ZÁRÓDOLGOZAT

Név: Kiss Gábor **Osztály:** 13.D

Noszlopy Gáspár Közgazdasági Szakközépiskola

Automatikus Videó Megfigyelő Rendszer

Tartalomjegyzék

---

1. Bevezetés

1.1 A téma kiválasztása

Megbízható megfigyelő rendszerek kiépítése alapvető fontosságú a biztonság és a védelem szempontjából. A megfigyelő rendszerek egyik kulcsfontosságú összetevője az emberi detektáló modell. A közelmúltban elért hardver- és beágyazott eszközökkel lehetővé válik, hogy alacsony költséggel valós idejű emberi detektáló rendszert készítsünk.

Ez a projekt beágyazott eszközökön, például Raspberry Pi-n került telepítésre. A datasetek jellemzői, a jellemzők kinyerési technikái és a gépi tanulási modellek kerülnek bemutatásra.

Egy egységes datasetet használunk különböző rendszerekhez a pontosság és a teljesítmény idő tekintetében. Az emberek pontos detektálása egy megfigyelő videóban a látáskutatás egyik fő témája, széleskörű alkalmazási lehetőségei miatt.

Kihívást jelent a megfigyelő videóból származó képek feldolgozása, mivel alacsony felbontású. A projekt során fejlett számítógépes látási technikákat használnak e probléma megoldására.

A Számítógépes Látás területén a "Objektum detektálás" nevű szakterület magában foglalja az objektumok azonosítását képek, videók vagy webkamera adatok alapján.

1.2 Alkalmazás tervezése

Az Automatikus Videó Megfigyelő Rendszer célja a biztonság növelése azáltal, hogy valós idejű megfigyelést, rögzítést és gyanús tevékenységek detektálását biztosítja különböző környezetekben, például nyilvános helyeken, irodákban és otthonokban. A rendszer nagy felbontású kamerákat és fejlett AI algoritmusokat használ a mozgások automatikus detektálására, a potenciális fenyegetések azonosítására és a biztonsági személyzet figyelmeztetésére.

1.3 Programozási nyelv kiválasztása, előkészítés

A projekt frontend és backend fejlesztésében használt különböző rendszereszközök bemutatásra kerülnek:

Python3

A Python egy magas szintű, értelmezett, interaktív és objektum-orientált szkriptnyelv. A Python úgy van tervezve, hogy nagyon olvasható legyen. A Python interpreter könnyen bővíthető új funkciókkal és adatszerkezetekkel, amelyeket C vagy C++ (vagy más C-ből hívható nyelvek) nyelven valósítanak meg. A Python alkalmas testreszabható alkalmazások kiterjesztésére is. A Python használatos webfejlesztésben, asztali alkalmazások fejlesztésében, robotikában, webes adatgyűjtésben, szkriptek írásában, mesterséges intelligenciában, adatelemzésben, gépi tanulásban, arcfelismerésben, színfelismerő alkalmazásokban, konzolos alkalmazásokban, audio-alapú alkalmazásokban, videó-alapú alkalmazásokban, vállalati alkalmazásokban és képekkel kapcsolatos alkalmazásokban. A nyelvet tudományos és matematikai számításokhoz, sőt AI projektekhez is használják. Sikeresen beágyazták számos szoftvertermékbe, beleértve a vizuális effektek összeállítóját, a Nuke-ot, 3D modellező és animációs csomagokat.

OpenCV

Az OpenCV egy programozási funkciók könyvtára, amely főként a valós idejű számítógépes látásra irányul. Az OpenCV (Open Source Computer Vision Library) egy nyílt forráskódú számítógépes látás és gépi tanulás szoftver könyvtára. Az OpenCV célja egy közös infrastruktúra biztosítása a számítógépes látás alkalmazások számára és a gépi érzékelés

kereskedelmi termékekben való használatának felgyorsítása. Mivel BSD-licenc alatt áll, az OpenCV megkönnyíti a vállalkozások számára a kód használatát és módosítását. A könyvtár több mint 2500 optimalizált algoritmust tartalmaz, amelyek átfogó készletét tartalmazzák a klasszikus és a legmodernebb számítógépes látási és gépi tanulási algoritmusoknak.

Objektumdetektálás

A Mély Tanulás területén az "Objektumdetektálás" nevű szakterület magában foglalja az objektumok azonosítását képek, videók vagy webkamera adatok alapján. Az objektumfelismerés egy általános kifejezés, amely a digitális fényképekben található objektumok azonosítását célzó kapcsolódó számítógépes látási feladatok gyűjteményét jelöli. A képfelismerés magában foglalja egy kép osztályának megjóslását. Az objektumlokalizálás arra vonatkozik, hogy azonosítsuk egy vagy több objektum helyét egy képen, és keretet rajzolunk az adott objektum köré. Az objektumdetektálás kombinálja ezeket a két feladatot, és lokalizálja, valamint osztályozza a képen található egy vagy több objektumot. Bemeneti képként / keretként egy vagy több objektummal osztálycímkéket ad ki minden detektálásra.

Flask

A Flask egy népszerű Python webkeretrendszer, ami azt jelenti, hogy egy harmadik fél Python könyvtár, amelyet webalkalmazások fejlesztésére használnak. A Webalkalmazás-keretrendszer, vagy egyszerűen Webkeretrendszer egy olyan könyvtárak és modulok gyűjteménye, amely lehetővé teszi a webalkalmazás fejlesztő számára, hogy alkalmazásokat írjon anélkül, hogy alacsony szintű részletekkel, például protokollokkal, szálkezeléssel stb. kellene foglalkoznia. A Flask egy Python nyelven írt webalkalmazás-keretrendszer. Armin Ronacher fejlesztette ki, aki egy nemzetközi Python-rajongókból álló csoport, a Pocco vezetője. A Flask a Werkzeug WSGI eszközkészletre és a Jinja2 sablonmotorra épül, mindkettő a Pocco projektek része. A Web Server Gateway Interface (WSGI) standardizált specifikációként került elfogadásra a Python webalkalmazás-fejlesztésében. A WSGI egy specifikáció a webkiszolgáló és a webalkalmazások közötti univerzális interfészre.

S3

Az Amazon S3 egyszerű webszolgáltatási interfészt kínál, amelyet bármilyen mennyiségű adat tárolására és visszakeresésére használhat, bármikor, bárhonnan az interneten. A hozzáférés-vezérlés határozza meg, hogy ki férhet hozzá az Amazon S3-ban lévő objektumokhoz és vödrökhöz, valamint a hozzáférés típusát (például OLVASÁS és ÍRÁS). Az azonosítási folyamat megerősíti annak a felhasználónak a személyazonosságát, aki megpróbál hozzáférni az Amazon Web Services (AWS) szolgáltatásokhoz.

SNS

Az Amazon Simple Notification Service (SNS) egy felhőalapú szolgáltatás, amely a push üzenetek szállításának koordinálására szolgál szoftveralkalmazásokból az előfizető végpontokra és kliensekre. Az Amazon SNS-re publikált összes üzenetet több elérhetőségi zónában tárolják, hogy megakadályozzák az adatvesztést.

Az Amazon SNS lehetővé teszi a felhasználók számára, hogy üzeneteket küldjenek Windows, Google, Apple és bizonyos internethez csatlakozó okoseszközök számára API vagy az AWS Management Console használatával. Amint egy üzenetet közzétesznek a szolgáltatáson, azt többször is el lehet küldeni különböző címzetteknek. A szolgáltatásfelhasználók rugalmasan küldhetnek közvetlen üzeneteket több eszközre vagy egyetlen előfizetőnek egyetlen közzétételi kérelem használatával.

Raspberry Pi

A Raspberry Pi egy sor egylapos számítógép neve, amelyet a Raspberry Pi Alapítvány készített, amely egy brit

jótékonysági szervezet, amelynek célja az emberek informatikai oktatása és az informatikai oktatáshoz való könnyebb hozzáférés biztosítása. A Raspberry Pi 2012-ben indult, azóta több iteráció és variáció is megjelent. Az eredeti Pi egy egymagos 700MHz CPU-val és mindössze 256MB RAM-mal rendelkezett, míg a legújabb modell egy négy magos 1.4GHz CPU-val és 1GB RAM-mal bír. A Raspberry Pi alapára mindig 35 dollár volt, és minden modell 35 dollár vagy annál kevesebb áron érhető el, beleértve a Pi Zerót is. Világszerte sokan használják a Raspberry Pi-t programozási készségek elsajátítására, hardverprojektek készítésére, otthoni automatizálásra, sőt ipari alkalmazásokban is. A Raspberry Pi egy nagyon olcsó számítógép, amely Linuxot futtat, de emellett biztosít egy sor GPIO (általános célú bemeneti/kimeneti) lábat is, amelyek lehetővé teszik az elektronikai alkatrészek vezérlését a fizikai számításhoz és az Internet of Things (IoT) felfedezéséhez.

1.4 Adatbázis

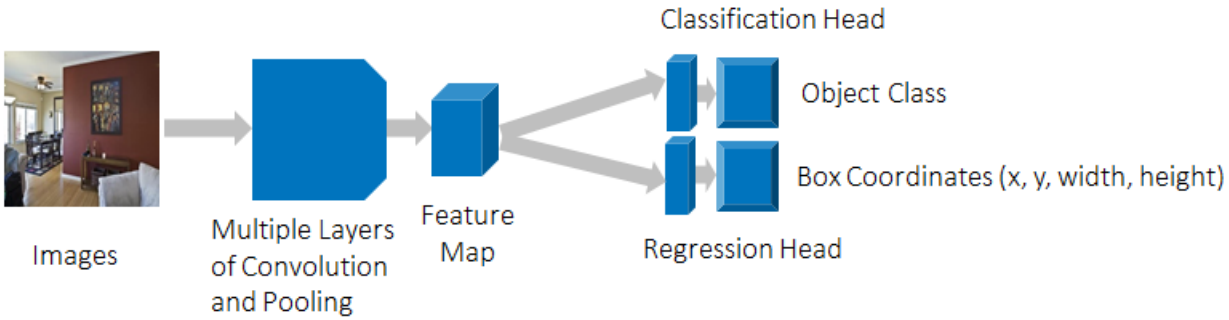
A megjegyzett adatok xml formátumban állnak rendelkezésre, amelyeket képekkel együtt pickle fájlba olvasnak és tárolnak, hogy a beolvasás gyorsabb legyen. Emellett a képeket egy fix méretre is átméretezik. A modell robusztusabbá tétele érdekében a különböző bemeneti objektumméretek és -formákhoz minden tanítóképet véletlenszerűen mintavételeznek az alábbi lehetőségek egyikével:

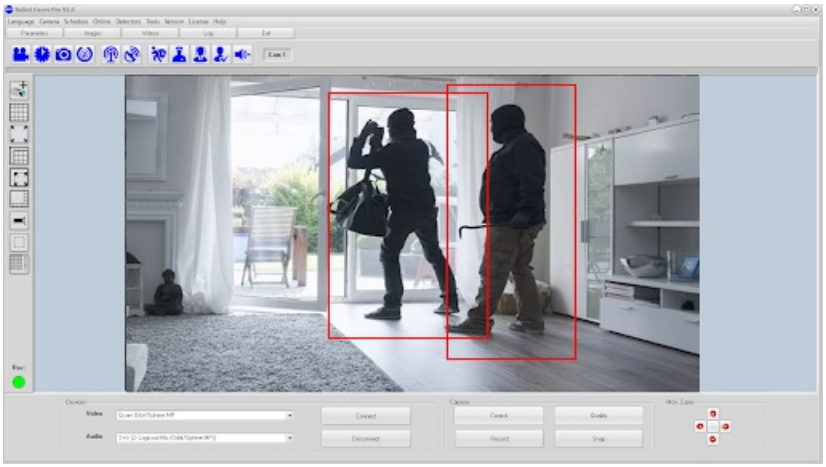
- Használja az eredeti bemeneti képet.
- Véletlenszerűen mintavételezzen egy foltot.

1. Felhasználói Kliens

A Raspberry Pi 3B+ mint kliens működik, amely élő videófolyamból képkockákat mintáz, és azokat a szerverre küldi.

2. Szerver a videóképkockák feldolgozásához





Amint bármilyen embert észlelnek, a rendszer azonnal SMS riasztást küld, egy körülbelül 10 másodperces rövid videóklippel a felhasználónak.

1.5 Köszönetnyilvánítás

Szeretnénk kifejezni őszinte hálánkat mindazoknak, akik hozzájárultak ennek a projektnek, az Automatikus Videó Megfigyelő Rendszer fejlesztéséhez és befejezéséhez.

Elsősorban szeretnénk köszönetet mondani mentoraiknak és tanácsadóiknak, akik szakértelmük és útmutatásuk elengedhetetlenek voltak a tervezési és megvalósítási fázisok során. Visszajelzéseik segítettek a rendszer architektúrájának finomításában és annak általános funkcionalitásának javításában.

Szeretnénk elismerni kollégáink és együttműködőink támogatását is, akik technikai meglátásaik és elkötelezettségük kulcsszerepet játszottak a projekt megvalósítása során felmerült különböző kihívások leküzdésében.

Külön köszönet a családunknak és barátainknak a folyamatos bátorításukért és megértésükért, amely motivált minket a kitartásra és a céljaink elérésére.

Végül hálásak vagyunk azoknak a szervezeteknek és intézményeknek, amelyek erőforrásokat, eszközöket és pénzügyi támogatást nyújtottak, lehetővé téve e projekt megvalósulását. Hozzájárulásuk alapvető fontosságú volt víziónk megvalósításában.

2. Felhasználói Dokumentáció

Az Automatikus Videó Megfigyelő Rendszer felhasználói dokumentációja egy átfogó útmutatót kíván nyújtani a felhasználók számára, hogy megértsék és hatékonyan működtethessék a rendszert. Tartalmaz minden szükséges utasítást a telepítéstől a napi használatig és a hibaelhárításig, biztosítva a zökkenőmentes és hatékony élményt.

2.1 Rendszerkövetelmények

Hardverkövetelmények

Komponens	Fejlesztés	Telepítés
OS / Processzor	Minimum Intel P4/berendezés	Inter server vagy IBM Main Frame
Merevlemez	Minimum 40 GB	Minimum 100 GB
RAM	Minimum 512 MB	Minimum 1 GB
Kijelző	Minimum 14 hüvelykes színes kijelző	Minimum 14 hüvelykes színes kijelző
Billentyűzet	108 Standard billentyűzet	108 Standard billentyűzet
Egér	Soros egér (lehetőleg görgetős)	Soros egér (lehetőleg görgetős)

Szoftverkövetelmények

Komponensek	Fejlesztés	Telepítés
OS	Linux és Mac	Linux és Mac
Keretrendszer	Python és Flask	Python és Flask
Tárolási szolgáltatás	AWS S3	AWS S3
Frontend	Python	Python
Dokumentáló eszközök	Google Docs	Google Docs

Ügyféleszközök

- Processzor: Minimum Intel P4; duálmagos vagy annál gyorsabb processzor ajánlott, 2.20 GHz sebességgel vagy magasabbal.
- RAM: Minimum 512 MB; 1 GB vagy magasabb ajánlott.
- Merevlemez: Minimum 40 GB szabad hely szükséges a rendszerlemezen, 100 GB ajánlott.
- Operációs rendszer: Linux, Mac vagy Windows (32/64-bit) operációs rendszerek, beleértve az újabb verziókat.
- Böngészők: A legfrissebb verziók, mint a Mozilla Firefox, Internet Explorer 11 vagy újabb, és Google Chrome ajánlottak az optimális teljesítményhez.

2.5 A Program Készítőjének Kapcsolati Információi

Email: kissgabor@gmail.com
Telefonszám: +36304559752

3.1 Adatbázis Tervezés

A tervezési fázis célja, hogy megoldást találjon a követelmények által meghatározott problémára. A rendszertervezés célja a rendszer moduljainak azonosítása, ezek specifikálása, és megérteni, hogyan lépnek kölcsönhatásba egymással a kívánt eredmény elérése érdekében. A tervezési folyamat célja egy olyan modell létrehozása, amely később felhasználható a rendszer felépítésére. Ezt a modellt rendszertervezésnek nevezzük.

A rendszertervezési folyamat magában foglalja a rendszer architektúrájának, komponenseinek, moduljainak, interfészeinek és adatainak meghatározását a megadott követelmények teljesítése érdekében.

Általában a tervezés két fázisban történik:

- Fizikai tervezés
- Adatbázis tervezés

Fizikai tervezés

A fizikai tervezés a rendszer grafikus ábrázolása, amely bemutatja a belső és külső entitásokat, valamint az adatok áramlását ezek között az entitások között. A belső entitás az adatokat a rendszeren belül alakítja át. A fizikai tervezést olyan diagramokkal ábrázolják, mint az Adatáramlási Diagramok (DFD), Entitás-Kapcsolat (E-R) diagramok, és Használati eset diagramok.

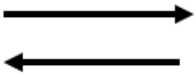
1. Adatáramlási Diagram (DFD)

Az adatáramlási diagramot (DFD) először LARRY CONSTANTINE fejlesztette ki, mint grafikus ábrázolási módot, amely moduláris tervezéshez vezetett. A DFD leírja, hogy milyen adatáramlás (logikai) van, nem pedig azt, hogyan dolgozzák fel őket, így nem függ a hardvertől, szoftvertől, adatszerkezettől vagy fájlok szervezésétől. Más néven „buborékdiagramnak” is nevezik. Az Adatáramlási Diagram egy strukturált elemzési és diagramkészítő eszköz, amely felhasználható folyamatábrázolásra információ-orientált és folyamat-orientált rendszerfolyamatábrák helyett vagy azokkal együtt. A DFD egy hálózat, amely leírja az adatok áramlását és a folyamatokat, amelyek megváltoztatják vagy átalakítják az adatokat a rendszerben. Ezt a hálózatot egy olyan szimbólumkészlettel építik fel, amely nem sugallja a fizikai megvalósítást. Célja a rendszer követelményeinek tisztázása és a fő átalakítások azonosítása, amelyek programokká válnak a rendszertervezés során. Ez a tervezési fázis kiindulópontja, amely a funkcionalitás lebontja a követelmények specifikációját a

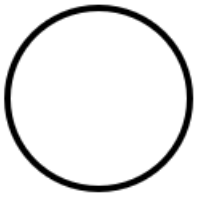
legalacsonyabb szintű részletességig.

A következő négy alapvető DFD szimbólum a következő:

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 Forráskód

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)
```

A szkript egy Flask webkiszolgálót állít be, amely HTML felületet biztosít videóstreaminghez. Az alapértelmezett kameráról videót rögzít az OpenCV használatával, feldolgozza a képkockákat, és elküldi őket egy meghatározott szerverre az imagezmq könyvtár segítségével. Ez lehetővé teszi a valós idejű videóstreaminget, amely egy webböngészőn keresztül elérhető.

Főbb komponensek

Importok: A szkript az alábbi szükséges könyvtárak importálásával kezdődik:

- **cv2:** A videó rögzítés és képkezelés kezelésére.
- **time és threading:** Az időzítés és a szálkezelés irányításához az alkalmazásban.
- **imagezmq:** Képek hálózaton történő küldésére.
- **Flask:** A webalkalmazás létrehozásához.

Client osztály:

A Client osztály inicializálja a képküldőt és a videó rögzítő objektumot. Az imagezmq ImageSender osztályának segítségével kapcsolódik a meghatározott szerverhez.
A `send()` metódus a kamerából képkockákat rögzít, átméretezi azokat a megadott felbontásra, és elküldi őket a szerverre. A metódus a kimeneti képkockák elérésénél biztosítja a szálbiztonságot egy zárolás használatával. Ez a metódus végtelen ciklusban fut, folyamatosan rögzítve és küldve a képkockákat.
A `generate()` metódus a jelenlegi kimeneti képkockát JPEG formátumba kódolja a streaminghez. Ellenőrzi, hogy a kimeneti képkocka elérhető-e, kódolja azt, és olyan formátumban adja vissza, amely alkalmas a HTTP-n történő streamingre.

Flask alkalmazás:

A szkript egy Flask alkalmazást állít be két fő útvonallal:

- A gyökér útvonal (/) egy HTML sablont szolgáltat, amely biztosítja a felhasználói felületet a videóstreamhez.
- A `/video_feed` útvonal visszaadja a generált videóstreamet válaszként, lehetővé téve a böngészők számára, hogy valós időben megjelenítsék a videofeedet.

Szálkezelés:

Az alkalmazás szálkezelést használ a videó rögzítésének és küldési folyamatának háttérben történő futtatására, miközben a Flask alkalmazást kiszolgálja. Ez lehetővé teszi a szerver számára, hogy kezelje a bejövő webes kéréseket anélkül, hogy blokkolná a videó rögzítést.

Futtatás:

A szkript fő blokkja inicializálja a Client objektumot, elindítja a Flask alkalmazást, és kezel egy billentyűzet megszakítást, biztosítva, hogy a program tisztán leálljon.

Használati eset

Ez az alkalmazás különböző valós idejű videóstreaming forgatókönyvekhez használható, például megfigyelő rendszerekhez, videokonferenciákhoz vagy távoli megfigyeléshez. A felhasználók webböngészőn keresztül hozzáférhetnek a videofeedhez, lehetővé téve a könnyű integrációt és az élő kameraképek megtekintését a hálózaton.

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)
```

Ez a Python szkript egy webalkalmazást valósít meg, amely videót rögzít egy kamerából (helyi kamerából vagy IP kamerából), és streameli azt egy hálózaton a Flask és az OpenCV segítségével. Íme a funkcionalitásának részletezése:

Importok

A szkript azzal kezdődik, hogy importálja a videófeldolgozáshoz, hanglejátszáshoz, szálkezeléshez és webalkalmazás-fejlesztéshez szükséges könyvtárakat.

Client osztály:

A Client osztály felelős a videóképkockák rögzítéséért és a hangriasztások kezeléséért.

- **Inicializálás:** Inicializálja az IP kamera pillanatképének URL-jét, és meghatározza a riasztáshoz szükséges hangfájlt. Beállít egy kimeneti képkockát és egy zárat a szálbiztos műveletekhez.
- **Riasztás lejátszása:** A `play_alarm()` metódus 8 másodpercig játssza le a megadott hangfájlt, amikor hívják.
- **Videó rögzítése és küldése:** A `send()` metódus képkockákat rögzít egy helyi kamerából vagy egy IP kamerából, átméretezi őket, és elküldi egy kijelölt szerverre. Kezeli a zárolási mechanizmust, hogy biztosítsa a kimeneti képkockához való biztonságos hozzáférést.

- **Képkocka generálás:** A `generate()` metódus a rögzített képkockákat JPEG formátumba kódolja a HTTP-n történő streaminghez.

Flask alkalmazás:

A fő blokk inicializál két Client objektumot: egyet a helyi kamerához és egyet az IP kamerához.
Beállítja a Flask webalkalmazást, definiálva az útvonalakat a kezdőlaphoz (/) és a videofeedhez (/video_feed).
Az alkalmazás úgy van megtervezve, hogy videóképkockákat streameljen egy webböngészőbe, lehetővé téve a valós idejű megtekintést.

Szálkezelés

Az alkalmazás szálkezelést használ a videó rögzítési és küldési folyamatok háttérben történő futtatására, lehetővé téve, hogy a Flask alkalmazás egyidejűleg kiszolgálja a webes kéréseket.

Futtatás

A szkript elindítja a Flask alkalmazást, és gondoskodik a billentyűzet megszakításának kezeléséről, hogy biztosítsa a tiszta leállítást.

""" imagezmq: OpenCV képek szállítása ZMQ-n keresztül.

Olyan osztályok, amelyek OpenCV képeket szállítanak egyik számítógépről a másikra. Például a Raspberry Pi kamerával gyűjtött OpenCV képek elküldhetők egy másik számítógépre a képek megjelenítésére a cv2.imshow() használatával, vagy további kép feldolgozásra. A részletekért lásd az API-t és a használati példákat.

Copyright (c) 2019 Jeff Bass. Licenc: MIT, a részletekért lásd a LICENSE-t. """

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;
```

```
    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").
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)
```

```
        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
```

Következetes Névhasználat

A metódusok most következetesen vannak elnevezve, jelezve, hogy képekkel vagy JPG-kkel dolgoznak.

Egyszerűsített Logika

A logika, amely ellenőrzi, hogy egy kép összefüggő-e, egyetlen hivatkozási pontra csökkent, a `send_image_reqrep` és a `send_image_pubsub` metódusokban, javítva a világosságot.

Jobb Dokumentáció

A docstringek kiterjesztésre kerültek, hogy tisztázzák az egyes metódusok célját, ami segíti a funkcionalitás megértését.

Inicializálás Kapszulázása

A socket inicializálási metódusokat különválasztották a világosság érdekében, és a hivatkozási logika a üzenetek küldésére egyértelműbbé vált. Ez megkönnyíti a kód olvasását és karbantartását. Ha további módosításokra van szükséged, szólj!

3.6 Admin Panel

Az Admin Panel a központi központ a Automatikus Videó Megfigyelő Rendszer kezeléséhez és konfigurálásához. Felhasználóbarát felületet biztosít az adminisztrátorok számára a rendszer működésének felügyeletéhez, a felhasználók kezeléséhez és a beállítások konfigurálásához a megfelelő teljesítmény és biztonság biztosítása érdekében.

3.7 Fejlesztési Lehetőségek

A Fejlesztési Lehetőségek szakasz vázolja az Automatikus Videó Megfigyelő Rendszerbe integrálható potenciális fejlesztéseket és funkciókat, amelyek javíthatják a funkcionalitást, a felhasználói élményt és az alkalmazkodást a fejlődő technológiai trendekhez. Ezek a lehetőségek nemcsak a rendszer képességeit bővítik, hanem összhangban állnak az iparági fejlesztésekkel és a felhasználói igényekkel.

3.8 Felmerült Kihívások

A Automatikus Videó Megfigyelő Rendszer fejlesztése és végrehajtása során számos kihívással szembesültünk, amelyek gondos mérlegelést és stratégiai problémamegoldást igényeltek. E kihívások kezelése nemcsak a végterméket formálta, hanem értékes betekintést is nyújtott a jövőbeli projektekhez.

Az egyik fő kihívás a fejlett technológiák integrációja volt. Az AI-alapú funkciók, mint például az arcfelismerés és a mozgásérzékelés megvalósítása nehézségeket okozott az algoritmusok összetettsége, az adatok tanítása és a változó körülmények közötti pontos eredmények biztosítása terén. Továbbá, kompatibilitási problémák merültek fel különböző hardver- és szoftverkomponensek integrálásakor, ami extra fejlesztési időt és erőforrásokat igényelt a zökkenőmentes működés biztosításához.

Az adatvédelmi és biztonsági aggályok is jelentős kihívásokat jelentettek. A GDPR-nek való megfelelés biztosítása érdekében robusztus adatvédelmi intézkedésekre és világos felhasználói beleegyezési folyamatokra volt szükség, ami bonyolulttá tette a rendszer tervezését. Ezen kívül a bizalmas videófelvételek védelme a potenciális adatlopásokkal szemben fejlett titkosítási és biztonsági protokollok bevezetését igényelte.

Ahogy a rendszer növekedett, a skálázhatóság biztosítása egyre kritikusabbá vált. Az erőforrások hatékony kezelése, miközben optimális teljesítményt és válaszidőt tartottunk fenn, kihívást jelentett, különösen a felhasználók és kamerák

számának növekedésével. Olyan skálázható tárolási megoldások kifejlesztése, amelyek képesek voltak a növekvő adatigények kielégítésére anélkül, hogy veszélyeztették volna a rendszer teljesítményét, további bonyodalmat jelentett.

A felhasználói elfogadás és képzés szintén létfontosságú szempontok voltak. Egy diverz közönség kiszolgálása, amely magában foglalja a műszaki személyzetet és a nem műszaki személyeket, erős fókuszt igényelt a felhasználói élmény tervezésére és átfogó képzési anyagokra a hatékony rendszerhasználat megkönnyítése érdekében. A meglévő rendszerekről átálló szervezeteknél a változás ellenállásának leküzdése érdekében a változásmenedzsment erőfeszítései és az új rendszer értékének bemutatása volt szükséges.

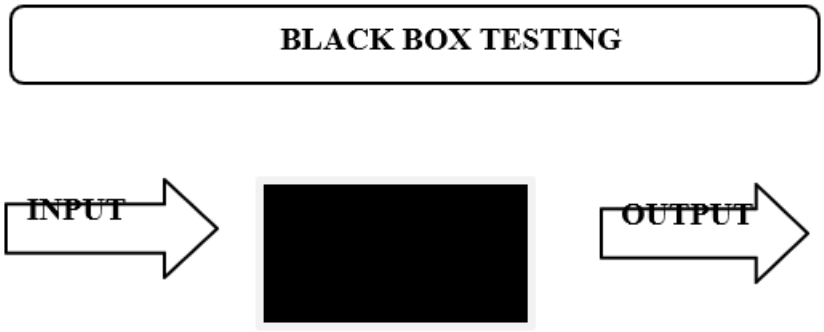
Végül a valós idejű feldolgozási képességek elérése saját kihívásokat jelentett. Az alacsony késleltetés iránti igény a videófolyamok feldolgozása során, miközben magas minőségű kimenetet tartottunk fenn, különösen nehéz volt, különösen nagy forgalmú vagy több egyidejű stream esetén. Ezenkívül a megbízható hálózati kapcsolatok biztosítása kritikus volt az élő megfigyeléshez, ami problémás lett olyan környezetekben, ahol a kapcsolatok ingadozóak voltak.

Ezeknek a kihívásoknak a navigálása együttműködő megközelítést igényelt, kihasználva a funkcionális csapatok szakértelmét és fenntartva a nyílt kommunikációs vonalakat az érdekelt felekkel. Az ezekből a tapasztalatokból levont tanulságok nemcsak a jelenlegi rendszert javították, hanem erős alapot is teremtettek a jövőbeli fejlesztésekhez a videómegfigyelési technológiában.

3.9 Teszt Dokumentáció

1. Fekete dobozos tesztelés

A fekete dobozos tesztelés során csak a szoftverrendszer bemeneteire és kimeneteire összpontosítunk anélkül, hogy a szoftverprogram belső működésével foglalkoznánk.
Ez a tesztelési módszer alkalmazható minden egyes szoftvertesztelési szinten, mint például egység, integráció, rendszer és elfogadási tesztelés.



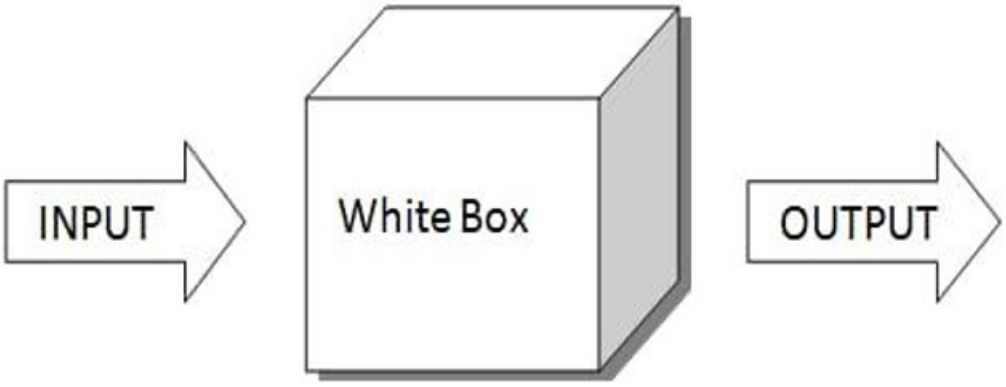
Ez a módszer azért kapta ezt a nevet, mert a szoftverprogram a tesztelő szemében olyan, mint egy fekete doboz; amelynek belseje nem látható. Ez a módszer arra törekszik, hogy hibákat találjon a következő kategóriákban:

- Helytelen vagy hiányzó funkciók
- Felület hibák
- Hibák az adatszerkezetekben vagy külső adatbázisok elérésében
- Viselkedési vagy teljesítménybeli hibák
- Inicializálási és leállítási hibák

2. Fehér dobozos tesztelés

A fehér dobozos tesztelés a kód belső logikájának és struktúrájának részletes vizsgálatát jelenti.
A fehér dobozos tesztelés egy olyan tesztelési technika, amely megvizsgálja a program struktúráját, és a program logikájából/kódjából származó tesztadatokat derivál.
A fehér dobozos tesztelés más elnevezései közé tartozik az üvegdobozos tesztelés, a tiszta dobozos tesztelés, a nyílt dobozos tesztelés, a logikaalapú tesztelés vagy az útvonalalapú tesztelés vagy struktúraalapú tesztelés.

A fehér dobozos tesztelés magában foglalja a kód szerkezetének vizsgálatát. Amikor ismered a termék belső struktúráját, a tesztek végrehajthatók annak biztosítása érdekében, hogy a belső műveletek a specifikáció szerint történjenek, és hogy minden belső komponens megfelelően legyen tesztelve.



A fehér dobozos tesztelés előnyei:

1. Mivel a tesztelő ismeri a forráskódot, nagyon könnyű megállapítani, hogy milyen típusú adatok segíthetnek a tesztelésben.
2. Kényszeríti a tesztfejlesztőket, hogy alaposan átgondolják a megvalósítást.
3. További kódvonalak eltávolíthatók.
4. Felfedi a "rejtett" kódban található hibákat.
5. Felfedi a kódot vagy más problémákat a legjobb programozási gyakorlatokkal kapcsolatban.

3. Szürke dobozos tesztelés

A szürke dobozos tesztelés egy olyan technika, amely a szoftvertermék vagy alkalmazás tesztelésére irányul, részleges ismeretekkel az alkalmazás belső működéséről.
A szürke dobozos tesztelés egy olyan tesztelési technika, amelyet a rendszer belső funkcióival kapcsolatos korlátozott információk birtokában hajtanak végre. A szürke dobozos tesztelők hozzáférnek a követelmények részletes tervezési információihoz.

A szürke dobozos tesztelés előnyei:

- A szürke dobozos tesztelés mind a fehér dobozos, mind a fekete dobozos tesztelés előnyeit egyesíti.
- A szürke dobozos tesztelők az interfészdefiníciókra és a funkcionális specifikációkra támaszkodnak, nem a forráskódra.
- A szürke dobozos tesztelők kiváló tesztscenáriókat tudnak tervezni a kommunikációs protokollok és az adattípusok kezelésével kapcsolatban a rendelkezésre álló korlátozott információk miatt.
- A tesztelés a felhasználói nézőpontból történik, nem a tervező szempontjából.
- A tesztelést magas szintű adatbázis-diagramok és adatáramlási diagramok alapján végzik.

A szürke dobozos tesztelés hátrányai:

- A tesztelők nem férnek hozzá a forráskódhoz, és előfordulhat, hogy kihagynak bizonyos sebezhetőségeket.
- A szürke dobozos tesztelés nem ideális algoritmusok tesztelésére.
- Minden lehetséges bemenet tesztelése túl időigényes és irreális, ami azt jelenti, hogy bizonyos programutakat nem tesztelnek.

□ IMPLEMENTÁCIÓS MEGKÖZELÍTÉSEK

Az implementációs tervezés a szükséges tevékenységek vázlatos felvázolásának gyakorlata, amely biztosítja, hogy a projekt eleme a tervezett módon elérhető legyen a végfelhasználók számára.
Az implementációs tervezésnek világos összhangot kell biztosítania a szabályok, politikák, üzleti folyamatok és az implementáció között.

Amikor egy számítógép képet lát (képet fogad be bemenetként), egy pixelértékekből álló tömbként látja azt. A kép felbontásától és méretétől függően egy 480 x 480 x 3 tömböt lát számokkal. E számok közül mindegyik 0 és 255 közötti értéket kap, amely leírja a pixel intenzitását azon a ponton.
Az elképzelés az, hogy a számítógépnek ezt a számokból álló tömböt kell megadni, és az kimenetként olyan számokat ad vissza, amelyek leírják, hogy a kép valószínűsége egy bizonyos osztályba tartozik.
Most, hogy ismerjük a problémát, valamint a bemeneteket és a kimeneteket, gondolkodjunk el a megközelítésen. Azt szeretnénk, ha a számítógép képes lenne megkülönböztetni az összes neki megadott képet, és kiderítené azokat az egyedi jellemzőket, amelyek egy személyt alkotnak.

Az architektúra hasonlít az emberi agyban lévő neuronok összekapcsolódási mintázatára, és inspirálta a vizuális kéreg

szerveződése.

KÖVETKEZTETÉS

Összegzésként megemlíthető, hogy a projektet lelkiismeretes erőfeszítéssel dolgoztam ki. A legtöbb követelmény teljesült az elvárt színvonalon. Az MI kulcsfontosságú képesség a legtöbb számítógépes és robotikai látórendszer számára.

Bár az utóbbi években jelentős előrelépés figyelhető meg, és néhány jelenlegi technika már része a fogyasztói elektronikának (például az arcfelismerés az autófókuszhoz okostelefonokban) vagy integrálódott az asszisztált vezetési technológiákba, még mindig messze vagyunk az emberi szintű teljesítmény elérésétől, különösen a nyílt-világ tanulás tekintetében.

Meg kell jegyezni, hogy az objektumfelismerést sok olyan területen nem használták még ki, ahol nagy segítséget nyújthatna. Ahogy a mobil robotok és általában az autonóm gépek egyre szélesebb körben terjednek (például drónok, négyrotoros drónok és hamarosan szolgáltató robotok), az objektumfelismerő rendszerek iránti igény egyre nagyobb jelentőséggel bír.

Végül figyelembe kell vennünk, hogy szükségünk lesz objektumfelismerő rendszerekre olyan nanorobotok vagy robotok számára, amelyek az ember által még nem látott területeket kutatnak fel, például a tenger mély részeit vagy más bolygókat, és ezeknek a felismerő rendszereknek képesnek kell lenniük új objektumosztályok megtanulására, amikor találkoznak velük. Ilyen esetekben a valós idejű, nyílt-világ tanulási képesség kulcsfontosságú lesz.

Egy pontos és hatékony objektumfelismerő rendszer került kifejlesztésre, amely eléri a jelenlegi csúcstechnológiás rendszerekhez hasonló mutatókat. Ez felhasználható valós idejű alkalmazásokban, ahol objektumfelismerésre van szükség az előfeldolgozási folyamatban.

Fontos fejlesztési lehetőség lenne a rendszer betanítása videószekvenciákon, hogy használható legyen követési alkalmazásokban. Egy időbeli konzisztens hálózat hozzáadása lehetővé tenné a simább felismerést, és optimalizáltabb lenne a képkockánkénti felismerésnél.

A RENDSZER JÖVŐBELI KILÁTÁSAI

A projekt célja annak biztosítása, hogy a termék életképes legyen a mai, kihívásokkal teli világban. Itt minden funkciót megvalósítottak és teszteltek. Jelenleg a rendszer csak korlátozott számú klienskamerával működik. A terv az, hogy kibővítik több kamera adatfolyamának egyszerre történő feldolgozására, így javítható a hatékonyság. Lehetséges egy SaaS megoldás létrehozása, amely figyelmeztető rendszert biztosít integrált hardveres megoldással. Olyan rendszer, amely pontosabb eredményeket nyújt gyorsabb észleléssel.

- 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**