**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: IoT система за отдалечен достъп, контрол и анализ на устройства

|  |  |
| --- | --- |
| Дипломант: | Научен ръководител: |
| *Огнян Барух* | *Атанас Атанасов* |

СОФИЯ

2 0 2 1

СЪДЪРЖАНИЕ

**УВОД ........................................................................................................................... 4**

**ГЛАВА I. МЕТОДИ И ТЕХНОЛОГИИ НА РЕАЛИЗАЦИЯ. ПРОУЧВАТЕЛНА ЧАСТ.**

1.1 Лицево засичане

1.2 Лицево верифициране

1.3 NumPy

1.4 MQTT

1.5 Дигитални близнаци

1.6 Bosch IoT Suite

1.7 Bosch IoT Hub

1.8 Bosch IoT Things

1.9 Bosch IoT Edge Services

1.10 Умни устройства

1.11 ONVIF

1.12 Z-Wave

1.13 Flutter

1.14 Firebase

**ГЛАВА II. ИЗИСКВАНИЯ КЪМ ПРИЛОЖЕНИЕТО. СТРУКТУРА НА СИСТЕМАТА ОТ УСТРОЙСТВА.**

2.1 Изисквания към приложението

2.1.1 Изисквания към модела за лицево верифициране

2.1.2 Изисквания към системата от свързани устройства

2.1.3 Изисквания към мобилното приложение

2.2 Съображения за избор на програмни средства

2.2.1 Алгоритми за машинно самообучение

2.2.2 Устройства в системата

2.3 Основни алгоритми за лицево разпознаване

2.3.1 Алгоритъм за лицево засичане

2.3.2 Алгоритъм за лицево верифициране

УВОД

През последните години терминът „IoT“ (Internet of Things – Интернет на нещата) придобива огромна популярност, тъй като такива решения улесняват нашето ежедневие. Вече съществуват решения, които ни позволяват по-лесно да контролираме нашите домове, коли, градини и други. Интернет на нещата навлиза и в проекти, свързани със сигурността, измерването и предаването на данни, както и с идентификация на лице, глас и пръстов отпечатък.

Машинното самообучение също става основен фактор в множество софтуерни и хардуерни решения през последните години. Развитието на машините позволява да бъдат тренирани по-сложни алгоритми с цел постигане на по-точни резултати. Машинното самообучение навлиза в света на технологиите все повече и повече, което ни позволява да заменим човешките усилия с работа на машини. То е използвано както за лични проекти, така и за глобални решения с цел подобряване на услуги като градски транспорт, имейл, персонални асистенти, преводи и други.

Целта на настоящата дипломна работа е да бъде изградена система, която комбинира две от най-широко използваните технологии и която създава по-лесен и по-сигурен начин за влизане в нашия дом, както и начин да следим кой е влязъл или се опитва да влезне в нашия дом. Основните компоненти на дипломната работа са камера, която засича обекти, и алгоритъм за машинно самообучение, който проверява дали засечените обекти са хора и дали тези хора са в списъкa с позволени хора, за да отключи вратата и да ги пусне в техния дом.

**ГЛАВА I. МЕТОДИ И ТЕХНОЛОГИИ НА РЕАЛИЗАЦИЯ. ПРОУЧВАТЕЛНА ЧАСТ.**

**1.1 Лицево засичане**

**1.1.1 Определение**

Лицевото засичане е технология, използвана в множество решения за идентифициране на човешки лица в изображения или във видео връзки на живо. То произлиза от засичането на обекти, като в случая търсените обекти са човешки лица. Един алгоритъм за лицево засичане се тренира върху множество изображения на различни човешки лица – различни полове, различни раси, различни черти на лицето. При изпълняване на алгоритъма той обхожда пиксел по пиксел даденото изображение и сравнява пикселите със съществуващите снимки на лица, за да открие приликите между тях. В зависимост от приликите между потенциалното засечено лице и снимките на познатите лица, алгоритъмът „взима решение“ дали даденият обект е лице или не, като резултатът е процентът сигурност, че разглежданият обект е лице. Съществуват алгоритми, които се тренират по време на изпълнение – при засечено лице алгоритъмът го добавя към множеството от познати лица. По този начин всяко следващо разпознато лице довежда до по-точни резултати.

**1.1.2 Приложения**

Лицевото засичане намира множество приложения в различни сфери –системи за лицево разпознаване, автоматичен фокус във фотографията, разпознаване на емоции, както и четене по устни. С развиването на алгоритмите и моделите за лицево засичане, то набира все повече и повече популярност в различни решения.

**1.2 Лицево верифициране**

**1.2.1 Определение**

Лицевото верифициране използва резултатите от лицевото засичане, за да сравни непознато лице с познати такива. За разлика от лицевото разпознаване, което отговаря на въпроса: „Чие е това лице?“, лицевото верифициране отговаря на въпроса: „Това ли е правилното лице?“. Един алгоритъм за лицево верифициране сравнява характерните черти на непознатото и познатите лица – разстоянието между зениците на двете очи, разстоянието между външния и вътрешния ъгъл на всяко око, разстоянието между носа и устата и други. Тази информация се записва като вектор със 128 измерения, а впоследствие се сравняват данните от всички измерения, за да се верифицира непознатото лице.

**1.2.2 Приложения**

Лицевото верифициране намира широко приложение в системи, свързани със сигурността, като това могат да бъдат както лични системи, така и публични такива. Лицевото верифициране се използва за контрол на достъпа до различни помещения, както и за отключване на нашите смартфони.

**1.3 NumPy**

NumPy е най-използваната библиотека за работа с многоизмерни масиви, написана на Python. Библиотеката предлага множество функционалности, свързани с създаване и обработване на масиви с много измерения. Освен това NumPy има огромна документация, която много улеснява използването на функционалностите, коио предлага. Библиотеката позволява запазването на такива масиви в специални файлове с разширение .npz, което представлява компресиран многоизмерен масив. Друга широко използвана функционалност е че масивът, използван в NumPy, е обект, който съдържа и друга лесно достъпна информация като брой на измеренията, големина на всяко измерение и други.

**1.4 MQTT**

**1.4.1 Определение**

MQTT представлява стандартен протокол за комуникация, който намира широко приложение в проектите, свързани с „Интернет на нещата“.

**1.4.2 Принцип на работа**

MQTT работи на принципа на публикуване и абониране за съобщения, което позволява едно съобщение да достига до множество крайни устройства. Всяко устройство може да се абонира за даден канал и да получава всички съобщения в този канал.

**1.4.3 Защо MQTT е толкова разпространен?**

Най-силните страни на MQTT, е че има много висока скорост и много ниска консумация. Една MQTT заявка се изпълнява над 10 пъти по-бързо от една HTTP заявка. Освен това този протокол е с малко, но все пак достатъчни възможности, за да се постигнат необходимите функционалности за работа с устройства. MQTT заявките са с много малък размер, което позволява на устройства с ограничени възможности да се възползват от услугите на този протокол. MQTT позволява едно съобщение да бъде получено от безброй много абонати, докато при HTTP ще трябва да се изпращат отделни съобщения до всеки получател.

**1.5 Дигитални близнаци**

Дигитален близнак на едно устройство представлява съвкупността от информация, описваща съществуващо устройство. Терминът добива популярност в началото на XIX век с навлизането в ерата на „Интернет на нещата“. Дигиталният близнак може да се разглежда като база от данни за едно устройства, съдържаща само текущата информация. Ако устройството има връзка със своя дигитален близнак, то тогава то може да сигнализира за промяна в неговата информация, която да се отбележи в дигиталния близнак. Съществуването на дигитални репрезентации позволява на хората да следят от разстояние информацията и работата на всяко устройство.

**1.6 Bosch IoT Suite**

Bosch IoT Suite представлява платформа за свързване на устройства, тяхното контролиране, както и обработването на данните, които изкарва всяко устройство. Услугите в Bosch IoT Suite са разделени спрямо функционалността си – свързване на устройства, дигитална репрезентация на устройства, управление на устройства и подновяване на софтуера върху устройства. Към днешна дата над 15 милиона устройства са свързани към Bosch IoT Suite. Също така над 250 международни проекта се възползват от услугите, предлагани от тази платформа.

**1.7 Bosch IoT Hub**

Bosch IoT Hub представлява облачна услуга, която свързва устройства с приложения, които ги управляват. Чрез тази услуга разработчиците на софтуер получават бърз и лесен начин да свържат най-различни устройства към своите приложения. Тя поддържа няколко протокола за комуникация, най-известните от които са HTTP, MQTT и AMQP. Свързването на устройства чрез Bosch IoT Hub е лесно и най-вече сигурно, тъй като се разчита на информация, достъпна само за притежателя на устройствата.

**1.8 Bosch IoT Things**

Bosch IoT Things представлява облачна услуга, която позволява на всеки потребител да съхранява и обработва дигителните близнаци на устройствата си. Това може да се случва през сайта на Bosch IoT Things, през HTTP API или посредством код с HTTP заявки и MQTT съобщения. Функционалностите на услугата са регистриране и обработване на дигитални близнаци, контролиране на достъпа до тях, както и търсене на дигитални близнаци по динамични стойности. Bosch IoT Things е част от Bosch IoT Hub, което позволява на всички услуги в рамките на Bosch IoT Suite да обменят информация с цел подобраване на качеството на услугата и улесняването на процеса на контрол на дигиталните близнаци за потребителите.

**1.9 Bosch IoT Edge Services**

Bosch IoT Edge Services представлява връзката между устройствата и тяхната дигитална репрезентация в Bosch IoT Suite. Освен връзката, Bosch IoT Edge Services осъществява и комуникацията между едно устройство и Bosch IoT Suite посредством протокола MQTT. Те позволяват на устройства, работещи върху всякакви протоколи, да се свържат със своя дигитален близнак. По този начин всяко устройство, което работи с Bluetooth, Z-Wave, ONVIF, ZigBee и други може по много бърз и лесен начин да се свърже и управлява чрез Bosch IoT Suite.

**1.10 Умни устройства**

През последните години всички устройства, използвани в ежедневието, биват заменяни от умни устройства. Умни устройства са устройства или сензори, които са свързани към други устройства или към други машини и които могат да се използват интерактивно. Съществуват множество протоколи за комуникация между отделни устройства, като най-известните са Bluetooth, Wi-Fi, Z-Wave, ZigBee, ONVIF и други. Такива устройства биват смартфони, умни автомобили, умни часовници, умни камера, умни хладилници, умни ключалки и други. Появата на умните устройства улеснява много човешкото ежедневие и тяхната употреба става необходима във все повече индустрии.

**1.11 Z-Wave**

Z-Wave представлява протокол за безжична комуникация, като се използва предимно в решения, свързани с „Интернет на нещата“. Най-често протоколът се използва, за да се осъществи комуникацията с устройства в дома като автоматични ключалки, автоматични прозорци, автоматични врати за гаражи, термостати и други. Вълните достигат до 100 метра, което позволява пълен контрол в дома и градината. Всяко устройство, поддържащо Z-Wave, може да бъде контролирано през Wi-Fi, което улеснява работата с тези устройства.

**1.12 ONVIF**

ONVIF е глобална организация, която цели да стандартизира работата с базираните на IP устройства, свързани със сигурността. Създаденият стандарт описва връзката и комуникацията между различни устройства за видеонаблюдение и централни машини, които следят тяхната работа и потока, който записват. Друга основна цел на организацията е да се осигури съвместимостта между продукти на различни компании, за да се постигне по-сигурна и стандартна система.

**1.13 Flutter**

Flutter е пакет за разработка на софтуер, създаден от Google. Първата версия излиза 2017 г. и Flutter за кратко време става една от най-използваните платформи за създаване на мобилни приложения. Във Flutter се използва обектно-оринтираният езикът за програмиране Dart. Основната единица в едно Flutter приложение е т.нар „widget“, което представлява клас за даден компонент в приложението (бутон, параграф, изображение), който има отделни параметри и функционалности. Едно от най-големите предимства на Flutter е че позволява компилиране за iOS и Android, което означава, че може да се създаде едно приложение на един език за двете операционни системи вместо две отделни приложения на два отделни езика за програмиране. Освен това съществуват множество курсове и форуми за Flutter, което помага на по-начинаещи да направят своите първи стъпки в разработката на мобилни приложения

**1.14 Firebase**

Firebase е платформа, разработена от Google за създаване на мобилни и уеб приложения. Платформата съдържа 18 продукта, които улесняват разработката на приложения. Всеки продукт има отделна документация, в която е описан процесът на инсталиране и функционалностите, които предлага. Най-често процесът на инсталиране изисква няколко реда код, което помага на разработчиците да се фокусират върху функционалностите на приложението, което създават.

**1.14.1 Firebase Authentication**

В почти всички приложения се изисква имплементирането на някакъв вид автентикация. Firebase Authentication позволява да се вградят над 10 вида автентикация в приложението. Firebase позволява изграждането на база от данни, където се пази потребителските имена и пароли за използване от Firebase Authentication. Този продукт поддържа сесии, което позволява по всяко време да се знае дали някой потребител е влезнал в своя профил.

**ГЛАВА II. ИЗИСКВАНИЯ КЪМ ПРИЛОЖЕНИЕТО. СТРУКТУРА НА СИСТЕМАТА ОТ УСТРОЙСТВА.**

**2.1 Изисквания към приложението**

**2.1.1 Изисквания към модела за лицево верифициране**

• Моделът за лицево верифициране да включва програма, която да позволи на потребителя да отвори камерата на машината си и да снима изображения на потребителя, с които да се тренира алгоритъма.

• Моделът за лицево верифициране да включва модел за лицево засичане, който да засича всички лица на хора в едно изображение и да ги запазва в масив.

• Процесът за лицево верифициране да прекъсва изпълнението си, ако няма засечени лица в изображението.

• Моделът за лицево верифициране да включва програма, която да тренира алгоритъма, използвайки заснетите снимки за познати лица.

• Моделът за лицево верифициране да включва програма, която да приема изображение чрез линк или чрез файл с възможни разширения *.jpg*, *.png* и *.jpeg*. Програмата трябва да засече всички лица в изображението и да провери кои лица са на познати хора. Алгоритъмът трябва да връща масив с познатите засечени лица или „Unknown“, ако сред засечените лица няма нито едно познато.

**2.1.2 Изисквания към системата от свързани устройства**

• Системата да включва Raspberry Pi, което да има инсталирани Bosch IoT Edge Services, за да може да се свържат устройствата в система към това Raspberry Pi.

• Системата да включва камера, която да е свързана към Raspberry Pi посредством Bosch IoT Edge Services. Тази камера трябва да засича движение, да прави снимка и да изпраща линк на снимката към Raspberry Pi, за да се верифицират човешките лица, ако има такива.

• Системата да включва ключалка, която да е свързана към Raspberry Pi посредтсвом Bosch IoT Edge Services. Ключалката да се отключва за 5 секунди, ако моделът за лицево верифициране разпознае лице. Съобщенията към ключалката да се изпращат и получават посредством Z-Wave контролер, свързан към Raspberry Pi.

**2.1.3. Изисквания към мобилното приложение**

• Мобилното приложение да може да изпраща HTTP заявки към Bosch IoT Suite с цел достъп, четене и редактиране на дигиталните близнаци на устройствата.

**2.2 Съображения за избор на програмни средства**

**2.2.1 Алгоритми за машинно самообучение**

За реализация на алгоритмите за лицево засичане и лицево верифициране се използва скриптовият език Python. Има много причини, защо Python е най-използваният език за реализиране на алгоритми за машинно самообучение. Този език е сред първите 10 на най-използваните в програмирането. Освен това Python е скриптов език и за изпълнението на програма, написана на Python, не се изискват допълнителни процеси като компилация, както при компилируеми езици като C и Java. Друго предимство на Python е че може да се поддържа от всякакви операционни системи, дори и на микроконтролери. Освен това съществуват множество библиотеки за машинно самообучение, написани на Python, което позволява на разработчиците на код да създават всякакви приложение, използвайки Python и съществуващите библиотеки.

**2.2.2 Устройства в системата**

Системата от свързани устройства включва камера и ключалка, свързани към Raspberry Pi посредством Bosch IoT Edge Services. Използва се този микроконтролер, тъй като той разполага с достатъчно добър процесор, за да поддържа връзките към устройствата, връзката към облачната услуга Bosch IoT Hub и да изпълнява тренирания алгоритъм за лицево верифициране.

Камерата, използвана за реализацията на дипломната работа, е Bosch FLEXIDOME IP 4000i. Камерата използва ONVIF за комуникация, което позволява лесното й свързване и контролиране от друга машина. Тя съдържа сензор за движение и сигнализира, когато има засечен обект от сензора. Камерата може да изпраща линк към изображение и линк към видео поток, които могат да се достъпят с автентикация с потребителско име и парола.

Ключалката, използвана за реализацията на дипломната работа, е Danalock V3 Smartlock. Тя може да бъде свързана към смартфон или към смарт часовник и да се контролира от съответното устройство. Ключалката използва протокола Z-Wave за комуникация. Този протокол позволява лесното ѝ свързване към Bosch IoT Suite, както и лесното задаване на команди към нея. Освен това тя може да бъде контролирана през Wi-Fi, а създателите на тази ключалка са разработили и мобилно приложение, което може да се свърже с нея и да я контролира.

**2.3 Основни алгоритми за лицево засичане и лицево верифициране**

**2.3.1 Алгоритми за лицево засичане**

Съществуват много алгоритми, създадени и оптимизирани специално за целта да засичат лица в снимки. Според много източници MTCNN е един от най-добрите модели откъм ефективност и постигнати резултати. Този модел се състои от 3 конволюционни невронни мрежи, като освен лица в снимка, те могат и да открият координатите на характеристични точки на лице на човек – двете очи, носа и двата края на устата. Пример за резултат от изпълнението на алгоритъма за лицево засичане:

{

"box": [282, 414, 479, 583],  
 "confidence": 0.9999996423721313,  
 "keypoints": {  
 "left\_eye": (405, 667),  
 "right\_eye": (611, 596),  
 "nose": (526, 732),  
 "mouth\_left": (476, 874),  
 "mouth\_right": (672, 814)

}  
}

Резултатът от изпълнението на алгоритъма e обект от тип JSON. Стойността на ключът „box“ е масив с 4 елемента, като първите два определят горния ляв ъгъл на правоъгълника, в който се намира лицето, а вторите два определят дължината и широчината на правоъгълника, откъдето можем да се намери долния десен ъгъл правоъгълника. По този начин се знае в рамките на кои пиксели алгоритъмът е засякъл лице. Стойността на ключът „confidence“ е между 0 и 1 и показва колко е „сигурен“ алгоритъмът, че обектът в правоъгълника е лице. Тази стойност е важна, тъй като алгоритъмът може да обрисува повече от един правоъгълник около едно и също лице и се използва, за да се определи кой правоъгълник е най-точен. Стойността на ключът „keypoints“ е обект от тип JSON, който съдържа стойности за координатите на характеристичните точки на човешкото лице. Стойностите на „left\_eye“ и „right\_eye“ показват координатите на зениците съответно на лявото и дясното око, „nose“ показва координатите на върха на носа, а „mouth\_left“ и „mouth\_right“ са координатите съответно на левия и десния връх на устата. Ако в изображението алгоритъмът засече повече от едно човешки лица, то тогава резултатът ще бъде масив с големина броят на засечените лица, а всеки елемент ще съдържа стойностите за отделно лице.

**2.3.2 Алгоритми за лицево верифициране**

Както при лицевото засичане, така и при лицевото верифициране невронните мрежи водят до най-точни резултати. Един от най-известните модели за лицево верифициране е VGG-Face, разработен от специалисти от Оксфордски университет. Алгоритъмът за лицево верифициране приема изображение, съдържащо лице и създава т.нар „*face embedding*“, което представлява вектор в 128 измерения, като всяко измерения представлява различна част от този *face embedding*. Този вектор пази информация за много разстояния между характеристичните части на човешкото лице – разстоянието между зениците на двете очи, разстоянието между вътрешния и външния ъгъл на всяко око и други.

При трениране на алгоритъма за лицево верифициране се използва метода „*triplet loss*“. За неговата реализация се използват три изображения на човешки лица – базова снимка, вярна снимка и грешна снимка. Базовата снимка е снимката, върху която се тренира, вярната снимка е на същото лице, а грешната снимка е на друго лице. По този начин алгоритъмът се „учи“ как да разграничава отделните лица по стойностите от различните измерения на вектора *face embedding*. Функцията на *triplet loss* изглежда така:

В тази формула е базовата снимка, е вярната снимка, а е грешната. Функцията представлява функцията за извличане на *face embedding* от изображение на лице, a е допустимата граница между вярната и грешната снимка. Тази граница се въвежда, за да има по-голямо разграничение между отделни лица и да може да се разграничават хора, които си приличат в лицето. За трениране на алгоритъма за лицево верифициране се използва следната формула:

Резултатът от тази формула е сумата от резултатите на функцията с всички снимки от колекцията от снимки за трениране. Този резултат се запазва във файл с разширениe *.npz* и впоследствие се използва при изпълнението на алгоритъма върху тестови лица.

**2.4 Структура на Flutter мобилно приложение**

Примерна файлова структура на едно Flutter приложение, което е разположено в папката „flutter-application“:

flutter-application /

├── android/

│ ├── app/

│ ├── build.gradle

│ └── gradle/

├── build/

├── fonts/

├── ios/

│ ├── Flutter/

│ ├── Podfile

│ ├── Pods/

│ └── Runner/

├── lib/

│ ├── main.dart

│ ├── models/

│ ├── screens/

│ └── services/

├── pubspec.yaml

└── README.md

В папката „android” е поместено приложението за устройства с Android операционна система. В подпапката „app“ се намира функционалния код на мобилното приложение, а в подпапката „gradle“ се намира кода за сглобяване на приложението. Файлът „build.gradle“ съдържа в себе си константи, които се използват при сглобяването на приложението.

В папката „build“ са поместени сглобените приложения за Android OS и iOS, както и сорс код на изтеглените външни библиотеки.

В папката „fonts“ се поставят семействата от шрифтове, използвани в приложението.

В папката „ios“ се намира приложението за устройства с операционната система iOS. В подпапката „Flutter“ е поместена логиката за сглобяване на мобилното приложение. В подпапката „Pods“ се намират необходимите външни библиотеки за сглобяване на приложението, а в подпапката „Runner” се пазят конфигурационни файлове за стартирането на приложението върху реално устройство. Във файлът „Podfile“ се пазят версиите и имената на необходимите външни библиотеки, които при сглобяване се изтелгят и се поставят в папката „Pods”.

В папката „lib“ е поместен основния функционален код на приложението, написан на програмния език Dart. Там се намира кодът, написан от разработчикът на едно мобилно приложение. При сглобяване на приложението кодът, написан на Dart, се „превежда“ на Swift (програмният език за разработване на приложения за iOS) и Kotlin (програмният език за разработване на приложения за Android OS), за да могат да се подготвят за стартиране приложенията за двете операционни системи. Подпапката „models“ съдържа различни помощни класове, които се използват в приложението. В подпапката „screens“ са поместени класовете за различните екрани в мобилното приложение. В подпапката „services“ се намират различни помощни класове, свързани с функционалностите на приложението. Файлът „main.dart“ е файлът, който съдържа началната страница на приложението и който се изпълнява при неговото стартиране.

**ГЛАВА III. ПРОГРАМНА РЕАЛИЗАЦИЯ НА СИСТЕМА ОТ СВЪРЗАНИ УСТРОЙСТВА.**