DEPARTMENT OF AEROSPACE ENGINEERING,
SMME, NUST, ISLAMABAD

Project Report

CS – 117

Applications of ICT

Project Group Members	Class/Section	CMS ID
Abdul Moeed	AE-03(B)	548105
Danyal Hassan	AE-03(B)	562680
M. Eisa Hassan	AE-03(B)	540374

Submitted to Engr. Kashan Ahmad

1. Table of Contents

2. Introduction

2.1 Background

With the rapid growth of digital surveillance systems, continuous video recording has become a common practice in security and monitoring applications. However, such systems generate massive amounts of data, most of which contains no meaningful information. Continuous recording leads to inefficient storage usage, increased power consumption, and difficulty in reviewing footage when an actual event occurs.

Modern ICT-based solutions aim to optimize data collection by incorporating intelligent decision-making into systems. Motion-based recording is one such approach that ensures only relevant activity is captured, significantly improving efficiency.

2.2 Introduction & Context

This project presents a motion-activated camera recording system developed using Python and OpenCV. The system continuously monitors a live camera feed while maintaining a short-duration rolling buffer of recent frames. When motion is detected, the system saves the buffered frames along with real-time footage, ensuring that no important context before the motion is lost.

Once motion ceases, recording continues for a brief duration before stopping, after which the system returns to idle monitoring mode.

2.3 Relevance to the Subject

The project directly applies ICT concepts such as:

- Automation
- Computer vision
- Efficient data handling
- Software-based system optimization

It demonstrates how ICT tools can be used to solve real-world engineering problems related to surveillance, storage management, and system intelligence.

3. Research

3.1 Flow Chart

The system follows a structured workflow:

1. Camera initialization
2. Idle monitoring mode
3. Continuous frame buffering using a deque
4. Motion detection via frame comparison
5. Triggering of recording
6. Saving buffered, active, and post-motion frames
7. Returning to the idle state

This logical flow ensures uninterrupted monitoring with optimized storage usage.

3.2 Projected / Expected Outcomes

The expected outcomes of the project include:

- Significant reduction in stored video data
- Efficient recording of meaningful events only
- Automated operation with minimal user intervention
- Improved surveillance efficiency

3.3 Discussion / Literature Review

Existing motion-based surveillance systems include commercial CCTV solutions and smart home cameras such as Ring and Nest. While these systems provide reliable monitoring, they are often expensive, proprietary, and offer limited customization.

Basic open-source motion detection scripts exist, but they frequently lack features such as pre-motion recording, efficient buffering, and fine-grained sensitivity control. This project addresses these limitations by offering a customizable, open-source, and cost-effective alternative.

4. Development

4.1 Methodology

The system is implemented using OpenCV for real-time video processing. Motion detection is achieved using:

- Frame subtraction between consecutive frames
- Thresholding to highlight motion regions
- Contour detection to identify significant movement

A deque (double-ended queue) data structure is used to store recent frames, allowing efficient insertion and removal with minimal memory overhead.

4.2 List of Components / Libraries

Hardware Components:

- Smartphone camera (used as live camera input)
- Computer system

Software Components:

- Python programming language
- OpenCV library
- GitHub for version control

4.3 Hurdles and Solutions

Several challenges were encountered during development:

- **False motion detection:** Reduced by adjusting threshold values and filtering small contours
- **Lighting variations:** Managed through sensitivity tuning
- **Storage management:** Solved using a rolling buffer mechanism

4.4 Final Project

The final implementation successfully records a complete motion event consisting of:

- Pre-motion footage
- During-motion recording
- Post-motion footage

This ensures contextual completeness while maintaining efficient storage usage.

5. Project Details

5.1 Project Demo

1. The camera operates in idle monitoring mode
2. Recent frames are continuously stored in a rolling buffer
3. Motion detection algorithm analyzes incoming frames
4. Upon detecting motion, buffered frames are written to a video file
5. Live recording continues until motion ends
6. Video file is saved and the system returns to idle mode

5.2 Error & Anomaly Considerations

- Sudden lighting changes may cause false positives
- Background movement can affect detection accuracy
- Camera noise may impact frame comparison

5.3 Scope

Future extensions of the project may include:

- AI-based object classification
- Multi-camera support
- IoT-based alerts and cloud storage
- User-adjustable graphical interface

5.4 Projection as a Product

The system can be developed into a low-cost surveillance product suitable for homes, offices, and small-scale industrial monitoring.

6. Discussion

This project demonstrates the effective use of ICT tools to design an intelligent surveillance system. The integration of motion detection with buffering ensures reliable event capture while minimizing unnecessary data storage.

7. Sustainability Considerations

- Reduced storage usage lowers energy consumption
- Minimizes unnecessary surveillance, improving privacy
- Uses open-source tools, making the solution affordable
- Lower storage requirements reduce electronic waste

8. Conclusion

The motion-activated camera recording system successfully applies ICT principles to address real-world surveillance challenges. By intelligently recording only relevant footage and preserving pre-motion context, the system achieves efficiency, reliability, and sustainability.

9. Appendix

Highlighted code.

```
import cv2  # to record and perform motion detection
import collections  # to store buffers in deque data structures
import time  # to for buffer and timeout time measurement
import datetime  # to display current time on recording

# ========================================
# SETTINGS
# ========================================

buffer = 3  # seconds of video to record before motion happened
timeout = 10  # seconds of video to record after motion has ceased
MIN_CONTOUR_AREA = 500  # Minimum area to consider as valid motion
OUTPUT_FILENAME = f"{datetime.datetime.now().strftime("%Y-%m-%d %H-%M")}.mp4"
vid_src = 0  # 0 for camera, filepath for video, url for mjpeg stream

# ========================================
# INITIALIZATION
# ========================================


# intialize video capture source and returns the cap object and video
properties.
def setup_camera(source):
    cap = cv2.VideoCapture(source)
    if not cap.isOpened():
        print("Error: Could not open camera.")
        return None, 0, 0, 0

    # get relevant information about the video stream
    frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)

    # if FPS is not reported or reportly 0, assume it to be 30
    if fps == 0.0:
        fps = 30.0

    return cap, fps, frame_width, frame_height


# makes the VideoWriter object for saving the recording to a file
def setup_video_writer(filename, fps, width, height):
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")  # specify file encode and
extension
    frame_size = (width, height)  # specify dimensions, taken from video
source
    return cv2.VideoWriter(filename, fourcc, fps, frame_size)  # return the
VideoWriter


# make a buffer of fixed size based on fps and how long of a buffer to keep
# uses the deque data structure which is more efficient than python lists
```

```
def initialize_buffer(fps, buffer_seconds):
    buffer_size = int(fps * buffer_seconds)
    return collections.deque(maxlen=buffer_size)


# ========================================
# MOTION DETECTION
# ========================================


# detect motion by applying background subtraction
def detect_motion_contours(frame, back_sub):
    # apply background subtraction
    fg_mask = back_sub.apply(frame)

    # apply erosion algorithm to remove noise
    kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (3, 3))
    mask_eroded = cv2.morphologyEx(fg_mask, cv2.MORPH_OPEN, kernel)

    # find contours edges, boundary of moving pixels
    contours, _ = cv2.findContours(
        mask_eroded, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )

    # filter contours by size
    valid_contours = [
        cnt for cnt in contours if cv2.contourArea(cnt) > MIN_CONTOUR_AREA
    ]

    return valid_contours


# draw boxes around contours to view movement and adjust sensitivity
def draw_bounding_boxes(frame, contours):
    for cnt in contours:
        x, y, w, h = cv2.boundingRect(cnt) # gives top left-corner(x,y) and
width and height(w,h) to draw rectangle ignoring rotation
        cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 200), 3) # draws
the rectangle onto the frame
    return frame


# ========================================
# PUT IT ALL TOGETHER
# ========================================


def start_surveillance(cam_source=0):
    # initializations

    # cam
    cap, fps, width, height = setup_camera(cam_source)
    if cap is None:
        return
    # buffer
    out = setup_video_writer(OUTPUT_FILENAME, fps, width, height)
    frame_buffer = initialize_buffer(fps, buffer)
```

```
    # MOG2 background subtractor (with shadows disabled)
    back_sub = cv2.createBackgroundSubtractorMOG2(detectShadows=False)

    # state variables for conditionals
    is_recording = False
    last_motion_time = time.time()

    print(f"FPS: {fps:.2f}. q to exit.")


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

            # Always add current frame to the circular buffer
            frame_buffer.append(frame)

            # detect motion
            motion_contours = detect_motion_contours(frame, back_sub)
            motion_detected_now = len(motion_contours) > 0 # if are drawing
even one contour that is counted as movement

            display_frame = frame.copy() # so we can show information like
boxes for debugging and adjustments on display frame

            if motion_detected_now:
                last_motion_time = time.time()
                draw_bounding_boxes(display_frame, motion_contours)

            # motion detected and we're not recording yet so add buffer first
            if motion_detected_now and not is_recording:
                is_recording = True
                print("motion weeeeee")

                # write buffer to recording
                for buffered_frame in list(frame_buffer):
                    out.write(buffered_frame)

            # start recording live
            cv2.putText(
                frame,
                datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
# text
                (10, 30),                       # position
                cv2.FONT_HERSHEY_SIMPLEX,       # font
                0.5,                            # font scale
                (255, 255, 255),                # color
                1,                              # thickness
                cv2.LINE_AA,                    # anti-aliasing for
smoooooothness
            )
            if is_recording:
                if time.time() - last_motion_time > timeout:
                    is_recording = False
```

```
                    print(f"Recording stopped. No motion for {timeout}s.")
                else:
                    # draw_bounding_boxes(display_frame, motion_contours)
                    out.write(frame)

            # display video
            cv2.imshow("Surveillance Feed", display_frame)

            if cv2.waitKey(1) & 0xFF == ord("q"):
                break

    finally:
        # Thanos SNAP
        print("Cleaning up resources...")
        cap.release()
        out.release()
        cv2.destroyAllWindows()


# ========================================
# ENTRY POINT
# ========================================

if __name__ == "__main__":
    start_surveillance(vid_src)
```

10. References

- OpenCV Documentation
- Python Official Documentation
- https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
- https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html
- https://opencv.org/blog/reading-and-writing-videos-using-opencv/
- https://learnopencv.com/moving-object-detection-with-opencv/
- AI for buffer solutions