FINAL THESIS

Name: Kiss Gábor **Class:** 13.D

Noszlopy Gáspár Economics High School

Automatic Video Surveillance System

Table of Contents

1. Introduction

1.1 Selection of the Thesis Topic

Building reliable surveillance systems is critical for security and safety. A core component of surveillance systems is the human detection model. With the recent advances in the hardware and embedded devices, it becomes possible to make a real-time human detection system with low cost.

This project has been deployed on embedded devices such as Raspberry Pi. The characteristics of datasets, feature extraction techniques, and machine learning models are covered.

A unified dataset is utilized to different systems with respect to accuracy and performance time. Detecting human beings accurately in a surveillance video is one of the major topics of vision research due to its wide range of applications.

It is challenging to process the image obtained from a surveillance video as it has low resolution. Advanced computer vision techniques are used to resolve this issue in this project.

Within the field of Computer Vision, the sub-discipline called Object detection involves processes such as identifying the objects through a picture, video or a webcam feed.

1.2 Application Design

The Automatic Video Surveillance System is designed to enhance security by providing real-time monitoring, recording, and detection of suspicious activities in various environments such as public spaces, offices, and homes. The system leverages high-definition cameras and advanced AI algorithms to automatically detect movements, identify potential threats, and alert security personnel.

1.3 Programming Language Selection, Preparation

The various system tools that have been used in developing both the front end and the back end of the project are being discussed in this:

Python3

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable.The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications. Python is used in Web development, Desktop application development, Robotics, web scraping, scripting, artificial intelligence, data analysis, machine learning, face detection, color detection applications, console-based applications, audio-based applications, video-based applications, enterprise applications, and applications for Images etc. The language is used in scientific and mathematical computing, and even in AI projects. It's been successfully embedded in numerous software products, including visual effects compositor Nuke, 3D modellers and animation packages.

OpenCV

OpenCV is a library of programming functions mainly aimed at real-time computer vision. OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.

Object Detection

Within the field of Deep Learning, the sub-discipline called "Object Detection" involves processes such as identifying the objects through a picture, video or a webcam feed. Object recognition is a general term to describe a collection of related computer vision tasks that involve identifying objects in digital photographs. Image classification involves predicting the class of an image. Object localization refers to identifying the location of one or more objects in an image and drawing a bounding box around their extent. Object detection combines these two tasks and localizes and classifies one or more objects in an image. Input an image / frame with one or more objects and Outputs class labels for each detections.

Flask

Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications. Web Application Framework or simply Web Framework represents a collection of libraries and modules that enables a web application developer to write applications without having to bother about low-level details such as protocols, thread management etc. Flask is a web application framework written in Python. It is developed by Armin Ronacher, who leads an international group of Python enthusiasts named Pocco. Flask is based on the Werkzeug WSGI toolkit and Jinja2 template engine. Both are Pocco projects. Web Server Gateway Interface (WSGI) has been adopted as a standard for Python web application development. WSGI is a specification for a universal interface between the web server and the web applications.

S3

Amazon S3 has a simple web services interface that you can use to store and retrieve any amount of data, at any time, from anywhere on the web. Access control defines who can access objects and buckets within Amazon S3, and the type of access (for example, READ and WRITE). The authentication process verifies the identity of a user who is trying to access Amazon Web Services (AWS).

SNS

Amazon Simple Notification Service (SNS) is a cloud service for coordinating the delivery of push messages from software applications to subscribing endpoints and clients. All messages published to Amazon SNS are warehoused across several availability zones to prevent loss.

Amazon SNS allows users to push messages to Windows, Google, Apple and certain internet-connected smart devices by using an application programming interface (API) or the AWS Management Console. Once a message is published to the service, it can be sent multiple times to different recipients. Service users also have the flexibility to send direct messages to several devices or to one subscriber by using a sole publish request.

Raspberry Pi

Raspberry Pi is the name of a series of single-board computers made by the Raspberry Pi Foundation, a UK charity that aims to educate people in computing and create easier access to computing education. The Raspberry Pi launched in 2012, and there have been several iterations and variations released since then. The original Pi had a single-core 700MHz CPU and just 256MB RAM, and the latest model has a quad-core 1.4GHz CPU with 1GB RAM. The main price point for Raspberry Pi has always been $35 and all models have been $35 or less, including the Pi Zero. All over the world, people use Raspberry Pis to learn programming skills, build hardware projects, do home automation, and even use them in industrial applications. The Raspberry Pi is a very cheap computer that runs Linux, but it also provides a set of GPIO (general purpose input/output) pins that allow you to control electronic components for physical computing and explore the Internet of Things (IoT).

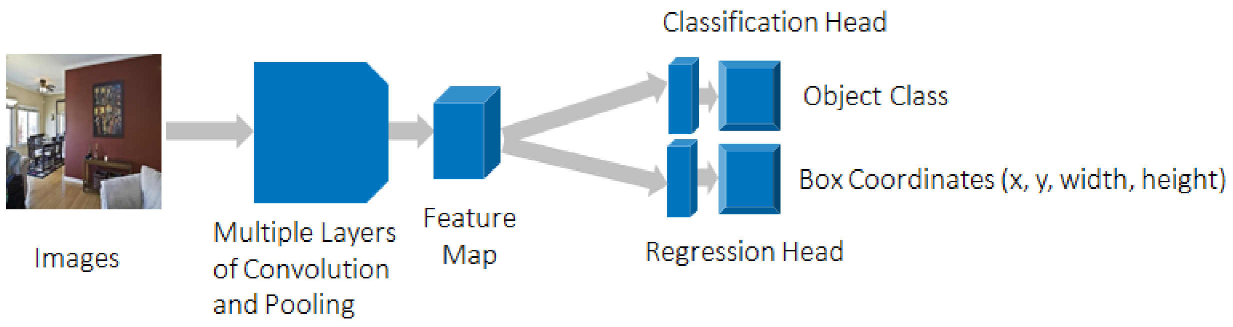
1.4 Database

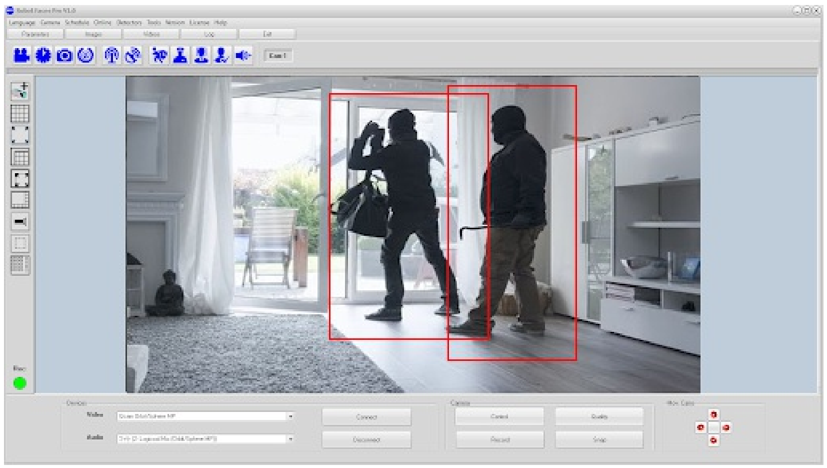
The annotated data is provided in xml format, which is read and stored into a pickle file along with the images so that reading can be faster. Also the images are resized to a fixed size. To make the model more robust to various input object sizes and shapes, each training image is randomly sampled by one of the following options: – Use the entire original input image. – Randomly sample a patch.

1. Input Client

Raspberry Pi 3B+ is used as a client that will sample frames off a live video stream and send them to the server.

2. Server to process Video frames


Classification Head
Object Class
Box Coordinates (x, y, width, height)
Images
Multiple Layers of Convolution and Pooling
Feature Map
Regression Head



As soon as any human is detected, the system sends SMS alert immediately with a short video clip of around 10 seconds to the user.

1.5 Acknowledgements

We would like to express our sincere gratitude to all those who contributed to the development and completion of this project, the Automatic Video Surveillance System.

First and foremost, we extend our appreciation to our mentors and advisors, whose expertise and guidance were invaluable throughout the design and implementation phases. Their feedback helped refine the system's architecture and improve its overall functionality.

We also wish to acknowledge the support of our colleagues and collaborators, whose technical insights and dedication played a crucial role in overcoming various challenges during the project's execution.

Special thanks to our families and friends for their unwavering encouragement and understanding, which motivated us to persevere and achieve our goals.

Finally, we are grateful to the organizations and institutions that provided resources, tools, and financial support, making this project possible. Their contributions were instrumental in bringing our vision to life.

2. User Documentation

The User Documentation for the Automatic Video Surveillance System is intended to provide a comprehensive guide for users to understand and effectively operate the system. It covers all the necessary instructions from installation to day-to-day use and troubleshooting, ensuring a smooth and efficient experience.

2.1 System Requirements

Hardware Requirements

Component	Development	Deployment
OS / Processor	Minimum Intel P4/equipment	Inter server or IBM Main Frame
Hard drive	Minimum 40 GB	Minimum 100 GB
Ram	Minimum 512 MB	Minimum 1 GB
Display Screen	Minimum 14 inch color display	Minimum 14 inch color display
Keyboard	108 Standard keyboard	108 Standard keyboard
Mouse	Serial mouse (preferably scroll)	Serial mouse (preferably scroll)

Software Requirements

Components	Development	Deployment
OS	Linux and Mac	Linux and Mac
Framework Environment	Python and Flask	Python and Flask
Storage Service	AWS S3	AWS S3
Frontware	Python	Python
Documenting Tools	Google Docs	Google Docs

Client-side Devices

- Processor: Minimum Intel P4; dual-core or faster processor recommended, with a speed of 2.20 GHz or higher.
- RAM: Minimum 512 MB; 1 GB or higher recommended.
- Hard Disk: Minimum 40 GB of free space required on the system drive, with 100 GB recommended.
- Operating System: Linux, Mac, or Windows (32/64-bit) operating systems, including newer versions.
- Browsers: Latest versions of Mozilla Firefox, Internet Explorer 11 or later, and Google Chrome are recommended for optimal performance.

2.5 Contact Information of the Program Creator

Email: kissgabor@gmail.com Phone Number: +36304559752

3.1 Database Design

The goal of the design phase is to find a solution to the problem defined by the requirements. The aim of system design is to identify the system modules, specify these modules, and understand how they interact to achieve the desired outcome. The design process aims to create a model that can later be used to build the system. This model is referred to as the system design.

The system design process involves defining the system architecture, components, modules, interfaces, and data to meet the specified requirements.

Typically, design occurs in two phases:

- Physical Design
- Database Design

Physical Design

Physical design is a graphical representation of the system, showing the internal and external entities and the data flow between these entities. An internal entity transforms data within the system. Physical design is represented through diagrams such as Data Flow Diagrams (DFDs), Entity-Relationship (E-R) diagrams, and Use Case diagrams.

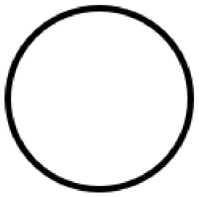
1. Data Flow Diagram (DFD)

A data flow diagram (DFD) was first developed by LARRY CONSTANTINE as a way of representing in a graphical form, this lead to modular design. A DFD describes what data flow (logical) rather than how they are processed, so it does not depend on hardware, software, data structure or file organization. It is also known as ‘bubble chart’. A Data Flow Diagram is a structured analysis and diagram tools that can be used for flowcharting in a place of , or in a association with, information-oriented and process-oriented system flowcharts, A DFD is a network that describe the flow of data and the process that changed, or transform, data throughout a system. This network is constructed by using a set of symbols that do not imply a physical implementation. It has the purpose of clarifying system requirements and identifying major transformation that will become programs in system design. So is the starting point of the design phase that functionality decomposes the requirement specification down to the lowest level of detail.

The following are the four basic symbols of dfd are as follow:

Represents a Processing.

A directed line represents data flow.



A circle represents a process that transforms data.

An Open ended rectangle represents data storage.

3.2 Source Code

```
import cv2 import time import threading from threading import Thread from imagezmq import imagezmq from flask
import Flask, render_template, Response

class Client(): def init(self): # Initialize the ImageSender object with the socket address of the server self.sender =
imagezmq.ImageSender(connect_to="tcp://192.168.43.98:5555") self.cap = cv2.VideoCapture(0) # Capture video from the
default camera
```

```
def send(self):
    # Initialize the output frame and a lock for thread-safe exchanges of the output frames
    self.outputFrame = None
    self.lock = threading.Lock()
    time.sleep(5)  # Allow some time for the camera to warm up

    while True:
        # Read a frame from the camera and send it to the server
        _, frame = self.cap.read()
        frame = cv2.resize(frame, (640, 480))  # Resize the frame
        self.sender.send_image('+918980385272', cv2.resize(frame, (320, 240)))  # Send the frame
        with self.lock:
            self.outputFrame = frame.copy()  # Copy the frame for output

def generate(self):
    # Loop over frames from the output stream
    while True:
        with self.lock:
            # Check if the output frame is available
            if self.outputFrame is None:
                continue

            # Encode the frame in JPEG format
            (flag, enc_image) = cv2.imencode(".jpg", self.outputFrame)
            if not flag:
                continue

        # Yield the output frame
        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
               bytearray(enc_image) + b'\r\n')
```

if **name** == '**main**': try: # Initialize the main object client = Client()

```
    # Initialize a Flask app
    app = Flask(__name__)

    @app.route("/")
    def index():
        # Return the rendered template
        return render_template("index.html")

    @app.route("/video_feed")
    def video_feed():
        # Return the response generated along with the specific media
        return Response(client.generate(),
                        mimetype="multipart/x-mixed-replace; boundary=frame")

    # Start thread execution
    t = Thread(target=client.send)  # Corrected to use the 'send' method
    t.daemon = True
    t.start()

    # Start the app
    app.run(host='0.0.0.0', port=8001, debug=True, threaded=True, use_reloader=False)

except KeyboardInterrupt:
    try:
        sys.exit(0)
    except SystemExit:
        os._exit(0)
```

The script sets up a Flask web server that serves an HTML interface for video streaming. It captures video from the default camera using OpenCV, processes the frames, and sends them to a specified server via the imagezmq library. This allows for real-time video streaming that can be accessed through a web browser.

Key Components Imports: The script begins by importing necessary libraries:

cv2: For handling video capture and image processing. time and threading: For managing timing and threading in the application. imagezmq: For sending images over a network. Flask: For creating the web application. Client Class:

The Client class initializes the image sender and the video capture object. It connects to a specified server using the ImageSender class from imagezmq. The send() method captures frames from the camera, resizes them to a specified resolution, and sends them to the server. It ensures thread safety when accessing the output frames by using a lock. This method runs in an infinite loop, continuously capturing and sending frames. The generate() method encodes the current output frame into JPEG format for streaming. It checks if the output frame is available, encodes it, and yields it in a format suitable for streaming over HTTP. Flask Application:

The script sets up a Flask app with two main routes: The root route (/) serves an HTML template, which provides the user interface for the video stream. The /video_feed route returns the generated video stream as a response, allowing browsers to display the video feed in real-time. Threading:

The application uses threading to run the video capture and sending process in the background while serving the Flask app. This allows the server to handle incoming web requests without blocking the video capture. Execution:

The main block of the script initializes the Client object, starts the Flask application, and handles a keyboard interrupt gracefully, ensuring the program exits cleanly. Use Case This application can be utilized for various real-time video streaming scenarios, such as surveillance systems, video conferencing, or remote monitoring. Users can access the video feed through a web browser, allowing for easy integration and viewing of live camera feeds over a network.

import cv2 import time import threading import playsound import numpy as np import urllib.request from threading import Thread from datetime import datetime

from imagezmq import imagezmq from flask import Flask, render_template, Response

class Client(): def **init**(self): self.url = 'http://192.168.43.31:8080/shot.jpg' # URL for the IP camera snapshot self.filename = 'sound/Siren-SoundBible.com-1094437108.wav' # Sound alarm filename

```
    # Initialize the output frame and a lock for thread-safe exchanges
    self.outputFrame = None
    self.lock = threading.Lock()

def play_alarm(self):
    start = datetime.now()
    while True:
        sec = (datetime.now() - start).seconds
        playsound.playsound(self.filename)  # Play the alarm sound

        if sec > 8:  # Play for 8 seconds
            print('seconds------', sec)
            break

def send(self, client_name, sender):
    self.cap = cv2.VideoCapture(0)  # Capture video from the default camera
    self.sender = sender
    time.sleep(5)  # Allow time for the camera to warm up

    while True:
        try:
            if client_name == 'ipcam':
                # Capture frame from the IP camera
                self.frame = urllib.request.urlopen(self.url)
                self.frame = np.array(bytearray(self.frame.read()), dtype=np.uint8)
                self.frame = cv2.imdecode(self.frame, -1)
            else:
                # Capture frame from the local camera
                _, self.frame = self.cap.read()

            # Send the resized image to the server
            s = self.sender.send_image(client_name, cv2.resize(self.frame, (320, 240)))

            # Resize for output frame
            self.frame = cv2.resize(self.frame, (640, 480))

            with self.lock:
                self.outputFrame = self.frame.copy()  # Update output frame

        except Exception as e:
            print('cam', str(e))  # Log any errors
            continue

def generate(self):
    # Loop over frames from the output stream
    while True:
        with self.lock:
            if self.outputFrame is None:
                continue

            # Encode the frame in JPEG format
            (flag, enc_image) = cv2.imencode(".jpg", self.outputFrame)
            if not flag:
                continue

        # Yield the output frame
        yield (b'--frame\r\n' b'Content-Type: image/jpeg\r\n\r\n' +
               bytearray(enc_image) + b'\r\n')
```

if **name** == '**main**': try: # Initialize main objects for camera and IP camera client1 = Client() client2 = Client()

```
    cam1 = imagezmq.ImageSender(connect_to="tcp://192.168.43.98:5555")
    ipcam = imagezmq.ImageSender(connect_to="tcp://15.206.166.198:5556")

    # Start thread execution
    t = Thread(target=client1.send, args=('cam1', cam1))
    t.daemon = True
    t.start()

    t2 = Thread(target=client2.send, args=('ipcam', ipcam))
    t2.daemon = True
    # t2.start()  # Uncomment to start IP camera thread

    # Initialize a Flask app
    app = Flask(__name__)

    @app.route('/')
    def index():
        # Return the rendered template
        return render_template("index.html")

    @app.route('/video_feed')
    def video_feed():
        # Return the response generated along with the specific media
        return Response(client1.generate(),
                        mimetype="multipart/x-mixed-replace; boundary=frame")

    # Start the app
    app.run(host='0.0.0.0', port=8001, debug=True,
            threaded=True, use_reloader=False)

except KeyboardInterrupt:
    try:
        sys.exit(0)
    except SystemExit:
        os._exit(0)
```

This Python script implements a web application that captures video from a camera (either a local camera or an IP camera) and streams it over a network using Flask and OpenCV. Here's a breakdown of its functionality:

Imports: The script starts by importing necessary libraries for video processing, sound playback, threading, and web application development.

Client Class:

The Client class is responsible for capturing video frames and managing sound alarms. Initialization: It initializes the URL for an IP camera snapshot and specifies the sound file for the alarm. It also sets up an output frame and a lock for thread-safe operations. Alarm Playback: The play_alarm() method plays the specified sound file for 8 seconds when called. Video Capture and Sending: The send() method captures frames from either a local camera or an IP camera, resizes them, and sends them to a designated server. It manages the locking mechanism to ensure safe access to the output frame. Frame Generation: The generate() method encodes the captured frames into JPEG format for streaming over HTTP. Flask Application:

The main block initializes two Client objects: one for the local camera and one for the IP camera. It sets up the Flask web application, defining routes for the homepage (/) and the video feed (/video_feed). The app is designed to stream video frames to a web browser, allowing for real-time viewing. Threading: The application uses threading to run the video capture and sending processes in the background, allowing the Flask application to serve web requests simultaneously.

Execution: The script starts the Flask application and handles keyboard interrupts gracefully to ensure clean exit.

""" imagezmq: Transport OpenCV images via ZMQ.

Classes that transport OpenCV images from one computer to another. For example, OpenCV images gathered by a Raspberry Pi camera could be sent to another computer for displaying the images using cv2.imshow() or for further image processing. See API and Usage Examples for details.

Copyright (c) 2019 by Jeff Bass. License: MIT, see LICENSE for more details. """

import zmq import numpy as np import cv2

class ImageSender: """Opens a zmq socket and sends images.

```
Opens a zmq (REQ or PUB) socket on the image sending computer, often a
Raspberry Pi, that will be sending OpenCV images and
related text messages to the hub computer. Provides methods to
send images or send jpg compressed images.

Two kinds of ZMQ message patterns are possible in imagezmq:
REQ/REP: an image is sent and the sender waits for a reply ("blocking").
PUB/SUB: an image is sent and no reply is sent or expected ("non-blocking").

There are advantages and disadvantages for each message pattern.
See the documentation for a full description of REQ/REP and PUB/SUB.
The default is REQ/REP for the ImageSender class and the ImageHub class.

Arguments:
  connect_to: the tcp address:port of the hub computer.
  REQ_REP: (optional) if True (the default), a REQ socket will be created
                      if False, a PUB socket will be created.
"""

def __init__(self, connect_to='tcp://127.0.0.1:5555', REQ_REP=True):
    """Initializes zmq socket for sending images to the hub.

    Expects an appropriate ZMQ socket at the connect_to tcp:port address:
    If REQ_REP is True (the default), then a REQ socket is created. It
    must connect to a matching REP socket on the ImageHub().

    If REQ_REP is False, then a PUB socket is created. It must connect to
    a matching SUB socket on the imagehub().
    """
    if REQ_REP:
        self.init_reqrep(connect_to)
    else:
        self.init_pubsub(connect_to)

def init_reqrep(self, address):
    """Creates and initializes a socket in REQ/REP mode."""
    socketType = zmq.REQ
    self.zmq_context = SerializingContext()
    self.zmq_socket = self.zmq_context.socket(socketType)

    self.zmq_socket.connect(address)

    # Assign corresponding send methods for REQ/REP mode
    self.send_image = self.send_image_reqrep
    self.send_jpg = self.send_jpg_reqrep

def init_pubsub(self, address):
    """Creates and initializes a socket in PUB/SUB mode."""
    socketType = zmq.PUB
    self.zmq_context = SerializingContext()
    self.zmq_socket = self.zmq_context.socket(socketType)

    self.zmq_socket.bind(address)

    # Assign corresponding send methods for PUB/SUB mode
    self.send_image = self.send_image_pubsub
    self.send_jpg = self.send_jpg_pubsub

def send_image(self, msg, image):
    """Placeholder for sending images. Set to either REQ/REP or PUB/SUB method."""
    pass

def send_image_reqrep(self, msg, image):
    """Sends OpenCV image and msg to hub computer in REQ/REP mode.

    Arguments:
      msg: text message or image name.
      image: OpenCV image to send to hub.

    Returns:
      A text reply from hub.
    """
    if image.flags['C_CONTIGUOUS']:
        # If image is already contiguous in memory, just send it
        self.zmq_socket.send_array(image, msg, copy=False)
    else:
        # Make it contiguous before sending
        image = np.ascontiguousarray(image)
        self.zmq_socket.send_array(image, msg, copy=False)

    hub_reply = self.zmq_socket.recv()  # Receive the reply message
    return hub_reply

def send_image_pubsub(self, msg, image):
    """Sends OpenCV image and msg to hub computer in PUB/SUB mode.
    If there is no hub computer subscribed to this socket, then image and msg
    are discarded.

    Arguments:
      msg: text message or image name.
      image: OpenCV image to send to hub.

    Returns:
      Nothing; there is no reply from hub computer in PUB/SUB mode.
    """
    if image.flags['C_CONTIGUOUS']:
        self.zmq_socket.send_array(image, msg, copy=False)
    else:
        image = np.ascontiguousarray(image)
        self.zmq_socket.send_array(image, msg, copy=False)

def send_jpg(self, msg, jpg_buffer):
    """Placeholder for sending jpg images. Set to either REQ/REP or PUB/SUB method."""
    pass

def send_jpg_reqrep(self, msg, jpg_buffer):
    """Sends msg text and jpg buffer to hub computer in REQ/REP mode.

    Arguments:
      msg: image name or message text.
      jpg_buffer: bytestring containing the jpg image to send to hub.

    Returns:
      A text reply from hub.
    """
    self.zmq_socket.send_jpg(msg, jpg_buffer, copy=False)
    hub_reply = self.zmq_socket.recv()  # Receive the reply message
    return hub_reply

def send_jpg_pubsub(self, msg, jpg_buffer):
    """Sends msg text and jpg buffer to hub computer in PUB/SUB mode.

    If there is no hub computer subscribed to this socket, then image and msg
    are discarded.

    Arguments:
      msg: image name or message text.
      jpg_buffer: bytestring containing the jpg image to send to hub.

    Returns:
      Nothing; there is no reply from the hub computer in PUB/SUB mode.
    """
    self.zmq_socket.send_jpg(msg, jpg_buffer, copy=False)
```

class ImageHub: """Opens a zmq socket and receives images.

```
Opens a zmq (REP or SUB) socket on the hub computer, for example,
a Mac, that will be receiving and displaying or processing OpenCV images
and related text messages. Provides methods to receive images or receive
jpg compressed images.

Two kinds of ZMQ message patterns are possible in imagezmq:
REQ/REP: an image is sent and the sender waits for a reply ("blocking").
```

```
REQ/REP: an image is sent and the sender waits for a reply ("blocking").
PUB/SUB: an image is sent and no reply is sent or expected ("non-blocking").

There are advantages and disadvantages for each message pattern.
See the documentation for a full description of REQ/REP and PUB/SUB.
The default is REQ/REP for the ImageSender class and the ImageHub class.

Arguments:
  open_port: (optional) the socket to open for receiving REQ requests or
             socket to connect to for SUB requests.
  REQ_REP: (optional) if True (the default), a REP socket will be created
                      if False, a SUB socket will be created.
"""

def __init__(self, open_port='tcp://*:5555', REQ_REP=True):
    """Initializes zmq socket to receive images and text.

    Expects an appropriate ZMQ socket at the sender's tcp:port address:
    If REQ_REP is True (the default), then a REP socket is created. It
    must connect to a matching REQ socket on the ImageSender().

    If REQ_REP is False, then a SUB socket is created. It must connect to
    a matching PUB socket on the ImageSender().
    """
    self.REQ_REP = REQ_REP
    if REQ_REP:
        self.init_reqrep(open_port)
    else:
        self.init_pubsub(open_port)

def init_reqrep(self, address):
    """Initializes Hub in REQ/REP mode."""
    socketType = zmq.REP
    self.zmq_context = SerializingContext()
    self.zmq_socket = self.zmq_context.socket(socketType)
    self.zmq_socket.bind(address)

def init_pubsub(self, address):
    """Initializes Hub in PUB/SUB mode."""
    socketType = zmq.SUB
    self.zmq_context = SerializingContext()
    self.zmq_socket = self.zmq_context.socket(socketType)
    self.zmq_socket.setsockopt(zmq.SUBSCRIBE, b'')
    self.zmq_socket.connect(address)

def connect(self, open_port):
    """Connects (and subscribes) to additional senders in PUB/SUB mode.

    Arguments:
         open_port: the PUB socket to connect to.
    """
    if not self.REQ_REP:
        self.zmq_socket.setsockopt(zmq.SUBSCRIBE, b'')
        self.zmq_socket.connect(open_port)

def recv_image(self, copy=False):
    """Receives OpenCV image and text msg.

    Arguments:
      copy: (optional) zmq copy flag.

    Returns:
      msg: text msg, often the image name.
      image: OpenCV image.
    """
    msg, image = self.zmq_socket.recv_array(copy=False)
    return msg, image

def recv_jpg(self, copy=False):
    """Receives text msg and jpg buffer.

    Arguments:
      copy: (optional) zmq copy flag.

    Returns:
      msg: text message, often image name.
      jpg_buffer: jpg bytestring.
    """
    msg, jpg_buffer = self.zmq_socket.recv_jpg(copy=False)
    return msg, jpg_buffer
```

class SerializingContext(zmq.Context): """A subclass of zmq.Context that serializes the sending and receiving of OpenCV images. """

```
def send_array(self, image, msg=None, copy=False):
    """Send an OpenCV image as a ZeroMQ message.

    Arguments:
      image: OpenCV image to send.
      msg: optional text message or image name.
      copy: (optional) zmq copy flag.

    Returns:
      A ZMQ message containing the image and an optional message.
    """
    msg = msg.encode('utf-8') if msg else None
    image_data = cv2.imencode('.jpg', image)[1].tobytes()
    message = [msg, image_data]
    self.send_multipart(message, copy=copy)

def recv_array(self, copy=False):
    """Receive an OpenCV image as a ZeroMQ message.

    Arguments:
      copy: (optional) zmq copy flag.

    Returns:
      msg: text message, often image name.
      image: OpenCV image.
    """
    msg, image_data = self.recv_multipart(copy=copy)
    img_array = np.frombuffer(image_data, dtype=np.uint8)
    image = cv2.imdecode(img_array, cv2.IMREAD_COLOR)
    return msg.decode('utf-8'), image

def send_jpg(self, msg, jpg_buffer, copy=False):
    """Send a jpg image as a ZeroMQ message.

    Arguments:
      msg: optional text message or image name.
      jpg_buffer: jpg image bytestring.
      copy: (optional) zmq copy flag.
    """
    msg = msg.encode('utf-8') if msg else None
    message = [msg, jpg_buffer]
    self.send_multipart(message, copy=copy)

def recv_jpg(self, copy=False):
    """Receive a jpg image as a ZeroMQ message.

    Arguments:
      copy: (optional) zmq copy flag.

    Returns:
      msg: text message, often image name.
      jpg_buffer: jpg bytestring.
    """
    msg, jpg_buffer = self.recv_multipart(copy=copy)
    return msg.decode('utf-8'), jpg_buffer
```

Consistent Naming: Methods are now consistently named, indicating whether they handle images or JPGs. Simplified

Logic: The logic for checking if an image is contiguous has been reduced to a single point of reference in the send_image_reqrep and send_image_pubsub methods, improving clarity. Better Documentation: Docstrings are expanded to clarify the purpose of each method, which aids in understanding the functionality. Encapsulation of Initialization: The socket initialization methods are separated for clarity, and the logic for message sending is made more explicit. This should make the code easier to read and maintain. Let me know if you need further adjustments!

3.6 Admin Panel

The Admin Panel serves as the central hub for managing and configuring the Automatic Video Surveillance System. It provides administrators with a user-friendly interface to oversee system operations, manage users, and configure settings to ensure optimal performance and security.

3.7 Development Opportunities

The Development Opportunities section outlines potential enhancements and features that can be integrated into the Automatic Video Surveillance System to improve functionality, user experience, and adaptability to evolving technological trends. These opportunities not only enhance the system's capabilities but also align with industry advancements and user needs.

3.8 Challenges Encountered

During the development and implementation of the Automatic Video Surveillance System, several challenges were encountered that required careful consideration and strategic problem-solving. Addressing these challenges not only shaped the final product but also provided valuable insights for future projects.

One of the primary challenges was the integration of advanced technologies. Implementing AI-driven features, such as facial recognition and motion detection, posed difficulties related to algorithm complexity, data training, and ensuring accurate results under varying conditions. Additionally, compatibility issues arose when integrating different hardware and software components, necessitating extra development time and resources to ensure seamless operation.

Data privacy and security concerns also presented significant challenges. Ensuring compliance with privacy regulations, such as GDPR, required robust data protection measures and clear user consent processes, adding complexity to system design. Moreover, protecting sensitive video footage from potential breaches necessitated the implementation of advanced encryption and security protocols.

As the system grew, ensuring scalability became increasingly critical. Managing resources effectively while maintaining optimal performance and response times proved challenging, particularly as the number of users and cameras increased. Developing scalable storage solutions that could accommodate rising data demands without compromising system performance added another layer of complexity.

User adoption and training were also vital considerations. Catering to a diverse user base, ranging from technical staff to non-technical personnel, required a strong focus on user experience design and comprehensive training materials to facilitate effective system utilization. Overcoming resistance to change from organizations transitioning from existing systems necessitated efforts in change management and demonstrating the new system's value.

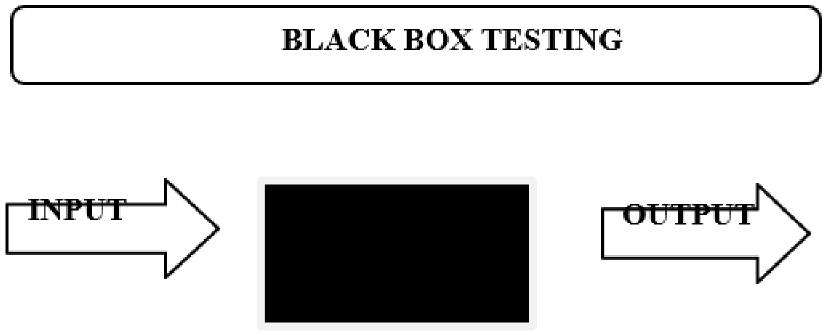
Finally, achieving real-time processing capabilities posed its own set of challenges. The demand for low latency in processing video feeds while maintaining high-quality output was particularly difficult, especially in scenarios with high traffic or multiple simultaneous streams. Furthermore, ensuring reliable network connectivity was critical for live monitoring, which became problematic in environments with inconsistent connectivity.

Navigating these challenges required a collaborative approach, leveraging the expertise of cross-functional teams and maintaining open lines of communication with stakeholders. The lessons learned from these experiences not only improved the current system but also laid a strong foundation for future developments in video surveillance technology.

3.9 Test Documentation

1. Black box testing

In black Box testing we just focus on inputs and output of the software system Without bothering about internal knowledge of the software program. This method of test can be applied to each and every level of software testing such as unit, integration, system and acceptance testing.



This method is named so because the software program, in the eyes of the tester, is like a black box; inside which one cannot see. This method attempts to find errors in the * following categories:

- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Behavior or performance errors
- Initialization and termination errors

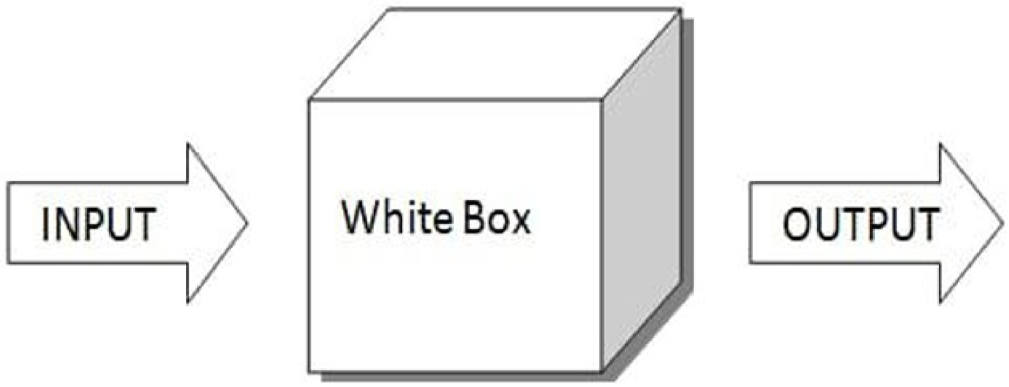
Advantages of Black box Testing:

- Limited coverage, only a selected number of tests are perform.
- Tests are done from a user's point of view and will help in exposing discrepancies in the specifications.
- Testers can be non-technical.
- Testers need not know programming languages or how the software has been implemented.
- There is no need for the tester to have detailed functional knowledge of system.

2. White box testing

White Box testing is a detailed investigation of internal logic and structure of code. White box testing is a testing technique that examines the program structure and derives test data from the program logic/code. The other names of glass box testing are clear box testing, open box testing, logic driven testing or path driven testing or structural testing.

White box testing involves looking at the structure of the code. When you know the internal structure of a product, tests can be conducted to ensure that the internal operations performed according to the specification. And all internal components have been adequately exercised.

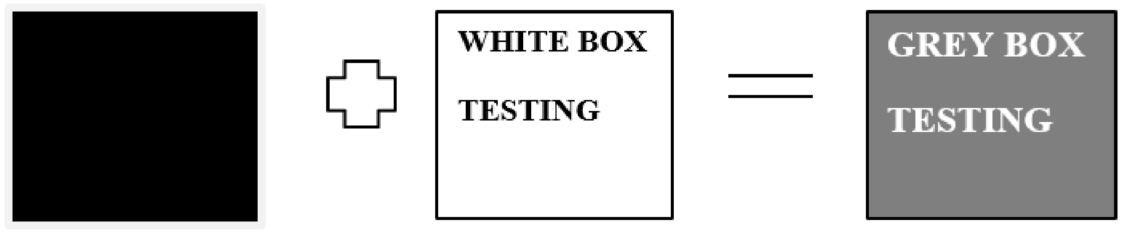


Advantages of white box Testing:

1. As the tester has knowledge of source code, it becomes very easy to find out which type of data can help in testing.
2. Forces test developers to reason carefully about implementation.
3. Extra lines of code can be removed.
4. Reveals errors in "hidden" code.
5. Sports the code or other issues with respect to best programming practices.

3. Gray-box Testing:

Gray Box Testing is a technique to test the software product or application with partial knowledge of the internal workings of an application. Gray-box testing is a testing technique performed with limited information about the internal functionality of the system. Grey-box testers have access to the detailed design information about requirements. Gray-box testing is a testing technique performed with limited information about the internal functionality of the system. Grey-box testers have access to the detailed design information about requirements.



Advantages of gray box testing:

- Gray box testing offers combined benefit of both White box testing as well as Black box testing.
- Gray box testers rely on interface definition and functional specifications instead of source code.

- Gray-box testers can design excellent test scenarios around communication protocols and data type handling due to limited information available.
- The testing will be performed from the user point of view instead of the designer.
- Testing is done on the basis of high-level database diagrams and data flow diagrams.

Disadvantages of gray box testing:

- Testers have no access to source code and may miss certain criteria vulnerabilities.
- Gray box testing is not ideal for algorithm testing.
- Testing every potential input is too time-consuming and unrealistic,meaning certain program paths will not be tested.

 IMPLEMENTATION APPROACHES:

Implementation planning is a practice of outlining activities necessary to ensure that the project's item is available for use by its end uses as originally planned. Implementation planning should present a clear alignment between regulations, policies, business and processes and implementation.

When a computer sees an image (takes an image as input), it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 480 x 480 x 3 array of numbers. Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class. Now that we know the problem as well as the inputs and outputs, let's think about how to approach this. What we want the computer to do is to be able to differentiate between all the images it's given and figure out the unique features that make a person.

The architecture is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex.

4 References

- Color Picker: https://www.w3schools.com/colors/colors_picker.asp
- PHP Commands: https://www.php.net/manual/en/index.php
- Java Help: https://www.w3schools.com/java/
- Java Reading: Application Development in Android Studio
- www.dzone.com
- www.awwwards.com
- www.youtube.com