FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Mapování rastrové grafiky na lidskou tvář ve smíšené realitě
Student:	Adam Pončák
Vedoucí:	Ing. Petr Pauš, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Počítačová grafika
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Cílem práce je vytvoření systému pro tvorbu rastrové grafiky, která bude následně mapována na tvář ve vstupním videu a vizualizována v rozšířené realitě (Augmented Reality). Vytvořená rastrová grafika tedy bude v reálném čase projektována na obličej ve videu.

Zadání:
1) Analyzujte možnosti knihovny OpenCV pro práci s videem, pro detekci lidské tváře a pro vizualizaci ve smíšené realitě.
2) Na základě analýzy vyberte vhodné funkcionality OpenCV.
3) Analyzujte dostupné frameworky pro tvorbu aplikace.
4) Proveďte návrh systému pro mapování grafiky na lidskou tvář a návrh prototypu aplikace ve zvoleném frameworku.
5) Implementujte celý prototyp dle návrhu.
6) Na prototypu proveďte vhodné testování.

Elektronicky schválil/a Ing. Radek Richtr, Ph.D. dne 17. února 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Mapovanie rastrovej grafiky na ľudskú tvár v zmiešanej realite

***Adam Ponč*ák**

Katedra softwarového inženýrství
Vedúci práce: Ing. Petr Pauš, Ph.D.

13. mája 2021

Poďakovanie

Chcem sa poďakovať svojmu školiteľovi Ing. Petr Pauš, Ph.D. za cenné rady a odbornú pomoc, ktoré mi poskytol pri písaní bakalárskej práce. Taktiež chcem poďakovať rodinným príslušníkom za neoblomnú podporu a trpezlivosť.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 13. mája 2021 . . . . . .

Odkaz na túto prácu

Pončák, Adam. *Mapovanie rastrovej grafiky na ľudskú tvár v zmiešanej realite*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Abstrakt

Táto práca sa zaoberá problematikou vytvárania jednoduchých tvárových filtrov za pomoci knižnice OpenCV. Cieľom je vytvoriť prototyp aplikácie schopnej tvorby a mapovania rastrovej grafiky na orientačné body tváre v zmiešanej realite. Na detekciu tvárí bola použitá *Haar-like casscade* detekcia a na tvárové črty algoritmus regresných lokálnych binárnych príznakov. Výsledkom je funkčná počítačová aplikácia pre systém Windows z jednoduchým užívateľským rozhraním. Umožňuje užívateľovi intuitívnu tvorbu jednoduchej grafiky, ktorú aplikácia následne pretransformuje a mapuje na črty tváre detekovanej vo vstupnom videu webkamery.

Klíčová slova zmiešaná realita, počítačové videnie, OpenCV, detekcia tváre, detekcia tvárových čŕt, tvárové filtre, webkamera

Abstract

This work deals with the issue of creating simple face filters using the OpenCV library. The aim is to create a prototype application capable of creating and mapping raster graphics on the landmarks of the face in augmented reality. *Haar-like casscade* detection was used for face detection and a regression local binary features algorithm for facial landmarks. Currently, a Windows computer application with a simple user interface is fully functional. It provides the user with intuitive creation of simple graphics, which the application then transforms and maps onto facial features detected in the webcam's input video.

Keywords augmented reality, computer vision, OpenCV, face detection, face feature detection, face filters, webcam

Obsah

Zoznam obrázkov

Úvod

Zmiešaná realita, ako odvetvie počítačovej grafiky, sa za posledné roky značne rozmohla medzi bežných užívateľov. Jedným z najpopulárnejších oblastí zmiešanej reality sa stali tvárové filtre, ktoré sú dnes obľúbené hlavne zásluhou sociálnych sietí.

Práca reflektuje potreby súčastného spotrebiteľa - laika a umožní mu intuitívnu tvorbu tvárových filtrov bez akýchkoľvek predchádzajúcich znalostí o problematike. Výsledok tejto práce dopomôže bežným užívateľom tvoriť a testovať jednoduché tvárové filtre, a zároveň zdokonaľovať ich tvorivosť a umelecké zručnosti.

Kvôli zmenám spôsobených pandémiou *COVID-19* sa webkamery stali pre jedincov potrebnou súčasťou každodenného života. Väčšina pracovnej činnosti sa začala koncentrovať v digitálnej podobe. To, spolu s faktom, že považujem zmiešanú realitu za príťažlivú a stále neprebádanú oblasť, ma ovplyvnilo k voľbe témy tejto práce.

V práci sa zaoberám analýzou, návrhom a implementáciou počítačového programu umožňujúceho vytvoriť rastrový obrázok, ktorý bude následne mapovaný na tvár detekovanú vo výstupe webkamery. Rastrové obrázky budú tvorené na prispôsobenej kresliacej ploche programu, s následnou možnosťou ich uloženia.

Teoretická časť bude pozostávať z analýzy knižnice OpenCV a technológií pre vývoj aplikácie za jej pomoci. Pokračuje časťami o analýze zmiešanej reality, detekcie tvárí a detekcie tvárových čŕt. Je ukočená časťou zameranou na analýzu požadovaných funkcionalít OpenCV pre prácu s rastrovou grafikou.

Praktická časť sa venuje návrhu programu, ktorý je založený na analýze z teoretickej časti. Nasleduje implementácia popisujúca fungovanie a ovládanie prototypu. Posledná časť tejto kapitoly je zameraná na predstavenie výsledkov testovaní funkčnosti prototypu.

Kapitola 1

Cieľ práce

Hlavným cieľom práce je návrh a tvorba počítačového programu umožňujúceho užívateľovi tvorbu rastrového obrázku, ktorý bude následne mapovaný na tvár detekovanú vo vstupnom videu webkamery. Na vytvorenom prototype je následne cieľom uskutočniť vhodné testovania.

Cieľom teoretickej časti bakalárskej práce je analýza funkcionalít knižnice OpenCV pre prácu s videom webkamery, detekciu ľudských tváre, detekciu tvárových čŕt a vizualizáciu rastrovej grafiky v rozšírenej realite. Ďalším cieľom teoretickej časti je analýza vývojového prostredia a podporných *frameworkov* pre tvorbu samotnej aplikácie. Získané znalosti budú základom pre návrh a implementáciu aplikácie.

Praktická časť má za cieľ výber vhodnej implementačnej platformy, výber vhodných funkcionalít OpenCV a návrh užívateľského rozhrania programu. Program bude postupne implementovaný v súlade s navrhnutým rozhraním. Hlavné časti programu, ako je tvárový detektor, detektor tvárových čŕt, mapovanie rastrov či kresliace prostredie, budú implementované na základe zvolených funkcionalít OpenCV.

Program má umožniť intuitívnu tvorbu rastrov (kreslenie) na prispôsobenej ploche, jednoduché ovládanie a jasnú vizualizáciu výsledku mapovania filtra na tvár.

Kapitola 2

Analýza

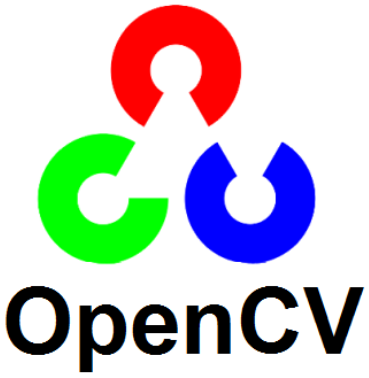
Kapitola je zameraná na analýzu funkcionalít OpenCV, ktoré budú nevyhnutné pre následný návrh a implementáciu finálneho programu. Ako prvá bude analyzovaná samotná knižnica OpenCV. Následne technológie tvorby aplikácie s danou knižnicou, medzi ktoré patrí platforma, programovací jazyk a *framework* pre tvorbu grafického užívateľského rozhrania. Budú rozobraná zmiešaná realita, tvárové detektory, detektory tvárových čŕt, základné princípy práce s videom a obrázkami v OpenCV a rôzne spôsoby úprav obrázkov potrebné na mapovanie rastru na oblasť tváre.

2.1 Knižnica OpenCV

Open Source Computer Vision Library je, ako už naznačuje anglický názov *open source* softvérová knižnica počítačové videnia a strojového učenia, vytvorená pre poskytovanie infraštruktúry pre aplikácie požadujúce počítačové videnie, ako aj pre akceleráciu princípov strojového vnímania v komerčných produktoch. Vďaka svojej licencii je výhodnou voľbou pre využitie v biznise, pretože povoľuje jednoduchú modifikáciu a využívanie kódu 2.1.

Hlavným zameraním knižnice je *real-time* počítačové videnie, čomu dopomáha aj pomerne nová podpora GPU akcelerácie, ktorá bola pridaná v roku 2011. Knižnica bola pôvodne vyvíjaná pod spoločnosťou *Intel*, ktorá predstavila jej alfa verziu v roku 2000 na konferencii *IEEE Conference on Computer Vision and Pattern Recognition*. Prvá verzia (1.0) odštartovala v roku 2006 a druhá (2.0) v roku 2009. Nasledovalo množstvo ďalších verzií, ktoré so sebou prinášali zmeny a vylepšenia, cez vylepšené C++ prostredie, po nové funkcie a funkcionality.

OpenCV je pôvodne napísaná v programovacom jazyku C++, mimo ten sú tiež dostupné rozhrania pre programovacie jazyky Python, Java a MATLAB. Je podporovaná operačnými systémami Windows, Linux, Android a Mac OS. Okrem toho sú takisto vyvinuté väzby pre viaceré programovacie jazyky, ako je napríklad *JavaScript*. Knižnica podporuje bezproblémovú prácu so *Stan-*



Obr. 2.1: OpenCV logo. [1]

dard Template Library (STL), čo je sada šablónových tried, ktoré zabezpečujú bežné programovacie dátové štruktúry a funkcie, ako sú listy, haldy, polia a podobne. [1, 2, 3]

Knižnica má modulárnu štruktúru, vďaka čomu je možné jednoducho pridávať nové moduly. Medzi hlavné z momentálne pridaných modulov patria:

Core functionality (core) – modul, v ktorom sú definované základné dátové štruktúry a jednoduché funkcie používané ostatnými modulmi. Príkladom definovanej štruktúry je viacrozmerné pole *Mat*, ktoré slúži ako základný kontajner pre rastrovú grafiku, alebo tiež *Scalar*, *Point* či *Range*.

Image Processing (imgproc) – modul zameraný na spracovávanie obrázkov, ktorý obsahuje „*histogramy, prevádzanie medzi farebnými priestormi, geometrické transformácie obrázkov (zmena veľkosti, afínna a projektívna transformácia), filtrovanie obrázkov a iné.*“ [4]

Video Analysis (video) – modul slúžiaci na analýzu videí, ktorý „*zahŕňa algoritmy na odhad pohybu, odčítanie pozadia či sledovanie objektov.*“ [4]

Camera Calibration and 3D Reconstruction (calib3d) – modul implementujúci základné algoritmy pre „*kalibráciu kamery, odhadu polohy objektov či prvky 3D rekonštrukcie.*“ [4]

2D Features Framework (features2d) – modul obsahujúci „*detektory význačných príznakov, deskriptory a porovnávače deskriptorov.*“ [4]

Object Detection (objdetect) – „*detekcia objektov a inštancií predefinovaných tried (napríklad tváre, oči, šálky, ľudia, autá, a iné).*“ [4]

High-level GUI (highgui) – „*ľahko použiteľné GUI prostredie, vhodné pre implementáciu jednoduchého užívateľského prostredia.*“ [4]

Video I/O (videoio) – „*ľahko použiteľné rozhranie pre videokodeky a na snímanie videa.*“ [4]

Knižnica taktiež obsahuje množstvo extra modulov, medzi ktoré patria moduly ako **„alphamat“** (*Alpha Matting*), **„barcode“** (metódy detekcie a

dekódovania čiarových kódov), **„dnn_objdetect“** (DNN využitá na detekciu objektov), **„img_hash“** (rôzne algoritmy hashovania obrázkov), **„videostab“** (stabilizácia videa) a mnoho ďalších, vrátane modulu **„face“** implementujúceho rôzne techniky rozpoznávania a analýzy tvárí, ako sú *Eigen*, *Fisher* či *Local Binary Pattern Histograms* (LBPH) metódy. Spomenutý modul taktiež zabezpečuje vyhľadávanie tvárových čŕt v snímke tváre, preto bude pre implementáciu potrebný. OpenCV extra moduly ale nie sú súčasťou oficiálneho vydania OpenCV. Sú však voľne dostupné na Github stránke projektu OpenCV, ako balík *opencv_contrib*. [5, 6]

2.2 Technológie

Analýza technológií pre tvorbu programu sa bude zaoberať zvolenou platformou, operačným systémom, programovacím jazykom a *frameworkom* pre samotnú tvorbu užívateľského prostredia aplikácie.

Aplikácie, zamerané na tvorbu rastrovej grafiky (kresliace editory), benefitujú z veľkosti pracovnej plochy. Záleží pri nich aj na dostupnosti periférií, ako je napríklad grafický tablet. Pre aplikácie pracujúce v zmiešanej realite je nesmierne dôležitý prístup ku kamere. Algoritmy počítačovej grafiky a strojového videnia sú taktiež náročné na hardvérové požiadavky.

Po zvážení rôznych zariadení, je stolový počítač zvolený za preferovanú platformu pre implementáciu programu. Smartfón síce spĺňa požiadavky na prístup ku kamere, no veľkosť pracovnej plochy a u niektorých aj dostupný hardvér, sú nedostatočné. PC zariadenia a notebooky spĺňajú všetky požiadavky, niekedy s výnimkou práve prístupu ku kamere. Tú je však možné jednoducho zaobstarať a pripojiť externe. Z dôvodu pandémie *COVID-19* je tiež v dnešnej dobe vlastniť počítač s webkamerou takmer normou.

Windows

„*Ide o operačný systém (OS) mierený na PC zariadenia, ktorý je vyvíjaný spoločnosťou Microsoft Corporation. Bol prvým operačným systémom, ponúkajúcim grafické užívateľské prostredie (GUI) pre počítače kompatibilné s IBM.*“[7] Windows je v dnešnej dobe najpopulárnejším operačným systémom pre osobné počítače. Prvá verzia Windows bola uvedená na trh v roku 1985. Od tej doby prešiel operačný systém množstvom zmien. Medzi najznámejšie vydania Windows patrí Windows 95 z roku 1995, Windows XP z roku 2001, alebo súčasná verzia Windows 10 z roku 2015.

Windows vychádza vo viacerých edíciách už od verzie Windows XP. Základ každej edície je rovnaký, no rôzne edície ponúkajú ďalšie funkcie za príplatok. Najbežnejšie edície sú *Windows Home* a *Windows Professional*. Windows je spustiteľný na štandardnom x86 hardvéri, ako sú Intel či AMD procesory. Keďže spoločnosť Microsoft poskytuje licenciu k ich OS viacerým výrobcom, spoločnosti ako je Dell, HP, Acer, Asus či Lenovo ponúkajú svoje zariadenia

priamo vybavené Windows OS. [7] Vzhľadom na rozľahlú klientelu a popularitu OS Windows, v spojení s predošlými skúsenosťami programovania aplikácií v tomto systéme, je zvolený Windows ako OS, v ktorom bude program vyvíjaný.

Programovacie jazyky

Ako už bolo spomenuté, OpenCV je postavená na programovacom jazyku *C++*, s naimplementovaným prostredím pre *Python*, *Java* alebo *MATLAB*. Následuje preto analýza týchto programovacích jazykov, za ktorou sa nachádza výber konkrétneho programovacieho jazyka využitého v tomto projekte.

C++

Objektovo orientovaný programovací jazyk, ktorý je známy tým, že je medzistupňom medzi nízko-úrovňovými a vysoko-úrovňovými programovacími jazykmi. Jeho tvorcom je *Bjarne Stroustrup*, ktorý ho vyvinul v *Bell Telephone Laboratories* v New Jersey. Jazyk bol spomenutý už v roku 1979, keď Bjarne pracoval na svojej doktorandskej práci. Jeho zámerom bolo vytvoriť flexibilný a dynamické programovací jazyk, ako je C, ktorý by bol doplnený o aktívnu kontrolu dátových typov, jednoduchú dedičnosť, triedy, predvolené argumenty funkcií a iné. Tak vznikol jazyk C++. [8]

Výhody C++:

1. **Prenosnosť** – Možnosť spustenia identického kódu na rôznych zariadeniach a v rôznych operačných systémoch nezávisle na platforme.

2. **Objektová orientácia** – „*Zahŕňa koncepty ako sú triedy, dedičnosť, polymorfizmus, abstrakcia dát a zapuzdrenie, ktoré umožňujú opätovné použitie kódu a robia program ešte spoľahlivejším.*“ [9] Ide o jednu z najvýznamnejších výhod jazyka. Dáta sú uložené a pracuje sa s nimi v podobe objektov, čo pomáha riešiť problémy z reálneho sveta. C++ bol vytvorený kombináciou funkcií nielen z jazyka C, ale aj Simula 67, prvého objektovo orientovaného programovacieho jazyka.

3. **Multi-paradigma** – Pojem „paradigma“ sa vzťahuje na štýl programovania a zahŕňa logiku, štruktúru a procedúru programu. Paradigmy C++ sú „Všeobecnosť“, „imperatívnosť“ a „objektová orientácia“, pričom všeobecné programovanie označuje použitie jedinej myšlienky na viac účelov a imperatívne programovanie označuje použitie príkazov, ktoré menia stav programu.

4. **Manipulácia na nízkej úrovni** – Vyplýva z reality, že C++ je úzko späté s procedurálny jazykom C, ktorý je blízko spojený so strojovým kódom.

5. **Správa pamäte** – C++ poskytuje v tomto ohľade programátorovi absolútnu kontrolu, čo môže byť považované aj ako prínos, aj ako slabina. Zvyšuje to totiž zodpovednosť užívateľa za správu pamäte.

6. **Veľká podpora komunity** – Rozľahlá komunita programátorov, vďaka ktorej existuje množstvo online kurzov a prednášok, platených či neplatených.

7. **Kompatibilita s programovací jazykom C** – Každý C program je spustiteľný aj v C++.

8. **Škálovateľnosť** – C++ je schopné fungovať s malým aj veľkým rozsahom dát.

Nevýhody C++:

1. **Ukazovatele** – Ide o pomerne komplikovaný koncept na pochopenie. Okrem toho, zaberajú množstvo pamäte a ich nesprávne využitie môže taktiež spôsobiť neobvyklé správanie systému, alebo jeho zlyhanie.

2. **Problémy s bezpečnosťou** – Najväčšími bezpečnostnými problémami jazyka C++ sú *friend* funkcie, globálne premenné a ukazovatele. Aj napriek tomu je však, vďaka využitiu objektovo orientovaného programovania bezpečnejší, ako iné programovacie jazyky, ktoré tento princíp nevyužívajú (napríklad C).

3. **Chýbajúci „Garbage Collector“** – Nakoľko C++ prenecháva užívateľovi plnú autonómnosť nad správou pamäte, chýba v ňom *garbage collector*, ktorý by automaticky odfiltrovával nepotrebné a nevyužívané dáta.

4. **Absencia vstavaných vlákien** – „*C++ nepodporuje žiadnu zabudovanú správu vlákien. Z pohľadu C++ ide o pomerne nový koncept, ktorý nebol pôvodne implementovaný do jazyka.*“ [9]

Python

Python je interpretovaný, univerzálny, interaktívny, objektovo-orientovaný, vysoko-úrovňový programovací jazyk, ktorý obsahuje moduly, správu výnimiek, dynamické dátové typovanie, triedy či *garbage collector*. Medzi hlavné paradigmy jazyka patrí objektovo-orientované, procedurálne a funkcionálne programovanie.

Navrhol ho Guido van Rossum v roku 1991. Bol značne inšpirovaný jeho predchádzajúcou prácou na programovacom jazyku ABC, pri ktorej získal skúsenosti v oblasti dizajnu a tvorby programovacích jazykov. Jedným z najväčších nedostatkov jazyka ABC bola podľa neho nedostatočná rozšíriteľnosť,

čo preňho bolo motiváciou k vytvoreniu jazyka Python. Jeho slovami:„*Pripadalo mi, že skriptovací jazyk so syntaxou podobnou jazyku ABC a prístupom k systémovým volaniam jazyka Amoeba by bol ideálnym. Uvedomil som si, že by bolo nezmyselné vytvoriť jazyk špecifický pre Amoeba, preto som sa rozhodol pre jazyk, ktorý by bol všeobecne rozšíriteľný.*“ [10]

Následne bol Python vyvíjaný organizáciou *Python Software Foundation*, čo je nezávislá, nezisková organizácia. Vlastní práva duševného vlastníctva k jazyku od jeho verzie 2.1. Hlavnou misiou tejto organizácie je propagovať, chrániť a zlepšovať programovací jazyk Python a podporovať a uľahčovať rast rozmanitej a medzinárodnej komunity programátorov Pythonu.

V roku 1994 bola uverejnená verzia 1.0, následne v roku 2000 verzia 2.0 a verzia 3.0 v roku 2008. V súčastnosti je poslednou stabilnou verziou verzia 3.9.5.

Oblasti, v ktorých sa využíva najčastejšie sú strojové učenie, umelá inteligencia, veda o dátach či *Internet of Things*(IoT). K týmto účelom ponúka množstvo knižníc, medzi ktoré patria knižnice ako *TensorFlow*, *Keras*, *SciPy*, *NumPy*, *Flask* alebo *django*. [10, 11, 12]

Výhody Python:

1. **Jednoduchý na čítanie, učenie i písanie** – Keďže je jeho syntax podobná anglickému jazyku, je uľahčené čítanie a porozumenie kódu. Je jednoduché s ním začať a naučiť sa v ňom pracovať, preto je odporúčaný začiatočníkom.

2. **Vylepšená produktivita** – Python je, vďaka jeho jednoduchosti jazyk zameraný na produktivitu. Umožňuje vývojárom zamerať sa na riešenie problému a nemusia teda tráviť príliš veľa času nad syntaxou alebo správaním programovacieho jazyka.

3. **Interpretovaný jazyk** – Python spúšťa kód riadok po riadku a ak narazí na chybu, zastaví beh programu a chybu nahlási, čím zjednodušuje *debugovanie* kódu.

4. **Dynamické typovanie** – Dátové typy sú do spustenia kódu v jazyku Python neznáme. Tie priradí automaticky počas behu programu. Programátor si preto nemusí robiť starosti s deklarovaním premenných a ich dátových typov.

5. **Zadarmo a Open-Source** – Keďže Python spadá pod licenciu open-source schválenú OSI, jeho použitie a distribúcia sú bezplatné. „*Stiahnutý zdrojový kód môžete upraviť a následne túto upravenú verziu distribuovať. To je užitočné pre organizácie, ktoré chcú upraviť niektoré špecifické správanie a použiť svoju verziu na vývoj.*“ [13]

6. **Široká podpora knižníc** – V štandardnej knižnici jazyka Python je možné nájsť takmer všetky funkcie potrebné pre väčšinu úloh. *Python package manager* (pip) taktiež uľahčuje vkladanie ďalších balíkov z *Python package indexu* (PyPi), ktorý zahŕňa viac ako 200 000 rôznych balíkov.

7. **Prenosnosť** – Za predpokladu, že nie sú vložené žiadne balíky závislé od platformy je možné program napísaný v jazyku Python spustiť na akejkoľvek platforme. [13]

Nevýhody Python:

1. **Pomalá rýchlosť** – Dynamickosť Pythonu je zodpovedná za jeho pomalú rýchlosť, nakoľko vykonáva kód riadok po riadku. Neodporúča sa preto Python používať v projektoch, ktorých dôležitým aspektom je rýchlosť.

2. **Nie je pamäťovo efektívny** – „*Programovací jazyk Python využíva veľké množstvo pamäte. To môže byť nevýhodou pri vytváraní aplikácií, v ktorých uprednostňujeme optimalizáciu pamäte.*“ [13]

3. **Slabý v *Mobile Computingu*** – Kvôli pamäťovej neefektívnosti a pomalému výpočtovému výkonu je Python primárne používaný v programovaní na strane servera.

4. **Prístup k databázam** – „*Prístupová vrstva databázy Pythonu je primitívna a v porovnaní s populárnymi technológiami ako JDBC a ODBC nedostatočne rozvinutá. Veľké podniky potrebujú plynulú interakciu zložitých starších údajov a Python sa tak v tejto oblasti používa len zriedka.*“ [13]

5. **Chyby počas behu** – Pretože je Python dynamicky typovaný jazyk, môže premenná na jednom riadku obsahovať reťazec a na ďalšom zasa celé číslo, čo môže pri nepozornosti programátora spôsobiť rôzne chyby. [13]

Java

Univerzálny, vysoko-úrovňový, objektovo-orientovaný programovací jazyk založený na triedach, ktorý bol navrhnutý pre prácu s menším množstvom implementačných závislostí. *„Je rýchla, bezpečná a spoľahlivá. Je široko používaná na vývoj Java aplikácií pre notebooky, dátové centrá, herné konzoly, vedecké superpočítače, mobilné telefóny atď.“* [14]

Java je programovací jazyk značne využívaný biznisom a priemyslom na výpočty na strane serverov. Hraje významnú úlohu aj v oblasti IoT. Pôvodne bol však vyvinutý ako vhodný, univerzálny jazyk pre stolné počítače. Bol

zverejnený v roku 1995 spolu s jazykmi ako Ruby, PHP a JavaScript. Tie chceli nahradiť jazyky Fortran, C či COBOL, ktoré boli v danej dobe primárnou voľbou.

Prenosnosť a veľké množstvo vstavaných knižníc ju takmer okamžite spopulárnili v oblasti *mainstream* programovania. Vďaka tejto možnosti je možné napísaný kód spustiť na takmer akejkoľvek platforme, čo z Javy vytvorilo výhodnú voľbu pre biznis aplikácie, ktoré potrebujú podporu viacerých platforiem.

Aj keď bola Java pôvodne známa svojou nízkou rýchlosťou, či už spúšťania alebo behu, dnes má miesto medzi najrýchlejšími programovacími jazykmi a zvláda spracovávať veľké množstvo dát, čo bolo podnetom *big data* revolúcie.

Vyvinula sa do programovacieho jazyka, ktorý je schopný vyriešiť takmer akýkoľvek programátorský problém. Významné bolo uvedenie verzie Java 8. Tá pridala dôležité funkcie prevzaté z funkčných programovacích idiómov, vďaka ktorým je kód kratší, spoľahlivejší a lepšie čitateľný. [14, 15]

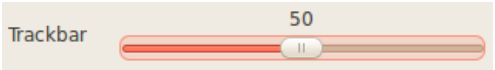
MATLAB

Programovací jazyk MATLAB je špecificky prispôsobený pre inžinierov a výskumníkov, založený na maticiach a maticových operáciach, čo umožňuje najprirodzenejšie vyjadrenie výpočtovej matematiky. Je využívaný v matematike, pre výpočty, analýzy, prieskumy a vizualizácie dát, vývoja algoritmov, tvorby modelov a aplikácií (vrátane GUI), simulácií, návrhov prototypov či vedeckú a inžiniersku grafiku. Významné oblasti využitia sú hlboké a strojové učenie, spracovanie a komunikácia signálov, spracovanie obrazu a videa, riadiace systémy, testovania a merania, výpočtové finančníctvo či výpočtová biológia.

MATLAB vynašiel programátor a matematik Cleve Moler v závislosti na jeho doktorandskej práci z roku 1960. Pôsobil ako profesor matematiky na *University of New Mexico*, kde začal vyvíjať MATLAB pre svojich študentov. Spolu so svojím jednorazovým konzultantom dizertačnej práce Georgom Forsytheom vypracoval pre MATLAB v roku 1967 základné programovanie založené na lineárnej algebre. Následne v roku 1971 vyvinul Fortran kód pre lineárne rovnice. Prvá verzia MATLABu bola finalizovaná koncom sedemdesiatych rokov a medzi verejnosť sa dostala vo februári 1979 v *Naval Postgraduate School* v Kalifornii. [16, 17, 18]

Záver

Pri výbere finálneho programovacieho jazyka hralo významnú úlohu uváženie predošlých skúseností. Tým sa pozornosť upriamila na jazyky C++ a Java. Po zvážení analyzovaných výhod a nevýhod bol zvolený programovací jazyk C++, ktorý je v porovnaní síce programátorsky menej prívetivý, no podstatne



Obr. 2.2: Ukážka *trackbaru* implementovaného v OpenCV module **highui**. Zobrazený *trackbar* pochádza z GUI knižnice GTK [23]

rýchlejší. Keďže ide o systém zameraný na prácu so snímkami v reálnom čase, rýchlosť zaberá dôležitú úlohu a je jej preto venovaná pozornosť.

GUI

OpenCV je plne uspôsobený využitiu vo veľkých projektoch vrámci rozsiahlych a komplikovaných UI *frameworkov*, ako sú *Qt*, *WinForms* alebo *Cocoa*. Môže byť tiež využívaný bez akéhokoľvek UI, čo je pre niektoré programy užitočnou vlastnosťou. Na tvorbu UI bez externých *frameworkov* ponúka OpenCV vlastný modul **„highui“**, ktorého hlavným cieľom je umožnenie jednoduchej vizualizácie výsledkov programu. Tento modul poskytuje nenáročné rozhranie pre vytváranie a manipuláciu okien, ktoré dokážu zobrazovať obrázky a „zapamätať si“ ich obsah (nie je potrebné riešiť prefarbenie obrazovky). Umožňuje do okien pridávať „trackbary“ 2.2, zvláda reagovať na jednoduché udalosti myši a klávesové príkazy. [19]

Tento modul závisí pri tvorbe užívateľského prostredia od rôznych GUI knižníc. Medzi podporované knižnice patrí *GTK* pre zariadenia Linux, *Win32UI* pre zariadenia Windows, *Cacao* pre zariadenia Mac. Tie sú pre zmienené operačné systémy predvolené. Alternatívou je *Qt*, podporované všetkými skôr zmienenými OS. Je ju nutné aktivovať pri kompilácií OpenCV, keďže nie je predvolenou možnosťou. Umožňuje naviac pridávanie tlačidiel, ponuky možností, nastavení, kontrolných panelov a ďalších prvkov do užívateľského prostredia, čo *Qt* činí podstatne silnejšou možnosťou pre tvorbu rozsiahlejšieho užívateľského prostredia.

Existuje taktiež možnosť implementácie vlastného užívateľského prostredie, čo je odporúčaná cesta pre väčšie, a hlavne komerčné projekty. Táto cesta je ale taktiež časovo najnáročnejšia. Vzhľadom na nízke požiadavky užívateľského prostredia programu, predošlé skúsenosti a časovú náročnosť, je zvolená knižnica *Win32UI*, ktorá je aj napriek faktu, že ponúka podstatne menšie množstvo UI prvkov a funkcionalít v porovnaní s ostatnými možnosťami, časovo najmenej zložitá na využitie a spĺňa všetky potrebné požiadavky. [20, 21, 22]

2.3 Zmiešaná realita

Zmiešanú realitu, alebo augmentovanú realitu (AR), je možné definovať ako obmenu virtuálneho prostredia, alebo virtuálnej reality (VR). Na rozdiel od

VR technológií, ktoré užívateľa plne ponoria do umelého sveta, brániac mu pozorovať svet reálny, AR technológie umožňujú vnímať reálny fyzický svet doplnený alebo pozmenený virtuálnymi objektmi. AR je zamerané na vylepšenie a doplnenie reálneho sveta, zatiaľčo VR na vytvorenie sveta vlastného. Zámerom je docielenie dojmu koexistencie virtuálneho objektu v priestore, v ktorom sa užívateľ nachádza. Možno ju považovať za medzistupeň medzi *teleprezenciou* (úplne skutočná) a VR (úplne syntetická).

Aby sa zabránilo viazaniu definície AR k AR špecifickým technológiám, je zmiešaná realita často definovaná ako systém, ktorý spĺňa následovné 3 podmienky:

1. Kombinuje virtuálny svet so svetom reálnym.

2. Je interaktívny, a to v reálnom čase.

3. Registruje virtuálne a reálne objekty v troj-rozmernom priestore.

AR ideálne alternuje reálne prostredie naprieč rôznymi zmyslovými vnemami, vrátane vizuálnych, sluchových, haptických či čuchových. *„Prekryté senzorické informácie môžu byť konštruktívne (aditívne k prirodzeného prostrediu) alebo deštruktívne (maskovanie prirodzeného prostredia).“* [24]

Pokročilé AR technológie, ako sú strojové učenie či počítačové videnie, sú schopné docieliť interaktívnosť a digitálnu manipulatívnosť informácií z okolia užívateľa. Zobrazované informácie, vložené do reálneho obrazu, môžu pochádzať ako z virtuálneho, tak z reálneho sveta. Príkladom zobrazovania informácie z reálneho sveta je vizualizácia reálnych nasnímaných a zmeraných dát (elektromagnetické rádiové vlny vizualizované v lokalite, kde sa podľa meraní nachádzajú vo fyzickom svete).

Medzi niektoré odvetvia ovplyvnené technológiami AR patria verejná bezpečnosť, zdravotníctvo, armáda, letectvo, videohry, turizmus, školstvo, šport a fitness, navigácie, *visual art*, priemyselná výroba, dizajn a plánovanie miest, architektúra, archeológia a v neposlednej rade zábava.

Head-up display (HUD) je technológia ktorá bola predchodcom dnešných AR zariadení. Ide o priesvitný displej, ktorý umožňuje zobrazovať dáta užívateľom bez potreby odvrátiť zrak. Pochádza z oblasti letectva, no rozšíril sa v oblasti armády (zbrane, tanky) či automobilového priemyslu.

Existuje a je vyvíjaných mnoho iných AR zariadení, cez rôzne druhy okuliarov displejov, až po AR kontaktné šošovky. Najvyužívanejším je však v dnešnej dobe bežný *smartfón* (s funkcionalitou *AR Enabled*), ktorý predstavil AR bežných užívateľom a pomohol technológiám AR nabrať na značnej popularite. Dnešné mobilné zariadenia sú vybavené hardvérom aj softvérom pre rýchlejšie a stabilnejšie fungovanie aplikácií založených na zmiešanej realite. Ich základom je najčastejšie platforma od spoločnosti *Google – ARCore*, ktorá bola zverejnená v roku 2017. [25, 24, 26, 27]

2.4 Detekcia tvárí

Táto sekcia je zameraná na analýzu najpoužívanejších tvárových detektorov podporovaných knižnicou OpenCV, ich výhody a nevýhody.

Detekcia tvárí je proces, ktorého cieľom je rozhodnúť o existencii tvárí v zadanom digitálnom obraze, prípadne videu, a v kladnom prípade určiť polohu a rozmery každej z nich. Zväčša je charakterizovaná ako druh detekcie objektov. Problematika detekcie tvárí disponuje viacerými komplikáciami. Jej presnosť je závislá od množstva faktorov, vrátane polohy držania hlavy, mimiky tváre, osvetlenia, zatienenia inými objektmi scény či prítomnosť štrukturálnych prvkov v oblasti tváre (brada, fúzy, okuliare...). [28]

V dnešnej dobe sú najvyužívanejšími systémami detekcie tvárí implementovanými v knižnici OpenCV *Haar-cascade* detekcia a *deep neural network (DNN)* detekcia.

Haar-cascade detekcia

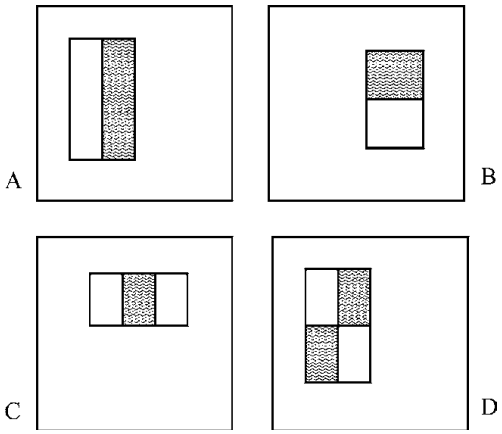
Zmienený spôsob detekcie je inšpirovaný výskumom autorov P. Viola a M. Jones. Na základe ich testov z obdobia publikácie (2001), dosahoval až pätnásťkrát rýchlejšiu detekciu tvárí, v porovnaní s ostatnými vtedy používanými spôsobmi detekcie. [29]

Jedná sa o detektor založený na princípe strojového učenia, v ktorom je kaskádová funkcia trénovaná na množstve pozitívnych (obsahujúcich tvár) a negatívnych (neobsahujúcich tvár) obrázkoch. Metóda hľadá príznaky pozostávajúce z čiernych a bielych obdĺžnikov, kde je súčet pixelov, ktoré ležia vrámci medzí bieleho obdĺžnika odčítaný od tých ležiacich v medziach čierneho obdĺžnika. Obrázok 2.3 zobrazuje príklady takýchto príznakov. [30]

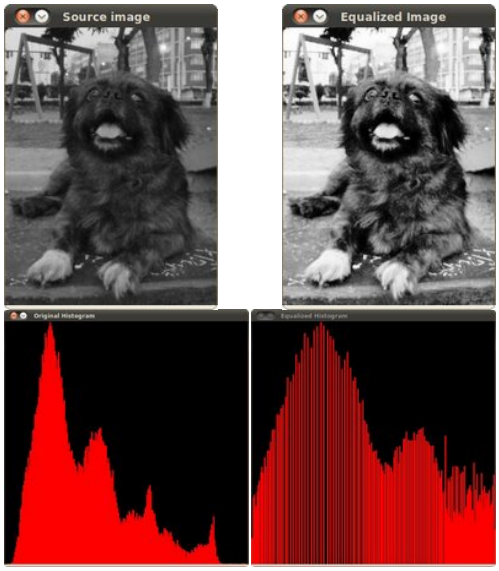
Postup fungovanie tohoto detektora je následovný. Pred spracovaním je nutné obraz previesť na podporovaný formát. Farebný obraz je najprv prevedený na čiernobiely. Pri veľkom rozlíšení sú dodatočne znížené jeho rozmery. Pôvodný detektor bol testovaný na snímkach rozmerov *384x288* pixelov. [29] Následne je potrebné previesť normalizáciu, ktorá zvýši kontrast obrazu 2.4. Dôvodom pre túto úpravu je skutočnosť, že detektor porovnáva vybrané časti obrazu s čiernobielymi príznakmi. Zvýšenie kontrastu zvýrazní rozdiely medzi svetlými a tmavými časťami obrazu, čo spresní výsledky spomínaných porovnaní.

Pri samotnom spracovávaní obrazu prevedie detektor obraz na integrálny obraz *(integral image)*, keďže nepracuje priamo s intenzitami obrazu. Integrálny obraz obsahuje na pozícii x, y súčet hodnôt pixelov smerom nahor a doľava (obrázok 2.5). Vďaka tejto reprezentácii je možné vypočítať ľubovoľné *Haar-like* príznaky v konštantnom čase.

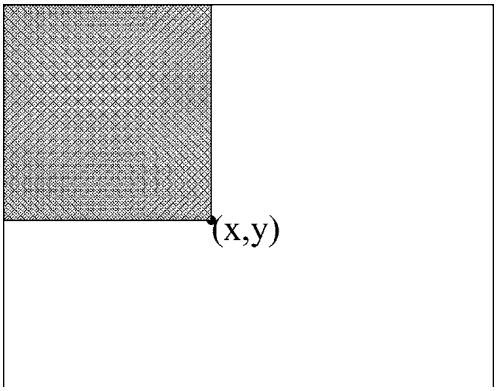
Za pomoci vytrénovaného operátora pozornosti je následne možné odfiltrovať až 50 % obrazu, pričom sa zachová 99 % tvárí. Zložitejšie výpočty sú prevádzané len na sľubných oblastiach obrazu. Tieto sú následne testované po-



Obr. 2.3: Príklady príznakov zobrazené relatívne k obklopujúcemu detekčnému oknu. Dvoj-obdĺžnikové príznaky sú zobrazené v (A) a (B). Časť (C) zobrazuje troj-obdĺžnikový príznak, a (D) štvor-obdĺžnikový príznak. [29, str. 3]



Obr. 2.4: Normalizovaný obraz. Na normalizáciu bola využitá funkcia knižnice OpenCV, **cv::equalizeHist**. [31, 32]



Obr. 2.5: Integrálny obraz. Hodnota v x, y je rovná súčtu hodnôt pixelov nahor a doľava. [29, str. 3]

stupnosťou klasifikátorov s rastúcou zložitosťou, združených do etáp. Oblasť je zamietnutá, ak na nej neuspeje akákoľvek etapa porovnávania. Ak naopak uspejú všetky, oblasť je označená za tvár.

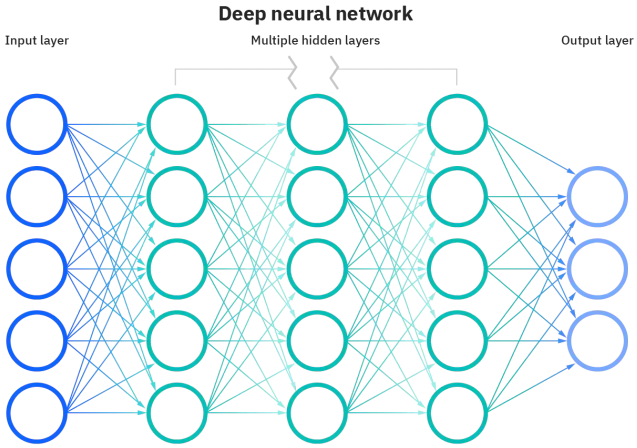
Keďže OpenCV obsahuje model priamo medzi stiahnutými súbormi, je možné tento konkrétny model porovnať s inými detektormi. Ide o rýchly detekčný algoritmus s jednoduchou štruktúrou, ktorý je schopný poradiť si s rôznymi veľkosťami tvárí. Nevýhodami je ale fakt, že nachádza množstvo falošných tvárí, nefunguje pod rôznymi uhlami a má značné problémy pri prekrytí tváre. [29, 33]

DNN detekcia

Neurónové siete (NN), alebo umelé neurónové siete (ANN) sú druh algoritmu z oblasti strojového učenia. Sú navrhnuté spôsobom, aby ich štruktúra napodobňovala biologické fungovanie neurónových prepojení v ľudskom mozgu, podľa čoho sú aj pomenované. Každý „neurón“ je reprezentovaný uzlom, ktoré sú súčasťou uzlových vrstiev. Uzly sú medzi sebou poprepájané pomocou váhy a prahovej hodnoty. Ak je prekročená prahová hodnota akéhokoľvek prepojenia, uzol sa aktivuje a zašle dáta do ďalšej uzlovej vrstvy. V opačnom prípade sa do ďalšej vrstvy siete neprenášajú žiadne údaje. Pred porovnaním hodnoty s prahovou hodnotou sa ale hodnota násobí váhou.

Uzlové vrstvy sú stavebnou jednotkou neurónových sietí. Každá neurónová sieť obsahuje vstupnú vrstvu a výstupnú vrstvu, kde vstupná predstavuje vstupné parametre algoritmu a výstupná zase výsledky výpočtov. Mimo týchto dvoch vrstiev obsahuje aspoň jednu skrytú vrstvu.

Váha, spolu s prahovou hodnotou sú parametre, ktoré sa postupne ladia a vylepšujú učením neurónovej siete, všetko preto, aby sa zvýšila ich presnosť. Po doladení sa neurónová sieť stáva silným nástrojom umelej inteligencie, ktorý umožňuje spracovávanie veľkého množstva dát v pomerne krátkom čase.



Obr. 2.6: Schéma, zobrazujúca štruktúru DNN algoritmu. [35]

V porovnaní s manuálnou identifikáciou ľudskými odborníkmi, ktorá môže trvať aj hodiny, sú úlohy spojené s rozpoznávaním reči alebo rozpoznávaním obrázkov vyriešené pomocou NN v priebehu niekoľkých minút. „*Jednou z najznámejších neurónových sietí je vyhľadávací algoritmus spoločnosti Google.*“ [34]

Rozdielom medzi klasickou NN a DNN je hĺbka danej siete. *Deep*, teda „hlboký“, referuje na takú NN, ktorá má viac ako tri skryté vrstvy. Diagram takejto siete je znázornený na obrázku 2.6. [35]

Detekcia tvárí pomocou DNN je realizácia spomínaných princípov. Do vstupnej vrstvy sú vložené hodnoty z malých okien obrázkov, alebo oblasti v obrázkoch. Následne rozhoduje o existencii tváre v danom okne v závislosti na:

- Tvárových črtách – Pomocou mapy hrán a heuristiky odstráni zoskupené hrany tak, aby boli zachované iba tie tvoriace kontúru tváre. Následne sa pokúša napasovať elipsu medzi oblasť hlavy a pozadie.

- Farbe pokožky – Farba ľudskej pokožky sa ukázala byť efektívnym príznakom pre veľa využití. Početné štúdie túto skutočnosť potvrdili aj napriek variácii farby pokožky u jedincov. Hlavný rozdiel totižto leží v intenzite, nie zafarbení.

- Textúre – Ľudská tvár má unikátnu textúru, ktorá ju odlišuje od iných objektov. Jedna z metód identifikuje v obrázkoch textúry podobné tvárám, pričom sú zvažované tri kategórie príznakov: „tvár“, „pokožka“ a „iné“. [28]

Výhody a nevýhody závisia od samotného modelu, keďže ten OpenCV priamo neponúka. Set obrázkov, na ktorých bol model trénovaný silne ovplyvňuje jeho vlastnosti. Aj napriek tomu, pre väčšinu sú následujúce tvrdenia pravdivé. Ide o veľmi presnú formu detekcie, ktorá je na procesore schopná bežať v reálnom čase. Dokáže si taktiež poradiť s rôznymi veľkosťami tvárí, orientáciami tvárí a aj prekrytím tváre. [33]

Záver

Po zvážení výsledkov analýzy bol aj napriek jednoznačne vyššej presnosti detekcie spôsobom DNN zvolený spôsob *Haar-cascade*. Zlomovým bodom bola absencia modelu v balíku súborov OpenCV, ako aj rýchlosť. Oba spôsoby detekujú tváre v reálnom čase, *Haar-cascade* však dosahuje lepších rýchlostných výsledkov. [33]

2.5 Detekcia tvárových čŕt

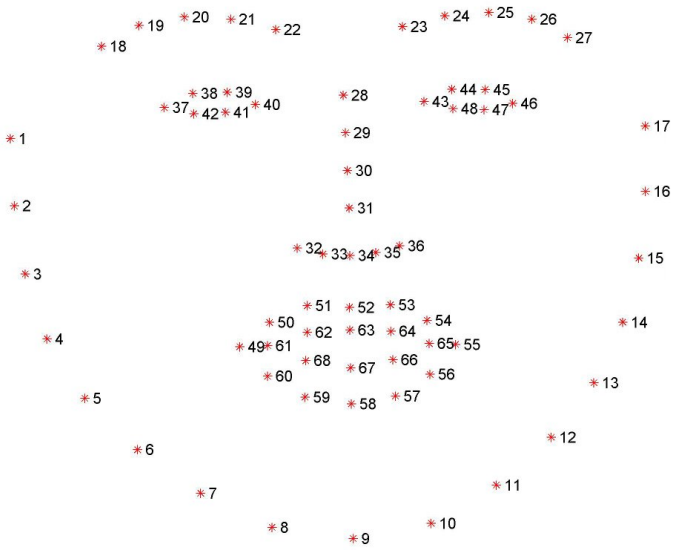
Pojem tvárová črta je v tejto práci definovaný ako významný sémantický orientačný bod tvárovej oblasti, ako je napríklad obrys očí, úst, nosa, obočia či silueta tváre. Tie sú často základom aplikácií zameraných na prácu s ľudskými tvárami, ako sú aplikácie riešiace rozpoznávanie, sledovanie, animovanie tvárí a analýza mimiky. Tak ako u detekcie tvárí, detekovanie čŕt je značne závislé na podmienkach ako komplikovanosť pózy a výrazu, osvetlenie či prekrytie objektov. Príklad vizualizácie bežných tvárových čŕt je zobrazený na obrázku 2.7.

Pre detekciu týchto bodov je dôležitá predchádzajúca detekcia tvárovej oblasti. Proces hľadania týchto bodov sa nazýva *face alignment*. Ide o komplikovaný proces založený na strojovom učení a počítačovom videní. Existuje mnoho modelov pre túto detekciu, cez *Robust Cascade Pose Regression* (RCPR) po regresné modely. Rôzne spôsoby detekcie tvárových čŕt sú zamerané na rôzne body tváre, ako aj na odlišný počet týchto bodov.

OpenCV extra modul „face“ ponúka API pre prácu s tvárovými črtami. Sú v ňom naimplementované tri triedy detekčných algoritmov, **FacemarkAAM**, **FacemarkKazemi** a **FacemarkLBF**. [36, 37, 38]

AAM

„Active Appearance Models (AAMs) sú štatistické modely tvarov a vzhľadov, ktoré sú z malého počtu parametrov modelu, ktoré kontrolujú variácie tvaru a vzhľadu, schopné generovať inštancie špecifických tried objektov (napr. tváre). Nepoužíva žiadne sofistikované tvary, robustné prvky či robustné normy na zlepšenie rýchlosť. Aj napriek tomu je ich výkonnosť pozoruhodná a v niektorých prípadoch porovnateľná s modernými postupmi.“ [40]



Obr. 2.7: Vizualizácia tvárových čŕt, zobrazujúca 68 hlavných orientačných bodov tváre. [39]

Kazemi

„Rieši problém face alignmentu pre jeden snímok. Využíva súbor regresívnych stromov na zisk pozícií tvárových orientačných bodov priamo z riedkej podmnožiny intenzít pixelov extrahovaných zo vstupnej snímky. Dosahuje vysoko kvalitné predikcii v priebehu niekoľkých milisekúnd. Za zrýchlenie v porovnaní so staršími metódami je zodpovedná identifikácia podstatných častí ostatných face alignment algoritmov a ich následné začlenenie v zjednodušenej formulácii do kaskády regresných funkcií vysokej kapacity za pomoci gradientného boostingu.“ [41]

LBF

„Vysoko efektívny, presný regresný prístup k problematike face alignment, ktorý priniesol dva nové komponenty: sadu local binary features (LBF) príznakov a princíp pre učenie sa týchto príznakov založený na lokalite. „Princíp lokality“ vedie k naučeniu sa sady silne diskriminačných LBF príznakov pre každý orientačný bod tváre nezávisle. Získané príznaky sú použité na spoločné naučenie sa lineárnej regresie pre konečný výstup. Tento prístup dosahuje prevratných výsledkov aj na najnáročnejších benchmarkoch. Keďže extrakcia regresných LBF je výpočtovo veľmi lacná prekonal tento systém rýchlosťou svojich predchodcov. Dosahuje rýchlosti až 3000 FPS na zariadeniach PC a až 300 FPS na mobilných zariadeniach.“ [42]

Záver

Aj napriek tomu, že sa OpenCV Facemark API skladá z troch rôznych implementácií, predtrénovaný model je dostupný len pre triedu **FacemarkLBF**. Keďže zámerom práce nieje trénovanie vlastného modelu, bolo rozhodnuté využiť práve tento predtrénovaný model dostupný priamo v balíku **„opencv_contrib“**, a tým aj spôsob detekcie pomocou LBF.

2.6 Práca s obrázkami a videom v OpenCV

Obrázky a rastrová grafika

Rastrová grafika je v OpenCV reprezentovaná maticovou formou (trieda **cv::Mat**). Táto trieda je zložená z dvoch častí, hlavičky (obsahujúcej informácie ako veľkosť matice, adresu k matici, apod.) a ukazovateľa na maticu obsahujúcu hodnoty pixelov. **Cv::Mat** podporuje viacero spôsobov ukladania obrazových informácií, cez rôzne farebné priestory, rôzne dátové typy a rôzne počty farebných kanálov. Najbežnejší je dátový typ *uchar/unsigned char*, ktorý reprezentuje hodnoty v rozsahu *0-255*, a farebný priestor *RGB (red, green, blue)*, v štandarde OpenCV ale *BGR*. Táto trieda tiež podporuje množstvo spôsobov pre explicitnú tvorbu instancií, či už rôzne formy konštruktorov, či funkcie ako **cv::Mat::create**, **cv::Mat::zeros**, **cv::Mat::ones** alebo **cv::Mat::eye** (viď. 2.8). [43, 44]

Modifikácia obrázkov

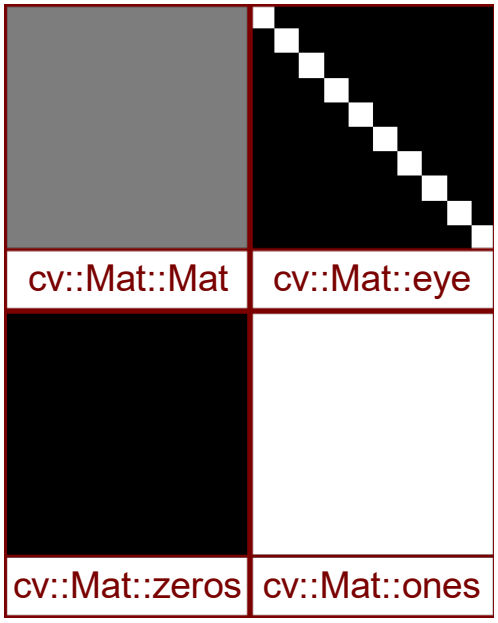
OpenCV podporuje základné funkcie vkresľovania jednoduchých útvarov do rastrových obrázkov. Funkcia kresliaca čiaru, **cv::line**, funkcia kresliaca kružnicu, **cv::circle**, funkcia kresliaca obdĺžnik, **cv::rectangle**, a iné. Medzi dôležité nastaviteľné parametre týchto funkcií patrí poloha, farba či hrúbka čiary, prípadne rozmery a možnosť vyplnenia útvaru (vyplnenie útvaru je možné dosiahnuť nastavením parametra *thickness* na hodnotu *-1*). [45, 46]

Práca s videom a webkamerou

Na prácu s webkamerou je v OpenCV modul **„videoio“**, ktorý obsahuje triedu **cv::VideoCapture**, zameraná na zachytávanie videa z videosúborov, sekvencií obrázkov či kamier.

Pre prácu s touto triedou v spojení s kamerami je najprv potrebné **cv::VideoCapture** inštancovať.

Následne je potrebné pripojiť sa ku kamere, k čomu slúži metóda **open**, ktorá berie za parameter index kamerového zariadenia v danom systéme (a prípadne index preferovaného *backendu* napr. **cv::CAP_FFMPEG**, **cv::CAP_IMAGES** či **cv::CAP_DSHOW**), alebo je možné tie isté parametre predať konštruktoru triedy. Na kontrolu korektného naviazania spojenia



Obr. 2.8: Explicitná tvorba triedy **cv::Mat**. Vizualizácia na čiernobielom obrázku rozmerov *10x10* pixelov. Vľavo hore využitý konštruktor s pridelenou hodnotou *125*, následne v smere hodinových ručičiek funkcie **cv::Mat::eye**, **cv::Mat::ones** a **cv::Mat::zeros**.

s kamerou slúži metóda **isOpened**, ktorá vracia *boolean* hodnotu označujúcu úspech alebo neúspech spojenie.

Trieda umožňuje čítanie a nastavovanie vlastností pomocou metód **get** a **set**, ktoré berú ako parameter index danej vlastnosti. Je napríklad možné nastaviť preferované FPS alebo rozlíšenie snímkov. Zoznam týchto vlastností a ich indexov sa nachádza v dokumentácii triedy.

Pre zisk následujúceho snímku je využívaná metóda **read** s alternatívou preťaženého operátoru », ktoré vracajú snímok v podobe matice (**cv::Mat**).

Po ukončení práce s triedou ju je nutné uvoľniť volaním metódy **release**. [47, 48]

Homografia

Alebo „projektívna transformácia“ je transformácia, ktorá mapuje dvoj-dimenzionálny priestor na iný dvoj-dimenzionálny priestor v homogénnych súradniciach. Homografie medzi sebou vzájomne mapujú dvojice bodov aj dvojice čiar, na rozdiel od afínnych transformácií však nezachovávajú súbežnosť. Je možné ich vyjadriť projektívnou maticou, ktorá nám umožňuje prevádzať súradnice bodov medzi danými rovinami. Znázornenie rovnice prevodu súradníc vstupného priestoru do súradníc výstupného priestoru za pomoci práve pro-

Obr. 2.9: Príklad využitia projektívnej transformácie v panoramatických fotografiách. [52]

jektívnej matice:

$$\begin{pmatrix} x'_1 \\ x'_2 \\ x'_3 \end{pmatrix} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Proces je úzko súvislý s projektívnou geometriou. Pre jednoduché pochopenie, imituje čo by sa stalo s obrazom, ak by sa zmenil uhol pohľadu. Homografia túto zmenu uhlu pohľadu zachytáva matematickými vzorcami.

Tento princíp je užitočný pri mapovaní rastrovej grafiky podľa určených dvojíc súradníc. Ide o pomerne jednoduchý a lacný spôsob simulácie perspektívy obrazu. Oblasťou, kde je využívaná, sú panoramatické fotografie a teda zlučovanie viacerých snímkov podľa istých dvojíc orientačných bodov 2.9. Je tiež často využívaný v AR. [49]

V OpenCV sú naimplementované funkcie pre prácu s Homografiami. Ako prvé je nutné získať transformačnú maticu. Tento problém riešia funkcie **cv::findHomography** a **cv::getPerspectiveTransform**. Obe tieto funkcie počítajú transformačnú maticu za pomoci dvojíc bodov z dvoch „uhlov pohľadu“, vstupným parametrom **cv::getPerspectiveTransform** sú presne 4 takéto dvojice, zatiaľčo **cv::findHomography** pracuje aj z väčším počtom dvojíc snažiac sa minimalizovať chybu pri spätnej projekcii. **cv::warpPerspective** a **cv::perspectiveTransform** sú funkcie, ktoré následne zo získanou transformačnou maticou pracujú. **cv::warpPerspective** aplikuje vypočítanú projekciu na obrázok, narozdiel **cv::perspectiveTransform** transformuje akýkoľvek vektor, no nieje odporúčané využívať ho v spojení s obrázkami. [50, 51]

Kapitola 3

Návrh

Táto kapitola sa zaoberá návrhom prototypu samotného programu založeného na predchádzajúcej analýze. Prvá časť tejto kapitoly je zameraná na charakterizáciu vlastností, ktoré bude prototyp spĺňovať. Nasleduje návrh užívateľského prostredia, hlavného cyklu programu a nakoniec časť návrhu tvárového detektora.

3.1 Charakteristika programu

„Úlohou tohto projektu je tvorba intuitívnej aplikácie na vytváranie jednoduchých tvárových filtrov.“ Výstupom má byť prototyp, ktorý spĺňa zmienenú charakteristiku, je schopný riešiť následujúce problémy:

- tvorbu rastrovej grafiky,
- prácu s webkamerou,
- detekciu tváre a tvárových čŕt v snímke,
- mapovanie rastrovej grafiky v závislosti na množine bodov.

Prototyp bude obsahovať základné funkcionality pre splnenie požiadavok. Nebudem sa z tohto dôvodu zameriavať na návrh a implementáciu komplikovaného kresliaceho prostredia, zložitého grafického prostredia či bezchybného mapovania grafiky na tvárové črty.

3.2 Užívateľské prostredie

V návrhu užívateľského prostredia som sa zameral na jednoduchosť a intuitívnosť. GUI bude rozdelené na 3 hlavné časti.

Prvou je kresliaca plocha, ktorá slúži na tvorbu následne mapovaných obrázkov. Užívateľ by mal byť schopný pri tvorbe grafiky (teda pri kreslení)

bez ťažkostí odhadnúť, na aké miesto tváre bude určitá časť grafiky mapovaná. Pre dosiahnutie tohoto cieľa je potrebné na pozadie kresliacej plochy vizualizovať tvárové body detekované detektorom tvárových čŕt. Výsledok by mohol pripomínať obrázok 2.7.

Druhá časť bude obsahovať samotný výstupný snímok webkamery. Na ňom bude znázornená mapovaná grafika z kresliacej plochy, prípadne iné prvky, ako znázornené tvárové črty či hodnota FPS.

Poslednou časťou je tzv. panel nástrojov. Ten bude obsahovať UI prvky zodpovedné za zmeny a nastavovania rôznych hodnôt za behu aplikácie. Medzi takéto prvky patria *trackbary*, texty, prípadne tlačidlá. Nastaviteľné hodnoty za behu budú napríklad hrúbka kreslenej čiary, farba kreslenia či zapínanie a vypínanie rôznych vizualizačných prvkov (viď. predchádzajúci bod).

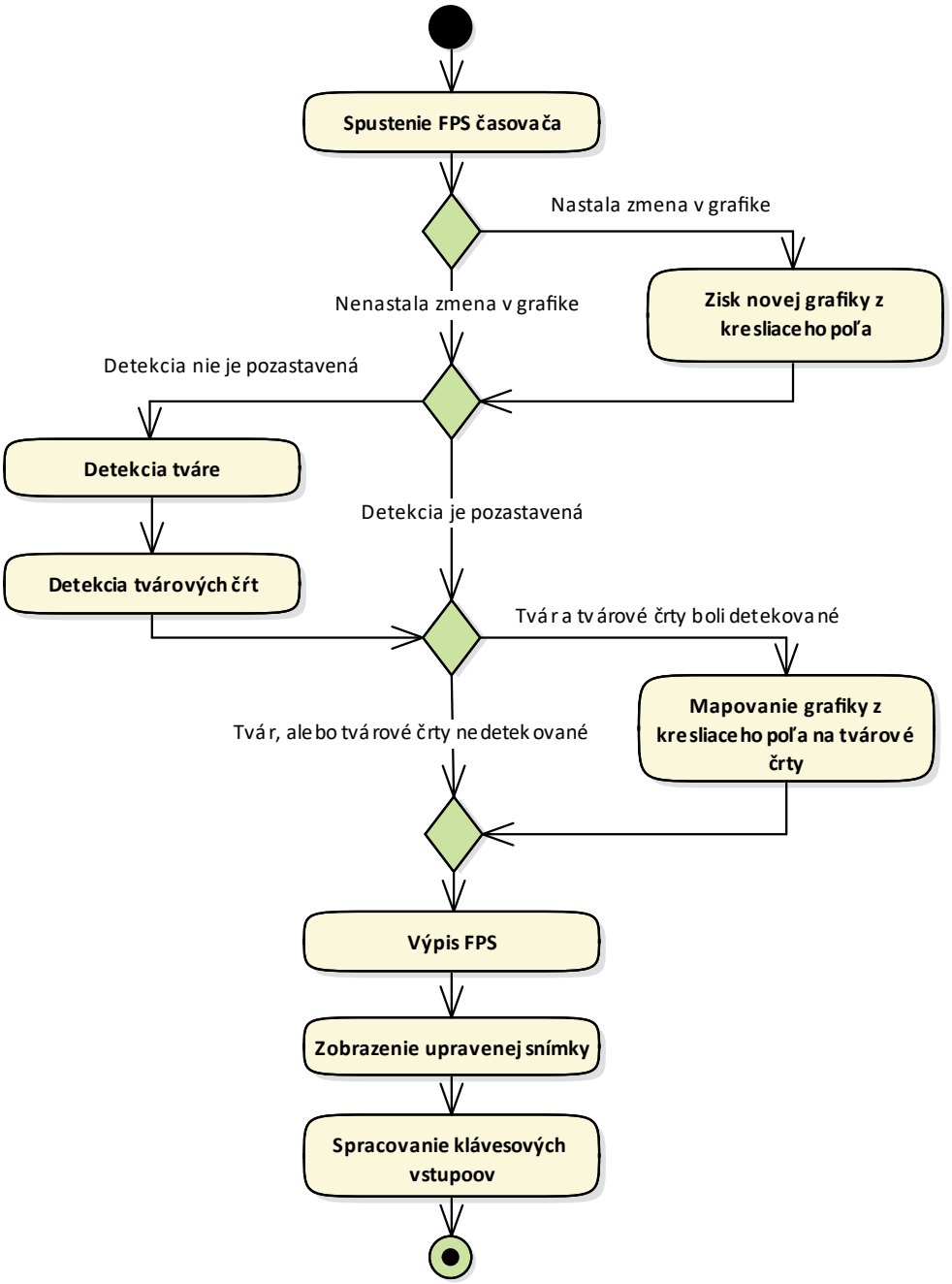
3.3 Spracovanie snímky z webkamery

Aplikácia bude pracovať v dvoch hlavných etapách, príprava a cyklus behu. V cykle behu bude každú iteráciu načítaná snímka z webkamery, ktorá následne prejde sledom úprav až vznikne výsledný obraz. Medzi tieto kroky patrí detekcia tváre, detekcia tvárových čŕt, či mapovanie grafiky na danú snímku. Schéma 3.1 zobrazuje návrh cyklu behu prototypu.

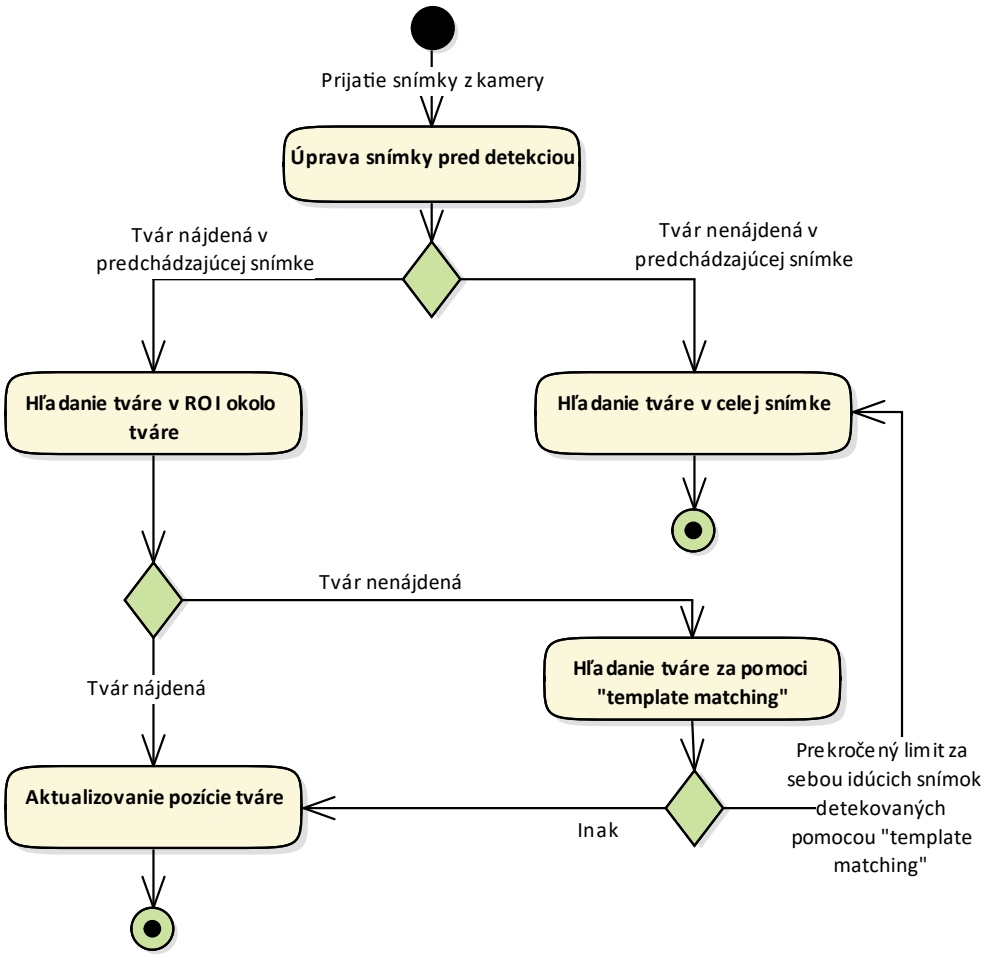
3.4 Tvárový detektor

Hlavnou vlastnosťou, ktorá je od detektora tváre v tomto programe požadovaná, je rýchlosť. Práve preto bol v analýze zvolený spôsob detekcie za pomoci *Haar-like cascades*. Ten má ale viacero slabých stránok, ktoré je nutné zobrať v úvahu pri návrhu samotnej detekcie. Taktiež je pre dosiahnutie vyššej rýchlosti detekcie nutné prispôsobiť detekčný algoritmus potrebám programu.

Táto aplikácia nepotrebuje detekovať viacero tvárí naraz, čo je skutočnosť, ktorej využitím viem detekciu podstatne zrýchliť. Pri zameraní sa výhradne na *region of interest* (ROI) v okolí poslednej detekovanej tváre zahodím veľké množstvo snímku, čo zjednoduší detektoru prácu. V prípade, že nebola tvár z minulého snímku v ROI nájdená je silne pravdepodobné, že nastalo zlyhanie detektora pri hľadaní tváre. V takejto situácii preto využijem *template matching* na hľadanie časti snímku podobnej záznamu obrázku tváre z minulej iterácie. V prípade prvotnej nesprávnej detekcie sa však môže tento postup zaseknúť na detekovaní v nesprávnej časti obrazu, preto som nastavil limit počtu za sebou idúcich detekcií pochádzajúcich z *template matching*. 3.2 znázorňuje schému fungovania môjho tvárového detektora inšpirovanú prácou. [53]



Obr. 3.1: Schéma návrhu cyklu behu aplikácie.



Obr. 3.2: Schéma návrhu algoritmu tvárového detektora.

Kapitola 4

Implementácia

V tejto kapitole sa zameriavam na postupy a rozhodnutia vykonané pri implementácii programu. V sekcii 4.1 zhrniem využité technológie implementácie. Nasleduje podkapitola 4.1, v ktorej sa zaoberám hlavnými triedami aplikácie. V sekcii 4.3 riešim tvorbu jednoduchého užívateľského rozhrania. V časti 4.4 sa sústredím na fungovanie detektoru tváre a tvárových čŕt. Následne v časti 4.5 vysvetľujem tvorbu a mapovanie grafiky v tejto aplikácii.

4.1 Technológie

Ako prvé sa zameriam na výsledky analýzy z pohľadu technológií. Aplikácia je založená na práci s knižnicou OpenCV a jej externými modulmi. Je vyvíjaná pre OS Windows a ako programovací jazyk je využité C++. Na detekciu tvárí využívam *Haar-cascade* prístup a tvarové črty detekujem pomocou LBF.

4.2 Základné triedy

Chod aplikácie je z väčšej časti postavený na funkčnosti troch základných tried, ktoré hrajú významnú úlohu v cykle behu programu, či už pri spracovávaní snímky z webkamery, alebo pri tvorbe rastrovej grafiky a mapovaní.

FaceDetector

Trieda zameraná na detekciu tváre v snímke. Snímka je do funkcie predávaná ako parameter metódy **Detect**. Na lokalizovanie tvárí využíva *Haar-cascade* detekciu. Sústredí sa na detekciu výhradne jednej tváre v snímke, ostatné zahadzuje. Volanie metódy **GetFace** túto detekovanú tvár vracia. Tvár, resp. obdĺžnik v ktorom sa tvár nachádza je možné v ľubovoľnom obrázku vizualizovať volaním metódy **Show**.

LandmarkDetector

Ide o triedu detekujúcu konkrétne orientačné body tváre v predtým nájdenej tvárovej oblasti. Je štruktúrou podobná triede **FaceDetector**. Taktiež sa v nej nachádza metóda **Detect**. Tá na detekciu využíva triedu **FacemarkLBF** z externého OpenCV modulu **„face“**, ktorá je založená na princípe LBF. Narozdiel od triedy tvárového detektora má implementované dve metódy na získavanie týchto detekovaných bodov. Prvá má názov **GetOne**, ktorá vracia práve jeden detekovaný bod, a to podľa indexu zaslaného ako parameter. **GetAll** vracia všetkých 68 detekovaných bodov. Podobne fungujú aj vizualizačné metódy **ShowOne** a **ShowAll**.

DrawBoard

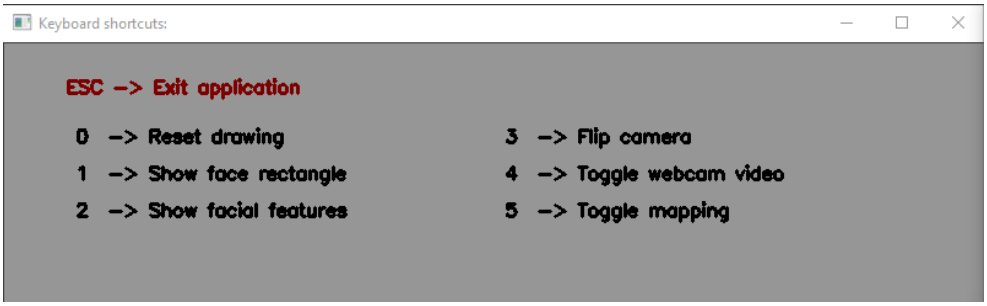
Trieda zaobstaravajúca kresliace okno a špecifickú plochu pre tvorbu rastrovej grafiky. Metóda **Init** vykoná pomerne zdĺhavú inicializáciu kresliaceho okna, pričom načíta nastavenia, inicializuje okno a vytvorí kresliacu plochu. **Show** aktualizuje a zobrazí užívateľom vytvorenú grafiku na kresliacej ploche. Táto metóda je volaná v hlavnom cykle po volaní metódy **IsChanged**, ktorá vracia *boolean* hodnotu reprezentujúcu informáciu o prípadnej zmene na grafike v kresliacom poli. **GetDrawing** slúži na získavanie vytvorenej grafiky. Na vymazanie vytvorenej grafiky slúži metóda **Reset**. Pri mapovaní grafiky je z kresliaceho okna taktiež potrebné získať polohy orientačných bodov, aby bolo možné využiť homografiu. S tým súvisí aj kontrola počtu týchto bodov. Spomínané problémy riešia metódy **GetMarkPositions** (zisk súradníc polôh bodov) a **GetMarkCnt** (zisk počtu bodov).

4.3 Užívateľské prostredie

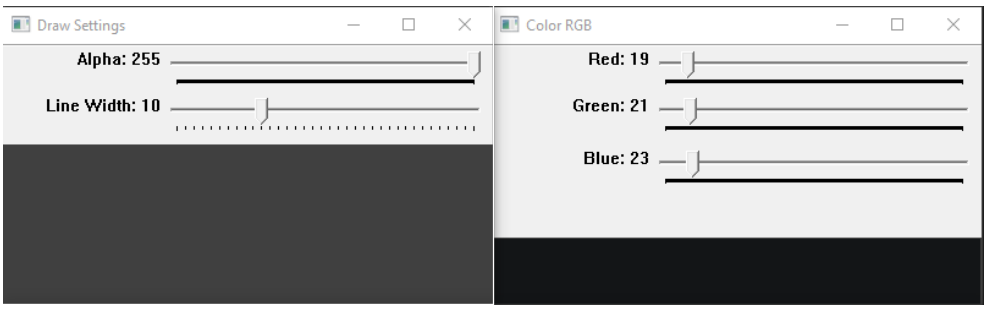
Ako bolo spomenuté v návrhu, GUI je rozdelené na 3 základné časti, kresliaca plocha, panel nástrojov a upravená snímka z webkamery.

Panel nástrojov je ale rozdelený na menšie časti. Dôvodom je spôsob, akým fungujú v OpenCV *trackbary*. Keďže nie je možné určiť ich šírku, výšku ani polohu a vieme ovplyvniť len ich poradie, bolo nutné rozdeliť panel nástrojov na väčšie množstvo menších okien. Najväčšie okno zobrazuje zoznam funkčných klávesových skratiek, na ktoré aplikácia reaguje. Je nimi možne napríklad vypínať a zapínať vizualizáciu orientačných bodov tváre či zrkadliť vstup webkamery. Plný zoznam klávesových skratiek je zobrazený na obrázku 4.1. Menovite:

- *Escape* – ukončí program,
- *0* – resetuje nakreslenú grafiku,
- *1* – vypína a zapína vizualizáciu tváre,



Obr. 4.1: Ukážka okna panela nástrojov zobrazujúceho klávesové skratky.



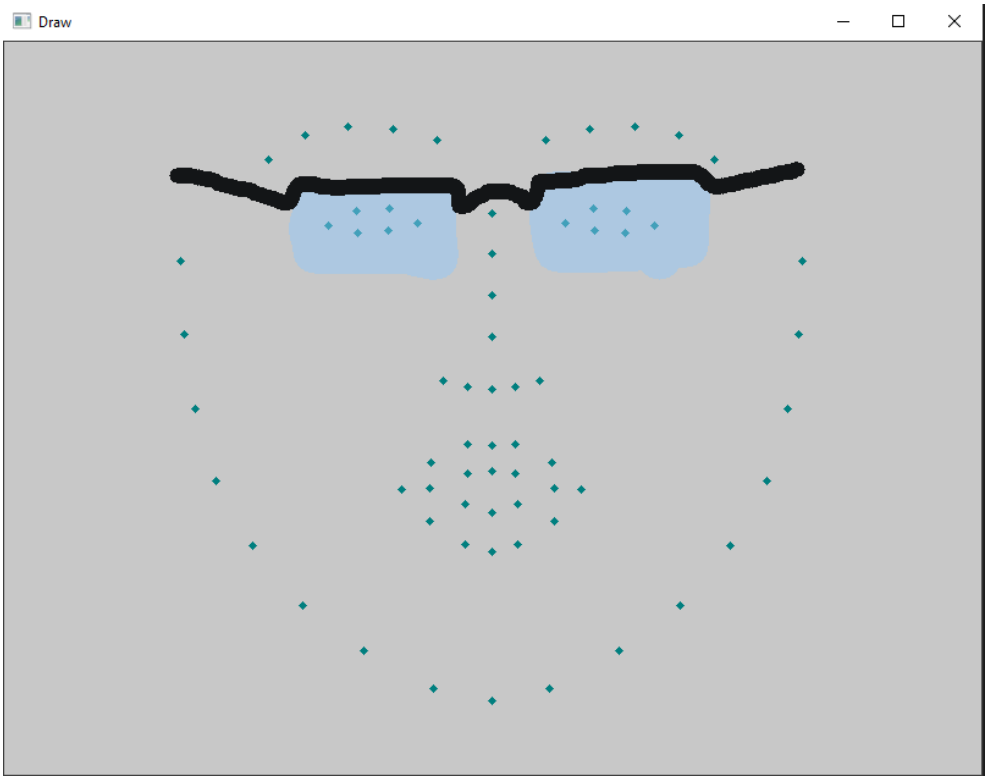
Obr. 4.2: Ukážka okna panela nástrojov umožňujúca nastavenia farby hrúbky a priesvitnosti kreslenia.

- *2* – vypína a zapína vizualizáciu orientačných bodov tváre,
- *3* – zrkadlí vstup webkamery,
- *4* – pozastaví načítanie nových snímkov z webkamery,
- *5* – pozastaví mapovanie grafiky na tvár.

Prostredné okno v hornej rade obsahuje *trackbar* na nastavenie hodnoty priesvitnosti čiary a ďalší *trackbar*, ktorý nastavuje hrúbku čiary. Posledné okno panela nástrojov umožňuje nastavenie farby pomocou troch *trackbarov*, pričom každý reprezentuje jednu z hodnôt RGB. Pod týmito *trackbarmi* je ukážka zvolenej farby. Ukážku týchto okien je možné vidieť na obrázku 4.2.

V spodnej rade na pravej strane sa nachádza okno predstavujúce kresliacu plochu aplikácie. Na pozadí sú zobrazené pozície orientačných bodov, ktoré slúžia pre intuitívnu orientácie. Užívateľ vďaka týmto bodom jednoducho zistí, kam sa jeho grafika na tvár mapuje. Toto okno je spravované triedou **DrawBoard**, ktorá bola popísaná v sekcii 4.2. Implementácia samotného mapovania je vysvetlená v sekcii 4.5. Ukážka daného okna sa nachádza na obrázku 4.4.

Na pravej strane spodnej rady užívateľského prostredia sa nachádza okno zobrazujúce výslednú upravenú snímku z webkamery. V opise okna panela

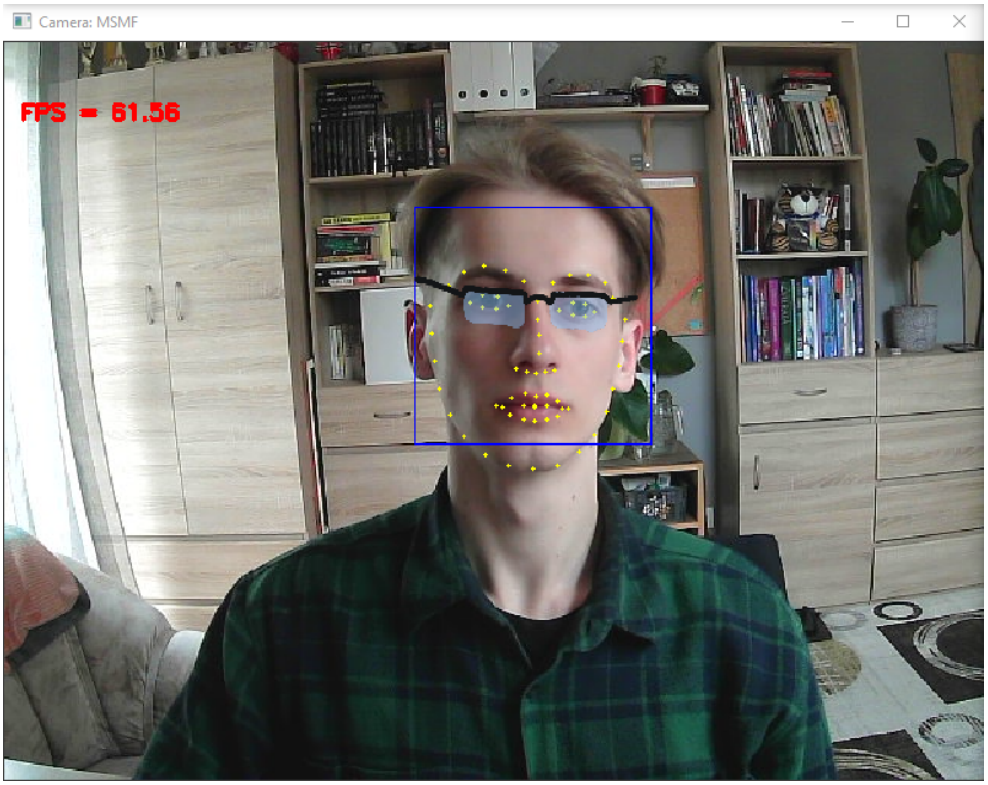


Obr. 4.3: Ukážka okna kresliacej plochy.

nástrojov už bol zmienený zoznam klávesových skratiek, z ktorých väčšina slúži na ovplyvnenie správania tohto okna. Jediná bod, ktorý nebol spomenutý je ukazovateľ FPS, nachádzajúci sa v ľavom hornom rohu okna. Je k nemu dôležité poznamenať, že nejde o FPS spojené z výkonom webkamery, ale o FPS behu aplikácie. Hodnota teda môže prekročiť maximálnu hodnotu FPS použitej webkamery. Pri nízkom rozlíšení webkamery môže tiež nastať zhoršenie kvality obrazu kvôli úprave veľkosti snímky na veľkosť okna. Na obrázku 4.4 je spomínané okno vizualizované.

Ukážku celého GUI je možné vidieť na obrázku 4.5. Tvorba okna užívateľského prostredia je pomerne jednoduchá a postup je takmer identický pre všetky doposiaľ zmienené okná. Následujúci kód ukazuje časť funkcie **createKeyWindow**, ktorá vytvára a zobrazuje okno panela nástrojov so zoznamom klávesových skratiek.

```
void createKeyWindow()
{
    cv::namedWindow ( KeyWndName, cv::WINDOW_NORMAL );
    cv::moveWindow  ( KeyWndName, KeyWndPos.x, KeyWndPos.y );
    cv::resizeWindow( KeyWndName, KeyWndSize );
```



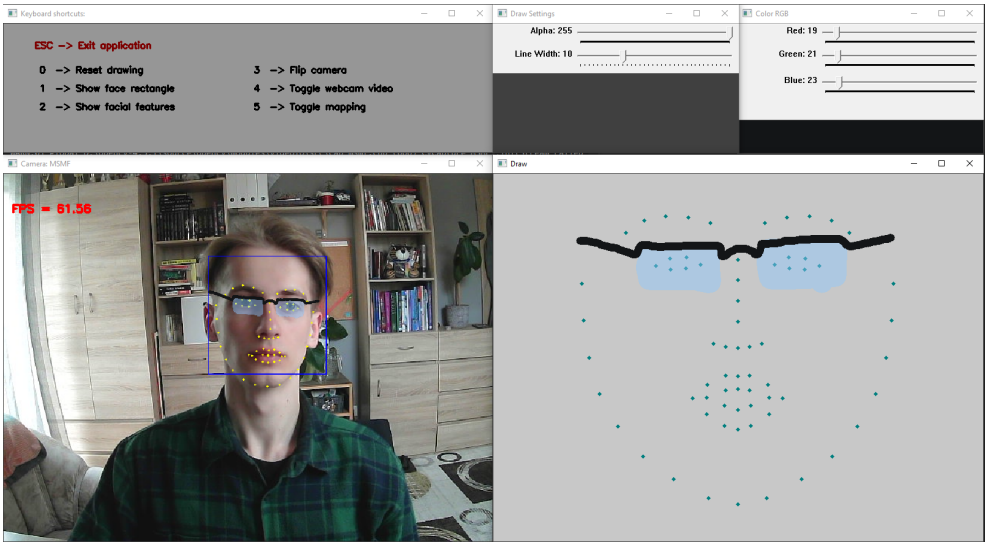
Obr. 4.4: Ukážka okna zobrazujúceho upravenú snímku s mapovanou grafikou na tvári. Sú tiež vizualizované výsledky detekčných algoritmov a počítadlo FPS.

```
    cv::Mat keys( KeyWndSize, CV_8UC3, cv::Scalar(150, 150, 150));
    cv::putText( keys, "ESC -> Exit application",
            cv::Point(50, 40), cv::FONT_HERSHEY_SIMPLEX,
            0.5, cv::Scalar(0, 0, 125), 2 );
    ...

    cv::imshow( KeyWndName, keys );
}
```

4.4 Detekcia tvárí a tvárových čŕt

V tejto časti sa zameriam na implementáciu dvoch detektorov. Jedným je detektor tváre a druhým detektor tvárových čŕt. V časti 4.2 som už spomínal, že oba tieto detektory majú naimplementovanú vlastnú triedu a opísal som



Obr. 4.5: Ukážka grafického užívateľského prostredia programu. Tri okná v hornej rade reprezentujú „panel nástrojov“. Pravé okno v spodnej rade zobrazuje kresliacu plochu a ľavé upravenú snímku z webkamery.

tiež rozhranie týchto tried. V tejto časti popíšem, čo sa deje vo vnútri týchto tried.

Detekcia tváre

V kapitole 3.4 už bola uvedená schéma návrhu tvárového detektora. Verzia, ktorá je súčasťou programu funguje presne podľa tejto schémy 3.2, preto v tejto časti nebudem opisovať základnú štruktúru detektora. Namiesto toho rozoberiem bod po bode časti kódu, ktoré túto schému privádza do reality. Kód ktorý pripravuje snímku na detekciu je následovný:

```
cv::cvtColor( frame, gray, cv::COLOR_BGR2GRAY);
cv::resize( gray, gray,
            cv::Size( gray.size[1] * m_ResizeScale,
                      gray.size[0] * m_ResizeScale),
            0, 0, cv::INTER_LINEAR_EXACT);
cv::equalizeHist(gray, gray);
```

Ako už bolo spomenuté v analýze, prvým krokom je prevod snímky z farebného formátu na formát čiernobiely. Na to slúži v tomto prípade funkcia **cvtColor**. Snímka je následne zmenšená funkciou **resize** a **equalizeHist** zvýši kontrast snímku.

Detekcia tvárí v celej snímke je riešená volaním metódy **detectMultiScale** na premennú typu **CascadeClassifier**. Premenná tohto typu reprezen-

tuje *Haar-cascade* detektor a je inicializovaná parametrom udávajúcim cestu k priečinku, ktorý obsahuje daný naučený kaskádový model. Metóda **detectMultiScale** prijíma parametre ktoré udávajú vlastnosti detektora, ako je maximálna a minimálna veľkosť detekovanej tváre či minimálny počet susediacich detekcií. Výstupom je *std::vector* obsahujúci obdĺžniky, ktoré definujú pozície nájdených tvárí.

Detekcia tvárí v ROI prebieha identicky, rozdielom je však vstupná matica, ktorá zahŕňa len oblasť v ktorej sa nachádzala tvár v minulej iterácii. Táto oblasť bola práve v minulej iterácii po úspešnej detekcii uložená. Pri nájdení väčšieho počtu tvárí v jednom ROI sa vyberie tá, ktorej súradnice sú najmenej odlišné od tých z predchádzajúcej iterácie.

Hlavná časť kódu funkcie pracujúcej s postupom *template matching*:

```
bool FaceDetector::FindWithMatching(cv::Mat& gray)
{
  ...
    cv::TemplateMatchModes matchMode = cv::TM_SQDIFF;
    cv::matchTemplate( gray, m_PrevFaceImg, compRes, matchMode );
    cv::normalize( compRes, compRes, 0, 1,
                   cv::NORM_MINMAX, -1, cv::Mat() );

    minMaxLoc( compRes, &minVal, &maxVal,
               &minLoc, &maxLoc, cv::Mat() );
    matchLoc = minLoc;
  ...
}
```

Funkcia **matchTemplate** zoradí v snímku oblasti v závislosti na podobnosti k obrázku, ktorý je doň zaslaný ako parameter. V skutočnosti ide skôr o priradenie hodnoty podobnosti než o „zoradenie“. Tieto hodnoty je nutné normalizovať. Následne hľadáme oblasť s najnižšou hodnotou podobnosti (pre iné metódy ako **cv::TM_SQDIFF** to môže byť najvyššia hodnota).

Detekcia tvárových čŕt

Implementácia tohto detektora má pomerne jednoduchú štruktúru. Postačuje vytvoriť objekt typu **cv::Ptr¡cv::face::Facemark¿**, čo je vlastne ukazovateľ objekt detekujúci tvárové črty. Pre práca s týmto objektom je najskôr potrebné vytvoriť detektor metódou **cv::face::FacemarkLBF::create()**. Ďalším krokom je volanie **loadModel** s parametrom „cesta k súboru obsahujúcemu model LBF“, čo zo súboru LBF model načíta. Samotnú detekciu vykonáva volanie metódy **fit**, ktorá ukladá súradnice nájdených orientačných bodov do „poľa polí bodov“ (*std::vector<std::vector<cv::Point()>>*).

4.5 Tvorba a mapovanie grafiky

Tvorba grafiky je založená na využití *callback* funkcií, ktoré modul **highui** umožňuje napojiť na udalosti kliku či pohybu myši. *Callback* funkcia je teda naviazaná na kresliace okno a je aktivovaná pri pohybe myši. Čo sa deje pri zavolaní funkcie znázorňuje následujúci kód:

```
if (flags & cv::EVENT_FLAG_LBUTTON) {
    cv::line( m_Drawing, m_PrevMousePos, cv::Point(x, y),
              cv::Scalar(m_DrawData.blue, m_DrawData.green,
                         m_DrawData.red, m_DrawData.alpha),
              m_DrawData.width + 1);
    m_PrevMousePos = cv::Point(x, y);
    m_IsDrawingChanged = true;
}
```

Najprv program zistí, či je stlačené ľavé tlačidlo myši. Ak áno, je do kreslenej grafiky (**m_Drawing**) pridaná čiara, a to z posledného uloženého bodu do bodu, v ktorom sa myš momentálne nachádza. Farbu, hrúbku či priehľadnosť tejto čiary definujú premenné ovládané *trackbarmi* spomenutými v časti 4.3. Uložením novej pozície pre ďalšie volanie a nastavením premennej **m_IsDrawingChanged** reprezentujúcej zmenu kresliacej plochy na kladnú hodnotu volanie funkcie končí.

Mapovanie grafiky na tvár je implementované podľa analýzy (časť 2.6).

```
cv::Mat homography =
        cv::findHomography( drawBoard.GetMarkPositions(),
                            landmarkDetector.GetAll() );
cv::warpPerspective( drawing, warped,
                     homography, drawing.size() );

overlayImage(frame, warped, frame, cv::Point(0, 0));
```

Ako prvá je vypočítaná projektívna matica za pomoci bodov z kresliaceho okna a bodov z detektora tvárových čŕt. Tá je následne využitá pri transformácii perspektívy obrázku nakreslenej grafiky. Nakoniec je táto transformovaná grafika zlúčená s obrázkom z webkamery funkciou **overlayImage**.

4.6 Testovanie

Táto krátka sekcia obsahuje testy naimplementovaného prototypu programu a ich výsledky. Testovania prototypu aplikácie prebehli kvôli momentálnej situácii odvíjajúcej sa od situácie *COVID-19* len s malým počtom testovacích

subjektov. Testy sú primárne zamerané na rýchlosť a intuitívnu prácu s programom.

4.6.1 Test intuitívnosti GUI

Cieľom testu bolo ohodnotiť intuitívnosť užívateľského prostredia. Na testovaní sa podieľali 3 osoby. Každý z týchto subjektov dostal rovnaké zadanie: „Pomocou tejto aplikácie nakresli na svoju tvár okuliare so žltým rámom a modrými polo-priehľadnými sklami. Následne ich vymaž, zapni vizualizáciu detekcie tváre a prevráť obraz z kamery v zvislej osy.“ Test mal vyskúšať užívateľovu schopnosť orientovať sa v aplikácii a zahŕňal takmer každú implementovanú funkcionalitu programu.

Výsledky, nakoľko zväčša pozitívne poukázali na niekoľko nedostatkov. Formát RGB, ktorý aplikácia využíva na nastavenie farby čiary nie je intuitívny pre užívateľa, ktorý nie je technickejšie zameraný. Taktiež sa stalo, že sa subjekt pokúšal kresliť do okna s webkamerou, nie do kresliaceho editoru čo poukazuje na neintuitívnosť užívateľského prostredia.

4.6.2 Test rýchlosti

Cieľom testu bolo zistiť rýchlosť prototypu. Tento test bol motiváciou k implementácii zobrazovania FPS. Pôvodné verzie nedosahovali uspokojivých výsledkov. Pred optimalizáciou detekcie tváre dosahoval prototyp rýchlosti okolo 12 FPS. To taktiež ovplyvňovalo reakcie kresliaceho prostredia, ktoré bolo miestami takmer nepoužiteľné.

Súčasná verzia pracuje s rýchlosťou okolo 25 FPS, čo predstavuje signifikantné zlepšenie oproti prvotným verziám.

Záver

Cieľom práce bolo, na základe analýzy knižnice OpenCV navrhnúť a vytvoriť prototyp programu, umožňujúci užívateľovi tvorbu rastrových obrázkov, ktoré by boli následne mapované a vizualizované na tvár detekovanú vo vstupnom videu webkamery. Ďalším cieľom bolo zrealizovať na vytvorenom prototype vhodné testovania.

V úvodnej časti bakalárskej práce bola prezentovaná stručná analýza knižnice OpenCV. Nasledovala analýza operačných systémov, programovacích jazykov a frameworkov pre tvorbu programu. Ďalšou časťou práce bola analýza zmiešanej reality. Boli tiež analyzované možnosti detekcie tvárí a detekcie tvárových čŕt, kde boli opísané a porovnané rôzne spôsoby, ktoré OpenCV pre detekciu ponúka. Následne boli analyzované funkcionality knižnice OpenCV pre prácu s obrázkami, videom a webkamerou, kde boli vysvetlené základné princípy knižnice pre čitateľov, ktorí s ňou nemali predchádzajúce skúsenosti.

V praktickej časti práce bola predložená charakteristika implementovaného programu s následným vysvetlením rozhodnutí pri voľbe konkrétnej formy implementácie. Nasledovala časť zameraná na návrh intuitívneho užívateľského rozhrania. Praktická časť pokračovala návrhom postupu spracovania snímky z webkamery. Časť o návrhu tvárového detektora, ktorou práca pokračovala, vysvetľovala postup optimalizácie hľadania tvárových oblastí v snímke. Nasledovala kapitola zameraná na implementáciu prototypu aplikácie. Táto kapitola začala rekapituláciou v analýze zvolených technológií. V ďalšej časti bolo popísané fungovanie základných tried programu. Implementačná časť práce pokračovala časťou zameranou na vysvetlenie tvorby užívateľského rozhrania aplikácie. Následne bolo vysvetlené vnútorné fungovanie tvárového detektora a detektora tvárových čŕt. Po tom bola na základe analýzy popísaná implementácia tvorby a mapovania rastrovej grafiky na orientačné body ľudskej tváre.

Praktická časť končila opisom výsledkov testovaní vytvoreného prototypu. Kvôli momentálnej situácii boli prevedené len testy s malým počtom testovacích subjektov. Tie poukázali na niektoré nedostatky, ale taktiež na zvýraznili niektoré silné stránky prototypu. Užívateľské testy vo väčšej škále doposiaľ neboli uskutočnené, keďže vhodná príležitosť pre tieto testovania nastane až po odovzdaní tejto práce.

Aj napriek tomu, že je prototyp plne funkčný, disponuje množstvom nedostatkov. Prostredie pre tvorbu rastrovej grafiky je obmedzené, ponúka málo možností pre tvorbu grafiky. Preto určite stojí za to v práci pokračovať a implementáciu optimalizovať.

Literatúra

[1] OpenCV About. https://opencv.org/about/, accessed: 2021-04-08.

[2] OpenCV Intel. https://opencv.org/intel/, accessed: 2021-04-08.

[3] Introduction to the Standard Template Library. https://justinmeiners.github.io/sgi-stl-docs/stl_introduction.html, accessed: 2021-04-02.

[4] OpenCV Introduction. https://docs.opencv.org/4.5.2/d1/dfb/intro.html, accessed: 2021-04-21.

[5] OpenCV modules. https://docs.opencv.org/master/, accessed: 2021-03-12.

[6] Repository for OpenCV's extra modules. https://github.com/opencv/opencv_contrib, accessed: 2021-02-07.

[7] Britannica, T. E. o. E.: Microsoft Windows operating system. https://www.britannica.com/technology/Windows-OS, 2020, accessed: 2021-04-11.

[8] History of C++. https://www.cplusplus.com/info/history/, accessed: 2021-05-02.

[9] Advantages and Disadvantages of C++ — Make your Next Move! https://data-flair.training/blogs/advantages-and-disadvantages-of-cpp/, accessed: 2021-04-07.

[10] General Python FAQ. https://docs.python.org/3/faq/general.html, accessed: 2021-04-21.

[11] Python History and Versions. https://www.javatpoint.com/python-history, accessed: 2021-04-27.

[12] Python Software Foundation. https://www.python.org/psf/, accessed: 2021-04-21.

[13] Python Advantages and Disadvantages – Step in the right direction. https://techvidvan.com/tutorials/python-advantages-and-disadvantages/, accessed: 2021-04-27.

[14] What is Java? Definition, Meaning & Features of Java Platforms. https://www.guru99.com/java-platform.html, accessed: 2021-04-21.

[15] Java's 20 Years Of Innovation. https://www.forbes.com/sites/oracle/2015/05/20/javas-20-years-of-innovation/, accessed: 2021-04-21.

[16] Introduction. https://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/whatismatlab.htm, accessed: 2021-04-24.

[17] What Is MATLAB? https://ch.mathworks.com/discovery/what-is-matlab.html, accessed: 2021-04-24.

[18] MATLAB. https://en.wikipedia.org/wiki/MATLAB, accessed: 2021-04-24.

[19] OpenCV High-level GUI. https://docs.opencv.org/master/d7/dfc/group__highgui.html, accessed: 2021-03-14.

[20] OpenCV Documentation High-level GUI. https://docs.opencv.org/4.5.0/d7/dfc/group__highgui.html, accessed: 2021-04-08.

[21] OpenCV Documentation Qt New Functions. https://docs.opencv.org/3.4/dc/d46/group__highgui__qt.html, accessed: 2021-05-02.

[22] OpenCV OpenCV configuration options reference. https://docs.opencv.org/master/db/d05/tutorial_config_reference.html, accessed: 2021-03-14.

[23] OpenCV Adding a Trackbar to our applications! https://docs.opencv.org/3.4/da/d6a/tutorial_trackbar.html, accessed: 2021-04-08.

[24] What is Augmented Reality. https://arenabled.com/, accessed: 2021-05-06.

[25] Azuma, R. T.: A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, ročník 6, č. 4, 1997: s. 355–385.

[26] ARCore overview. https://developers.google.com/ar/discover, accessed: 2021-05-06.

[27] Butterfield, A.; Szymanski, J.: head-up display. 2018, doi:10.1093/acref/9780198725725.013.2102. Dostupné z: https://www.oxfordreference.com/view/10.1093/acref/9780198725725.001.0001/acref-9780198725725-e-2102

[28] Rizvi, D. Q.: A Review on Face Detection Methods. *Journal of Management Development and Information Technology*, ročník 11, 02 2011.

[29] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, ročník 1, 2001, s. I–I, doi:10.1109/CVPR.2001.990517.

[30] OpenCV Face Detection using Haar Cascades. https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html, accessed: 2021-03-05.

[31] OpenCV Histogram Equalization. https://docs.opencv.org/3.4/d4/d1b/tutorial_histogram_equalization.html, accessed: 2021-03-15.

[32] OpenCV Documentation Image Processing, Histograms. https://docs.opencv.org/master/d6/dc7/group__imgproc_hist.html#ga7e54091f0c937d49bf84152a16f76d6e, accessed: 2021-03-15.

[33] Gupta, V.: Face Detection – OpenCV, Dlib and Deep Learning (C++ / Python). https://learnopencv.com/face-detection-opencv-dlib-and-deep-learning-c-python/, 10 2018, accessed: 2021-03-21.

[34] Neural Networks. https://www.ibm.com/cloud/learn/neural-networks, accessed: 2021-05-06.

[35] AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference? https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks, accessed: 2021-05-06.

[36] deng, J.; Sun, Y.; Liu, Q.; aj.: Low rank driven robust facial landmark regression. *Neurocomputing*, ročník 151, 2015: s. 196–206, ISSN 0925-2312, doi:https://doi.org/10.1016/j.neucom.2014.09.052. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0925231214012569

[37] Facemark : Facial Landmark Detection using OpenCV. https://learnopencv.com/facemark-facial-landmark-detection-using-opencv/, accessed: 2021-05-02.

[38] OpenCV cv::face Namespace Reference. https://docs.opencv.org/master/d4/d48/namespacecv_1_1face.html, accessed: 2021-05-02.

[39] Dlib 68 points Face landmark Detection with OpenCV and Python. https://www.studytonight.com/post/dlib-68-points-face-landmark-detection-with-opencv-and-python, accessed: 2021-05-02.

[40] Tzimiropoulos, G.; Pantic, M.: Optimization Problems for Fast AAM Fitting in-the-Wild. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2013.

[41] Kazemi, V.; Sullivan, J.: One Millisecond Face Alignment with an Ensemble of Regression Trees. In *CVPR*, 2014.

[42] Ren, S.; Cao, X.; Wei, Y.; aj.: Face Alignment at 3000 FPS via Regressing Local Binary Features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[43] OpenCV Mat - The Basic Image Container. https://docs.opencv.org/master/d6/d6d/tutorial_mat_the_basic_image_container.html, accessed: 2021-04-03.

[44] OpenCV Documentation cv::Mat Class Reference. https://docs.opencv.org/3.4/d3/d63/classcv_1_1Mat.html, accessed: 2021-04-03.

[45] OpenCV Basic Drawing. https://docs.opencv.org/3.4/d3/d96/tutorial_basic_geometric_drawing.html, accessed: 2021-04-03.

[46] OpenCV Documentation Drawing Functions. https://docs.opencv.org/3.4/d6/d6e/group__imgproc__draw.html, accessed: 2021-04-03.

[47] cv::VideoCapture Class Reference. https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html, accessed: 2021-04-10.

[48] Additional flags for video I/O API backends. https://docs.opencv.org/3.4/dc/dfc/group__videoio__flags__others.html, accessed: 2021-04-10.

[49] Radke, R. J.: CHAPTER 1 - Multi-View Geometry for Camera Networks. In *Multi-Camera Networks*, editace H. Aghajan; A. Cavallaro, Oxford: Academic Press, 2009, ISBN 978-0-12-374633-7, s. 3–27, doi:https://doi.org/10.1016/B978-0-12-374633-7.00003-3. Dostupné z: https://www.sciencedirect.com/science/article/pii/B9780123746337000033

[50] Geometric Image Transformations. https://docs.opencv.org/3.4/da/d54/group__imgproc__transform.html, accessed: 2021-04-11.

[51] Operations on arrays. https://docs.opencv.org/3.4.1/d2/de8/group__core__array.html, accessed: 2021-04-11.

[52] 2D Projective Geometry and Transformation. https://fzheng.me/2016/01/14/proj-transformation/, accessed: 2021-04-23.

[53] Hrastnik, M.; Shimamura, C.: Fast and robust face detection and tracking. https://github.com/hrastnik/face_detect_n_track, accessed: 2021-03-21.

Dodatok **A**

Zoznam použitých skratiek

OpenCV Open Source Computer Vision Library

OS Operačný Systém

API Application Programming Interface

FPS Frames Per Second

GUI Graphical User Interface

STL Standard Template Library

GPU Graphics Processing Unit

pip Python package manager

PyPi Python Package index

JDBC Java Database Connectivity

ODBC Open Database Connectivity

IoT Internet of Things

LBPH Local Binary Pattern Histograms

AR Augmented Reality

VR Virtual Reality

HUD Head-Up Display

NN Neural Network

ANN Artificial Neural Network

DNN Deep Neural Network

AAM Active Appearance Model

LBF Local Binary Features

RCPR Robust Cascade Pose Regression

ROI Region Of Interest

Dodatok **B**

Obsah priloženého USB Flash disku

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresár so spustiteľnou formou implementácie
- `src`
  - `impl` ..... zdrojové kódy implementácie
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF
  - `thesis.ps` ..... text práce vo formáte PS