ДОДАТОК А

Текст програми

ГЮИК. 502840.006–01 12 01

ЗАТВЕРДЖЕНО
ГЮИК. 502840.006–01 12 01-ЛЗ

Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва

Текст програми

ГЮИК. 502840.006–01 12 01

АРКУШІВ 11

2023р.

Міністерство освіти і науки України

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»

Керівник кваліфікаційної роботи

проф.Цимбал О. М.

Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва

Текст програми

ЛИСТ ЗАТВЕРДЖЕННЯ

ГЮИК. 502840.006–01 12 01-ЛЗ

РОЗРОБИВ:

ст. гр. КТРСм-22-1

Мілько Д.В.

2023р.

viev_system.py

```
# Import necessary libraries
import os
import time
import cv2
import threading
import queue
from bin.image_processing.image_drawing import DrawFunctions
from bin.image_processing.tracker_functions import tracking_process, distance_calculator
from bin.image_processing.image_functions import zoom
from bin.config_parser import parse_config
#from bin.communications.nrf24_communication import main_comm_loop
from bin.flightcontroller_utils.SITL_drone_control import drone_auto_control_thread
from bin.image_processing.neural_detection import detect_objects, load_model, load_labels, preprocess_image

# Parse configuration
parse_config()

# Initialize variables
object_lost = False
video_file = ".\\test.mp4"
cap = cv2.VideoCapture(video_file)
object_image_size = 40
cursor_x, cursor_y = -1, -1
tracking = False
bbox = None
ml_detection = False
start_flight = False
stop_flight = False
zoom_factor = 1.0
old_results = {}
real_object_width = 6
fps_limiter = 15
tracking_ok = False

# Initialize queues
img_buffer = queue.Queue()
ml_results = queue.Queue()
drone_control_queue = queue.Queue()
return_drone_info = queue.Queue()
input_comm_queue = queue.Queue()
output_comm_queue = queue.Queue()

# Load labels and model for neural detection
labels = load_labels()
model_path = 'Tensorflow/workspace/models/my_ssd_mobnet/export/saved_model/'
model = load_model(model_path)


# Function for ML detection process
def ml_detection_process():
    while True:
        time.sleep(0.025)
        if img_buffer.empty():
            continue
        else:
```

```
        ml_frame, screen_size = img_buffer.get()
        img = preprocess_image(ml_frame)
        res = detect_objects(model, img, 0.6)
        results_dict = {}

        for idx, result in enumerate(res):
            ymin, xmin, ymax, xmax = result['bounding_box']
            xmin = int(max(1, xmin * screen_size[1]))
            xmax = int(min(screen_size[1], xmax * screen_size[1]))
            ymin = int(max(1, ymin * screen_size[0]))
            ymax = int(min(screen_size[0], ymax * screen_size[0]))

            label = labels[result['class_id']]
            key = f'{label}_{idx}'
            results = [ymin, xmin, ymax, xmax]
            results_dict[key] = results
        ml_results.put(results_dict)


# Function for drone control thread
def drone_thread():
    drone_auto_control_thread(drone_control_queue, return_drone_info)


# Function for communication thread
# def communication_thread():
#     main_comm_loop(input_comm_queue, output_comm_queue)


# Mouse callback function
def mouse_callback(event, x, y, flags, param):
    global cursor_x, cursor_y, tracking, bbox
    cv2.rectangle(frame, (
    x - int(object_image_size / 2), y - int(object_image_size / 2), object_image_size, object_image_size),
                  (0, 255, 0), 2)
    if event == cv2.EVENT_LBUTTONDOWN:
        cursor_x, cursor_y = x, y
        bbox = (x - int(object_image_size / 2), y - int(object_image_size / 2), object_image_size, object_image_size)
        tracking = True
        print("Start Tracking")


# Create a window for object tracking and set mouse callback
cv2.namedWindow('Object Tracking')
cv2.setMouseCallback('Object Tracking', mouse_callback)

# Start threads for ML detection, drone control, and communication
t_ml_detection = threading.Thread(target=ml_detection_process)
t_drone_control = threading.Thread(target=drone_thread)
#t_communications = threading.Thread(target=communication_thread())
t_ml_detection.start()
t_drone_control.start()
# t_communications.start()

# Main loop
while True:
    timer = cv2.getTickCount()
    ret, frame = cap.read()
    screen_size = frame.shape
    original_frame = frame.copy()

    if not ret:
```

```
    break

# Process tracking and key events
frame, bbox1, tracking, object_lost, tracking_ok, target_center = tracking_process(frame, tracking, tracking_ok,
                                                                    bbox, object_lost)

if os.getenv("NRF_ON"):
    key_quit = cv2.waitKey(fps_limiter)
    input_comm_queue.put("Debug_Info")
    key = output_comm_queue.get()
else:
    key = cv2.waitKey(fps_limiter)
    key_quit = cv2.waitKey(fps_limiter)

zoom_point = target_center

# Handle key events
if key == ord('w'):
    zoom_factor = zoom_factor + 0.1 if zoom_factor < 5.0 else zoom_factor
elif key == ord('s'):
    zoom_factor = zoom_factor - 0.1 if zoom_factor > 1.0 else zoom_factor
elif key == ord('q'):
    object_image_size = object_image_size + 10 if object_image_size < 100 else object_image_size
elif key == ord('a'):
    object_image_size = object_image_size - 10 if object_image_size > 10 else object_image_size
elif key == ord('y'):
    ml_detection = True
elif key == ord('c'):
    tracking = False
    tracking_ok = False
elif key == ord('t'):
    ml_detection = False
elif key == ord('g'):
    start_flight = True
    stop_flight = False
elif key == ord('h'):
    stop_flight = True

# Process ML results and draw on frame
if ml_results.empty():
    img_buffer.put([original_frame, screen_size])

if ml_detection:
    if not ml_results.empty():
        results_dict = ml_results.get()
        old_results = results_dict.copy()
    for key, results in old_results.items():
        ymin, xmin, ymax, xmax = results
        cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (0, 255, 0), 3)
        cv2.putText(frame, key, (xmin, min(ymax, screen_size[0] - 20)),
                    cv2.FONT_HERSHEY_COMPLEX, 0.5, (255, 255, 255), 2, cv2.LINE_AA)
    old_results = old_results.copy()

# Draw additional elements on the frame
frame = DrawFunctions(frame).draw_aim()
frame = zoom(frame, zoom_factor, zoom_point)
frame, target_distance = distance_calculator(frame, bbox1, real_object_width)

# Send control commands to the drone
if start_flight and tracking_ok:
    if not return_drone_info.empty():
        stop_flight = return_drone_info.get()
```

```
        else:
            drone_control_queue.put([target_center, screen_size, start_flight, stop_flight, target_distance])
    frame = DrawFunctions(frame).draw_control_keys()
    frame = DrawFunctions(frame).debug_info(cursor_x, cursor_y)

    # Display the frame
    cv2.imshow('Object Tracking', frame)

    if key_quit == ord(']'):
        break

# Release video capture and close windows
cap.release()
                                                    cv2.destroyAllWindows()
```

constants.py

```
class CameraConstants:
    def __init__(self, camera_choice='MAIN_CAMERA'):
        self.camera_choice = camera_choice

    @property
    def focal_length(self):
        if self.camera_choice == 'OV2640':
            return 0.00215
        elif self.camera_choice == 'OV2710':
            return 0.00369
        elif self.camera_choice == 'ISOCELL_GW3':
            return 0.025

    @property
    def matrix_y(self):
        if self.camera_choice == 'OV2640':
            return 3.59
        elif self.camera_choice == 'OV2710':
            return 5.85
        elif self.camera_choice == 'ISOCELL_GW3':
            return 6.5

    @property
    def matrix_x(self):
        if self.camera_choice == 'OV2640':
            return 2.68
        elif self.camera_choice == 'OV2710':
            return 3.27
        elif self.camera_choice == 'ISOCELL_GW3':
            return 4.86

    @property
    def image_size(self):
        if self.camera_choice == 'OV2640':
            return [800, 600]
        elif self.camera_choice == 'OV2710':
            return [1920, 1080]
        elif self.camera_choice == 'ISOCELL_GW3':
            return [1920, 1080]

    @property
    def fov(self):
        if self.camera_choice == 'OV2640':
            return 66
```

```
        elif self.camera_choice == 'OV2710':
            return 120
        elif self.camera_choice == 'ISOCELL_GW3':
            return 100
```

config.ini

```
[CAMERA]
#MAIN_CAMERA=OV2640
#MAIN_CAMERA=OV2710
MAIN_CAMERA=ISOCELL_GW3

[NRF_COMM]
NRF_ON = False
CE_PIN = 25
CSN_PIN = 8
NRF_CHANNEL = 0
PIPE_RX = 0xF0F0F0F0D2
PIPE_TX = 0xF0F0F0F0E1
PAYLOAD_SIZE = 32

[DRONE_CONTROL]
DRONE_IP= udp:127.0.0.1:14550
```

SITL_drone_control.py

```
import os
import queue
import time
from geopy.distance import distance, geodesic
from dronekit import connect, VehicleMode, LocationGlobalRelative
from bin.constants import CameraConstants

vehicle = connect('udp:127.0.0.1:14550', wait_ready=True)


def get_distance_metres(location1, location2):
    location1_coords = (location1.lat, location1.lon)
    location2_coords = (location2.lat, location2.lon)
    dist_meters = geodesic(location1_coords, location2_coords).meters
    return dist_meters


def arm_and_takeoff(aTargetAltitude):
    print("Basic pre-arm checks")
    while not vehicle.is_armable:
        time.sleep(1)

    print("Arming motors")
    vehicle.mode = VehicleMode("GUIDED")
    vehicle.armed = True

    while not vehicle.armed:
        time.sleep(1)

    print("Taking off!")
```

```
    vehicle.simple_takeoff(aTargetAltitude)

    while True:
        print("Altitude: ", vehicle.location.global_relative_frame.alt)
        if vehicle.location.global_relative_frame.alt >= aTargetAltitude * 0.95:
            print("Reached target altitude")
            break
        time.sleep(1)


def fly_to_target(location: LocationGlobalRelative, stop_flight):
    print("Flying forward...")
    vehicle.simple_goto(location)
    if stop_flight:
        position_for_now = vehicle.location.global_frame
        vehicle.simple_goto(position_for_now)


def calculate_target_gps(current_location, azimuth, target_distance):
    target_coords = distance(meters=target_distance).destination((current_location.lat, current_location.lon), azimuth)
    return LocationGlobalRelative(target_coords.latitude, target_coords.longitude, 1)


def tank_attack(target_image_point, image_width, distance):
    azimuth_drone_from_north = vehicle.heading
    camera_fov = 66.0
    target_x_on_image = target_image_point[0]
    target_y_on_image = target_image_point[1]
    azimuth_to_target_from_north = azimuth_drone_from_north + (target_x_on_image - image_width / 2) * (
            camera_fov / image_width)
    target_location = calculate_target_gps(vehicle.location.global_frame, azimuth_to_target_from_north, distance)
    fly_to_target(target_location, False)


#arm_and_takeoff(15)

#tank_attack([int(1920/2), 100], 1920, 100)


def drone_auto_control_thread(drone_control_queue: queue.Queue, return_drone_info: queue.Queue):
    arm_and_takeoff(15)
    while True:
        if not drone_control_queue.empty():
            target_image_point, screen_size, start_flight, stop_flight, target_distance = drone_control_queue.get()
            print("work")
            if start_flight:
                print("ok")
                screen_center_x = screen_size[1]/2
                now_drone_position = vehicle.location.global_frame
                azimuth_drone_from_north = vehicle.heading
                camera_fov = CameraConstants(os.getenv("MAIN_CAMERA")).fov
                target_x_on_image = target_image_point[0]
                angle_to_target = (camera_fov / screen_size[1]) * (target_x_on_image - screen_center_x)
                processed_angle = angle_to_target
                print(processed_angle)
                print(azimuth_drone_from_north)

                azimuth_to_target_from_north = (azimuth_drone_from_north + processed_angle)
                azimuth_to_target_from_north %= 360
                print(azimuth_to_target_from_north)
                target_location = calculate_target_gps(vehicle.location.global_frame, azimuth_to_target_from_north,
                                                       target_distance)
```

```
            fly_to_target(target_location, stop_flight)
            distance_from_gps = get_distance_metres(now_drone_position, target_location)
            print("Distance to target: ", distance_from_gps)
            time.sleep(0.025)
            if stop_flight:
                start_flight = False
                return_drone_info.put(start_flight)
        else:
            time.sleep(0.025)
    else:
        time.sleep(0.025)
        continue
```

image_drawing.py

```
import cv2


class Colors:
    red = (255, 0, 0)
    green = (0, 255, 0)
    blue = (0, 0, 255)
    black = (0, 0, 0)
    white = (255, 255, 255)
    yellow = (255, 255, 0)


class DrawFunctions:

    def __init__(self, frame):
        self.frame = frame

    def draw_aim(self):
        height, width, _ = self.frame.shape
        center_x = width // 2
        center_y = height // 2
        rect_size = 20
        rect_thickness = 2
        top_left = (center_x - rect_size, center_y - rect_size)
        bottom_right = (center_x + rect_size, center_y + rect_size)
        cv2.rectangle(self.frame, top_left, bottom_right, (0, 255, 0), rect_thickness)
        cv2.line(self.frame, (center_x, center_y - rect_size - 10), (center_x, center_y - rect_size - 100), Colors.green,
                 rect_thickness)
        cv2.line(self.frame, (center_x, center_y + rect_size + 10), (center_x, center_y + rect_size + 100), Colors.green,
                 rect_thickness)
        cv2.line(self.frame, (center_x - rect_size - 10, center_y), (center_x - rect_size - 100, center_y), Colors.green,
                 rect_thickness)
        cv2.line(self.frame, (center_x + rect_size + 10, center_y), (center_x + rect_size + 100, center_y), Colors.green,
                 rect_thickness)
        cv2.rectangle(self.frame, top_left, bottom_right, (0, 255, 0), rect_thickness)

        return self.frame

    def debug_info(self, cursor_x, cursor_y):
        cv2.putText(self.frame, f"Cursor: x={cursor_x}, y={cursor_y}", (50, 50), cv2.FONT_HERSHEY_DUPLEX, 0.75, (0, 0,
255),
                    2)
```

```
        # cv2.putText(frame, f"KCF TRACKER", (50, 250), cv2.FONT_HERSHEY_COMPLEX, 0.75, (0, 255, 255), 2)
        # cv2.putText(frame, f"CSRT TRACKER", (50, 250), cv2.FONT_HERSHEY_COMPLEX, 0.75, (0, 255, 255), 2)
        cv2.putText(self.frame, f"MIL TRACKER", (50, 250), cv2.FONT_HERSHEY_DUPLEX, 0.75, (0, 255, 255), 2)
        return self.frame

    def draw_control_keys(self):
        height, width, _ = self.frame.shape
        text = "Manual control keys:"
        x = 10
        y = height - 70
        cv2.putText(self.frame, text, (x, y), cv2.FONT_HERSHEY_DUPLEX, 0.75, (0, 255, 0), 2)
        text = "Turn ON/OFF ML: Y/T|Zoom IN/OUT: W/S|TrackerBox IN/OUT: Q/A|Stop Tracking: C|Start/Stop Flight:G/H"
        x = 10
        y = height - 50
        cv2.putText(self.frame, text, (x, y), cv2.FONT_HERSHEY_DUPLEX, 0.5, (0, 255, 0), 1)
        return self.frame
```

neural_detection.py

```
import re
import cv2
import tensorflow as tf
import numpy as np



def load_labels(path='labels.txt'):
    with open(path, 'r', encoding='utf-8') as f:
        lines = f.readlines()
        labels = {}
        for row_number, content in enumerate(lines):
            pair = re.split(r'[:\s]+', content.strip(), maxsplit=1)
            if len(pair) == 2 and pair[0].strip().isdigit():
                labels[int(pair[0])] = pair[1].strip()
            else:
                labels[row_number] = pair[0].strip()
    return labels


def load_model(model_path):
    model = tf.saved_model.load(model_path)
    return model


def preprocess_image(image):
    """Preprocess the input image."""
    image = cv2.resize(cv2.cvtColor(image, cv2.COLOR_BGR2RGB), (320, 320))
    image = (image * 255).astype(np.uint8)
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    return image


def detect_objects(model, image, threshold):
    input_tensor_info = model.signatures["serving_default"].inputs
    img_uint8 = (image * 255).astype(np.uint8)
    output_dict = model.signatures["serving_default"](tf.constant(img_uint8))
```

```
    boxes = output_dict['detection_boxes'].numpy().tolist()[0]
    classes = output_dict['detection_classes'].numpy().astype(int).tolist()[0]
    scores = output_dict['detection_scores'].numpy().tolist()[0]
    count = int(output_dict['num_detections'].numpy())

    results = []
    for i in range(count):
        if scores[i] >= threshold:
            result = {
                'bounding_box': boxes[i],
                'class_id': classes[i],
                'score': scores[i]
            }
            results.append(result)
    return results
```

image_functions.py

```
import cv2


def zoom(image, factor, zoom_center):
    if factor != 1:
        height, width, _ = image.shape

        new_width = int(width / factor)
        new_height = int(height / factor)

        roi_x = max(0, int(zoom_center[0] - new_width / 2))
        roi_y = max(0, int(zoom_center[1] - new_height / 2))
        roi_width = min(new_width, width - roi_x)
        roi_height = min(new_height, height - roi_y)

        zoomed_roi = cv2.resize(image[roi_y:roi_y + roi_height, roi_x:roi_x + roi_width], (width, height))
        return zoomed_roi

    return image
```

config_parser.py

```
            import configparser
            import os


            def parse_config(file_path='.\\config.ini'):
                config = configparser.ConfigParser()
                config.read(file_path)

                os.environ.update({
                    'MAIN_CAMERA': config.get('CAMERA', 'MAIN_CAMERA'),
                    'CE_PIN': config.get('NRF_COMM', 'CE_PIN'),
                    'CSN_PIN': config.get('NRF_COMM', 'CSN_PIN'),
                    'NRF_CHANNEL': config.get('NRF_COMM', 'NRF_CHANNEL'),
                    'PIPE_RX': config.get('NRF_COMM', 'PIPE_RX'),
                    'PIPE_TX': config.get('NRF_COMM', 'PIPE_TX'),
                    'PAYLOAD_SIZE': config.get('NRF_COMM', 'PAYLOAD_SIZE'),
```

```
        'DRONE_IP': config.get('DRONE_CONTROL', 'DRONE_IP')
    })
```

Tracker_functions.py

```
import datetime
import os
import cv2
from bin.constants import CameraConstants


class Trackers:
    tracker_1 = cv2.TrackerCSRT.create()
    tracker_2 = cv2.TrackerKCF.create()
    tracker_3 = cv2.TrackerMIL.create()


def draw_lines_to_edges(frame, bbox):
    x, y, w, h = [int(e) for e in bbox]
    center = (x + w // 2, y + h // 2)

    height, width, _ = frame.shape

    cv2.line(frame, center, (0, center[1]), (0, 255, 0), 2)
    cv2.line(frame, center, (width, center[1]), (0, 255, 0), 2)
    cv2.line(frame, center, (center[0], 0), (0, 255, 0), 2)
    cv2.line(frame, center, (center[0], height), (0, 255, 0), 2)

    return frame


def tracking_process(frame, tracking, tracking_ok, bbox, object_lost):
    height, width, _ = frame.shape
    center_x = width // 2
    center_y = height // 2

    if tracking:
        Trackers.tracker_1.init(frame, bbox)
        print(Trackers.tracker_1.init(frame, bbox))
        print(bbox)
        tracking_ok = True
        tracking = False
    bbox1 = [0, 0, 0, 0]
    target_center = [center_x, center_y]

    if tracking_ok:
        ret1, bbox1 = Trackers.tracker_1.update(frame)
        if ret1:
            bbox1 = [int(e) for e in bbox1]
            cv2.rectangle(frame, (bbox1[0], bbox1[1]), (bbox1[0] + bbox1[2], bbox1[1] + bbox1[3]), (0, 255, 0), 2)
            frame = draw_lines_to_edges(frame, bbox1)
            target_center = (bbox1[0] + bbox1[2] // 2, bbox1[1] + bbox1[3] // 2)
        else:
            object_lost = True
            tracking_ok = False
    return frame, bbox1, tracking, object_lost, tracking_ok, target_center


def distance_calculator(frame, bbox1, real_object_width):
```

```
    if not bbox1 == [0,0,0,0]:
        camera = CameraConstants(os.getenv("MAIN_CAMERA"))
        pixels_x, pixels_y = camera.image_size
        pixels_per_cm = pixels_x / camera.matrix_x
        x, y, w, h = [int(e) for e in bbox1]
        object_size = ((w / 2) / pixels_per_cm) / 100
        if object_size != 0:
            distance = (camera.focal_length * (real_object_width + object_size)) / object_size
        else:
            distance = 0
    else:
        distance = 0
    cv2.putText(frame, f"{round(distance, 2)} M", (100, 300), cv2.FONT_HERSHEY_DUPLEX, 0.6, (0, 255, 0), 2)
    return frame, distance
```

nrf24_communications.py

```
import time
import os
from nrf24 import NRF24

ce_pin = int(os.getenv("CE_PIN"))
csn_pin = int(os.getenv("CSN_PIN"))
radio = NRF24(ce_pin, csn_pin)
radio.begin(ce_pin, csn_pin)
channel = int(os.getenv("NRF_CHANNEL"))
radio.setChannel(channel)
pipe = [bytes(os.getenv("PIPE_RX")), bytes(os.getenv("PIPE_TX"))]
payload_size = int(os.getenv("PAYLOAD_SIZE"))
radio.setPayloadSize(payload_size)

radio.openReadingPipe(1, pipe[1])
radio.startListening()


def main_comm_loop(input_queue, output_queue):
    while True:
        if not input_queue.empty():
            message_data = input_queue.get()
            radio.openWritingPipe(pipe[0])
            radio.write(bytes(message_data, 'utf-8'))
            print(f"Data sent: {message_data}")
            radio.openReadingPipe(1, pipe[1])
            radio.startListening()
            time.sleep(1)

        if radio.available():
            received_data = 0
            radio.read(received_data)
            print(f"Received: {received_data}")
            output_queue.put(received_data)
            time.sleep(1)
```

ДОДАТОК Б

Апробація наукових досліджень

ЗАТВЕРДЖЕНО

Дослідження програмного методу визначення відстані до об’єкту за допомогою параметрів камери

Апробація наукових досліджень

АРКУШІВ 6

2023р.

Міністерство освіти і науки України

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»

Керівник кваліфікаційної роботи

проф.Цимбал О. М.

Дослідження програмного методу визначення відстані до об’єкту за допомогою параметрів камери

Апробація наукових досліджень

ЛИСТ ЗАТВЕРДЖЕННЯ

РОЗРОБИВ:

ст. гр. КТРСм-22-1

Мілько Д.В.

2023р.

Міністерство освіти і науки України

NURE

Харківський національний університет радіоелектроніки

ЗБІРНИК

студентських наукових статей

«Автоматизація та приладобудування»

«Automation and Development of Electronic Devices»

ADED-2023

(Випуск 2)

[електронне видання]

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

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

http://kafea.kdu.edu.ua

Харків 2023

УДК 004.415.25

ДОСЛІДЖЕННЯ ПРОГРАМНОГО МЕТОДУ ВИЗНАЧЕННЯ ВІДСТАНІ ДО ОБ'ЄКТУ ЗА ДОПОМОГОЮ ПАРАМЕТРІВ КАМЕРИ

Д. В. Мілько
Харківський національний університет
радіоелектроніки
Україна, 61166, Харків, пр. Науки 14
Email: denys.milko@nure.ua

Анотація: Стаття обговорює програмну реалізацію методу з використанням Python та OpenCV, визначення відстані за допомогою формули тонкої лінзи та оцінює точність вимірювань відстані за параметрами матриці камери. Висновок підкреслює обмежену точність методу, але його доцільність у системах комп'ютерного зору при обмеженому бюджеті.

Ключові слова*:* Далекомір, комп'ютерний зір, OpenCV, Python.

RESEARCH OF THE SOFTWARE METHOD FOR DETERMINING THE DISTANCE TO AN OBJECT USING CAMERA PARAMETERS

D. Milko
Kharkiv National University of Radio Electronics
Ukraine, 61166, Kharkiv, Nauky av.,14
Email: denys.milko@nure.ua

Anotations: The article discusses the software implementation of the method using Python and OpenCV, distance determination using the thin lens formula, and evaluates the accuracy of distance measurements using camera sensor parameters. The conclusion emphasises the limited accuracy of the method, but its feasibility in computer vision systems with a limited budget.

Keywords: Rangefinder, computer vision, OpenCV, Python.

Визначення відстані до об'єктів - завдання, яке може бути вирішено різними методами в залежності від конкретного контексту та умов вимірювань. Один із широко використовуваних методів – тригонометрична триангуляція. Цей підхід ґрунтується на вимірюванні кутів та використанні трикутникової геометрії для розрахунку відстані між спостерігачем і об'єктом. Інший ефективний спосіб - використання лазерних далекомірів. Ці пристрої використовують лазерне випромінювання для вимірювання часу, який лазерному променю потрібно, щоб відбитися від об'єкта та повернутися до приладу.

Не зважаючи на ефективність цих методів, їх інтеграція може бути складною та вимагати значних фінансових затрат. Зокрема, використання лазерних далекомірів може вимагати значних витрат на розробку та інтеграцію системи. У випадках, коли бюджет обмежений або розглядається використання в невеликих системах, доцільним є використання інших доступних методів вимірювання відстаней, які можуть бути більш економічними та пристосованими до конкретних вимог. У ході дослідження, було проведено аналіз точності вимірювання відстані за допомогою параметрів матриці камери.

Для успішної програмної реалізації використаного методу необхідно розробити базову частину програми комп'ютерного зору. Ця базова частина включатиме в себе набір алгоритмів та функцій, спрямованих на обробку та аналіз зображень, отриманих від камери. Зокрема, вона буде відповідальною за обробку вхідних даних, взаємодію з обраною камерою, імплементацію вибраного методу визначення відстані та здійснення подальших досліджень.

Ця базова частина програми буде надавати необхідний інтерфейс для взаємодії з методом вимірювання відстані та дозволить проводити експерименти та дослідження. Вона є ключовим компонентом для успішної реалізації та оптимізації вибраного методу в контексті комп'ютерного зору. [1].

Для реалізації даного методу вибрано мову програмування Python та використано бібліотеку комп'ютерного зору OpenCV. Використання Python забезпечує зручність програмування та велику кількість наявних бібліотек, що сприяє швидкій та ефективній розробці. Бібліотека OpenCV є потужним інструментом для обробки зображень та комп'ютерного зору, надаючи широкий спектр функцій для роботи з відео та зображеннями. Об'єднуючи Python і OpenCV, дослідники отримали надійний інструментарій для впровадження та тестування розробленого методу визначення відстані до об'єкта. [2].

Для визначення відстані до об'єкта застосовується формула тонкої лінзи, що є загальноприйнятим методом у фотографії та оптиці. Ця формула, яка зазвичай використовується в об'єктивах звичайних камер, враховує лінійний розмір об'єкта, його розмір на матриці камери, і фокусну відстань лінзи. За допомогою цієї формули можна точно визначити відстань до об'єкта, що є важливим елементом у різних областях, таких як комп'ютерний зір та геодезія. (рис. 1).

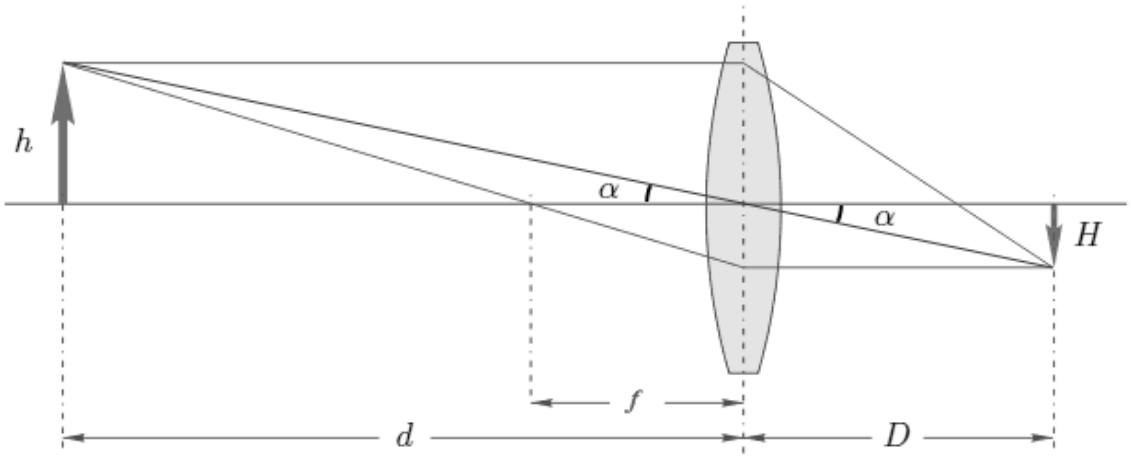

Рисунок 1 – Схема тонкої лінзи

При даному типі визначення відстані, необхідно мати сталі данні, які використовуються при розрахунку [3]. А саме розмір об’єкта, фокусна відстань лінзи та розмір об’єкта на матриці об’єктива.

Розрахунок проводиться за формулою:

$$d = \frac{(f(H+h))}{H}. \quad (1)$$

де:

h – лінійний розмір об’єкта;

H – розмір об’єкта на матриці;

f – фокусна відстань лінзи.

Далі в процесі визначення відстані до об'єкта необхідно звернутися до параметрів обраної камери, якою буде здійснюватися фотозйомка сцени з досліджуваним об'єктом. Обраною опцією є камера із матрицею Samsung ISOCELL GW3, яка має фізичні розміри 6.5 на 4.86 міліметра. Фокусна відстань об'єктива цієї камери складає 25 міліметрів, а розмір вихідного зображення становить 1080 на 1920 пікселів.

На основі цих вхідних даних можна визначити розмір об'єкта на матриці камери, використовуючи зв'язок між фізичними розмірами об'єкта і його відображенням на матриці. Ці дані стають ключовими у подальших розрахунках відстані до об'єкта, дозволяючи враховувати специфікації камери для отримання більш точних результатів вимірювань [4-6].

Розроблена програма починається ініціалізацією параметрів, таких як шлях до відеофайлу, визначення трекера, розмір реального об'єкта, параметри камери (такі як фокусна відстань та розміри матриці), і створення вікна для відстеження.

Система отримує відеофрейми, і якщо обрано відстеження об'єкта, ініціалізує трекер для вказаного прямокутника [7-9]. Потім трекер використовується для визначення положення об'єкта на кожному кадрі. Якщо відстеження успішне, виводиться інформація про відстань до об'єкта.

Крім того, код включає можливість вибору об'єкта для відстеження за допомогою миші та виводить додаткову інформацію, таку як кількість кадрів на секунду та час втрати об'єкта із зони відстеження [10-13].

Для тестування методу було створено відеозапис автомобіля, лінійний розмір якого дорівнює приблизно 2 м. Результати тестування представлені на рисунку 1.

Рисунок 2 – Результат тестування

Виходячи із отриманих результатів, можна зробити висновок, що використання цього методу дає не достатньо точні дані о відстані, так як розмір одного і того ж об’єкту може змінюватися залежно від його кута розміщення. Також можуть впливати чинники викривлення зображення лінзами, неточний розрахунок розмірів об’єкту у пікселях та інші.

Але при обмеженому бюджеті, такий варіант може доцільно використовуватися у системах комп’ютерного зору та нагляду.

ЛІТЕРАТУРА

1. OpenCV Documentation // Сайт Docs OpenCV, 2023. URL: https://docs.opencv.org/3.4/
2. Джо Мінічіно, Learning OpenCV 4 Computer Vision with Python 3. 2020. –253 с.
3. Henry Lipson, Optical Physics, 2012. – 590 с. – ISBN 9780511763120.
4. Дэвід Форсайт, Computer Vision: A Modern Approach, 2002. – 800 с.
5. Intel Articles [Електронний ресурс] URL: www.intel.com/content/www/us/en/developer/articles/ (дата звернення 03.10.2023)
6. Пратік Джоші, OpenCV By Example, / Девід Міллан Ескріва, Вінісіус Годой. – Packt, 2016. – 296 с. – ISBN 9781785280948.
7. TutorialsPoing. [Електронний ресурс] URL: https://www.tutorialspoint.com/opencv/opencv_overview.htm. (дата звернення 10.10.2023)
8. Packt Pub. [Електронний ресурс] URL: https://subscription.packtpub.com/book/application-development/. (дата звернення 15.09.2023)
9. Рейнхард Клетте, Computer vision. Theory and algorithms – Springer, 2019, – 506 с.
10. Габріель Крейман, Biological and Computer Vision. – Cambridge University Press, 2021. – 274 с.
11. Лінда Г. Шапіро, Computer Vision. – Pearson, 2001. – 608 с.
12. Навеенкумар Махамкалі, OpenCV for Computer Vision Applications / Вадівель Айясамі. – Trichy, 2015. – 135 с.
13. Machine Learning Made Easy [Електронний ресурс] – URL: https://www.mathworks.com/matlabcentral/fileexchange/50232-machinelearning-made-easy. (дата звернення 07.10.2023)

Науковий керівник: *Цимбал Олександр Михайлович, д.т.н., професор кафедри КІТАР Харківського національного університету радіоелектроніки*

ДОДАТОК В

Демонстраційний матеріал

ЗАТВЕРДЖЕНО

Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва

Демонстраційний матеріал

АРКУШІВ 6

2023р.

Міністерство освіти і науки України

Харківський Національний Університет Радіоелектроніки

«ЗАТВЕРДЖУЮ»

Керівник кваліфікаційної роботи

проф.Цимбал О. М.

Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва

Демонстраційний матеріал

ЛИСТ ЗАТВЕРДЖЕННЯ

РОЗРОБИВ:

ст. гр. КТРСм-22-1

Мілько Д.В.

2023р.

ДОДАТОК Г

Відомості кваліфікаційної роботи

№	Формат	Позначення	Найменування	Кіл.	Прим.
			Документація текстова		
1	А4	ГЮІК. 505400.016 ПЗ	Пояснювальна записка	76 с.	
			Інші документи		
2	А4	ГЮІК. 502840.006 -01 12 01	Додаток А	11 с.	
			Текст програми		
3	А4		Додаток Б	6 с.	
			Апробація наукових		
			досліджень		
4	А4		Додаток В	7 с.	у форматі *.ppt
			Демонстраційний		
			матеріал		

Змін.	Арк.	Номер докум.	Підп.	Дата	ГЮІК. 505400. 016 ВД			
Розроб.		Мілько Д. В.			Розроблення розподіленої системи керування засобами відеоспостереження інтегрованого виробництва Відомість кваліфікаційної роботи	Літера	Аркуш	Аркушів
Перевір.		Цимбал О.М						1
						ХНУРЕ кафедра КІТАР		
Н.контр.		Стародубцев М.Г.						
Затв.		Невлюдов І.Ш.						