VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE OCHRANNÝCH POMŮCEK V OBRAZOVÉM SIGNÁLU

DETECTION OF PROTECTIVE MEANS IN VIDEO SIGNAL

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. VOJTĚCH BURDÍK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JIŘÍ PŘINOSIL, Ph.D.

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Vojtěch Burdík *ID:* 115158
Ročník: 2 *Akademický rok:* 2013/2014

NÁZEV TÉMATU:

Detekce ochranných pomůcek v obrazovém signálu

POKYNY PRO VYPRACOVÁNÍ:

V rámci práce proveďte experimentální návrh algoritmu pro detekci použití ochranných pomůcek na zabezpečených pracovištích pomocí obrazového signálu ze záznamových kamer. Jako sledované ochranné pomůcky přitom uvažujte ochranný oděv a přilbu. Navržený algoritmus implementuje pomocí knihovny OpenCV a otestujte na referenční sadě videonahrávek.

DOPORUČENÁ LITERATURA:

[1] Jain, A.K.:Fundamentals of Digital Image Processing, Prentice Hall, ISBN: 978-0133361650.

Termín zadání: 11.2.2013 ***Termín odevzdání:*** 30.5.2014

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.
Konzultanti diplomové práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

Tato práce se věnuje poměrně novému počítačovému oboru – počítačovému vidění. Zaměřuje se na rozpoznávání osob, určení polohy a detekce barvy oděvu umístěné na osobě. Cílem práce je sestavit algoritmus, který bude schopen vyhledat osobu v obraze a podrobit jí testování barvy oděvu a přilby. Ke zpracování obrazu byly využity funkce z knihovny OpenCV a z algoritmu byl sestaven program, který tento problém řeší. Výstupem programu je pak odpověď jakou barvu má na sobě osoba na určených místech a pokud se barva přilba a oděvu shodují, je osoba vyhodnocena jako správně oblečená. Výsledný program je poté rozebrán a části jeho kódu jsou podrobně popsány v této práci. Je zde vysvětleno, jak správně použít každou funkci OpenCV použitou v programu a jaké jsou jejich výhody při použití pro zpracování obrazových informací.

ABSTRACT

This work is devoted to the relatively new field of computer – computer vision. It focuses on the recognition of people, positioning and colour detection of clothing placed on person. The aim is to build an algorithm that would be able to locate the person in the picture and would make colours tests of clothing and helmets. For image processing were used OpenCV library functions and from algorithms was compiled program solving this problem. The output of the program is the answer, what colour is person at stated locations wearing, and if clothing and helmet are the same colour, the person is evaluated as properly dressed. The resulting program is then disassembled and parts of the code are in detail described in this work. There is explained how to use correctly each OpenCV function used in program.

KLÍČOVÁ SLOVA

Knihovna OpenCV, zpracování videa a obrazu, detekce osob, rozpoznávání oděvu, určení barvy objektů, segmentace obrazu, programování v jazyce C++.

KEY WORDS

OpenCV library, Video and Image processing, Detection of people, Recognition of clothing, Determining colors of objects, Image segmentation, Programming in C++.

BIBLIOGRAFICKÁ CITACE

BURDÍK, V. *Detekce ochranných pomůcek v obrazovém signálu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 40 s. Vedoucí diplomové práce Ing. Jiří Přinosil, Ph.D..

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Detekce ochranných pomůcek v obrazovém signálu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil(-a) autorská práva třetích osob, zejména jsem nezasáhl(-a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(-a) následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonu (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ……. .....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu své diplomové práce Ing. Jiřímu Přinosilovi, Ph.D., za odbornou pomoc a cenné rady při zpracování této práce.

V Brně dne ……. .....

podpis autora

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

OBSAH

ÚVOD

Sledování míst a pracovišť bezpečnostními kamerami je dnes samozřejmosti v každé firmě, která dbá na pracovní morálku, bezpečnost a pořádek na pracovišti. V mnoha podnicích je nutné dbát na zvýšenou bezpečnost a kontrolovat pracovníky, zda jsou korektně oděni, aby při práci nedošlo ke zranění nebo vážnějšímu úrazu. Příkladem může byt správný pracovní oděv či vhodné ochranné prostředky. Bohužel člověk není dokonalý a může se stát případ, kdy si na sebe zapomene dotyčná osoba pracující v nebezpečných podmínkách obléci některý z důležitých ochranných prostředků a v nebezpečí se rázem může ocitnout lidský život. Přitom by tomu mohl zamezit systémem sledovacích kamer zapojených spolu do zařízení zpracovávající tento obraz a vyhodnotit vzniklou nebezpečnou situaci. Při detekci problému se pracovník upozorněn pomocí určité signalizace a vzniklá situace by mohla být včas vyřešena.

Následující práce je zaměřena na řešení podobné situace pomocí počítačového programu. Primárním úkolem v práci je tento program navrhnout a realizovat pomocí knihovny OpenCV a programovacího jazyku C++.

1 VYUŽITÍ OBRAZOVÝCH INFORMACÍ

1.1 Počítačové vidění (Computer vision)

Počítačové vidění je věda, která vyvíjí prostředky pro interakci počítače/stroje s okolním prostředím s použitím běžných senzorů (kamer nebo infračervených čidel). Počítačové vidění umožní neživému stroji částečné napodobení lidského vidění. Robot, jakožto stroj složený z počítače a čidel, se může následně sám pohybovat v prostoru a rozhodovat se o následných akcích. Počítačové vidění je důležité pro vytvoření reálné umělé inteligence, stroj musí umět vnímat okolní prostředí.

Cílenější studium tohoto oboru začalo až v pozdních 70. letech, kdy počítače již dokázaly zpracovávat velké množství dat. Od té doby vzniklo množství metod pro řešení nejrůznějších dobře definovaných úloh počítačového vidění, ve kterých byly metody řešení často velmi specifické pro danou úlohu, a jen zřídka je bylo možné zevšeobecnit pro širokou škálu aplikací. Mnohé z těchto metod a aplikací jsou stále ve stavu základního výzkumu, ale mnoho metod si již našlo cestu do komerčních produktů, kde často tvoří součást větších systémů, které mohou řešit složité úkoly (např. v lékařství, kontroly jakosti). Ve většině praktických aplikací počítačového vidění jsou použity počítače předem naprogramované k řešení konkrétního úkolu, ale stále běžnější jsou také metody založené na učení.

Příklady nasazení počítačového vidění:

- ovládání procesů (např. průmyslový robot, autonomní vozidla)
- detekce událostí (např. pro vizuální sledování a počítání lidí)
- pořádání informací (např. pro indexování databáze obrazů a obrazových sekvencí)
- trojrozměrné modelování objektů nebo prostředí (např. průmyslové kontroly, lékařské analýzy obrazu nebo topografické modelování)

Počítačové vidění může být také popsáno jako doplněk či opak biologického vidění. U biologického vidění je zkoumáno vizuální vnímání člověka a různých zvířat, což má za následek tvorbu modelů popisujících jak tyto systémy pracují ve smyslu fyziologických procesů. U počítačového vidění se na druhou stranu studují a popisují umělé systémy vidění, které jsou implementovány v softwaru či hardwaru. Výměny a aplikace vědomostí mezi obory biologického a počítačového vidění se projevily zlepšením rozvoje obou těchto oblastí.

Počítačové vidění řeší úlohu vytvoření popisu objektů v obraze. Počítačové vidění vytváří 2D nebo 3D modely z obrazových dat. V poslední době je celkem žádané vytváření 3D modelů především ve strojírenském průmyslu, zdravotnictví nebo při počítačové rekonstrukci historických předmětů. K převodu objektu do počítačového 3D modelu je zapotřebí více snímacích zařízení rozprostřených okolo skenovaného objektu nebo použití pouze jedné kamery otáčející se okolo objektu se znalostí změny polohy.

1.2 Zpracování obrazu (Image processing)

Zpracování obrazu značnou částí zasahuje do počítačového vidění. Zpracování a analýza obrazu se zaměřuje na přeměnu jednoho 2D obrazu v jiný se stejným rozměrem (změna kontrastu, jasu, korekce chyb v obrazu, odstranění šumu) nebo na geometrickou transformaci obrazu (otáčení, přemisťování části scény). To znamená, že zpracování obrazu a analýza nevyžaduje znalost obsahu obrazu.

Nejběžnějším typem zpracování obrazu je digitální fotografie. Při tomto procesu je obraz zachycen pomocí fotoaparátu nebo digitální kamery, a aby bylo možné vyrobit z fotografie fyzický obraz, musí být obraz vhodně zpracován. U digitálních fotografií je obraz uložen jako počítačový soubor obrazového typu. Tento soubor je přeložen pomocí fotografického softwaru a je vygenerován skutečný obraz. Před odesláním na tiskové zařízení je možné upravit vlastnosti fotografie, jako jsou barvy, expozice nebo stínování, které jsou zachyceny v okamžik focení (Obr. 01).

Postup při zpracování obrazu:

- snímání obrazu a digitalizace, uložení obrazu v počítači
- předzpracování – základní úprava
- rozdělení obrazu na objekty
- popis objektů
- porozumění obsahu obrazu (klasifikace objektů)

Obr. 01: Ukázka zpracování obrazu: a) překrytí barvou, b) kontrast, c) hranový detektor

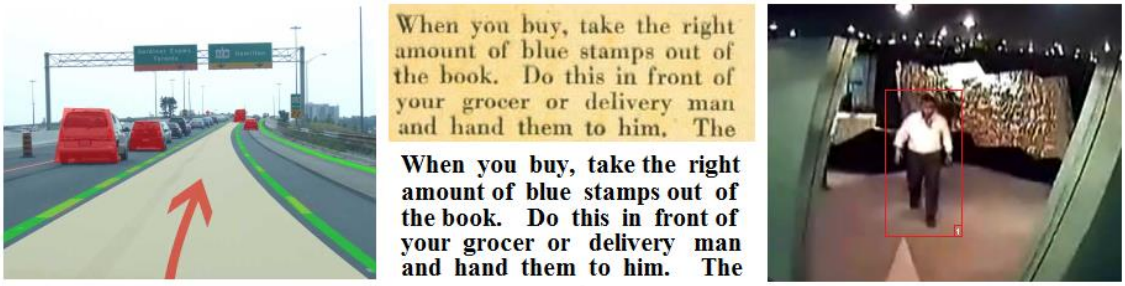
1.3 Rozpoznávání objektů (Object detection)

Rozpoznávání objektů v obrazu je důležitým vědním oborem. Setkáváme se s ním na každém kroku. Předmětem zpracování a případné rozpoznání objektu je obrazová informace o reálném světě, která do počítače vstupuje nejčastěji kamerou. Zpracovávání probíhá na základě předem definovaných pravidel, při kterých se obraz skenuje, hledají se důležité a jedinečné rysy. Získaná data jsou dále reprezentovaná ve formě výstupu (textová informace, označení oblasti, aj.).

Klasickým problémem počítačového vidění, zpracování obrazu a strojového vidění je zjišťování, zda obrazová data obsahují některý konkrétní objekt, funkci nebo činnost. Tento úkol je obvykle možné vyřešit robustně a bez zásahu člověka, ale dosud není tento problém uspokojivě vyřešen pro obecný případ, tj. libovolný objekt v libovolné situaci. Stávající metody pro řešení tohoto problému mohou v nejlepším případě vyřešit pouze konkrétní případy, jako jsou jednoduché geometrické objekty, lidské tváře, tištěné nebo ručně psané znaky, vozidla, a to ve specifických situacích, typově popsaných, za dobrého definovaného osvětlení pozadí a jen v určité poloze vztažené ke kameře. [1]

Další druhy problémů [1]:

- *Rozpoznávání:* může být rozpoznán jeden nebo několik předem definovaných nebo naučených objektů či tříd objektů, obvykle spolu s jejich pozicí ve 2D obrazu
- *Identifikace:* rozpoznává se instance určitého typu objektu
- *Detekce:* v obrazových datech jsou vyhledávány konkrétní události



a) *b)* *c)*

Obr. 02: Ukázky použití rozpoznávání obrazu: a) navigace automobilu bez řidiče, b) rozpoznávání textu, c) detekce pohybu v bezpečnostních kamerách

Využití rozpoznávání obrazu (viz Obr. 02):

- *Navigace aut bez řidiče* – pomocí webkamer je možné snímat obraz před a za autem a pomocí vyhodnocení dat určit následné korekce jízdy
- *Rozpoznávání psaného textu (OCR)* – možnost psaní na dotykových obrazovkách a následný převod do textové formy, překládání knih na jejích elektronické verze
- *Rozpoznání čárových kódu* – bez nutnosti mít speciální přístroj; existují programy pro mobilní telefony, kde se pomocí webkamery nasnímá kód, dekóduje se a pokud má mobilní přístroj přístup na internet, zobrazí se aktuální informace o čárovém kódu
- *Použití v kamerách* – detekce pohybu v bezpečnostních kamerách, rozeznávání SPZ na autech v kamerách policejních radarů nebo v automatizovaném systému výběru mýtného, rozeznání tváří u přístupu k bezpečnostním trezorům
- *Augmentovaná realita* – v chytrých mobilních telefonech je možné zobrazit pomocí kamery okolní svět doplněný daty staženými z internetu
- *Detekce buněk nebo tkání u lékařských snímků* – rozeznání rakovinných buněk od buněk zdravých

1.4 Detekce pohybu (Motion detection)

Tento obor velkou měrou zasahuje do oboru rozpoznávání objektů. Prakticky je detekce pohybu založena na určení místa změny obrazu v sérii po sobě jdoucích snímků. Místo změny je lokalizováno, zaměřeno a následně jsou provedeny další akce jako vykreslení míry změn v obraze či přiblížení objektu pro jeho následnou identifikaci. Důležité je, aby se snímací zařízení nepohybovalo a bylo stále stacionární, jinak by byl detekován pohyb všeho ve snímaném prostoru.

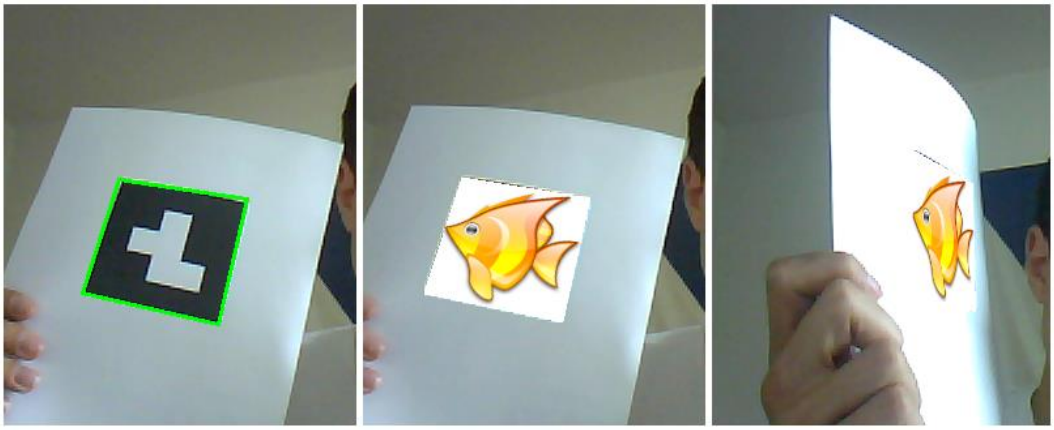
Detekce pohybu se nejčastěji využívá u bezpečnostních kamer, které reagují na pohyb v jejich zorném poli. Rovněž se používá v kvalitních fotoaparátech při focení, kde detekce pohybu zabraňuje vyfocení rozmazané fotografie.

1.5 Rozšířená realita (Augmented reality)

Rozšířená realita (AR) je variace virtuálního prostředí nebo také virtuální reality, jak se obvykle nazývá. Technologie virtuálního prostředí zcela ponoří uživatele do umělého prostředí. Zatímco je uživatel uvnitř prostředí, nevidí skutečný svět kolem sebe. Oproti tomu AR umožňuje uživateli vidět reálný svět doplněný o virtuální objekty, které jej překrývají nebo se jinak skládají s reálným světem. Proto AR doplňky reality spíše doplní, než zcela nahradí reálný pohled. Virtuální objekty jsou spojeny ve 3D prostoru s reálným světem a vztahují se k reálným předmětům. AR je spojení mezi virtuálním prostředím (kompletně umělé) a obrazovým přenosem (úplně reálné). [2]

V mobilních telefonech a „chytrých“ kamerách je snímaná scéna (například nějaký významný objekt, hrad či zámek) rozpoznána a na základě získaných informací ze scény, případně i z jiných zdrojů (GPS, kompas), je vytvořena virtuální nadstavba, která se na obrazovce zařízení zobrazí spolu s doplňkovými informacemi. Zdroj těchto informací může být databáze uložená v zařízení nebo se může stahovat online přímo z internetu pomocí bezdrátového připojení.

„Marker“ (Obr. 03 a), jak se nazývá obrazec či piktogram vyjadřující určitý význam, může být součástí dlaně a na základě rozpoznání „markeru“ a jeho pohybu může být prováděna interakce s jiným zařízením, například ovládáním robotické paže. Tohoto jevu se využívá u nejnovějších herních konzolí, kde již není nutné kupovat drahé herní ovladače a je možné použít část těla jako elementu, který se objeví ve hře a ovládá určitý virtuální předmět. Pomocí AR dále můžeme nahradit tyto „markery“ v počítačovém pohledu jinými objekty, jejichž vytvoření v realitě by nebylo ani možné.



a) *b)* *c)*

Obr. 03: Ukázka rozšířené reality

Na Obr. 03 je znázorněna jednoduchá ukázka rozšířené reality. „Marker“ vytištěný na papíru nebo i jinak zobrazený, je pomocí počítačového vidění rozpoznán, je rozeznán druh piktogramu a pomocí 2D kalibrace je určen jeho úhel natočení a poloha vzhledem ke kameře. Jeho obrys se poté pomocí získaných informací z počítačového zpracování vyplní texturou, obrázkem (viz obr. 03b) nebo libovolným 3D objektem. Tento 3D objekt i obrázky se natáčí a pohybují podle snímané polohy piktogramu a se změnou úhlu natočení se mění i parametry vykreslovaného objektu. To platí ve všech osách natočení piktogramu (viz Obr. 03 c).

2 PROGRAMOVÉ PROSTŘEDÍ

Samotné programování funkcí pro práci s obrazovými daty by bylo příliš náročné a zdlouhavé. Proto vznikají projekty, které se snaží komplexně sdružit řadu náročných funkcí. Komunita programátorů připojená k určitému projektu neustále vylepšuje algoritmy těchto funkcí a optimalizuje je pro rychlejší práci a s dalšími verzemi projektu se přidávají nové funkce. Cílem je maximálně zjednodušit programování složitých aplikací a přiblížit použití jednotlivých funkcí začínajícím programátorům. Funkce jsou definovány v knihovnách a po připojení knihovny k projektu je ve vývojovém prostředí možné vybírat z jednotlivých funkcí. Často jsou funkce v knihovnách dobře dokumentované a jsou u nich uvedeny i příklady použití a ukázkové kódy. Také je možné dohledat zdrojový kódy funkcí a pomáhat při vývoji. Nejrozšířenější knihovnou v oboru zpracování obrazu je knihovna OpenCV

2.1 Knihovny pro zpracování obrazu OpenCV

Odkaz na stránky projektu: http://www.opencv.org/

OpenCV, neboli „Open Source Computer Vision“, je rozsáhlá knihovna funkcí pro zpracování obrazu, extrahování informací a práci s obrazovými daty šířená svobodně pod licencí BSD. Je možné jí svobodně používat, šířit a upravovat pro akademické i komerční použití. OpenCV byla od roku 2000 vyvíjena Intelem, který ji od roku 2006 dále nepodporoval a uvolnil její zdrojové kódy volně ke stažení. Vývoje se následně ujala společnost Willow Garage, která ji vyvíjí dodnes pod licencí BSD, což umožňuje její bezplatné použití i pro komerční a vědecké účely. Tato firma se specializuje na výrobu inteligentních personálních robotů a softwaru pro jejich ovládání. K ovládání robotů používá mimo jiné i prostředky OpenCV.

Knihovna je napsána v prostředí C a C++, což zajišťuje podporu na platformách operačních systému Windows a Linux (Unix). Po kompilaci je možné využít knihovnu i v jiných programovacích prostředích, zejména Python, Octave nebo Matlab. Existují i nadstavby, které umožňují využít dané funkce i v jiných jazycích jako jsou: „OpenCVDotNet“ pro .NET aplikace, „Emgu CV“ pro aplikace psané v C# nebo Visual Basicu [3]. Tyto odvozené knihovny však nejsou vytvářeny vývojáři OpenCV a podpora závisí na jejích tvůrcích.

Stručný přehled možností v OpenCV:

- načítání videa ze snímacího zařízení nebo obrazových či video souborů z disku
- ukládání a konverze video souborů
- úprava obrazu (jas, kontrast, změna barevnosti, spektra barev, filtry pro vyhlazení nebo rozmazání, změna velikosti obrazu, apod.)
- extrahování dat z obrazu (textura, hrany, velikost objektu, orientace, směr pohybu, …)
- vytváření 3D scény ze dvou 2D video zdrojů

2.2 Instalace knihovny OpenCV

Instalace OpenCV je velmi jednoduchá a při použití běžného operačního systému a vývojového prostředí není potřeba kompilovat zdrojové kódy. V automatickém instalátoru OpenCV jsou obsaženy binární knihovny pro nejčastěji využívané vývojové prostředí, samotný zdrojový kód a několik tutoriálů s příklady použití OpenCV knihovny. Nejjednodušší cestou pro zprovoznění OpenCV v počítači je stáhnout si instalační soubor z oficiálních stránek projektu (v našem případě soubor „opencv-2.4.9.exe“). Při instalaci je nutné pouze zadat umístění, kam se soubory rozbalí (obvykle do složky „C:\“) a následně je potřeba soubory nalinkovat do nastavení projektu ve vývojovém prostředí podle návodu níže.

Nastavení parametrů ve vlastnostech projektu [4]:

- Přidat mezi složky ve „VC++ Directories“:
  - Include Directories: C:\OpenCV\build\include;
  - Library Directories: C:\OpenCV\build\x86\vc10\lib;
  - Source Directories: C:\OpenCV\build\x86\vc10\bin;

- Přidat k souborům do „Linker > Input > Additional Dependencies“:
  - Additional Dependencies: opencv_highgui249d.lib;
    opencv_core249d.lib;
    opencv_imgproc249d.lib;
    opencv_video249d.lib;

Do záložky „Additional Dependencies“ se uvádějí pouze ty knihovny, jejichž funkce v projektu využíváme. Tyto nalinkované knihovny jsou potřeba pro úspěšnou činnost kompilátoru, avšak nejsou zahrnuty do výsledného spustitelného souboru, protože v našem případě se jedná o „debug“ kompilaci. Debug kompilace znamená, že se kompiluje jen samotný kód programu, aby bylo možné ladit program za chodu. Proto jsou použity knihovny OpenCV, jejichž název končí písmenem „d“ označující právě použití „debug“ verze knihovny.

Nastavení uživatelských proměnných v operačním systému:

- Přidat do proměnného prostředí „Počítač > Vlastnosti > Upřesnit nastavení systému > Proměnné prostředí“ novou proměnnou s cestou:
  - PATH: C:\OpenCV\build\x86\vc10\bin;

Toto nastavení umožní zkompilovanému programu automaticky vyhledat knihovny OpenCV bez toho, aby musely být přítomny přímo ve složce s programem.

2.3 Prostředí Microsoft Visual Studio 2010

Pro vytváření algoritmu využívám programovací prostředí Microsoft Visual Studio 2010 pro svou jednoduchost a velké množství nástrojů, které urychlují psaní kódu.

Původní práce byla psána v čistém C, ale jelikož knihovna OpenCV pomalu opouští C pro svou složitost implementace, bylo nutné algoritmus vypracovat nad C++. Tento jazyk je obsahuje mnoho rozšíření, díky kterým mohou vznikat i složitější funkce s jednoduchou strukturou. Nespornou výhodou je objektová orientovanost, možnost využití tříd, přetěžování funkcí a dědičnosti.

V textu jsou obsaženy části kódu, které jsou přeneseny přímo ze zdrojového kódu programu. Avšak při popisování jsou vynechány funkce a části kódu, které nemají nic společného se samotným algoritmem, ale pouze umožňují zpřehlednit výstup programu a poskytují funkce, které nejsou nutné k dosažení výsledku. Pokud je vynechána určitá funkce je možné dohledat si její popis v samotném zdrojovém kódu (příloha A.3).

Z vývojového prostředí je zachována barevnost kódu včetně tabulátorů pro přehlednější vymezení popisovaného kódu. Využito i stejné písmo, aby nebylo možné zaměnit kód s běžným textem. Význam jednotlivých barev je uveden v seznamu níže.

Barevné odlišení v popisu kódu programu:

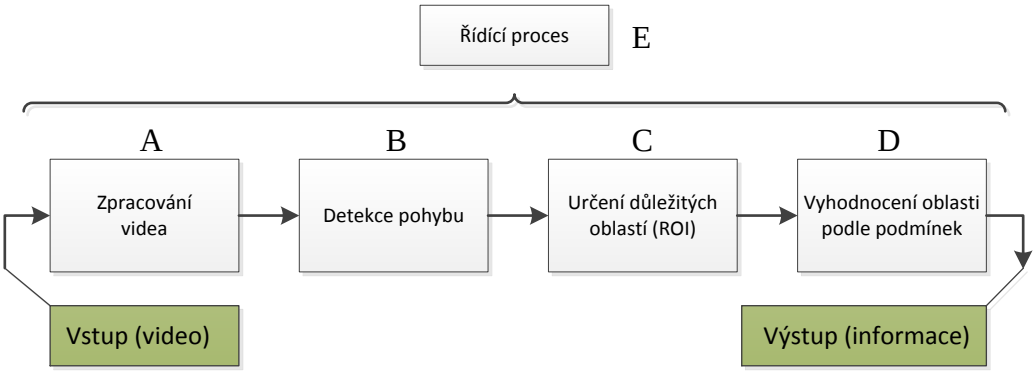
- `modré písmo` – vyznačuje datové typy, cykly (`for` nebo `while`), podmínky (`if`, `else`, `switch` a `case`) a vkládání hlavičkových souborů
- `červené písmo` – označuje textové řetězce včetně uvozovek
- `zelené písmo` – značí se tak komentáře nebo popis prováděného kódu
- `černé písmo` – běžný kód programu (podmínky, proměnné, funkce hodnoty proměnných, apod.)

3 ŘEŠENÍ DANÉ PROBLEMATIKY

Cílem práce bylo navrhnout algoritmus a sestavit program, pomocí něhož bude ve videozáznamu detekován pohybující se člověk, konkrétně pracovník jdoucí do provozu, přičemž bude vyhodnoceno, zda je správně oblečen (správný pracovní oděv, ochranné pomůcky). Bylo nutné najít postup, pomocí kterého ze vstupní videonahrávky dostaneme informaci, zda jsou splněny podmínky kladené na pracovní oděv. Byl proto vytvořen jednoduchý postup, který rozdělí rozsáhlý úkol na několik menších úkolů.

3.1 Stručný popis fungování programu

Celý algoritmus programu je rozdělen na několik bloků, jak zobrazuje Obr. 04. V bloku A vyextrahujeme ze vstupního videa vždy jeden snímek, ten zpracujeme pomocí obrazových funkcí knihovny OpenCV a výstup se pošle do bloku B. Zde je v sérii po sobě jdoucích snímku nalezena plocha, kde se pohybující osoba vyskytuje (detekce pohybu). Tyto informace o poloze přecházejí do bloku C, kde se plocha rozčlení na segmenty a v bloku D dojde k určení, zda jednotlivé segmenty odpovídají daným podmínkám. Poté je získána výstupní informace o tom, jak je daná část těla oblečena. Podmínky, jež jsou předdefinovány, představují rozhodující proces, který dle nalezené barvy odešle na výstup informaci o detekovaných pracovních pomůckách. Algoritmus je schopen rozlišit celkem 11 barev, přičemž správným barvám ochranných pomůcek odpovídají pouze žlutá nebo bílá (na základě zkoumání dodaných videonahrávek). Algoritmus tedy vyhodnotí, zda osoba má na sobě oblečen ochranný oděv nebo pomůcky. Jako oděv sledujeme oblek a kalhoty, jako ochrannou pomůckou detekujeme přilbu.

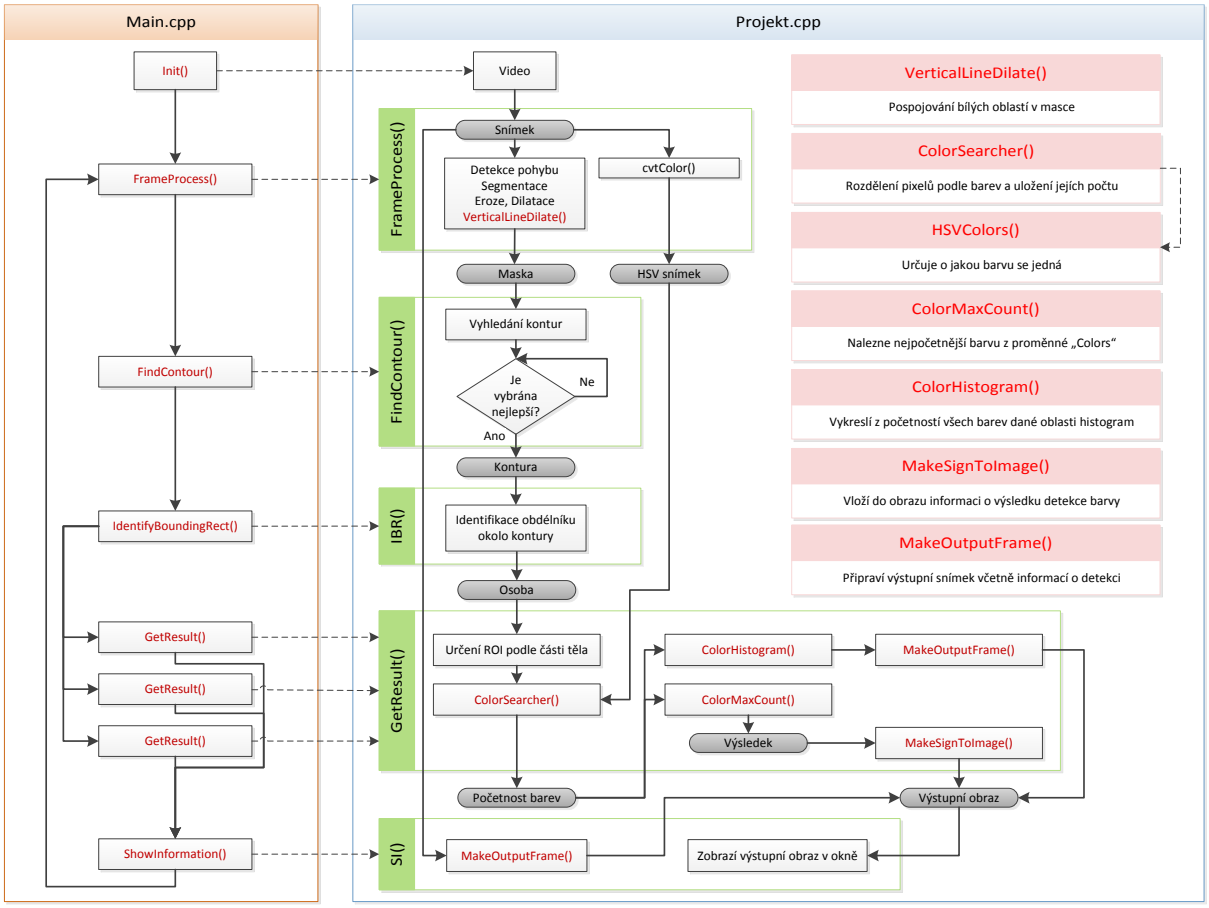


Obr. 04: Návrh činností algoritmů

Správné určení ochranných pomůcek je tedy závislé pouze na barvě nikoliv na tvaru pomůcky/objektu. Uvažujeme přitom 3 oblasti detekce, kterým odpovídá jistá část těla: přilba (hlava), oblek (hrudník) a kalhoty (nohy). Využíváme přitom znalosti anatomie člověka a tyto oblasti jsou pevně definované v určitém poměru k výšce osoby. Pro hlavu 0 % z celkové výšky, hrudník je v pozici 25 % výšky postavy a nohy jsou detekovány v 80 % výšky postavy.

3.2 Rozdělení kódu programu

Program je tudíž rozdělen na dvě samostatné entity a to „Řídící proces“ a „Ostatní procesy“. „Řídící proces“ zajišťuje postupné volání metod pro zpracování obrazu a je vyčleněn jako samostatný soubor „Main.cpp“. „Ostatní procesy“ jsou součástí třídy „Projekt“ a vystupují jako jednotlivé komponenty (funkce) v ní použité. Třída projekt je umístěna v samostatném souboru „Projekt.cpp“, což zajišťuje výbornou přehlednost algoritmu. Všechny „vlastní“ metody této třídy je možné vidět na Obr. 05Obr. 04 včetně vazeb mezi nimi.



Obr. 05: Diagram zobrazující všechny funkce programu

Na Obr. 05 vidíme kompletní schéma fungování algoritmu včetně předávání volání metod a proměnných. Diagram je rozdělen na dva oddíly odpovídající rozdělení kódu. Uvnitř třídy „Projekt“ čili stejnojmenného souboru „Projekt.cpp“ jsou obsaženy dva typy metod: metody označené zeleně mají nastaven modifikátor přístupu na `public`, jsou tedy viditelné i mimo danou třídu, metody označené červeně jsou s modifikátorem přístupu `private` a slouží proto pro volání jen v rámci třídy. Červeným písmem jsou označeny vlastní navrhnuté metody.

Třetím souborem v Programu je třída „StopWatch“, která plní funkci stopování doby běhu zpracování celého kódu. Tato třída je však doplňková a není obsažena v zadání, proto není v diagramu na Obr. 05 zahrnuta.

3.3 "header.hpp" - Hlavičkový soubor

Pro podporu různých knihoven a spolu s nimi i různých funkcí se v programu musí funkce zanést pomocí hlavičkových souborů a vložit je do zdrojového kódu, tak aby je kompilátor a vývojové prostředí mohlo načíst. Tento soubor obsahuje seznam všech globálně použitých hlavičkových souborů. Jsou definovány na jednom místě a do každého souboru projektu s příponou „`*.cpp`“ jsou voženy přes jeden jednoduchý `#include "header.hpp"`.

```
#include <stdio.h>
#include <windows.h>

#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/video/background_segm.hpp>

using namespace std;
using namespace cv;
```

Popis vložených hlavičkových souborů [5]:

- `stdio.h` – standardní knihovna obsahující funkce používané pro různé vstupní a výstupní operace; umožňuje například jednoduchou práci s textovými řetězci, zobrazování výstupu v konzolové obrazovce, čtení znaku z klávesnice nebo přeložení číselných proměnných na řetězce znaků; použité funkce: `printf()`, `sprintf()`
- `windows.h` – knihovna obsahuje funkce pracující s aplikačním rozhraním systému Windows, funkce zprostředkovávají rozhraní mezi programem a operačním systémem, z knihovny využíváme prostředky pro načtení souboru použité ve funkci `GetFileName()` v souboru „Main.cpp“
- `imgproc.hpp` – základní knihovna pro zpracování obrazu a práci s obrazovými daty, obsahuje v sobě většinu funkcí dostupných v OpenCV, mezi ně patří zejména obrazové filtry, geometrické transformace, rozšířené obrazové transformace, detekce objektů a pohybu, sledování objektů, kalibrace kamery apod., dále obsahuje definice datových typů a dynamických struktur definované knihovnou OpenCV
- `highgui.hpp` – knihovna OpenCV obsahující základní funkce pro interakci s grafickým rozhraním programu (GUI – grafické uživatelské prostředí) a funkce pro práci se záznamovým zařízením a video soubory
- `background_segm.hpp` – knihovna obsahující detekční metody pohybu v obraze, využíváme z něj detektor `BackgroundSubtractorMOG2()` uložený v proměnné `detector`

Dále se v programu používají dva jmenné prostory `std` a `cv`. Není poté nutné uvádět před každou funkcí z knihovny OpenCV direktivu `cv::` a pro standardní knihovnu direktivu `std::`.

3.4 "Main.cpp" – Spouštěcí soubor programu

Hlavní soubor projektu se stará o postupné spouštění jednotlivých metod z třídy Projekt. Je zde přítomný omezený počet proměnných (více jich není potřeba). Jsou zde inicializovány třídy pro `Projekt` a `StopWatch` a dále cesta k vstupnímu souboru `file_name`, proměnná `key`, která slouží k pozastavení zpracovávání programu, a proměnná `continues` je zde z důvodu ochrany proti chybě při vytváření nastavení třídy metodou `Init()`.

```
char  file_name[255];

int   key = 0;
bool  continues;

Projekt     projekt;
StopWatch   stopwatch;

// Funkce pro zobrazení dialogu pro výběr souboru

bool GetFileName(char *file_name)
{
   ...   // Popsáno ve zdrojovém kódu
}

// Vlastní kód hlavního programu

void main (void)
{
   GetFileName(file_name);

   continues = projekt.Init(file_name);

   while (continues && key != 27)
   {
      continues = projekt.FrameProcess();

      projekt.FindContour();

      projekt.IdentifyBoundingRect();

      projekt.GetResult("prilba",  3, 0);
      projekt.GetResult("oblek",   4, 0.25);
      projekt.GetResult("kalhoty", 5, 0.8);

      projekt.ShowInformation(stopwatch.GetTime());
   }
}
```

Po spuštění programu a inicializaci proměnných se ihned přejde k hlavní funkci `main()`. Uvnitř ní se nejprve spustí dialog pro výběr testovací videonahrávky pomocí zavolání funkce `GetFileName()` (podrobný popis funkce je uveden ve zdrojovém kódu). Po úspěšném výběru videosouboru (povoleny jsou pouze s příponou *.avi) se uloží cesta k souboru do referenční proměnné `file_name` a poté se přejde k nastavení třídy metodou `Init()`. Její jediný vstupní parametr je právě cesta k souboru, po jejím vykonání se uvnitř třídy `Projekt` soubor otevře a provede se kontrola, zda je v pořádku. V případě že ano, je navrácena hodnota `true` a může se přejít k vykonávání smyčky `while` a procházení a zpracovávání jednotlivých snímků videa.

V každém jednotlivém cyklu se načte další snímek pomocí `FrameProcess()`. Zde je přítomna také návratová hodnota typu `bool`, a to z důvodu, pokud by již nemohl být načten žádný další snímek z videa v případě „doběhnutí“ videa nakonec – cyklus a posléze program se musí ukončit.

V cyklu se dále provádí vyhledání kontur, určení rozměru a pozice kontury a vyhledání výsledků nad třemi specifikovanými oblastmi osoby. Podrobně jsou funkce popsány v následujících kapitolách.

3.5 "Projekt.cpp" – Zpracování obrazových dat

Jak již bylo řečeno na začátku kapitoly, dochází v této části programu ke zpracování vstupního obrazu a k veškerým obrazovým operacím od hledání pohybu v obraze po určení barvy jednotlivých obrazových bodů scény. Je také jádrem a stěžejní částí celého projektu, od toho je odvozen název pro soubor „Projekt.cpp“.

3.5.1 Definování proměnných

```
VideoCapture                capture;
BackgroundSubtractorMOG2    detector;

Mat      output, frame, frame_output, maska, maska2;
Rect     rect_osoba, rect_roi;

vector<vector<Point>> kontury;

bool     continues, result;
double   vel_kont_temp, vel_kont_best;
int      index_best_kont;
int      pocet_pixelu_in_roi;
int      nejpocet_barva;

string   f_name;
double   fps, ratio;

int static const  colors_count = 12;

enum { NONDEF,WHITE,GRAY,BLACK,RED,ORANGE,YELLOW,GREEN,CYAN,BLUE,PURPLE,PINK };

struct { string name; Scalar color; int count; } colors[colors_count];
```

Na samotném začátku třídy je nutné definovat a inicializovat proměnné. Všechny proměnné ve třídě mají nastaven modifikátor přístupu na `private`, proto jsou dostupné pouze uvnitř třídy. To je výhodné z hlediska ochrany před změnou z vnějšku během chodu programu. Názvy proměnných byly voleny tak, aby měly co největší vypovídající hodnotu a byl tak kód přehledný takřka pro každého.

Je zde použita řada různých datových typů jak ze standardní knihovny, tak z OpenCV. Nejzákladnějším datovým typem v OpenCV je typ `Mat`. Jedná se o 3D matici pro uložení obrazových dat, kde každé pole matice odpovídá jednomu pixelu obrazu s informacemi o jeho

barvě a dalších vlastnostech (novější obdoba staršího typu `IplImage` z knihovny pro jazyk C). Matice má v sobě definovaný typ obrazu `type`, což je údaj o barevné hloubce a počtu kanálu vloženého obrazu. V projektu jsou použity pouze dva typy obrazu: `CV_8UC3` pro barevné snímky (8-bitová barevná hloubka na 3 kanály BGR) a `CV_8UC1` pro monochromatické snímky/masky (8-bitová hloubka na 1 jasový kanál).

Dalším v programu hodně používaným datovým typem je `Rect`, který reprezentuje vlastnosti obdélníku. Nese v sobě údaje souřadnice polohy na ose `x` a `y`, a informace o šířce a výšce `width` a `height` (všechny hodnoty typu `int`).

Je zde i několik netradičních datových typů jako je `VideoCapture` pro uložení informací o zdrojovém zařízení/souboru obsahující záznam (v našem projektu se načítají videosoubory) nebo `BackgroundSubtractorMOG2`, který se stará o definování proměnné pro detekci pohybu. Dále to jsou „`vektory<vektorů<bodů>>`“ do níž se ukládají nalezené kontury, jelikož jedna kontura obsahuje více bodů `Point` a každý vektor jedné kontury je uložen v poli (vektoru) všech kontur. K jednotlivým prvkům vektorů se přistupuje stejně jako k poli, a to pomocí indexů.

Enumerátor `enum` přiřazuje pojmenovaným hodnotám určité konstantní číslo, které odpovídá určité barvě, tzv. kód barvy. Struktura `colors` obsahující informace o barvách (název, barevný kód BGR a četnost barev), index struktury poté odlišuje jednotlivé barvy. Tato „kolekce“ všech barev se naplňuje hodnotami v konstruktoru třídy a při detekci barev se ke každému indexu barvy připočítává hodnota počtu barvy do vnitřní proměnné `count`.

3.5.2 Metoda "Init()"

```
bool Init(char *file_name)
{
   capture.open(file_name);

   if (!capture.isOpened())
      return false;

   capture.read(frame);

   if (frame.empty())
      return false;

   return true;
}
```

V této části třídy dochází k otevření souboru videa (`capture.open();`) a načtení prvního snímku z videosekvence (`capture.read();`). Po otevření v programu se soubor automaticky uzamkne, aby ho nebylo možné smazat nebo upravovat v jiném programu. Samozřejmostí je ošetření, zda je možné soubor otevřít a zda byl úspěšně načten snímek, v opačném případě je navrácena hodnota `FALSE` procesu, který metodu zavolal a ten ukončí chod programu.

3.5.3 Metoda "FrameProcess()"

Tato metoda a všechny popsané níže již pracují uvnitř cyklu `while`, který zajišťuje pomocí této metody postupné načítání snímku z videa chronologicky, jak jdou za sebou.

```
bool FrameProcess()
{
   capture.read(frame);

   if (frame.empty())
   {
      continues = false;
      return false;
   }
   else
      continues = true;

   detector(frame, maska);

   threshold(maska, maska, 128, 255, CV_THRESH_TOZERO);

   erode(maska, maska, Mat(), Point(-1,-1), 2);

   maska2 = maska.clone();

   dilate(maska, maska, Mat(), Point(-1,-1), 4);

   VerticalLineDilate(maska, 16, rect_osoba.y);

   cvtColor(frame, frame, CV_BGR2HSV);

   return true;
}
```

Provede se načtení snímku a kontrola, zda je snímek přítomen (podmínka pro nalezení konce videa). Následně snímek podstupuje řadu obrazových úprav, abychom z něj získali souřadnice v obraze, kde se nachází místo pro detekci správného oblečení. Klíčovou funkci pro toto je `detector()` definovaný jako `BackgroundSubtractorMOG2`.

`BackgroundSubtractor` (zkratka BS) je algoritmus hojně využívaný pro detekci pohybu v obraze při použití statické kamery, např. počítání procházejících lidi pomocí kamery nad dveřmi nebo počítání projíždějících aut. Algoritmus pracuje na principu odečtu pozadí několika po sobě jdoucích snímku sekvence, kdy statické pozadí není nijak zvýrazněno ve výstupním obrazu algoritmu [7]. Toto zvýraznění se přenáší do obrazu s maskou `maska`, kde je jakýkoliv pohyb vyznačen bílou plochou, vše ostatní zůstává černé. V případě, že by byla kamera v pohybu, detekce by nefungovala, protože algoritmus na toto není naprogramován.

V OpenCV jsou zastoupeny tři modifikace tohoto algoritmu: `MOG`, `MOG2` a `GMG` [6]. Tyto algoritmy jsou v OpenCV implementovány až od verze 2.3 a jsou dostupné pouze s využitím knihovny pro programovací jazyk C++ (na rozdíl od C).

Obr. 06: Srovnání výstupních masek jednotlivých metod BS: a) originální snímek, b) varianta MOG, c) varianta MOG2, d) varianta GMG

Varianta `MOG` je založena na metodě modelování každého pixelu obrazu do Gaussova rozdělení. Váhy v rozdělení směsi pixelů pak představují časové proporce, po kterých pixely zůstávají v obraze bez změny (definují pozadí), čili pravděpodobnější barvy pozadí jsou ty, které zůstávají delší a statické [7]. Background Subtractor `MOG2` je vylepšená a optimalizovaná varianta předchozí metody, kde dochází i k detekci stínů. Oproti předchozí metodě se pro detekci využívá jak jasová složka, tak i barevná složka obrazu. Přitom se porovnávají rozdíly mezi pixely pozadí (statický obraz) a pohybujícím se objektem (dynamická část obrazu). Pokud jsou rozdíly v obou barevných i jasových složkách v určité mezní hodnotě, je daný pixel považován za stín (vykreslení šedou barvou na Obr. 06 c) [9]. Poslední varianta `GMG` kombinuje metody statistického odhadu pozadí a Bayesianovy segmentace. Používá několik prvních snímků k modelování pozadí a pravděpodobnostní statistiku k určení popředí pomocí segmentačního algoritmu, který identifikuje případné objekty v popředí pomocí Bayesianova rozhraní [7], [9].

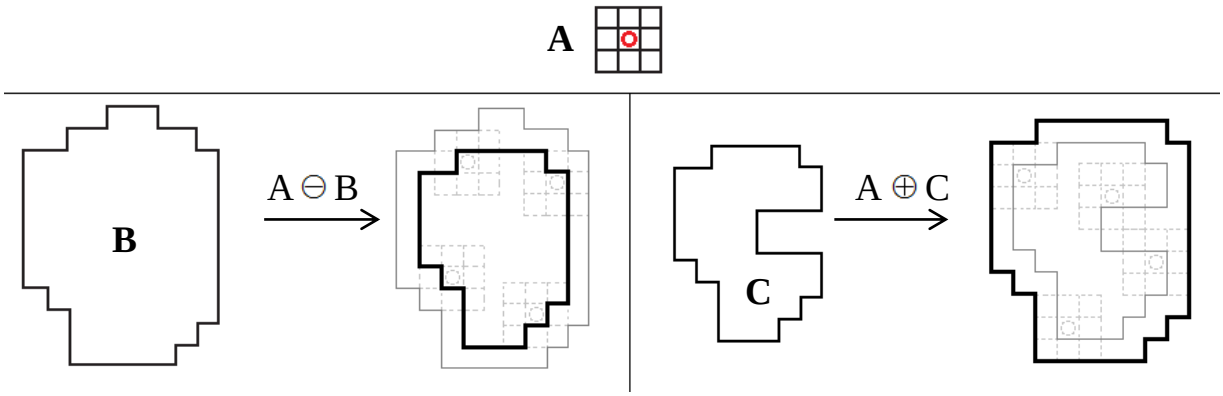
Tab. 01: Tabulka s naměřenou průměrnou dobou zpracování BS

Měřená funkce	**Čas [ms]**
`BackgroundSubtractorMOG()`	141,68
`BackgroundSubtractorMOG2()`	15,22
`BackgroundSubtractorGMG()`	88,62

Z výsledků měření doby potřebné ke zpracování jednoho snímku (uvedených v Tab. 01) je patrné, že pro reálné použití se jako nejvhodnější detektor pohybu jeví varianta `MOG2` s nejkratší dobou zpracování snímku. Je zřejmé, že tato funkce je v knihovně OpenCV výborně optimalizována a vykazuje nejlepší výsledky výstupní masky v porovnání s ostatními variantami (Obr. 06). Varianta b) na Obr. 06 vykazuje nedostatečnou detekci pohybu, protože k určení využívá pouze jasovou složku [8] a testovací nahrávky jsou potemnělé. Na druhou

stranu varianta d) z Obr. 06 vykazuje obstojnou detekci, avšak je velice citlivá na míru šumu ve vstupním obraze, a proto není vhodná pro použití v našem programu.

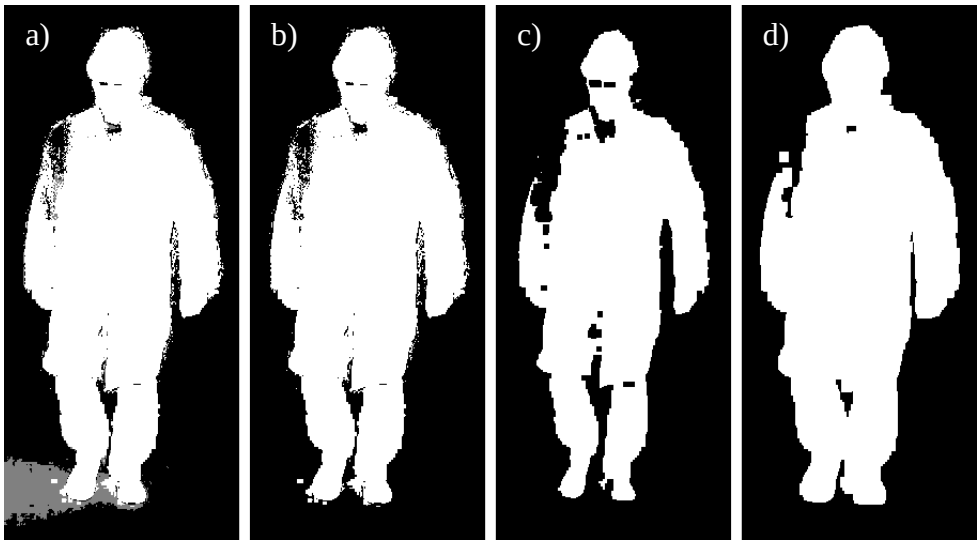
Po detekci pohybu se z masky odstraní stíny, které v tomto případě nejsou pro určení polohy osoby nutné. Provede se nad maskou prahování pomocí funkce `threshold()`, která jasové hodnoty všech pixelů masky od určité hranice vypustí (0–127) a ostatní ponechá (128–255). Maska stále vykazuje určitou míru šumu, ale tento šum je sotva patrný a jednoduše se odstraní erozí za použití funkce `erode()`. Pro úplné odstranění šumu i v hodně zarušených videonahrávkách je zapotřebí provést dvě iterace (průchody) této funkce, to se provede nastavením posledního parametru na hodnotu 2. Maska se poté uloží do proměnné `maska2` pro pozdější práci při určování barev v metodě `ColorSearcher()` (kapitola 3.5.8).



Obr. 07: Vlevo eroze, vpravo dilatace (A – element, B a C – vstupní objekty)

Po erozi došlo kromě odstranění šumu k odstranění okraje užitečné bílé plochy, proto je vhodné navrátit původní tvar pomoci dilatace. Provedou se 4 průchody funkce `dilate()`, která způsobí nabobtnání masky do původních rozměrů a vyplnění děr v bílé ploše masky, které se vytvořily nedokonalou detekcí. Příklad, jak funguje eroze a dilatace, jsou znázorněny na Obr. 07. Element A je defaultně nastaven na hodnotu 3×3 px. Při průchodu elementu obrazem se element zastaví u bílé plochy v masce, po jejíž hraně se začne posouvat určitým směrem, dokud neobkrouží celý objekt a nedostane se zpět na začátek, kde iterace začala. Při tomto pohybu se u eroze drží hrana elementu vnitřní hrany objektu a střed ořezává objekt, u dilatace se střed elementu drží vnitřní hrany plochy a přesahy elementu přes objekt jsou vyplňovány [10].

`VerticalLineDilate()` je metoda speciálně navržená pro pospojování větších souvislých ploch v masce. To je zapotřebí při špatné detekci, kdy oblečení osoby splývá s pozadím a následně není detekována souvislá kontura. Metoda tomuto jevu předchází, a je blíže popsána v samostatné kapitole 3.5.7.



Obr. 08: Postupné zpracování snímku pro následné vyhledání kontur, jednotlivé masky po operaci: a) detekci pohybu, b) prahování, c) erozi, d) dilatace

Nakonec se provede poslední obrazová operace, a to převedení aktuálního snímku z barevného prostoru BGR do prostoru HSV. Snímek následně bude zapotřebí při volání metody `ColorSearcher()`, jež je popsaná v kapitole 3.5.8.

3.5.4 Metoda "FindContour()"

Provádí se zde vyhledání všech kontur v masce a nalezení té největší z nich (nejvhodnější).

```
void FindContour()
{
   if (!continues) return;

   vel_kont_temp = vel_kont_best = 0, index_best_kont = -1;

   findContours(maska, kontury, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_TC89_KCOS);

   for (int i=0; i<kontury.size(); i++)
   {
      vel_kont_temp = contourArea(kontury[i]);

      if (vel_kont_temp >= vel_kont_best && vel_kont_temp > 40000)
      {
         vel_kont_best = vel_kont_temp;
         index_best_kont = i;
      }
   }

   if (index_best_kont >= 0)
      continues = true;
   else
      continues = false;
}
```

Metoda začíná podmínkou, pokud bylo v předchozím kroku vše v pořádku (snímek byl načten), může se pokračovat dále v hledání kontur. Dále se nastaví proměnné pro hledání největší kontury na výchozí hodnoty.

Vyhledání kontur se provádí pomocí funkce `findContours()`. Vstupním obrazem je `maska` získaná v předchozí metodě. Dalšími parametry funkce jsou `kontury` sloužící k uložení kontur, které byly nalezeny. Následuje parametr určující hloubku vyhledávání kontur. Funkce dokáže vyhledávat kontury i uvnitř již nalezených kontur, tomu zabráníme nastavením tohoto parametru na konstantu `CV_RETR_EXTERNAL`, ta definuje, že vyhledávat se bude pouze v jedné úrovni. Poslední parametr určuje metodu použitou k přeskládání nalezených bodů na výstup. Jako výchozí metoda je nastavena `CV_CHAIN_APPROX_SIMPLE`, která je použita díky jednodušší práci s jejím výstupem v podobě sekvence koncových bodů seřazených za sebou. Algoritmus hledání kontur pracuje na principu bezeztrátového přiblížení, které kóduje horizontální, vertikální a diagonální segmenty vrstevnice s vrcholovými body bílé plochy, která je aktuálně procházena. Každý nalezený obrys objektu (bílé plochy) je reprezentován seznamem vrcholů tohoto mnohoúhelníku [10].

Z nalezených kontur se musí vybrat ta nejvhodnější, přitom se předpokládá, že je to kontura s největším počtem obsažených pixelů. Pomocí cyklu `for` se projdou všechny kontury v seznamu a u každé se provede výpočet velikosti její plochy pomocí funkce `contourArea()`. Tato funkce obsahuje jediný parametr a to konturu, kterou právě procházíme, a navrací číselnou hodnotu velikosti kontur v pixelech. Poté dojde k porovnání, zda je aktuální kontura větší než doposud největší nalezená (její velikosti je uložena v proměnné `vel_kont_best`). Pokud kontura vyhovuje podmínce a je větší než zadaný práh 40 000 px, uloží se její velikost do konkrétní proměnné a její index do proměnné `index_best_kont`. Experimentálně bylo zjištěno, že ideální kontura pro detekci barvy oděvu je v rozmezí 30–120 kPx. Obr. 09 ilustruje vykreslení kontury do obrazu zelenou barvou.

Obr. 09: Vyznačení kontury v obraze (zelení obrys)

Na závěr metody se provede kontrola, zda byla opravdu nalezena nějaká kontura a zda je k ní přiřazen index v proměnné `index_best_kont`. Bez toho indexu se nemůže pokračovat k dalšímu kroku.

3.5.5 Metoda "IdentifyBoundingRect()"

Tato metoda je velmi jednoduchá a složí k určení oblasti, kde se pohybující osoba nalézá, k tomu se využije nalezená kontura z předešlého kroku.

```
void IdentifyBoundingRect()
{
   if (!continues) return;

   rect_osoba = boundingRect(kontury[index_best_kont]);

   ratio = (double) rect_osoba.height / rect_osoba.width;

   if (ratio > 2.0 && ratio < 4.0)
      continues = true;
   else
      continues = false;
}
```

Na začátku se zkontroluje, zda byla kontura v předešlém kroku nalezena. Poté se provede nalezení souřadnic a velikosti plochy, kterou kontura objímá. K tomu se používá funkce `boundingRect()`, jejíž jediný vstupní parametr je zmíněná kontura. Funkce projde všechny body kontury a určí z nich všechny potřebné parametry pro výstup, který je typu `Rect`. Tento typ má v sobě uložené přesné souřadnice na osách `x` a `y` a velikost obdélníku okolo kontury. Pokračuje se ve výpočtu poměru stran obdélníku a to z důvodu, abychom si byli jistí, že kontura odpovídá proporcím člověka. Z několika různých videonahrávek bylo zjištěno, že poměr výšky k šířce osoby je v rozmezí 2,0–4,0. Pokud detekovaná oblast (osoba) odpovídá tomuto rozmezí, přejde se k další metodě.

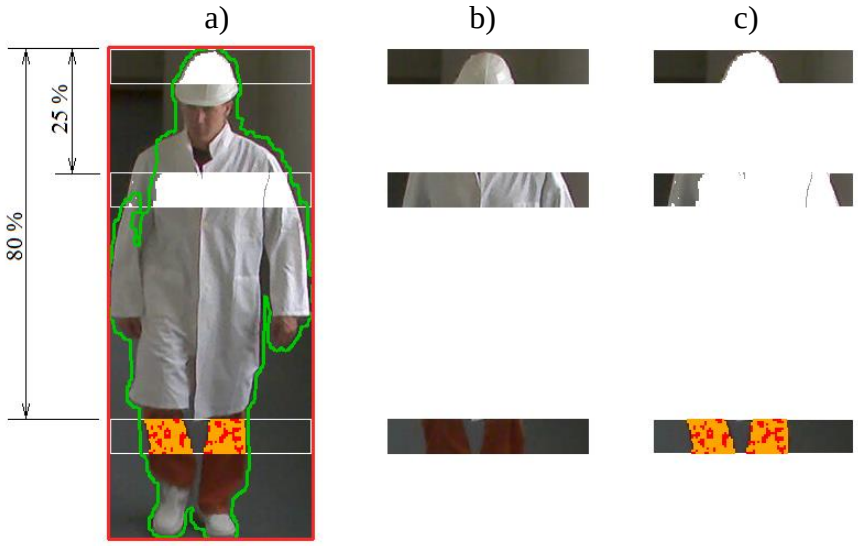
3.5.6 Metoda "GetResult()"

Zde se provádí výběr určité oblasti osoby, nad kterou je určena barva a rozhoduje se nad výsledkem, zda byl detekován ochranný oděv nebo nikoliv.

```
void GetResult(string name, int output, double top_offset)
{
   if (!continues) return;

   rect_roi = Rect(rect_osoba.x + 2,
                   rect_osoba.y + (rect_osoba.height*top_offset) + 2,
                   rect_osoba.width - 4,
                   rect_osoba.height/14);

   pocet_pixelu_in_roi = ColorSearcher(rect_roi);

   nejpocet_barva = ColorMaxCount();

   result = (nejpocet_barva == WHITE || nejpocet_barva == YELLOW);

   if (result)
      result = (colors[nejpocet_barva].count*2 > pocet_pixelu_in_roi);
}
```

Tato metoda má jako jediná vstupní parametry, které se předávají při jejím volání ze souboru „Main.cpp“. První dva parametry se týkají vykreslení histogramu do výstupního snímku (popsáno ve zdrojovém kódu), poslední parametr udává procentuální poměr k výšce osoby, kde dojde k vybrání oblasti pro určení barvy.

Proměnná `rect_roi` udává, podobně jako `rect_osoba` z předchozí metody, ohraničení okolo nějaké oblasti v obraze. Obdélník vložený do proměnné `rect_roi` má šířku stejnou jako `rect_osoba` menší na každou stranu o 2 pixely (uvažuje se negativní vliv dilatace). První parametr nastaví souřadnici na ose `x` (na šířku) podle souřadnice osoby. Druhý parametr nastavuje souřadnice na ose `y` (na výšku), a to na výšku osoby navýšenou o `top_offset`, který definuje posunutí hledací oblasti od shora dolů a může nabývat hodnot 0–1 (0 pro nejvyšší polohu a 1 pro nejnižší polohu oblasti `rect_osoba`). Offset je nastaven pro přilbu na hodnotu 0, oblek 0,25 (25 %) a kalhoty 0,8 (80 %), tak jak znázorňuje Obr. 10. Třetím parametrem je zadaná šířka a posledním parametrem je výška této oblasti a nastavuje se automaticky na 1/14 výšky osoby.



Obr. 10: Výřezy důležitých oblastí z obrazu: a) oblast rect_osoba, b) výřezy rect_roi před a c) po provedení metody ColorSearcher()

Po detekování oblasti pro určení barvy se přenesou informace do metody `ColorSearcher()`, kde dojde k postupnému prohledání všech pixelů a uložení počtu jednotlivých barev do globální struktury `colors`. Jako návratová hodnota je vrácen počet pixelů, nad kterými se provedlo určení barvy (slouží pro určení jednoznačnosti určení). Po provedení metody se ve snímku označí každý procházený pixel barvou, která byla nad ním určená (viz Obr. 10 c).

Ve struktuře `colors` jsou nyní uloženy informace o tom, kolik bylo nad danou oblastí nalezeno pixelů určité barvy. Celkem přitom rozlišujeme mezi 11 barvami. Metoda `ColorMaxCount()` prochází tuto strukturu a vyhodnotí, které barvy se napočítalo v dané oblasti nejvíce. Návratovou hodnotou je celé číslo, které odpovídá kódu barvy z enumerátoru barev (viz kapitola 3.5.1).

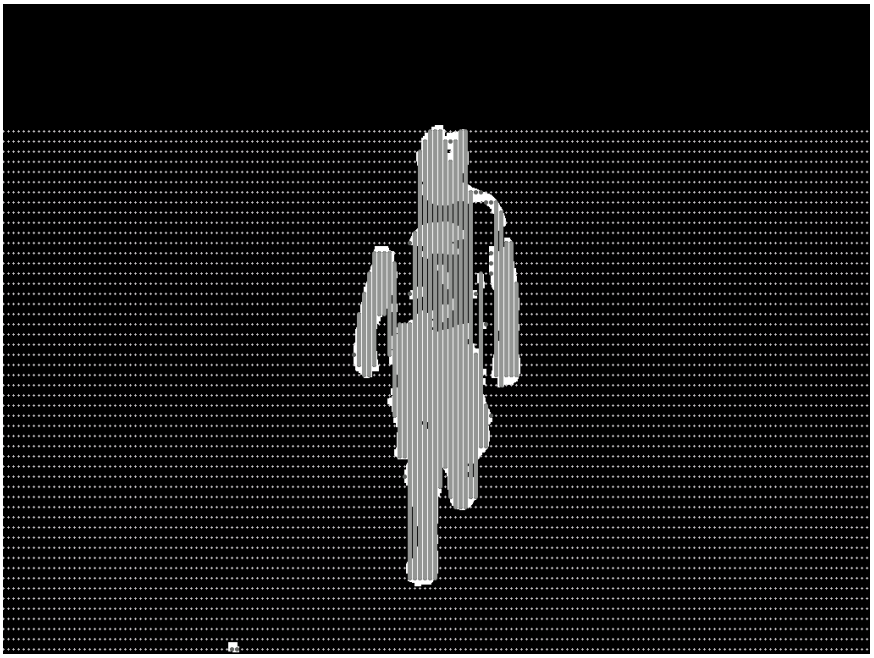
V této části programu již známe barvu, můžeme tedy rozhodnout, zda je nebo není daná část těla správně oblečena – tj. zdali obsahuje ochrannou pomůcku. Pokud je tedy kód nejpočetnější barvy shodný s kódem bílé (`WHITE`) nebo žluté (`YELLOW`) barvy, uloží se do proměnné `result` typu `bool` hodnota `TRUE`, v opačném případě `FALSE`. Za předpokladu, že byla určena správná barva, se provede kontrola jednoznačnosti určení. Kontrola spočívá v tom, že se ujistíme, že i když byla tato barva nejpočetnější, tak byla v dané oblasti zastoupena nejméně z 50 %. V případech kdy je detekce na první pohled správná, ale zjistí se, že dvojnásobek nejpočetnější barvy není větší než celkový počet procházených pixelů, je nakonec `result` nastaven na `FALSE` pro nejednoznačnost určení – k tomuto dochází hlavně v případech, kdy není osoba správně oblečena.

3.5.7 Metoda "VerticalLineDilate()“

Metoda provádí vertikální spojení bílých částí masky tak, aby maska pro detekci kontur tvořila jednu velkou nepřerušovanou (ideálně souvislou) oblast. Tato metoda předchází situaci, kdy oblečení osoby splývá s podlahou, a nejsou detekovány některé části těla.

```
void VerticalLineDilate(Mat maska, int step, int move_y)
{
   uchar maska_pixel;
   Point point1, point2, is_white;

   for (int x=0; x < maska.cols; x+=step/2)                    // Proch. po šířce
   {
      point1 = point2 = is_white = Point(0, 0);

      for (int y=move_y+step; y < maska.rows; y+=step)         // Proch. po výšce
      {
         maska_pixel = maska.at<uchar>(y, x);

         if (maska_pixel > 0)
         {
            if (!is_white.x && !is_white.y)
               is_white = Point(x, y);
            else
            {
               if (!point1.x && !point1.y)
               {
                  point1 = is_white;

                  is_white = Point(0, 0);
               }
               else
                  point2 = Point(x, y);
            }
         }
         else if (is_white.x && is_white.y)
            is_white = Point(0, 0);
      }

      if (point1.x && point1.y && point2.x && point2.y)
         line(maska, point1, point2, Scalar(150,150,150), step/2);
   }
}
```

Metoda pracuje na principu postupného procházení masky a hledání větších bílých ploch, které spojí. Procházení probíhá po sloupcích od shora dolů a procházení začíná na ose `y` (na výšku) vždy v místě nejvyššího bodu detekované osoby z předchozího snímku. Pokud se tedy osoba pohybuje směrem dolů z pohledu kamery, prostor masky nad osobou se nebere v potaz (ustupující řady teček na Obr. 11). V některých případech se v horní části obrazu nalézá bílý šum, který je nežádoucí a před tímto vylepšením značně metodu ovlivňoval a pospojovány byly i části, které nenáleží tělu osoby. Nejvyšší bod detekované osoby se do metody dostává vstupním parametrem `move_y` při volání z metody `FrameProcess()` (kapitola 3.5.3).



Obr. 11: Demonstrace výstupní masky metody VerticalLineDilate()

V metodě je definován krok `step`, jedná se rozlišovací schopnost pro zachycení bíle plochy v horizontálním směru, ve vertikálním je rozlišení poloviční. Proces procházení masky postupuje po sloupcích v definovaném kroku a v každém dalším sloupci si vynuluje proměnné `point1`, `point2` a `is_white`. `Point1` označuje první bod bílé plochy v daném sloupci, `Point2` poslední bod ve sloupci a `is_white` slouží k testování, zda se jedná o plochu, která v daném sloupci zaujímá alespoň dva po sobě testovací body (šedé tečky na Obr. 11).

Při procházení sloupce se procházejí pixely masky, a pokud je aktuální pixel světlý bod a není nastavená proměnná `is_white`, uloží se do ní souřadnice pixelu. V dalším bodu sloupce je již tato proměnná nastavená, a pokud je i nyní nalezen světlý bod, uloží si do `point1` souřadnice předešlého bodu (začátek bílé plochy). Postupuje se dále po sloupci a nyní se již každý další bod ukládá do `Point2`, avšak opět dochází k testování, zda jde o bílou plochu s velikostí nejméně dvou po sobě jdoucích testovaných bodů (pomocí proměnné `is_white`).

Po nalezení obou proměnných typu `Point` dojde ke spojení těchto bodů linkou pomocí funkce `line()` s šířkou odpovídající polovině kroku `step` a vytvoří se tak jeden spojitý celek (viz osoba na Obr. 11).

3.5.8 Metoda "ColorSearcher()"

Slouží pro vyhledávání nejpočetnější barvy ze vstupního parametru definujícího oblast v obraze, nad kterou se detekce bude provádět.

```
int ColorSearcher(Rect rect_roi)
{
   int color_def   = 0;
   int pixel_count = 0;

   uchar maska_pixel;
   Vec3b frame_pixel;

   for (int i=0; i < colors_count; i++)
      colors[i].count = 0;

   for (int x=rect_roi.x; x < (rect_roi.x+rect_roi.width); x++)
   {
      for (int y=rect_roi.y; y < (rect_roi.y+rect_roi.height); y++)
      {
         maska_pixel = maska2.at<uchar>(y, x);
         frame_pixel = frame.at<Vec3b>(y, x);

         if (maska_pixel > 0)
         {
            color_def = HSVColors(frame_pixel[0], frame_pixel[1], frame_pixel[2]);

            frame_output.at<Vec3b>(y,x) = Vec3b(colors[color_def].color[0],
                                                colors[color_def].color[1],
                                                colors[color_def].color[2]);
            colors[color_def].count++;

            pixel_count++;
         }
      }
   }
   return pixel_count;
}
```

Na začátku metody se inicializují proměnné, které jsou pro prohledávání obrazu zapotřebí. Globální struktura `colors` slouží jako počítadlo barev, přičemž se jedná o pole hodnot a každá barva (kód barvy) odpovídá jednomu indexu struktury. Tuto strukturu je nutné před započetím vyplnit nulami jako výchozí hodnoty, aby se nepočítalo s barvami detekovanými pro jinou oblast. Proměnná `color_def` v tomto případě slouží jen jako pracovní proměnná pro ukládání dočasného výsledku z metody `HSVColors()` (viz kapitola 3.5.9). Do proměnné `pixel_count` se počítá počet všech pixelů, které prošli barevnou klasifikací a na konci metody slouží jako návratová hodnota.

Následující dva cykly `for` prohledají každý řádek a sloupec oblasti obrazu definované v `rect_roi` a po jednom pixelu načítají barvu jak z obrázku, tak i z masky `maska2` (maska vytvořená po erozi, viz kapitola 3.5.3). Poté se zjišťuje, zda je v masce na určitém bodě detekována část těla pohybující se osoby čili, zda je jasová hodnota pixelu větší než 0 (0 – hodnota černé barvy, 255 – hodnota bílé barvy). Tato podmínka se provádí z důvodu, abychom do početnosti barev nezapočítávali barvu pozadí, ale jen užitečnou část obrazu. Dále

se zavolá metoda `HSVColors()` (viz kapitola 3.5.9), která zajistí roztřídění barev podle vstupních parametrů H, S a V získaných z vektoru `frame_pixel`, kde index 0 odpovídá barevnosti H, index 1 odpovídá saturaci S a index 2 odpovídá jasu V.

Po určení barvy určitého pixelu se tento označí přímo detekovanou, již kategorizovanou barvou podle Tab. 02. Toto vylepšení umožňuje přímo na snímku sledovat, jaké barvy byly v oblasti na jednotlivých pixelech detekovány (zobrazeno na Obr. 10 c).

Následně se v počítadle barev `colors[„index barvy“].count` přičte hodnota 1 pro určitou barvu vybranou indexem ve struktuře.

3.5.9 Metoda "HSVColors()"

Do metody vstupují veličiny H, S a V přímo jako parametry hodnoty odstínu, sytosti a jasu. Na základě těchto hodnot je klasifikována barva a na výstupu této metody se vrátí číselný kód barvy. Tento kód barvy odpovídá stejným barvám v celém kódu programu.

```
int HSVColors(double H, double S, double V)
{
   int color = 0;

   if (S < 80)
   {
      if       (V > 90)    color = WHITE;
      else if  (V > 40)    color = GRAY;
      else                 color = BLACK;
   }
   else
   {
      if       (H <=   4)  color = RED;
      else if  (H <=  18)  color = ORANGE;
      else if  (H <=  37)  color = YELLOW;
      else if  (H <=  80)  color = GREEN;
      else if  (H <= 100)  color = CYAN;
      else if  (H <= 140)  color = BLUE;
      else if  (H <= 157)  color = PURPLE;
      else if  (H <= 172)  color = PINK;
      else if  (H <= 180)  color = RED;
      else                 color = NONDEF;
   }

   return color;
}
```

Tato metoda je velice jednoduchá a využívá principů navržených v článku [11]. Ve zmíněné práci studovali barevný model HSV (někdy také označován HSI) a definovali parametry jednotlivých hodnot „Hue“, „Saturation“, „Value/Intensity“, které odpovídají daným barvám modelu HSV. Barevný prostor HSV je unikátním tím, že jednotlivé odstíny barev je možné definovat jako rozsahy a určovat je podle jedné proměnné. V prostoru RGB (respektive BGR) je velmi složité určit konkrétní barvu složením všech tří hodnot do jedné výsledné informace o odstínu konkrétní barvy.

V programu využíváme pouze tři barevné prostory: monochromatický pro masky, BGR pro standardní barevné snímky a HSV pro rozdělení barev. V metodě `FrameProces()` (kapitola 3.5.3) dochází za pomoci funkce `cvtColor()` k převodu mezi barevným prostorem BGR vstupního snímku a HSV prostorem na výstupu.

Obr. 12: Zobrazení snímku obsahující barevný prostor HSV

Barevná konverze z BGR do HSV se v OpenCV provádí podle následujících vzorců:

$$V = \max(R, G, B) \tag{3.1}$$

$$S = \begin{cases} 1 - \frac{\min(R,G,B)}{V} & pokud\ V \neq 0 \\ 0 & v\ ostatních\ případech \end{cases} \tag{3.2}$$

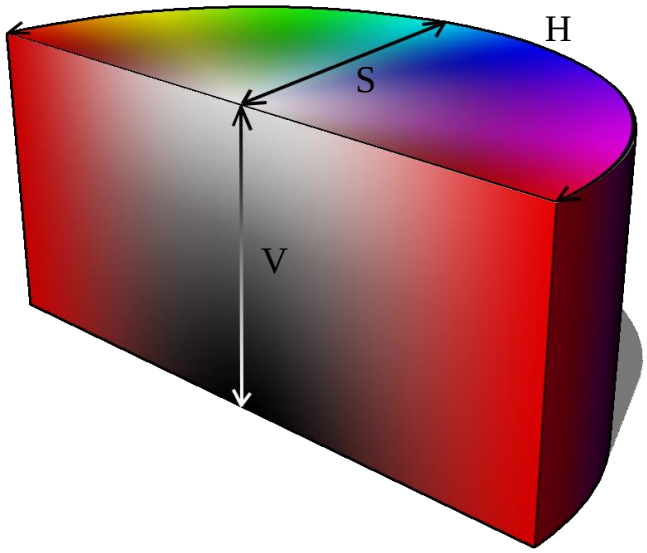
$$H = \begin{cases} 60 \cdot \frac{G-B}{S} & pokud\ je\ V\ hodnota\ R \\ 120 + 60 \cdot \frac{B-R}{S} & pokud\ je\ V\ hodnota\ G \\ 240 + 60 \cdot \frac{R-G}{S} & pokud\ je\ V\ hodnota\ B \end{cases} \tag{3.3}$$

Na výstupu této konverze jsou hodnoty V a S v rozmezí 0–1 a hodnota H je v rozmezí 0–360. Hodnota H může po výpočtu nabývat i záporných hodnot, proto se před výstupem provádí ve funkci kontrola, a pokud je vyhodnoceno H < 0, poté se H přiřadí hodnota 360. Barevná konverze dále zajišťuje přepočet na 8-bitovou barevnou hloubku podle vztahu 3.4:

$$V = 255 \cdot \mathrm{V},\ \ \mathrm{S} = 255 \cdot \mathrm{S},\ \ \mathrm{H} = \mathrm{H}/2 \tag{3.4}$$

Zde je nutné upozornit, že hodnoty H a S jsou po výstupu z funkce ve standardní 8-bitové barevné hloubce, avšak hodnota H je v rozmezí 0–180. To je dáno jednoduchým přepočtem,

kde je hodnota H podělena dvěma, aby bylo možné rozsah 0–360 uložit do 8-bitové matice obrázku [12]. Barevný prostor HSV používaný v OpenCV se od originálního modelu mírně liší a můžeme si ho představit jako polovinu válce originálního modelu HSV, kde po obvodu odečítáme hodnotu H, od okraje ke středu S a na výšku půlválce hodnotu V (viz Obr. 13).



Obr. 13: Barevný model HSV zobrazený podle definice OpenCV

Barevné rozsahy veličiny H definované v práci [11] je tedy nutné přepočítat tak, aby odpovídaly výstupním hodnotám barevného modelu HSV použitém v OpenCV. Veškeré rozsahy použité v programu znázorňuje Tab. 02.

Určení barvy v této metodě probíhá pomocí kaskádovitého porovnávání podmínek `if`. Porovnáváním se postupně vylučují barevné rozsahy, které hodnotě neodpovídají, až se dojde k rozsahu, kterým vstupní hodnotě odpovídá a do proměnné `color` se vloží kód příslušné barvy. Tato hodnota je poté navrácena metodě, kterou tuto metodu zavolala.

Nejprve se barva třídí podle saturace. Zde se rozděluje, zda jde o hodnoty chromatického nebo jasového pixelu. U jasového uvažujeme pouze jeden kanál a to je informace o světelnosti označovaný „Value“. Pro tento případ rozlišujeme pouze tři úrovně jasu: bílá, šedá a černá. U barevného pixelu nezáleží na jeho světelnosti a jeho rozřazení k určité barvě se provede podle zkoumání hodnoty barevného odstínu, označovaný jako „Hue“. Zde se rozlišuje celkem 6 rozsahů pro detekovatelné barvy: červená, oranžová, žlutá, zelená, azurová, modrá, fialová a růžová (jednotlivé barvy a rozsahy jsou uvedeny v Tab. 02).

Tab. 02: Přehled barev a jejich rozdělení podle hodnot H, S a V

Kód barvy	Název barvy	Saturation	Hue	Value
1	Bílá	0–79	–	91–255
2	Šedá	0–79	–	41–90
3	Černá	0–79	–	0–40
Kód barvy	**Název barvy**	**Saturation**	**Hue (originál)**	**Value**
4	Červená	80–255	0–4, 173–180 (0–9 °, 345–360 °)	–
5	Oranžová	80–255	5–18 (10–37 °)	–
6	Žlutá	80–255	19–37 (38–75 °)	–
7	Zelená	80–255	38–80 (76–160 °)	–
8	Azurová	80–255	81–100 (161–200 °)	–
9	Modrá	80–255	101–140 (201–280 °)	–
10	Fialová	80–255	141–157 (281–315 °)	–
11	Růžová	80–255	158–172 (316–344 °)	–

3.6 "StopWatch.cpp" – Výpočet doby zpracování

Pomocí této třídy probíhá měření doby zpracování určitého kódu, jedné funkce nebo i celé jedné smyčky cyklu `while`. Měření vždy probíhá mezi zavoláním funkcí `Start()` a `Stop()`, ve kterých se vždy načte aktuální počet taktů procesů, poté se tyto hodnoty od sebe odečtou a nakonec se vydělí rychlostí procesorů (počet taktů za 1 s) [13]. Takto získáme čas, který mezi zavoláním těchto funkcí uběhl v jednotkách mikrosekund. Pro přepočet na milisekundy vynásobíme hodnotu číslem 1000.

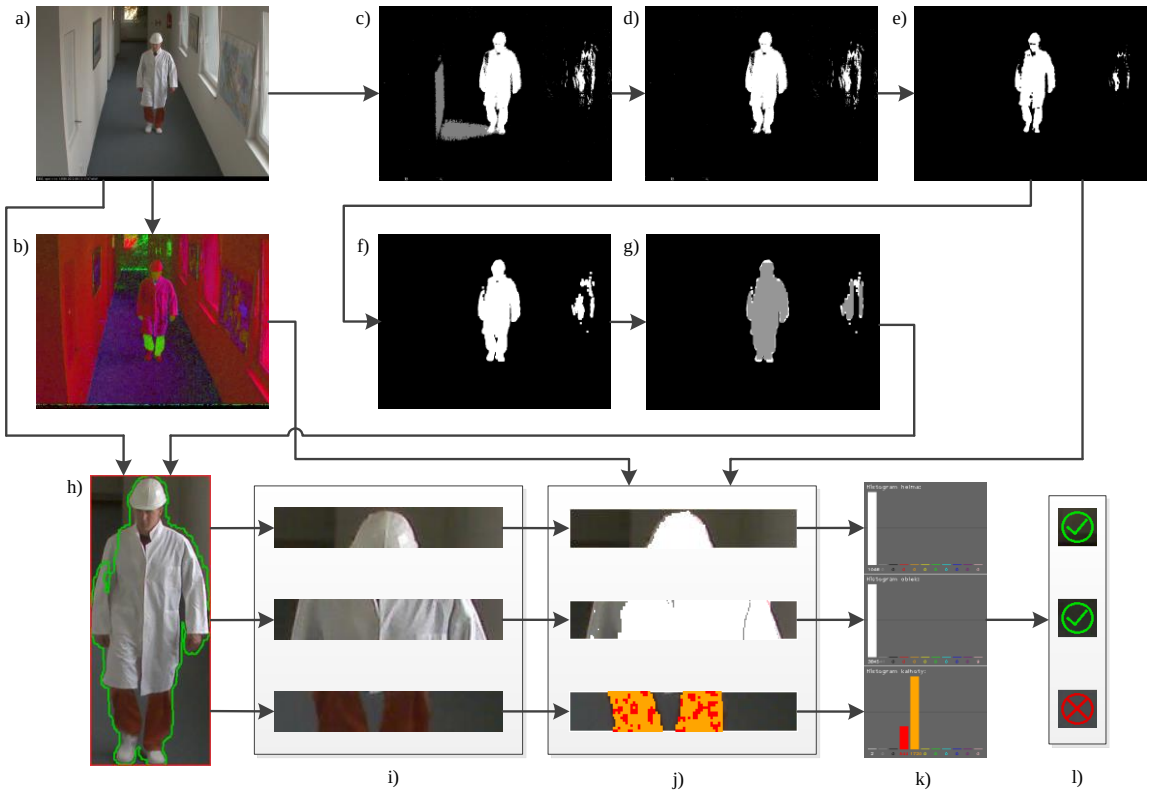
Měření se vyhodnocuje vždy po několika cyklech měření (průměrování), aby bylo zaručeno, že výsledky budou relevantní. Všechny vypočtené časy se přičítají do proměnné `time_celkovy`, dokud není dosaženo určitého počtů cyklů, přitom se počítá, kolik cyklů již proběhlo. Po určitém počtu cyklů se podělí celkový čas počtem cyklů a vynulují se proměnné pro nové počítání. Proměnná `time_counted` se naplní výsledkem a metodou `GetTime()` se tento čas může s třídy načíst.

```
private:

   // Definování proměnných třídy

   int   pocet_cyklu;
   int   pocet_cyklu_all;

   double   tick_freq;
   double   time_start, time_stop, time_celkovy;

   double   time_counted;

public:

   // Konstruktor třídy + inicializace proměnných

   StopWatch()
   {
      time_counted = 0.0;
      pocet_cyklu = 0;
      pocet_cyklu_all = 10;

      tick_freq = getTickFrequency();
   }

   // Spuštění počítání času (výchozí čas)

   void Start()
   {
      time_start = (double)getTickCount();
   }

   // Ukončení počítání času + výpočet (pokud již proběhl dostatečný počet cyklů)

   void Stop()
   {
      time_stop = (double)getTickCount();

      if (pocet_cyklu >= pocet_cyklu_all)
      {
         time_counted = time_celkovy / pocet_cyklu;

         time_celkovy = 0.0;
         pocet_cyklu = 0.0;
      }
      else
      {
         time_celkovy += ((time_stop - time_start) / tick_freq) * 1000;

         pocet_cyklu++;
      }
   }

   // Vrácení vypočtené hodnoty rozdílů časů včetně zprůměrování

   double GetTime()
   {
      return time_counted;
   }
```

3.7 Stručný přehled zpracování snímků v programu

Na Obr. 14 jsou pro přehlednost vyobrazeny všechny snímky, tak jak jsou postupně zpracovávány. Zde je jejich stručný popis:

- a) vstupní snímek z videa
- b) převedený snímek do prostoru HSV
- c) maska po detekci pohybu
- d) maska po provedení prahování (odstranění stínů)
- e) maska po erozi (ořezání)
- f) maska po dilataci (nabobtnání)
- g) maska po provedení metody VerticalLineDilate()
- h) výřez pohybující se osoby z původního snímku, zobrazení kontury
- i) získání důležitých oblastí osoby pro detekci barev
- j) důležité oblasti osoby s vykreslením detekované barvy
- k) histogram jednotlivých oblastí
- l) vyhodnocení výsledků detekce



Obr. 14: Jednotlivé snímky vytvářené při zpracování obrazu

4 TESTOVÁNÍ ALGORITMU

4.1 Testování videonahrávek

Testování práce algoritmu bylo provedeno nad sérií 26 videonahrávek (umístěné v příloze A.5). Výhodou algoritmu je automatické zobrazení dialogu pro výběr souboru pro načtení videa. Není tedy nutné program spouště z příkazové řádky a zadávat cestu jako argument nebo pokaždé kompilovat zdrojový kód s pevně nastavenou cestou k souboru.

Při vytváření algoritmu se často provádělo testování, zda je ta konkrétní úprava vhodná a zda funguje nad určitým počtem vzorků. To vedlo k častým úpravám kódu a parametrů nejrůznějších funkcí a podařilo se díky tomu vyladit program do nynější podoby.

Tab. 03: Správnosti určení programu

Název videa	Přilba		Oblek		Kalhoty	
	Barva	Určení	Barva	Určení	Barva	Určení
CAP_pop.1.1.avi	Nic	NE	Žlutá	ANO	Žlutá	ANO
CAP_pop.1.2.avi	Nic	NE	Bílá	ANO	Bílá	ANO
CAP_pop.1.3.avi	Nic	NE	Žlutá	ANO	Žlutá	ANO
CAP_pop.2.1.avi	Bílá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_pop.2.2.avi	Žlutá	ANO	Bílá	ANO	Bílá	ANO
CAP_pop.2.3.avi	Bílá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_pop.3.1.avi	Bílá	ANO	Nic	NE	Nic	NE
CAP_pop.3.2.avi	Bílá	ANO	Nic	NE	Nic	NE
CAP_pop.3.3.avi	Žlutá	ANO	Nic	NE	Nic	NE
CAP_pop.4.1.avi	Žlutá	ANO	Nic	NE	Bílá	ANO
CAP_pop.4.2.avi	Žlutá	ANO	Nic	NE	Bílá	ANO
CAP_pop.4.3.avi	Bílá	ANO	Bílá	ANO	Nic	NE
CAP_pop.5.1.avi	Nic	NE	Nic	NE	Nic	NE
CAP_pop.5.2.avi	Nic	NE	Nic	ANO	Nic	NE
CAP_pop.5.3.avi	Nic	NE	Nic	NE	Nic	NE
CAP_vz.1.1.avi	Žlutá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_vz.1.2.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.3.avi	Žlutá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_vz.1.4.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.5.avi	Žlutá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_vz.1.6.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.7.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.8.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.9.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO
CAP_vz.1.10.avi	Žlutá	ANO	Žlutá	ANO	Žlutá	ANO
CAP_vz.1.11.avi	Bílá	ANO	Bílá	ANO	Bílá	ANO

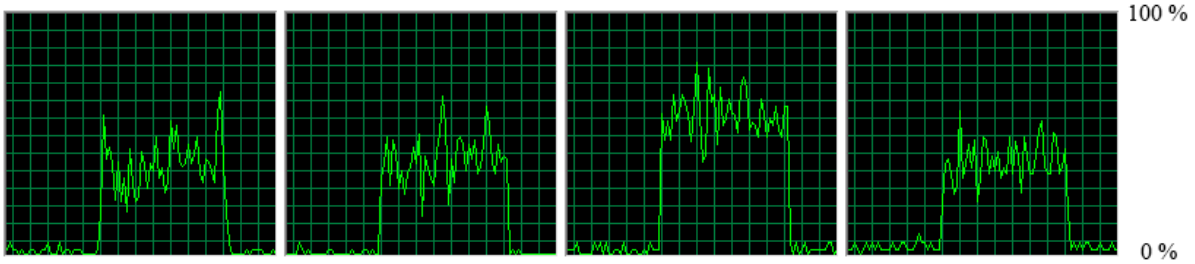
V Tab. 03 je uvedeno u kterých videí prošel program s úspěšnou detekcí, pouze u jednoho videa měl problém, protože osoba na sobě měla oblečené obyčejné triko se světlým motivem a program to vyhodnotil jako bílou barvu. Nepomohla ani podmínka jednoznačnosti, jelikož osoba splývala s podlahou a detekce zbytku trika byla nedokonalá.

4.2 Výpočetní náročnost a rychlost programu

Projekt je velmi náročný na výpočetní výkon. Při testování výkonu se měření provádělo s „debug“ kompilací a to mělo za následek vyšší nároky při volání debugovacích nástrojů. Při využití „Release“ kompilace by byl program daleko rychlejší, avšak doba zpracování u některých funkcí by nemusela být měřitelná.

Testovací sestava:

- OS: Windows 7 Professional 64-bit SP1
- CPU: Intel® Core i5-3350P @ 4 x 3.10GHz
- GPU: Nvidia GeForce GTX 650, 1 GB GDDR5
- RAM: 8,00GB Dual-Channel DDR3 @ 798MHz (11-11-11-28)
- HDD: SSD Kingston SV300S37A, 60 GB



Obr. 15: znázornění vytížení procesoru

Na Obr. 15 je zobrazen průběh programu během testování videa. Na všech čtyřech jádrech procesoru je znatelný nárůst spotřeby výkonů. Výpočet probíhal paralelně a zobrazoval se průměrnou rychlostí cca 11 FPS (snímku za sekundu). Pro srovnání snímkovací rychlost videa je 10 FPS.

Zkompilovaná „debug“ verze programu zabírá na disku 105 kB a při svém běhu zabírá přibližně 109 800 kB paměti RAM.

Z Tab. 04 je jasné, že největší poměr času spotřebovává funkce detektoru, která vyhledává pohyb v obraze. V dřívějších verzích programu byla použita funkce `convertTo()`, která se používá k úpravám jasu a kontrastu. Avšak pro její početní náročnost není v aktuální verzi programu zahrnuta.

Při průchodu více videí byla naměřena odchylka, která byla velmi malá a oproti hodnotám v Tab. 04 se lišily jen o +/- 1 %.

Doba zpracování:

- bez nalezené vyhovující kontury: 78-80 ms
- s nalezenou konturou: 95-100 ms

Tab. 04: Měření doby chodu programu

Měřená funkce	**Čas [ms]**
FrameProcess()	**78,470**
- detector(frame, maska)	16,415
- threshold(maska, maska, 128, 255, CV_THRESH_TOZERO)	0,149
- erode(maska, maska, Mat(), Point(-1,-1), 2)	3,757
- dilate(maska, maska, Mat(), Point(-1,-1), 4)	6,707
- VerticalLineDilate()	0,742
- MakeOutputFrame(maska, 9, "Maska po vlastni dilataci")	9,901
- cvtColor(frame, frame, CV_BGR2HSV)	7,363
- frame.convertTo(frame, -1, 2, 0) - nepoužívá se!	31,197
FindContour()	**2,212**
- findContours(maska, kontury, ..., ...)	2,202
- contourArea(kontury[i])	0.006
IdentifyBoundingRect()	**0,004**
- boundingRect(kontury[index_best_kont])	0.004
GetResult(string name, int output, double top_offset)	**5,748**
- ColorSearcher(rect_roi)	2,721
- ColorHistogram()	0,361
- MakeOutputFrame(ColorHistogram(), output, "Histogram " + name)	2,735
- ColorMaxCount()	0,000
- MakeSignToImage(result, ...)	0,169
Celková doba zpracování 1 snímku:	86,434

Veškerá měření času uvedené v Tab. 04 byla prováděna na nahrávce: CAP_pop.4.3.avi

ZÁVĚR

Tuto práci jsem si vybral na základě dobrých zkušeností s knihovnou OpenCV. Cílem práce bylo vytvořit program na zpracování obrazu referenčních videozáznamů z průmyslové kamery a detekci ochranných pomůcek osob, které jsou v záběru kamery.

V současné době neexistuje mnoho studií zabývajících se řešením problematiky, které se tato práce věnuje. Na trhu lze jistě nalézt hotová řešení pro sledování osob, avšak ve většině případů se jedná o proprietární řešení, která jsou současně firemním tajemstvím, a k širšímu okruhu lidí se nedostanou.

Mnou navrhnutý algoritmus je schopen najít v obraze osobu, určit barvu přilby, obleku a kalhot a to i během pohybu osoby. V testovacích videonahrávkách nebyly nejlepší světelné podmínky, přesto si s tímto nedostatkem algoritmus poradil a určil s téměř 100% úspěšností barvu jednotlivých detekovaných oblastí. Určitou výjimku tvořily osoby, které byly nevhodně oblečeny.

Výsledky práce by mohly být aplikovány do hotového řešení s několika vylepšeními, které by jednoznačně určili, zda bylo určení správnosti oblečení po celou dobu detekce v pořádku. Toho by se dalo snadno docílit průměrováním výsledků, kdy by se hlídalo, zda bylo určeno správné oblečení alespoň z 50 % času průchodu osoby před kamerou.

Dalším vylepšením by mohlo být přidání vhodnější detekční metody, aby bylo možné vyhodnotit správnost oblečení i při průchodu více než jedné osoby před kamerou. Poté by již bylo možné tento algoritmus nasadit do podniků, kde je zvýšené riziko styku s nebezpečnými látkami nebo zvýšené riziko úrazu. Program by mohl být aplikován v chemickém průmyslu, strojírenství nebo stavebnictví. Na všech těchto pracovištích je dle zákona povinnost nosit ochranné pomůcky.

LITERATURA

[1] Wikipedia. *Computer vision* [online]. 10/2012 [citováno 2012-12-02]. Dostupné z: <http://en.wikipedia.org/wiki/Computer_vision>

[2] AZUMA, R. T. A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments. 1997, roč. 6, č. 4, s. 355–385. ISSN 1531-3263.

[3] *Emgu CV: OpenCV in .NET (C#, VB, C++ and more)* [online]. 05/2014 [citováno 2014-15-29]. Dostupné z: <http://www.emgu.com/wiki/index.php/Main_Page>.

[4] *How to build applications with OpenCV inside the Microsoft Visual Studio* [online]. 05/2014 [citováno 2014-05-29]. Dostupné z: <http://docs.opencv.org/trunk/doc/tutorials/introduction/windows_visual_studio_Opencv/windows_visual_studio_Opencv.html#windows-visual-studio-how-to>.

[5] LAGANIÉRE, Robert. OpenCV 2 computer vision application programming cookbook. 1st ed. Brimmingham: Packt Publishing, 2011, III, 287 s. Quick Answers to Common Problems. ISBN 978-1-84951-324-1.

[6] *Motion Analysis and Object Tracking* [online]. 04/2014 [citováno 2014-05-29]. Dostupné z: <http://docs.opencv.org/2.4.8/modules/video/doc/motion_analysis_and_object_tracking.html?highlight=backgroundsubtractor#BackgroundSubtractor : public Algorithm>.

[7] MORDVINTSEV, A.; ABID, K. *Background Subtraction* [online]. 05/2014 [citováno 2014-05-29]. Dostupné z: <http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html>.

[8] KAEWTRAKULPONG, P.; BOWDEN, R. Computer Vision and Distributed Processing. *An Improved Adaptive Background Mixture Model for Realtime Tracking with Shadow Detection*. Video-Based Surveillance Systems. Boston, MA: Springer US, 2002, s. 135–144. DOI 10.1007/978-1-4615-0913-4_11.

[9] GODBEHERE, A., B.; MATSUKAWA, A., GOLDBERG, K. *Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation*. American Control Conference (ACC), Montreal, 2012, s. 4305–4312. ISSN 1055-4181.

[10] BRADSKI, G.; KAEHLER, A. *Learning OpenCV*. 1. vyd, Sebastopol: O'Reilly Media, 2008. 576 s. ISBN 978-0-596-51613-0.

[11] YUAN, S.; TIAN, Y.; ARDITI, A. *Technology and Disability. Clothing Matching for Visually Impaired Persons*. 2011, roč. 23, č. 2, s. 75–85. ISSN 1055-4181.

[12] *Miscellaneous Image Transformations* [online]. 04/2014 [citováno 2014-05-29]. Dostupné z: <http://docs.opencv.org/2.4.9/modules/imgproc/doc/miscellaneous_transformations.html?highlight=cvtcolor#cv2.cvtColor>.

[13] *Utility and System Functions and Macros* [online]. 04/2014 [citováno 2014-05-29]. Dostupné z: <http://docs.opencv.org/2.4.9/modules/core/doc/utility_and_system_functions_and_macros.html>.

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO MÉDIA

A.1 Diplomová práce v elektronické verzi

Název souboru: „Diplomova_prace.pdf“

Soubor je spustitelný v prohlížečích PDF s podporou verze 1.6. Testovaný v programu Adobe Reader verze 9.5.0.

A.2 Příloha k diplomové práci v elektronické verzi

Název souboru: „Priloha.pdf“

Soubor je spustitelný v prohlížečích PDF s podporou verze 1.6. Testovaný v programu Adobe Reader verze 9.5.0.

A.3 Kompletní projekt

Název souboru: „Projekt_kod.zip“

Kompletní kód programu, který je popsán v této práci. Obsahuje také komentáře ke každému řádku a k jednotlivým metodám. Projekt je spustitelný v Microsoft Visual Studiu 2010 a novějším. Kódy projektu je možné přenést do jiných kompilátoru nebo vývojových prostředí.

A.4 Spustitelný program

Název souboru: „Projekt.exe“

Projekt je spustitelný pouze v systému Windows a ke spuštění je nutné mít nainstalovanou knihovnu OpenCV verze 2.4.9 (příloha A.6) nebo rozbalené DLL knihovny z přílohy A.7.

A.5 Testovací videonahrávky

Název složky: „Testovaci videa“

Složka obsahuje sérii 26 videí, které byly využity při vytváření a testování programu popsaného v této práci.

A.6 Instalátor knihovny OpenCV 2.4.9

Název souboru: „opencv-2.4.9.exe“

Automatický instalátor v současnosti nejaktuálnější verze OpenCV 2.4.9. Obsahuje binární soubory a zdrojové kódy knihovny pro všechny verze Microsoft Visual Studia.

A.7 Binární soubory a zdrojový kód OpenCV 2.4.9

Název souboru: „OpenCV_2.4.9_vs2010.zip“

Rozbalovací ZIP archív obsahující binární soubory a zdrojové kódy knihovny OpenCV 2.4.9 pro Microsoft Visual Studio 2010.

B VLOŽENÉ LISTY

B.1 Zobrazení výstupních snímků programu

Zobrazení výstupních snímků včetně vykreslení histogramů a doprovodných informací.

B.2 Seznam proměnných

Seznam všech proměnných obsažených v programu včetně popisu.

B.3 Zdrojový kód Programu

Jedná se o kompletní zdrojový kód programu, který byl popisován a vytvářen pro diplomovou práci. Kompletní zdrojový kód je také dostupný v elektronické podobě (příloha A.3).