabel("Time [s]"); plt.ylabel("Command [arb]"); plt.title("PID pointing commands over time")
plt.legend()
plt.tight_layout()
plt.savefig("fpv_sim_outputs/fig5_pid_cmds.png", dpi=150)

# ---------- Figure 6: Ground-plane projection of tracked object (sample points) ----------
# Take a subset of frames to visualize image->ground mapping for the compensated track
idxs = np.linspace(0, N_int-1, 40, dtype=int)
u = trk0c_cx[idxs]
v = trk0c_cy[idxs]
XY = np.array([image_to_ground(ui, vi) for ui,vi in zip(u,v)])
plt.figure()
plt.plot(XY[:,0], XY[:,1], marker="o", linestyle="-", label="Tracked path on ground plane (proj)")
plt.xlabel("X_ground [arb]"); plt.ylabel("Y_ground [arb]"); plt.title("Homography-based ground-plane path (Object 0)")
plt.legend()
plt.tight_layout()
plt.savefig("fpv_sim_outputs/fig6_ground.png", dpi=150)

# Optional: show plots when running interactively
# plt.show()
```

```
    # ---------- Save key metrics to CSV for later comparison with experiments -----
-----
    df = pd.DataFrame({
        "time_s": time,
        "gt0_cx": gt0_cx, "gt0_cy": gt0_cy,
        "det0_cx": det0_cx, "det0_cy": det0_cy,
        "kf0_cx": trk0_cx, "kf0_cy": trk0_cy,
        "kf0c_cx": trk0c_cx, "kf0c_cy": trk0c_cy,
        "iou0_uncomp": iou0_uncomp, "iou0_comp": iou0_comp,
        "abs_err_cx_uncomp_px": err_uncomp, "abs_err_cx_comp_px": err_comp,
        "yaw_cmd": yaw_cmds, "pitch_cmd": pitch_cmds
    })
    df.to_csv("fpv_sim_outputs/sim_metrics.csv", index=False)

    print("Saved figures and CSV to ./fpv_sim_outputs")
```

**ДОДАТОК Б**

Апробація результатів кваліфікаційної роботи

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

# ЗБІРНИК

**студентських наукових статей**

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

**ADED-2025**

(Випуск 2)

[електронне видання]

KITAP

http://nure.ua/department/kafedra-komp-yuterno-integrovanih-tehnologiy-avtomatizatsiyi-ta-mehatroniki-kitam

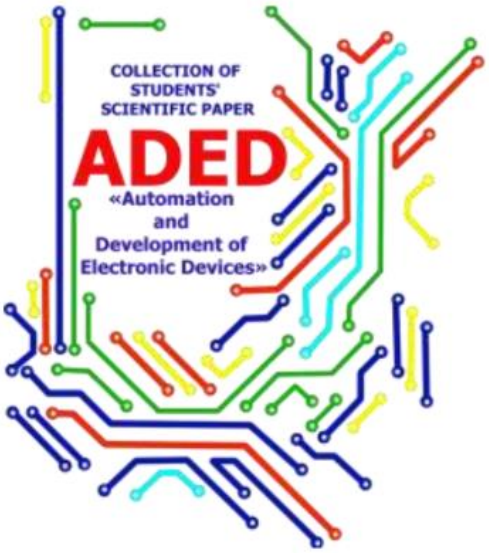
iTED

http://itez.zntu.edu.ua/

http://kafea.kdu.edu.ua

Харків 2025

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки
кафедра комп'ютерно-інтегрованих технологій, автоматизації та робототехніки
(КІТАР)




# ЗБІРНИК

**студентських наукових статей**

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

**ADED-2025**

(Випуск 2)

[електронне видання]

Харків 2025

Автоматизація та Приладобудування («Automation and Development of Electronic Devices» ADED-2025) [Електронний ресурс] : збірник студентських наукових статей / Харківський національний університет радіоелектроніки ; [редкол.: І.Ш. Невлюдов та ін.]. – Харків : ХНУРЕ, 2025. – Вип. 2. – 207с.

Collection of Students' Scientific Paper «Automation and Development Of Electronic Devices» ADED-2025 Part 2 (Key infrastructure 2025) - Kharkiv/ The Editorial.: Nevlyudov I.Sh. (head), that all. Kharkiv: Kind of Kharkiv National Universitety of Radio Elektronics [electronic edition], 2025. – 207p with.

Рекомендовано рішенням
Науково-технічної ради
Харківського національного
університету радіоелектроніки
протокол №6 від 29.11.2018

Рекомендовано рішенням Вченої ради
факультету Автоматики і комп'ютеризованих технологій
Харківського національного
університету радіоелектроніки
протокол № 4 від 21 листопада 2025

Збірник містить наукові статті здобувачів першого (бакалаврського), другого (магістерського) рівнів вищої освіти кафедри комп'ютерно-інтегрованих технологій, автоматизації та робототехніки (КІТАР) Харківського національного університету радіоелектроніки, кафедри Інформаційних технологій електронних засобів (ІТЕД) Запорізького національного технічного університету та кафедри Електронних апаратів (ЕА) Кременчуцького національного університету ім. М. Остроградського які навчаються за спеціальностями: 151 Автоматизація та комп'ютерно-інтегровані технології, 174 Автоматизація, комп'ютерно-інтегровані технології та робототехніка; 172 Телекомунікації та радіотехніка, 171 Електроніка та 163 Біомедична інженерія. Статті надані в авторській редакції.

УДК 681.5

# АНАЛІЗ СИСТЕМ КЕРУВАННЯ FPV ДРОНІВ З ВИКОРИСТАННЯМ НЕЙРОННИХ МЕРЕЖ

**Пара І.І.**
Харківський національний університет радіоелектроніки
Україна, 61166, Харків, пр. Науки, 14
E-mail: illia.para@nure.ua

**Анотація:** У роботі розглянуто підходи до створення системи ідентифікації та розпізнавання об'фєктів для FPV (First Person View) дронів. Проаналізовано методи керування БПЛА, а також основні інструменти комп'ютерного зору, такі як згорткові нейронні мережі (CNN), алгоритми трекінгу об'єктів в реальному часі. Реалізація розробленої системи наддасть змогу ідентифікувати об’єкти в реальному часі.

**Ключові слова:** комп’ютерний зір, розпізнавання об’єктів, керування БПЛА, згорткові нейронні мережі (CNN), обробка зображень.

# OBJECT IDENTIFICATION AND RECOGNITION SYSTEM FOR THE COMPUTER VISION OF FPV DRONES

**Para I.I.**
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, pr. Sciences, 14
E-mail: illia.para@nure.ua

**Abstract:** This paper examines approaches to developing an object identification and recognition system for FPV (First Person View) drones. UAV control methods and key computer vision tools, such as convolutional neural networks (CNN) and real-time object tracking algorithms, are analyzed. The implementation of the proposed system enables real-time object identification.

**Keywords:** computer vision, object recognition, UAV control, convolutional neural networks (CNN), image processing.

FPV дрони стають усе популярнішими у різних сферах діяльності, від розваг до професійних завдань. Однією з найважливіших функцій FPV дронів є здатність до автономної роботи, що залежить від ефективності системи комп'ютерного зору.

Метою цієї роботи є розроблення системи, яка забезпечує точну ідентифікацію та розпізнавання об'фєктів в реальному часі, враховуючи обмеження обчислювальних ресурсів FPV дронів.

Комп’ютерний зір (Computer Vision) – одна з областей штучного інтелекту, яка дає змогу комп’ютерам і системам отримувати корисну інформацію з цифрових зображень, відео, візуальних даних та виконувати дії або давати рекомендації на основі отриманої інформації. Комп’ютерний зір дозволяє машинам бачити, спостерігати й розуміти. Комплексні рішення на базі Computer Vision вже активно використовуються у рІзних сферах бізнесу та виробництва [2].

Система комп'ютерного зору має багато складових:

Ідентифікація об'єктів. Система аналізує отриману візуальну інформацію та ідентифікує окремий об'єкт на фото або відео, наприклад, система зможе знайти певну ручку серед інших або відрізнити одну кішку від іншої [3].

Класифікація об'єктів. Система аналізує отриману візуальну інформацію та призначає розпізнаним об'єктам певну категорію/класу на основі їх характеристик.

Стеження за об’єктом. Система обробляє отримане відео, знаходить об’єкт за заданими критеріями пошуку після чого відстежує його переміщення.

Також сюди можна віднести таке комплексне поняття, як розпізнавання об’єктів. Розпізнавання об’єктів- це техніка комп’ютерного зору, яка використовується для ідентифікації, визначення місцезнаходження та класифікації об’єктів на цифрових зображеннях або реальних сценаріях.

МЕТОДИ КЕРУВАННЯ БПЛА. Методи управління БПЛА включають різні техніки та алгоритми. Існує декілька режимів керування дронами: ручний (пілотажний), напівавтоматичний (навігаційний) та автоматичний (диспечерський) [4].

Пілотажний метод керування.


Рисунок 2.1 – Структура пілотажного методу керування БПЛА

У цьому випадку керування БПЛА виконується безпосередньо за допомогою пункту керування, з якого передаються кути розвороту, параметри роботи силових машин, тощо.

Навігаційний метод керування.

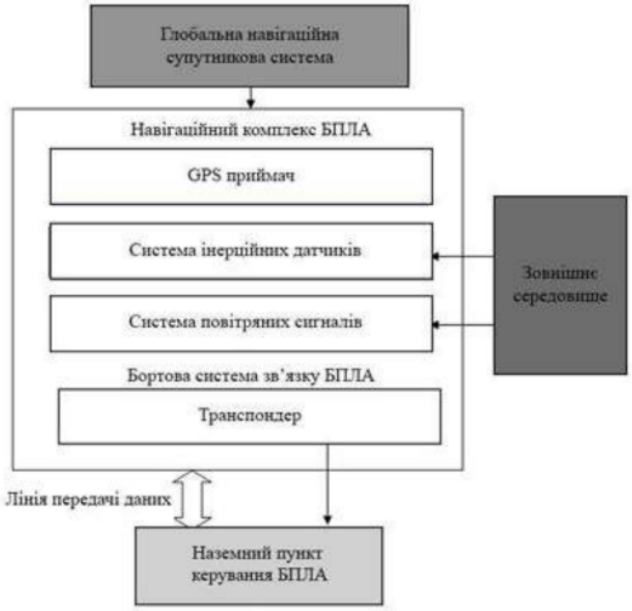



Рисунок 2.2 – Структура навігаційного методу керування БПЛА

Керування БПЛА здійснюється не передачею йому команд для виконання маневрів, а шляхом завдання точок маршруту щодо земної поверхні.

Диспечерський метод керування.


Рисунок 2.3 – Структура диспечерського методу керування БПЛА

Для його використання має бути створена внутрішня система управління функціонуванням БПЛА. Вона покликана реалізувати алгоритми функціонування алгоритмів внутрішніх систем.

ВИДИ КАМЕР НА БОРТУ БПЛА ДЛЯ ОТРИМАННЯ ВІДЕО/ФОТО. Камера в квадрокоптері є одним з ключових компонентів. У її функції входить не тільки фото- та відеозйомка, а й маневреність агрегату в польоті. Існує кілька різновидів камер для дронів.

FPV камера - це пристрій, що встановлюється на дрон і забезпечує передачу зображення в режимі реального часу з точки зору дрона. Завдяки FPV камері оператор може керувати дроном, орієнтуючись на зображення, передане на екран, окуляри чи шолом.

RGB камера - це стандартний тип цифрової камери, яка захоплює зображення в трьох основних кольорах: червоному (Red), зеленому (Green) та синьому (Blue). Ці три канали використовуються для формування повнокольорового зображення, яке відповідає сприйняттю людського ока.

Теплова камера - це пристрої, які дозволяють виявляти та візуалізувати теплове випромінювання об’єктів. Вони широко застосовуються в різних галузях для діагностики, контролю, безпеки та аналізу.

Екшн камера - це компактна, легка, і надзвичайно міцна відеокамера, призначена для зйомки в екстремальних умовах. Завдяки своїм характеристикам, вона широко використовується в незвичайних умовах, які потребують мобільності та високої якості відео

CNN. Конволюційні нейронні мережі (Convolutional Neural Networks, CNN) — це тип глибоких нейронних мереж, які надзвичайно ефективні для задач комп'ютерного зору, таких як класифікація зображень, розпізнавання об'єктів, сегментація та багато інших. CNN використовують конволюційні операції, які дозволяють виявляти просторові та ієрархічні залежності в зображеннях.

АРХІТЕРКТУРА CNN. Існує декілька архітектурних рішень:

1. LeNet-5. Перша CNN, яка показала ефективність на задачах розпізнавання рукописних цифр.

2. AlexNet. Згорткова нейронна мережа, що застосовується для розпізнавання зображень. Виграла ImageNet Challenge у 2012, стала проривом у комп’ютерному зорі.

3. VGGNet. Глибока мережа з простими архітектурними блоками.

4. ResNet. Представила механізм резидуальних зв’язків, який дозволяє тренувати дуже глибокі мережі.

5. Inception (GoogLeNet). Ефективна структура з використанням паралельних фільтрів різних розмірів.

6. EfficientNet. Оптимізовані моделі для балансу між точністю і швидкістю.

Конволюційні нейронні мережі (CNN) мають значні переваги, які зробили їх ключовим інструментом для вирішення задач комп'ютерного зору. До переваг можна віднести: автоматичне виділення ознак, ієрархічність аналізу даних, зменшення кількості параметрів, стійкість до змін у даних, стабільність до "шуму", масштабованість, використання просторової інформації та гнучкість у виборі архітектури.

ВИСНОВКИ. У цій роботі розглянуто актуальність використання систем комп'ютерного зору на основі конволюційних нейронних мереж (CNN) для автономної роботи FPV дронів. Проаналізовано методи ідентифікації та розпізнавання об'єктів, алгоритми трекінгу в реальному часі, а також ефективність різних архітектур CNN. Результати дослідження підтверджують, що такі системи є ключовим елементом для забезпечення точності ідентифікації об'єктів та стабільної роботи дронів у реальних умовах. Реалізація запропонованих рішень дозволить підвищити точність і швидкість розпізнавання об’єктів та забезпечити автономність роботи FPV дронів завдяки адаптації до обмежених обчислювальних ресурсів. Подальші дослідження можуть бути зосереджені на оптимізації алгоритмів трекінгу об’єктів, інтеграції інших технологій комп’ютерного зору та підвищенні енергоефективності системи.

ЛІТЕРАТУРА

1. Nevliudov, I., Yevsieiev, V., Maksymova, S., Gopejenko, V., & Kosenko, V. (2025). Development of mathematical support for adaptive control for the intelligent gripper of the collaborative robot manipulator. Advanced Information Systems, 9(3), 57-65.

2. Невлюдов, І., Євсєєв, В., Максимова, С., & Артюх, Р. (2025). Математична модель адаптивного ієрархічного високорівневого керування триланкового колаборативного робота-маніпулятора. Сучасний стан наукових досліджень та технологій в промисловості, (2 (32)), 58-68.

3. Moisieiev M. Research on Methods for Controlling a Group of Mobile Robots Under Uncertainty / M. Moisieiev, V. Yevsieiv // Manufacturing & Mechatronic Systems 2025 : Thesises of Reports of IX-st International Conference, October 25-26, 2025. - Kharkiv, 2025. - P. 26-29.

4. Yevsieiev V. Comparative Analysis of Neural Network Architectures for Intelligent Microclimate Control in Production / V. Yevsieiev, I. Holod // Manufacturing & Mechatronic Systems 2025 : Thesises of Reports of IX-st International Conference, October 25-26, 2025. - Kharkiv, 2025. - P. 15-17.

5. Yevsieiev V. Implementation of STEM education in distance learning conditions during martial law in Ukraine: challenges, tools and prospects for training future engineers / V. Yevsieiev, S. Starikova // Theoretical and Applied Aspects of Device Development on Microcontrollers and FPGA (MC&FPGA-2025) : VII International Scientific and Practical Conference, June 27-28, 2025. – Kharkiv : NURE. – P. 21-25.

***Науковий керівник**: Євсєєв Владислав В’ячеславович, професор кафедри КІТАР Харківського національного університету радіоелектроніки*

**ДОДАТОК В**

Демонстраційний матеріал

| Формат | Позиція | ПОЗНАЧЕННЯ | НАЙМЕНУВАННЯ | Кількість | Примітки |
|---|---|---|---|---|---|
| | | | Текстові документи | | |
| | | | | | |
| А4 | 1 | ГЮИК. 442351.022 ВД | Пояснювальна | 90 с. | |
| | | | записка | | |
| | | | | | |
| | | | Інші документи | | |
| | | | | | |
| | 2 | | Додаток А | 17 с. | |
| | | | Код численного | | |
| | | | моделювання | | |
| | | | відстеження об’єктів | | |
| | | | | | |
| | 3 | | Додаток Б | 9 с. | |
| | | | Апробація | | |
| | | | результатів | | |
| | | | кваліфікаційної | | |
| | | | роботи | | |
| | | | | | |
| | 4 | | Додаток В | 14 с. | |
| | | | Демонстраційний | | |
| | | | матеріал | | |

| Змін | Арк. | № документу | Підпис | Дата | ГЮИК. 442351.022 ВД | | |
|---|---|---|---|---|---|---|---|
| Розробив | | Пара І.І. | | | Розроблення програмного модуля розпізнавання об’єктів для системи комп’ютерного зору FPV дрону | Літера | Аркуш | Аркуш |
| Перевірив | | Євсєєв В.В. | | | | Н | 1 | 1 |
| | | | | | | Кафедра КІТАР ХНУРЕ | | |
| Норм. контроль | | Демська Н.П. | | | | | | |
| Затвердив | | Невлюдов І.Ш. | | | | | | |