VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

DETEKCE POHYBUJÍCÍCH SE OBJEKTŮ VE VIDEU
DETECTION OF MOVING OBJECTS IN VIDEO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Petr Hanek**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Martin Rajnoha**
SUPERVISOR

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Petr Hanek ***ID:*** 174303
Ročník: 3 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Detekce pohybujících se objektů ve videu

POKYNY PRO VYPRACOVÁNÍ:

Prostudujte knihovnu OpenCV. Seznamte se s metodami na odečtení pozadí z obrazu. Vytvořte aplikaci a na vhodných příkladech demonstrujte detekci pohybujících se objektů.

DOPORUČENÁ LITERATURA:

[1] BRADSKI, G., KAEHLER, A. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[2] PICCARDI, M. Background subtraction techniques: a review. In: Systems, man and cybernetics, 2004 IEEE international conference on. IEEE, 2004. p. 3099-3104.

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 8.6.2017

Vedoucí práce: Ing. Martin Rajnoha
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

ABSTRAKT

Tato bakalářská práce se zabývá knihovnou OpenCV a jejími metodami. Vytvořená aplikace je schopna detekovat pohybující se objekty ve videu ze statické kamery za pomocí metod pro odečítání pozadí obrazu. Tuto aplikaci je možné využívat v různých módech: detekování v oblasti vypočítanou pomocí BFS algoritmu a dva téměř podobné módy na detekci přechodu přes linii. Aplikace funguje na více vláknech, jelikož vytvořené grafické uživatelské rozhraní je náročné na výpočetní výkon. V aplikaci je implementovaný Kálmánův filtr pro detekci více objektů zároveň a implementace maďarské metody, která řeší přiřazovací problém.

KLÍČOVÁ SLOVA

OpenCV, JavaFX, BFS, detekce pohybujících se objektů, odečítání pozadí, počítačové vidění, Kálmánův filtr, Maďarská metoda.

ABSTRACT

This bachelor thesis focuses on OpenCV library and it’s methods. Created application is able to detect moving objects from static camera video thanks to background subtraction methods. This application can be used different modes: detection in area which is calculated by BFS algorithm and two slightly different modes for crossing line detection. The application is multi thread because of graphical user interface demands on processor performance. This application also has implemented Kalman filter for multi target tracking and Hungarian method which solves assignment problem.

KEYWORDS

OpenCV, JavaFX, BFS, detection of moving objects, background substraction, computer vision, Kalman filter, Hungarian method.

HANEK, P. Detekce pohybujících se objektů ve videu. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. 39 s. Vedoucí bakalářské práce Ing. Martin Rajnoha.

PROHLÁŠENÍ

Prohlašuji, že svoji bakalářskou práci na téma „Detekce pohybujících se objektů ve videu“ jsem vypracoval samostatně na vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že v souvislosti s vytvoření této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (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.

Brno ……. …….

Podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Martinu Rajnohovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ……...

……

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 bakalářské práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu 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

Seznam obrázků

Úvod

V dnešní době jsou na počítače kladeny obrovské nároky a díky financím proudícím do sféry informačních technologií jsou tyto nároky plněny. Jedním z odvětví počítačového průmyslu je tzv. počítačové vidění. Jde o způsob naučit počítač vidět v obraze věci, které tam vidí lidské oko. Počítače jsou schopné automaticky rozeznávat pohybující se objekty, detekovat části lidského těla jako je obličej a zaznamenávat fotky těchto obličejů do databáze nebo monitorovat silniční provoz a zaznamenávat počet projetých aut.

Musíme si uvědomit, že pokud přenášíme náš 3D svět na 2D kameru dochází ke ztrátě informací, tudíž nemůžeme počítat se sto procentním výsledkem detekce. Problémy, jako jsou detekce nechtěného pohybu například stromů, listí nebo ptáků, různorodost velikosti a barvy objektů, velké změny nasvícení scény nebo problém ztracení objektu za překážkou, musí být brány v potaz.

Tato bakalářská práce se zabývá detekcí pohybujících se objektů ve videu ze statické kamery. V práci se používá open source (volně dostupná a programovatelná) knihovna s názvem OpenCV, která má spoustu metod a algoritmů na řešení těchto problému. V zásadě bude použit především filtr na odečtení pozadí. Cílem této práce je prostudovat knihovnu OpenCV, seznámit se s jejími metodami pro odečtení obrazu a demonstrovat detekci pohybujících se objektů ve vlastní vytvořené aplikaci.

Práce je strukturovaná následovně: v druhé kapitole Teoretický úvod, je rozebráno, co je potřeba vědět při práci nejen s knihovnou OpenCV, ale při práci s obrazem jako takovým. Ve třetí kapitole se práce zabývá implementací řešení na výše zmíněné problémy. Ve čtvrté kapitole výsledky jsou ukázány nejdůležitější výsledky bakalářské práce a funkčnost aplikace je na různých videích otestována.

1 Teoretický úvod

Detekování pohybujících se objektů se stává čím dál tím více důležitějším prvkem počítačové techniky. Všechny zdroje napomáhající tomuto cíli se neustále finančně zvýhodňují, především co se týče levných a vysoce výkonných procesorů. Počítačové vidění se snaží za pomocí algoritmů napodobit lidské vidění a vnímání obrazu [12]. V dnešní době existují knihovny pro práci s obrazem jako například Halcon, Matrox nebo OpenCV.

1.1 Proč OpenCV

Knihovna OpenCV je vydána na licencí BSD[1] a je díky tomu volně použitelná pro jak akademické, tak komerční účely. Knihovna je napsaná v jazyku C/C++ a je schopná využít více jader počítače. Podporuje její používání v programovacích jazycích C++, C, Python a Java. Podporuje operační systémy Windows, Linux, Mac OS, iOS a Android. OpenCV samo o sobě neustále prochází změnami a jeho algoritmy se neustále zlepšují. Na knihovně stále pracuje spoustu lidí, a proto je výborně dokumentována, podporována a aktualizována [1].

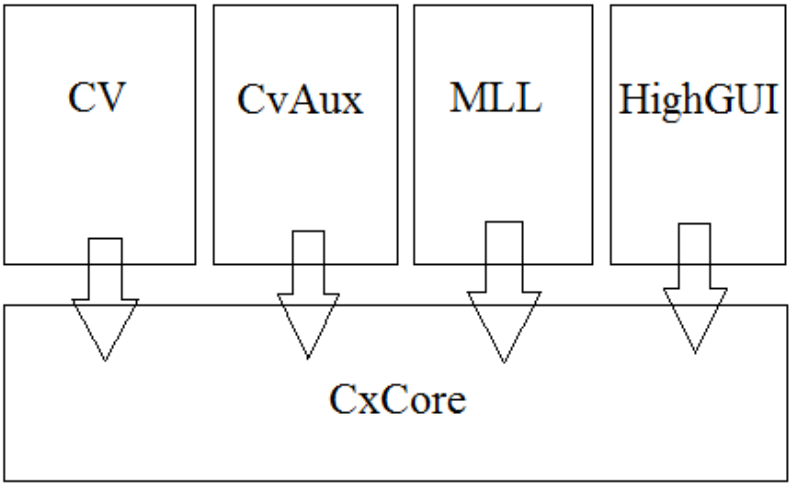
1.2 Historie OpenCV

OpenCV vznikla ve výzkumném centru společnosti Intel pro podporu náročných výpočtů pro procesor. Postupně se k projektu přidávali další skupiny a organizace především díky implementaci a optimalizací expertů z Ruska. Hlavní ředitel Ruského týmu byl Vadim Pisarevsky, který dokázal naprogramovat a optimalizovat většinu OpenCV a je pořád centrem všech aktivit okolo této knihovny. Spolu s ním pomohl vybudovat první infrastrukturu knihovny Victor Eruhimov a Varley Kuriakin, který byl manažerem celé laboratoře pracující na tomto projektu. S přibývajícím počtem lidí zajímajících se o knihovnu, rostla i obsáhlost knihovny [1]. Nejnovější verze knihovny 3.2.0 byla vydána 23.12. 2016 a je volně dostupná na internetu.

[1] Berkeley Software Distribution – licence pro svobodný software [7]

1.3 Struktura OpenCV

OpenCV je strukturovaná do pěti hlavních komponentů, jak je vidět na Obr. 1. Komponent CV[2] obsahuje základní nástroje na zpracování obrazu a pokročilé algoritmy ke schopnosti počítačů vidět. MLL[3] je knihovna, která obsahuje popisné funkce a nástroje pro shlukování videa. HighGUI[4] obsahuje I/O[5] metody a funkce pro ukládání a nahrávání obrázků, CXCore[6] obsahuje základní data, struktury a obsah. Část CvAux obsahuje jak funkce na rozpoznání různých částí lidského těla, tak experimentální algoritmy na segmentaci pozadí [1].



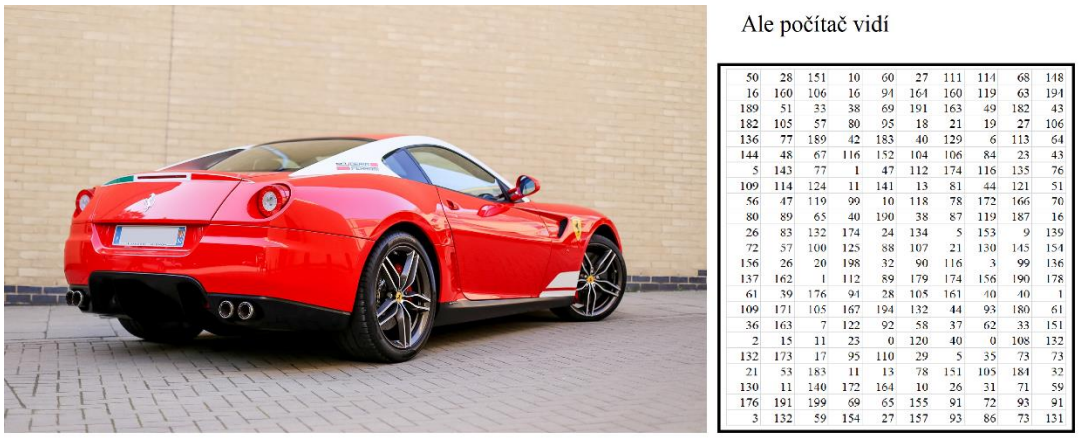
Obr. 1 Struktura OpenCV, převzato z [1].

1.4 Schopnost počítačů vidět

Schopnost počítačů vidět je založena na transformaci dat z na něco čemu bude počítač rozumět. Jelikož pro člověka je schopnost vidět vrozená vlastnost je snadné si myslet,

[2] Computer Vision – Počítačové vidění
[3] Machine Learning Library – Knihovna strojového učení
[4] High-Level Graphical User Interface – Grafické Uživatelské Rozhraní vyšších řádů
[5] Input/Output – Vstup a výstup
[6] Jádro knihovny

že naučit počítač vidět stejným stylem jako my je snadné. Počítač dostane z kamery pouze mnoho čísel, kterým je potřeba dát nějaký smysl viz, Obr. 2. Na Obr. 2 je vidět zrcátko auta, které počítač vidí jako dlouhý výčet čísel. Jakékoliv jedno číslo z tohoto seznamu má v sobě obrovský šum a dává počítači pouze malou dávku informací o tom co vlastně na obrázku je, a přesto to je všechno co má počítač k dispozici [1].



Obr. 2 Jak vidí počítač části obrazu, převzato z [1].

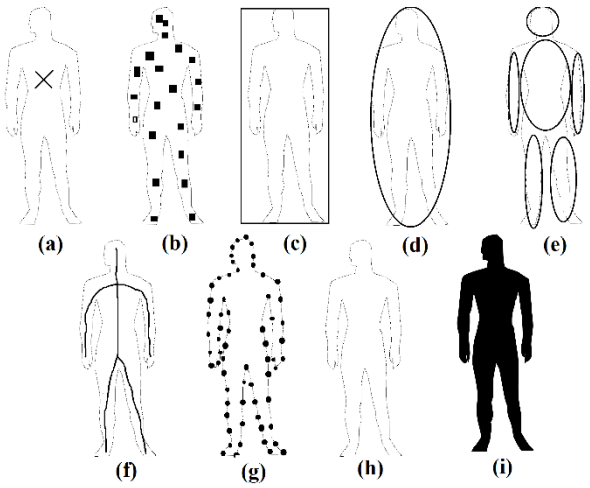
Úkolem knihoven jako jsou OpenCV je dát uživatelům možnost použít základní nástroje pro vyřešení problému počítačového vidění. V některých případech je nutnost použít těch nejlépe vytvořených algoritmů pro vyřešení komplexního problému. Po většinu času, avšak stačí používat základní nástroje, které jsou sami o sobě víc než schopné vyřešit většinu problémů. V některých případech se používají tzv. metody „pokus omyl“, kde se první vyzkouší vyřešit problém pomocí OpenCV následně se zjistí kde má ona metoda slabinu a tu se pokusíme vlastními metodami ošetřit [1].

1.5 Druhy objektů

Knihovnou OpenCV je možné detekovat velkou škálu objektů, a tudíž je pouze na uživateli, který si vybere a určí jako důležitý v další práci. Například to můžou být lodě na vodě, ryby v akváriu, vozidla na cestě, letadla ve vzduchu, lidé jdoucí po chodníku a mnohé další. Objekty jsou reprezentovány jejich tvarem a velikostí a na každý druh objektu se hodí jiný typ detekce [12].

Cílem detekování objektů zjistit kde se objekt nachází na každém snímku videa. Detekce taky může poskytnout část obrazu, ve které se objekt nachází. Objekt může nabývat různých tvarů a vzhledů jako například:

- **Bod** – objekt je reprezentován bodem, který je centrem objektu (Obr. 3 (a)) nebo více body (Obr. 3 (b)). Tento typ detekování je vhodný pouze pro objekty, které v obraze zabírají pouze malý prostor [12].
- **Geometrické tvary** – objekt je reprezentován například obdélníkem nebo elipsou (Obr. 3 (c), (d)). Tento typ detekování se především používá pro detekci tuhých (nezměnitelných) objektů [12].
- **Siluety a kontury objektu** – Kontura reprezentuje ohraničení objektu v obrázku (Obr. 3 (g), (h)). Oblast uvnitř kontury se nazývá silueta objektu (Obr. 3 (i)). Siluety a kontury objektu jsou vhodné pro detekování rychle měnitelných objektů [12].



Obr. 3 Druhy objektů, převzato z [12].

1.6 Výběr správné metody

Výběr správné metody hraje kritickou roli v detekování pohybujícího se objektu. Nejdůležitějším faktorem je schopnost rozeznat objekt na okolí. Výběr metody je úzce spjatý s tím, jak je objekt reprezentován. Většina algoritmů používá kombinaci více metod [12].

- **Detekování pomocí rozdílu barev** – Samotná barva objektu je zapříčiněna především úrovní světla, které na objekt dopadá a schopností objektu toto světlo odrážet. Využívají se různá barevná spektra jako například RGB[7]nebo HSV[8] ale tato spektra jsou velice náchylná na šum. Každý systém pracuje lépe za pomocí jiného barevného spektra. [12].
- **Hrany** – Hranice objektu většinou generují velké změny v obraze. Detekování hran je používáno pro detekování právě těchto změn. Důležitou vlastností hran je, že jsou mnohem méně náchylné na změnu intenzity světla v porovnáních s detekováním pomocí rozdílu barev [12].
- **Světelný tok** – Světelný tok je husté orientované vektorové pole, které definuje každý pixel v dané oblasti. V této metodě se využívá omezení jasu, při kterém se předpokládá určitá stálost jasu každého pixelu v po sobě jdoucích snímcích [12].

Většina metod je vybíraná manuálně uživatelem a záleží především na použití aplikace. Jednou z možností je taky automatický výběr nejlepší možné metody [12].

1.7 Odečítání pozadí obrazu

Odečítání pozadí je široce využívaný postup k detekování pohybujících se objektů ze statické kamery nebo videa. Hlavní myšlenkou této metody je zaměřit se pouze na pixely, které se liší ve dvou po sobě jdoucích obrázcích. Využívá se taky uložení obrázku scény, na které se nic nepohybuje, aby se dalo zjistit, co na scéně přibylo. Tento obrázek musí být často obnoven, jelikož se může měnit intenzita světla na scéně. Více komplexní metody byly rozšířeny do takových rozměrů, že už přesahují samotný význam slov odečítání pozadí obrazu [8].

1.7.1 Rozdíl dvou obrázků

Metoda rozdílu dvou obrázků je jednou z nejjednodušších metod na odečítání pozadí obrazu. Pro vypočtení pozadí obrázku se využívá obrázek předcházející, jakožto předloha pro pozadí. Jelikož tato metoda využívá pouze jeden předcházející obrázek, nemusí být schopná rozeznat některé pohyby rovnoměrně zbarvených objektů [9].

[7] Red, Green, Blue – česky Červená, Zelená, Modrá
[8] Hue, Saturation, Value – ve významu Tón barvy, Sytost barvy, Hodnota jasu

1.7.2 Mediánový filtr

Tento filtr je jeden z nejpoužívanějších metod na odečítání pozadí obrazu. Odhad pozadí se definuje jako medián každého pixelu všech obrázku v zásobníku. Předpoklad této metody je, že pixel zůstane v pozadí pro více jak polovinu obrázků v zásobníku [9].

1.7.3 Mix gaussiánů

V praxi jsou pravděpodobné velké změny osvětlení scény snímané kamerou, ať už se jedná o postupné venkovní změny počasí či denní doby nebo náhlé jako je rozsvícení světla uvnitř místnosti. Objekt může kdykoliv na scéně přibýt nebo již stávající zmizet. V této metodě se využívá časté obnovy tzv. trénovacích obrázků pro naučení pozadí obrazu. V uživatelem určeném časovém úseku se obnovují trénovací obrázky a znovu se odečítá pozadí obrazu, avšak na trénovacích obrázcích, které byly smazány, mohou být některá důležitá data objektů, která nebyla součástí pozadí [13].

1.8 JavaFX

JavaFX je platforma pro vývoj tzv. bohatých internetových aplikací. Díky této technologii je možné vytvářet jednoduše použitelné interaktivní webové aplikace. V těchto aplikacích se většinou klade důraz na jednoduchou použitelnost z hlediska uživatele. Tato platforma poskytuje velkou škálu grafických a multimediálních rozhraní. Jelikož se jedná o Javu, je zaručeno, že aplikace vyvinuté na této platformě se budou chovat ve všech zařízeních stejně [3].

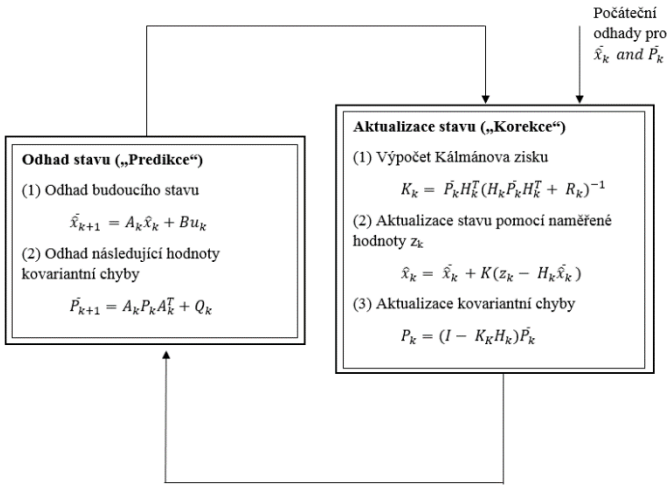
1.9 Prohledávání do šířky, BFS

Prohledávání do šířky (BFS – Breadth First Search) funguje na způsobu prohledávání stavového prostoru na základě sousedů daného bodu. Nejprve jsou prohledávány stavy nejbližší počátečnímu bodu a postupně se rozšiřuje dál. Algoritmus funguje na existenci dvou množin. Množina s názvem `Closed` obsahuje již navštívené stavy a na začátku běhu algoritmu je zcela prázdná, naopak množina s názvem `Open`, do které se na začátku algoritmu zavádí počáteční bod je množina otevřených stavů, které se budou algoritmem procházet [2]. Algoritmus se často používá na jak prohledání celého stavového prostoru, tak vyhledání nejkratší cesty na počátečního bodu k určitým stavům. Možnou variantou algoritmu BFS je varianta bez kontroly opakování průchodu

stavů. Taková varianta se hodí pouze v případě, že se v daném stavovém prostoru nenachází smyčky. Pokud tuto variantu použijeme a ve stavovém prostoru se přesto smyčky nachází, nemusí algoritmus konvergovat k řešení vůbec. Pokud ale tuto metodu zvolíme a algoritmus funguje správně, můžeme získat časovou a paměťovou úsporu [2].

1.10 Kálmánův filtr

V roce 1960 publikoval Rudolf Emil Kálmán práci o matematickém algoritmu, který dokáže na základě hodnot získaných z různých měření odhadnout stav lineárního systému. Za posledních pár desítek let našel tzv. Kálmánův filtr uplatnění v mnoha oblastech jako jsou například počítačové vidění, navigační a sledovací systémy nebo třeba ekonomika. Tato uplatnění jsou mnohdy způsobena obrovským pokrokem v oblasti výpočetních technologií. Kálmánův filtr je sofistikovaný algoritmus, který podporuje měření minulých, aktuálních, a dokonce i budoucích hodnot a tyto měření dokáže využít i když nezná přesnou podstatu modelovaného systému [11]. Algoritmus Kálmánova filtru probíhá ve dvou neustále se opakujících krocích tzv. predikce a korekce. Krok predikce pomocí rovnic zobrazených v Obr. 4 předpoví následující stav a vypočte novou chybu měření. V druhém kroku nazvaném korekce se nejprve vypočte hodnota K_k reprezentující Kálmánův zisk, která určuje váhu nové informace oproti předchozí. Následně se zaktualizuje stav pomocí naměřených hodnot, a nakonec se odhaduje chyba měření.



Obr. 4 Algoritmus Kálmánova filtru, převzato z [11]

Matice *A*, které se vyskytuje v kroku predikce je translační matice o velikosti n × n, která určuje vztah mezi po sobě jdoucími kroky (k a k+1). Matice *B* o velikosti n × l se nazývá kontrolní matice. Matice označená *H* má velikost n × m a nazývá se matice měření. Kálmánův zisk je druh šumu, se kterým dokáže algoritmus pracovat. Další šum, se kterým Kálmánův filtr pracuje je v rovnicích označen jako *R* a jeho velikost záleží na nepřesnosti měření. *Q* zachycuje chyby při zaokrouhlování a *P* je odhad chyby výpočtu [11] [6].

1.11 Maďarská metoda

Maďarská metoda byla vynalezena a publikována roku 1955 matematikem jménem Harold Kuhn, který jí pojmenoval „Hungarian method“ (maďarská metoda), jelikož se z velké části zakládá na práci dvou maďarských matematiků Dénes König a Jenő Egerváry [5]. Pomocí této metody se řeší tzv. přiřazovací problém, který řeší přiřazení *n* zdrojů k *n* úkolům. Každý zdroj má známou cenu ke každému úkolu a cílem maďarské metody je najít nejefektivnější přiřazení jednoho zdroje k jednomu úkolu. Řešení tohoto typu problému se provádí v několika krocích.

- První krok se nazývá primární redukce matice, kde v každém sloupci matice odečteme nejmenší hodnotu. To samé poté provedeme u řádků.
- Druhým krokem je nakreslení minimálního počtu krycích čár, které překrývají všechny nuly v matici. Tyto čáry mohou být pouze svislé nebo vodorovné. Pokud by se počet čar rovnal *n* tedy počtu sloupců a řádků, metoda tímto krokem končí. Pokud se počet čar nerovnal *n* je nutné matici dále redukovat.
- Třetím krokem je teda další redukce. V matici se najde nejmenší čárou nepřekrytá hodnota. Tato hodnota se odečte na všech hodnot, které nejsou čárou překryté čárou. Naopak se tato hodnota přičte ke všem hodnotám, které byli překryté jak svislou, tak i vodorovnou krycí čárou. Opakujeme druhý krok.

Tímto způsobem je nalezeno optimální přiřazení *n* zdrojů k *n* úkolům [10] [5].

2 Implementace řešení

V této kapitole je popsána vlastní praktická implementace bakalářské práce, tedy detekce pohybujících se objektů ve videu. V první části kapitoly jsou popsány nejdůležitější třídy a metody aplikace. Druhá část se pak zabývá problémy, které se vyskytly a jejich řešením.

2.1 Implementace JavaFX

Co se týče implementace JavaFX do vývojového prostředí je nutno nainstalovat speciální JavaFX balíček. V této bakalářské práci byl dále použit pro vytvoření jednoduchého grafického uživatelského rozhraní program Scene Builder, který umožňuje jednoduše přidávat na plochu různé objekty, které později můžeme pomocí Scene Builderu provázat s proměnnými. Pro použití Scene Builderu je nutné v JavaFX projektu vytvořit FXML dokument, který obsahuje popis všech objektů na ploše grafického uživatelského rozhraní popsaných v jazyce XML (Extensible Markup Language, česky rozšiřitelný značkovací jazyk) [3].

2.2 Třídy a metody

V této kapitole jsou postupně popsány vlastní třídy aplikace a zobrazeny jejich nejdůležitější metody. Pomocí přiložených obrázků jsou vysvětlovány jednotlivé metody a jejich funkce v aplikaci.

2.2.1 Třída Main

Třída `Main` je javaFX třída, která spouští grafické uživatelské rozhraní a následně i druhé vlákno aplikace, jelikož celé hlavní vlákno bude zaneprázdněno vykreslováním obrázků do určených oken je nutné pro zbytek aplikace především třídu `Detect` použít jiné vlákno (viz Obr. 5).

```
@Override
public void start(Stage primaryStage) {

    Parent root = null;

    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("GUI.fxml"));
        root = (Parent) loader.load();
        jedna = loader.getController();
    } catch (IOException e) {
        e.printStackTrace();
    }

    Scene scene = new Scene(root);
    primaryStage.setScene(scene);
    primaryStage.show();
    app.controller = jedna;

    MyTask task = new MyTask();
    task.controller = jedna;
    new Thread(task).start();

}

public static void main(String[] args) {
    launch(args);

}
```

Obr. 5 Ukázka třídy Main.

2.2.2 Třída Detect

Ve třídě `Detect` se odehrává nejdůležitější část programu, a sice samotná detekce pohybujících se objektů ve videu. Běží na vlastním vlákně, a tak nebrzdí zbytek aplikace v jejím chodu. Uživatelem vybrané video se rozčlení na jednotlivé obrázky, které se zpracovávají do vektoru matic formátu `Mat` a následně se aplikuje metoda `processFrame(Mat mRgba ,Mat mFGMask,BackgroundSubstractor mBGSub)` (Obr. 6).

```
protected void processFrame(Mat mRgba, Mat mFGMask, BackgroundSubtractor mBGSub) {
    mBGSub.apply(mRgba, mFGMask, controller.learningRate_slider.getValue());
    Imgproc.cvtColor(mFGMask, mRgba, Imgproc.COLOR_GRAY2BGRA, 0);
    Mat dilate = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(5, 5));
    Mat openElem = Imgproc.getStructuringElement(Imgproc.MORPH_RECT, new Size(7, 7), new Point(1, 1));
    final Size kSize = new Size(controller.kernelSize_slider.getValue(), controller.kernelSize_slider.getValue());
    Imgproc.blur(mFGMask, mFGMask, kSize);
    Imgproc.dilate(mFGMask, mFGMask, dilate);
    Imgproc.morphologyEx(mFGMask, mFGMask, Imgproc.MORPH_OPEN, openElem);

}
```

Obr. 6 Metoda na zpracování jednoho obrázku.

Tato metoda nejprve aplikuje tzv. MoG[9] „background substractor“ (odečítač pozadí) nazvaný mBGSub a tak získá pozadí obrazu. V dalším kroku metoda černobílé spektrum obrázků do barevného spektra RGB. Poté jsou vytvořeny dva filtry `dilate` (dilatace), `openElem` (strukturovaný element), které jsou aplikovány na obrázek bez pozadí (pouze pohybující se objekty) a zvýrazní, tudíž zlepší přesnost detekování pohybu. Následně jsou objekty rozmazány pomocí funkce `blur` pro zlepšení kvality detekce.

Obr. 7 Ukázka chodu programu v základním módu detekce pohybu.

Na levé straně Obr. 7 je vidět načtené video bez jakýchkoliv filtrů a úprav a na straně druhé je vidět již zpracovaná posloupnost obrázků, která se zobrazuje jako video, na které je aplikovaná právě metoda `processFrame(Mat mRgba ,Mat mFGMask,BackgroundSubstractor mBGSub)`. Ihned po aplikování metody `processFrame` se na vzniklý černobílý obrázek aplikuje metoda `detectionCountours(Mat outmat, Controller controller)`, která vytvoří vektor typu `Rect`, do kterého se budou zaznamenávat jednotlivé pozice bílých (pohybujících se) objektů. Tento vektor se použije pro vykreslení obdélníků kolem pohybujících se objektů v původním barevném videu (Obr. 7).

[9] Mix gaussiánů

```
public static Vector<Rect> detectionContours(Mat outmat, Controller controller) {
    Mat HIERARCHY = new Mat();
    Mat image = outmat.clone();
    List<MatOfPoint> contours = new ArrayList<MatOfPoint>();
    Imgproc.threshold(image, image, 254, 255, Imgproc.THRESH_BINARY);
    Imgproc.findContours(image, contours, HIERARCHY, Imgproc.RETR_LIST,
            Imgproc.CHAIN_APPROX_TC89_L1);

    int maxAreaIdx = -1;
    Rect r = null;
    Vector<Rect> rect_array = new Vector<Rect>();

    for (int idx = 0; idx < contours.size(); idx++) {
        Mat contour = contours.get(idx);
        double contourarea = Imgproc.contourArea(contour);
        if (contourarea > controller.minArea_slider.getValue() && contourarea < controller.maxArea_slider.getValue()) {
            maxAreaIdx = idx;
            r = Imgproc.boundingRect(contours.get(maxAreaIdx));
            rect_array.add(r);
        }

    }

    image.release();
    return rect_array;
}
```

Obr. 8 Metoda na zjištění kontur.

Do metody `detectionContours(Mat outmat, Controller controller)` vstupuje matice typu `Mat` obrázku bez pozadí a pomocí funkce `findCountours()` z třídy `Imgproc` jsou do proměnné typu `List<MatofPoint> countours` vloženy souřadnice jednotlivých kontur pohybujících se objektů. Ve smyčce `for` se procházejí všechny body v proměnné `countours`, a do proměnné `r` typu `Rect` se zaznamenávají souřadnice obdélníků obkreslující tyto kontury (Obr. 8). Tyto souřadnice se následně v programu použijí na vykreslení obdélníků do původního barevného obrázku typu `Mat`, který se převede metodou `Mat2Image(Mat image)` (Obr. 9) na BMP[10] obrázek zobrazitelný v grafickém uživatelském rozhraní.

```
public static Image Mat2Image(Mat image) {
    MatOfByte byteMat = new MatOfByte();
    Highgui.imencode(".bmp", image, byteMat);
    return new Image(new ByteArrayInputStream(byteMat.toArray()));
}
```

Obr. 9 Metoda převedení matice na BMP obrázek.

Do této metody vstupuje vektor matic po všech úpravách a je pomocí knihovny OpenCV a třídy `HighGUI` (kapitola 1.3) převedena na obrázek, který je již možné zobrazit v okně aplikace.

[10] BitMaP – mapa bitů zobrazující obrázek

2.2.3 Třída Controller

Ve třídě `Controller` jsou vytvořeny všechny proměnné týkající se grafického uživatelského rozhraní aplikace například tlačítka „Stop“ (zastavit) a „Play“ (spustit), okna pro zobrazování obrázků vytvořených třídou `Detect` a „`RadioButtony`“ (výběrová tlačítka) pro změnu módu, ve kterém aplikace funguje. Pro všechny tyto proměnné jsou zde vytvořeny obslužné metody. Jedno z vytvořených oken je pouze okno pro zobrazení obrázku tedy `ImageView`, ale druhé je typu `Canvas`, jelikož je nutné do něj kreslit. Pro kreslení do okna typu `Canvas` jsou zde vytvořeny metody typu `EventHandler` zaměřené na pohyb a klik myši. Dále se zde nachází metoda `fileChooser()`, která uživateli umožní načíst z disku počítače jakékoliv video pro zpracování, metoda `getFirstImage()` pro zobrazení prvního obrázku v okně aplikace aby uživatel věděl, kam kreslí čáry pro ovládání programu v módu `DrawLine` a `DrawArea`.

```
    private void copyPointsToLine(Vector<Point> points) {
        DetectionLine line = new DetectionLine();
        for (int i = 0; i < points.size(); i++) {
            line.addPoint(points.get(i));
        }
        app.addLine(line);

    }

});

canvas.addEventHandler(MouseEvent.MOUSE_CLICKED, new EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent t) {
        if (radioButtonArea.isSelected() && t.getButton() == MouseButton.SECONDARY) {
            Point click = new Point(t.getX(), t.getY());
            lastPointToArea(click);

        }
        return;
    }

    private void lastPointToArea(Point point) {
        DetectionArea area = new DetectionArea();
        HashSet<Point> hashSet = new HashSet<Point>();
        app.gc = gc;
        app.imageView = imgView2;
        hashSet = app.vytvorOblast(point);
        area.addPoint(hashSet);
        app.addArea(area);

    }

});
```

Obr. 10 Metody na převedení nakreslených čár do proměnných v aplikaci.

Při kliknutí a tažení myši oknem typu `Canvas se podle` souřadnic myši kreslí červená čára o velikosti nastavené v hodnotě `pixelArea`, která je přednastavená na hodnotu 5. Pokud je zvolen mód `Draw Line` (nakreslit čáru) všechny tyto body jsou následně

ukládány přes metodu `addPoint()` do pomocné proměnné typu `Vector<Point>` a převedeny do vlastní třídy `DetectionLine` přes metodu `addLine()`. Pokud je zvolen mód `Draw Area` (nakreslit oblast) body, které kreslí uživatel programu jsou ignorovány a čeká se na klik pravého tlačítka myši (metoda `lastPointToArea(Point point)`), který spustí výpočet všech bodů v oblasti (viz Obr. 10).

2.2.4 Metoda na vytvoření nakreslené oblasti

Tato metoda počítá všechny pixely ohraničené červenou čárou, kterou nakreslil uživatel do okna typu `Canvas`. Jako vstup očekává bod kliknutí po nakreslení červené hranice. Tímto bodem uživatel odliší chce-li počítat body uvnitř oblasti nebo mimo ni. Nachází se ve třídě `DetectionAPP`, která pomáhá provázat objekty s kódem. Algoritmus, který uvnitř této metody probíhá funguje na principu vyhledávání BFS (viz kapitola 1.8). Algoritmus funguje za pomocí dvou množin typu `HashSet<Point>`, což znamená, že množina nemůže obsahovat dva stejné objekty. Tento typ množiny je velice důležitý, jelikož by došlo k velkému nárůstu času potřebného k výpočtu. Jedna z množin má název `open` a nachází se v ní všechny body, se kterými bude algoritmus pracovat. Algoritmus vypočítá z jednoho bodu jeho čtyři sousedy a zkontroluje jaká je jejich barva. Pokud je barva červená (nakreslená hranice) bod neprojde podmínkou a není zapsán do množiny `open`. Algoritmus iteruje přes množinu `open`, dokud není zcela prázdná. V každém kroku se bod, se kterým se pracuje vymaže z množiny `open` a přidá do množiny `closed` protože tento bod již prošel podmínkou o barvě a počítají se jeho sousedé, kteří následně procházejí podmínkou o barvě.

```
Color red = Color.web("0xff0000ff");
Color pixel;
open.add(clickVoblasti);
while (!open.isEmpty()) {

    Point tmpPoint = open.iterator().next();
    open.remove(tmpPoint);
    closed.add(tmpPoint);

    sousedniBody.clear();

    if (tmpPoint.x + 1 < 640) {
        sousedniBody.add(platno[(int) (tmpPoint.x + 1)][(int) tmpPoint.y]);
    }
    if (tmpPoint.y + 1 < 480) {
        sousedniBody.add(platno[(int) (tmpPoint.x)][(int) tmpPoint.y + 1]);
    }
    if (tmpPoint.x - 1 > 0) {
        sousedniBody.add(platno[(int) (tmpPoint.x - 1)][(int) tmpPoint.y]);
    }
    if (tmpPoint.y - 1 > 0) {
        sousedniBody.add(platno[(int) (tmpPoint.x)][(int) tmpPoint.y - 1]);
    }

    // System.out.println(sousedniBody);
    for (int j = 0; j < sousedniBody.size(); j++) {
        pixel = canvasImage.getPixelReader().getColor((int) sousedniBody.get(j).x,
                (int) sousedniBody.get(j).y);

        if (!closed.contains(sousedniBody.get(j)) && pixel.getRed() != red.getRed()) {
            open.add(sousedniBody.get(j));

        }
    }

}

System.out.println("velikost oblasti" + " " + closed.size());
return closed;
```

Obr. 11 Metoda na vytvoření oblasti.

V proměnné `red` typu `Color` se nachází hexadecimální hodnota červené barvy v barevném spektru RGB. Proměnné `tmpPoint` a `sousedniBody` jsou pouze pomocné proměnné, které jsou v každé iteraci jiné a vztahují se pouze k danému bodu. V proměnné `tmpPoint` se nachází bod, se kterým algoritmus zrovna pracuje a do seznamu `sousedniBody` se přidají jeho čtyři sousední body. Přidávání sousedních bodů je ošetřeno podmínkou, aby sousední bod nebyl mimo oblast videa tedy (bod > 0, bod < 680, bod < 480) , což odpovídá šířce a výšce videa. V dalším kroku již následuje samotná podmínka o barvě po jejímž splnění se bod zapíše do množiny `open` a budou se dále počítat jeho sousedé (Obr. 11).

2.3 Implementace Kálmánova filtru

Samotná implementace Kálmánova filtru byla otázkou zakomponování, jíž existujícího algoritmu do této aplikace. Zdrojový kód algoritmu Kálmánova filtru je převzatý z [4]. Tento algoritmus se skládá ze sedmi tříd a plně využívá možnosti objektově orientovaného programování, které nabízí Java. Ve třídě `Kalman` se nachází definice základních matic a proměnných pro Kálmánův filtr, nachází se zde metody jako `getPrediction()` a `update()`. Třída `JTracker` je rodičovská, abstraktní třída ke třídě `Tracker` a nachází se v ní inicializace proměnných, které budou použité ve třídě `Tracker`. Samotná třída `Tracker` obsahuje důležité metody na práci jako je `update(Vector<Rect> rectArray, Vector<Point> detections, Mat imag)` a `updateKalman(Mat imag, Vector<Point> detections)`. Každý detekovaný objekt má vlastní Kálmánův filtr, a tak i vlastní `Track` (trasa). V této třídě se s těmito trasami pracuje. Algoritmus dokáže vytvářet nové a mazat staré nebo dlouho nepoužité trasy podle nastavené hodnoty maximálního počtu snímku bez detekce daného objektu. Velmi podstatnou částí tohoto algoritmu jsou třídy `AssignmentOptimal` a `HungarianAlg3`, které se starají o přiřazení jednotlivých tras Kálmánova filtru k objektům, kterými byli vytvořeny. Samotná třída `HungarianAlg3` představuje implementaci tzv. Hungarian Algorithm (Maďarská metoda), která řeší přiřazovací problém. Pokud je stejný počet zdrojů a možných objektů k přiřazení maďarská metoda najde nejlepší způsob, jak jednotlivé zdroje k objektům přiřadit. Tento způsob přiřazení se v této aplikaci využívá při detekci více objektů najednou.

2.4 Problémy a jejich řešení

V této kapitole jsou popsány nejdůležitější problémy, které nastaly v rámci vypracování bakalářské práce. Každá následující kapitola nejprve popisuje daný problém a následně jak se při řešení problému postupovalo.

2.4.1 Problém s javaFX a jeho řešení

Problémem kompatibility této aplikace a javyFX je ten, že většinou není možné využít předem vytvořené funkce pro např. tlačítka play a stop, jelikož video na výstupu není video ale mnoho obrázku za sekundu, které javaFX musí obsluhovat. Prvním problémem byla metoda `update()`, která měla za úkol vykreslovat aktuální zpracovaný

obrázek do příslušných oken. Tato metoda i obrázky, které do ní byli vkládány byly správné, ale program se vždy při spuštění zaseknul. Docházelo k přetížení aplikace, která nestíhala projíždět nekonečnou smyčku a vykreslovat obrázky najednou. Řešení spočívá v tom udělat aplikaci více vláknovou a samotné zpracování obrázku přesunout na jiné vlákno. JavaFX tedy běží na hlavním vlákně, zatímco nekonečná smyčka, která zpracovává video na obrázky a dále s nimi pracuje, běží na vedlejším vlákně. Dalším problémem byla již vzpomínaná tlačítka play a stop. Tento problém je vyřešen proměnou typu `boolean` (pouze dvě hodnoty pravda nebo nepravda), která se použije v nekonečné smyčce pro podmínku, jestli má program provádět zpracování obrázků a dále je posílat do metody `update()` nebo pouze čekat.

2.4.2 Problém s metodou na vytvoření nakreslené oblasti a jeho řešení

Pro vytvoření oblasti pouze z jednoho bodu je použit algoritmus BFS (kapitola 1.9), který je mírně upraven pro potřeby aplikace. Problém použití normálního BFS v této aplikaci je ten, že z každého bodu se vytvoří 4 sousedé, a to i na stranu odkud byl prvotně vytvořen. Při velikosti obrazu 640x480 je celkový počet pixelů v obraze 307 200 což by se mělo rovnat i počtu iterací algoritmu, pokud jako oblast bereme celý obraz (nenakreslíme žádnou červenou linii a klikneme kdekoliv do kreslící plochy). Při použití normální množiny s možným opakováním bodů celkový čas výpočtu extrémně vzrostl. Dalším problém bylo zjištění barev jednotlivých pixelů a ošetřit, aby algoritmus přestal kontrolovat body, když se dostane na uživatelem nakreslenou červenou hranici. V aplikaci je proto vytvořen ukázkový pixel typu `Color` viz, Obr. 11, do kterého se zapíše hexadecimální hodnota červené barvy. Následně se barva každého pixelu pomocí funkce `pixelReader()` zapíše do proměnné pixel typu `Color`. Na obě dvě tyto proměnné se použije funkce `getRed()`, čímž docílíme vrácení hexadecimální hodnoty barvy pixelu a porovnáme je.

3 Výsledky

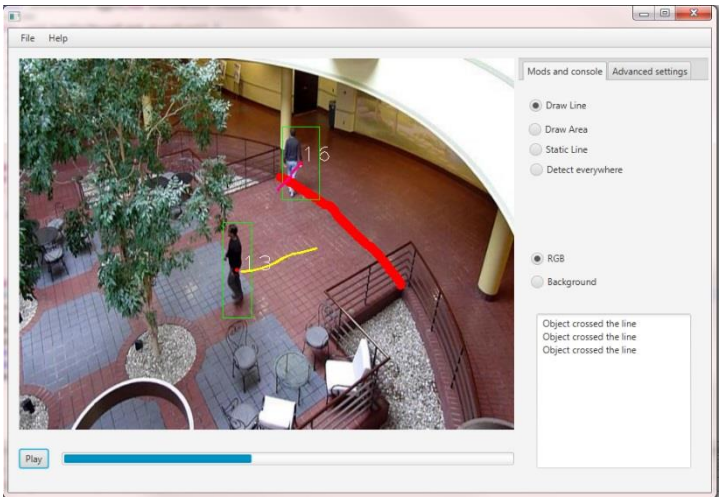
V bakalářské práci byla vytvořena aplikace s grafickým uživatelským rozhraním, která dokáže detekovat pohyb pohybujících se objektů ve videu a tento pohyb detekovat v různých módech. V této kapitole jsou demonstrovány výsledky práce na praktických příkladech. V druhé části bude na více příkladech ukázána funkčnost aplikace, co se týče detekce pohybujících se objektů ve videu.

3.1 Představení aplikace

Po spuštění aplikace se uživatel nachází v první záložce aplikace s názvem „Mods and console“, která slouží pro základní ovládání aplikace. Po rozkliknutí File v horní liště je možné vybrání video souboru, který bude aplikace zpracovávat. V horní liště se také nachází tlačítko Help s návodem na ovládání aplikace a popisem pokročilejších nastavení. Na první záložce aplikace si uživatel může zvolit jeden ze čtyř módů aplikace Draw Line, Draw Area, Static Line a Detect Everywhere. Na první záložce programu je také možné vybrat, vykreslování originálního obrazu nebo obraz po odečtení pozadí. K tomuto slouží dvě tlačítka RGB a „Background“ (pozadí). Pod tlačítky RGB a Background se nachází konzole, do které aplikace vypisuje různé hlášky, které budou ukázány níže. V druhé záložce nazvané Advanced settings může uživatel přesněji nastavit proměnné, které ovlivňují běh algoritmů na odečítání obrazu, detekci kontur v odečteném obraze, a nastavení parametrů Kálmánova filtru.

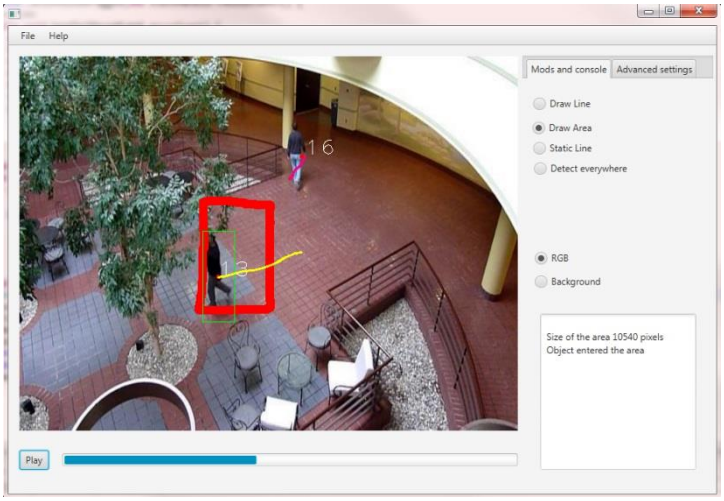
3.1.1 Záložka „Mods and console“

Při zvolení prvního módu Draw Line, je uživateli umožněno kreslit do okna aplikace táhnutím myši. Tímto je možné nakreslit linii, se kterou bude program následně pracovat. Po nakreslení linie a spuštění videa aplikace detekuje přechod středního bodu detekovaného pohybujícího se objektu přes tuto linii, přičemž tuto událost zaznamenává do konzole viz. Obr. 12. Pokud se stane, že středový bod detekovaného objektu jde přímo po linii program opakovaně vypisuje přechod objektu přes linii, jelikož není schopen rozeznat jeden objekt na druhého. Tato linie může mít jakoukoliv velikost a tvar, tudíž je složitější rozeznat, ze které strany objekt přichází a tato možnost je tak pouze v módu Static Line, který je popsaný níže.



Obr. 12 Představení módu Draw Line

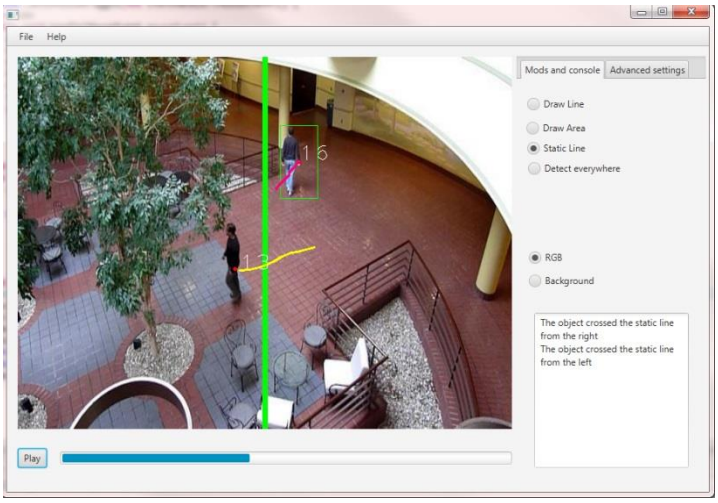
V módu Draw Area se na uživatele vyžaduje vymezit spojitou čárou oblast a následně kliknout pravým tlačítkem myši buď do oblasti nebo vně oblasti, a tak určit, jestli má aplikace detekovat pohyb jen uvnitř oblasti nebo všude kromě ní. Následně aplikace provede algoritmus popsaný v kapitole 2.2.4 a vypíše do konzole velikost oblasti v pixelech. Po spuštění videa aplikace detekuje pohyb například jen uvnitř oblasti, jak je znázorněno na Obr. 13.



Obr. 13 Představení módu Draw Area.

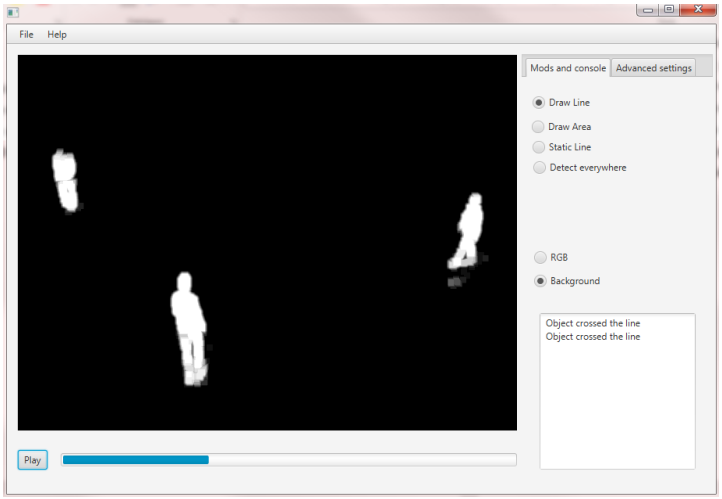
Pokud si uživatel vybere mód Static Line aplikace na kreslení nereaguje a po spuštění aplikace se zobrazí v kódu nadefinovaná linie přes střed obrazovky, přes kterou dokáže

aplikace detekovat pohyb a určit, jestli objekt přišel zleva či zprava jak je vidět na Obr. 14.



Obr. 14 Představení módu Static Line

Pokud si uživatel vybere mód Detect Everywhere aplikace na kreslení opět nereaguje a detekuje pohyb v celém obraze. Na předešlých obrázcích (Obr. 12, Obr. 13, Obr. 14) je použit mód RGB aby bylo vidět, jestli detekce opravdu proběhla. Na následujícím obrázku (Obr. 15) je použit mód background zobrazující pouze odečtené pozadí. Na obraze nejsou vidět žádné nakreslené čáry, ale aplikace o nich ví a neustále vypisuje hlášky pro daný mód, jak je vidět v konzoli.



Obr. 15 Představení módu Background

3.1.2 Záložka Advanced settings

V záložce advanced settings, jak již název napovídá, je možné přesněji nastavit jednotlivé hodnoty různých algoritmů pro (pokud jsou hodnoty zvoleny správně) zlepšení efektivity aplikace. Prvních pět posuvníků se týká nastavení Kálmánova filtru. Pomocí prvního z těchto pěti posuvníků je možné nastavit hodnotu tzv. `delta time` Kálmánova filtru, jejíž výchozí nastavení je 0.2. Dalším posuvníkem je možné nastavit hodnotu s názvem `Noise`, která dává možnost nastavit kolik má očekáváme na vstupu nepřesností (šumu). Posuvník s názvem `Distance` dává uživateli možnost nastavit, jak daleko mohou jednotlivé trasy Kálmánova filtru skočit při hledání ztraceného zdroje přiřazeného pomocí Maďarské metody. Posuvník s názvem `Skipped frames` umožňuje uživateli přesně určit, po jak kolika snímcích na poslední detekce objektu bude trasa Kálmánova filtru smazána. Poslední z těchto posuvníků má název `Trace length` a určuje kolik minulých hodnot Kálmánova trasa zaznamenává, jinak řečeno, jak dlouhá je vykreslená čára Kálmánovy trasy. Další dva posuvníky se týkají samotné detekce a jejího znázornění do obrazu. Uživatel má možnost nastavit, jak má být minimálně a maximálně velký objekt, který předpokládá, že bude chtít detekovat. Tyto posuvníky ovlivňují metodu `detectionContours(Mat outmat, Controller controller)` viz, Obr. 8. Poslední dva posuvníky ovlivňují metodu `processFrame(Mat mRgba ,Mat mFGMask,BackgroundSubstractor mBGSub)` viz, Obr. 6, ve které mění hodnotu Learning rate a velikost matice. Learning rate je hodnota, která udává, velikost kroku, po které algoritmus konverguje k řešení. V této záložce se také nachází tlačítko `Reset to default`, které nastaví všechny hodnoty na výchozí. Přesné výchozí hodnoty a veškerý popis jednotlivých hodnot lze najít také po rozkliknutí tlačítka `Help` v horní liště a následně vybrání `More about Advanced Settings`.

3.2 Funkčnost aplikace na různých testovacích videích

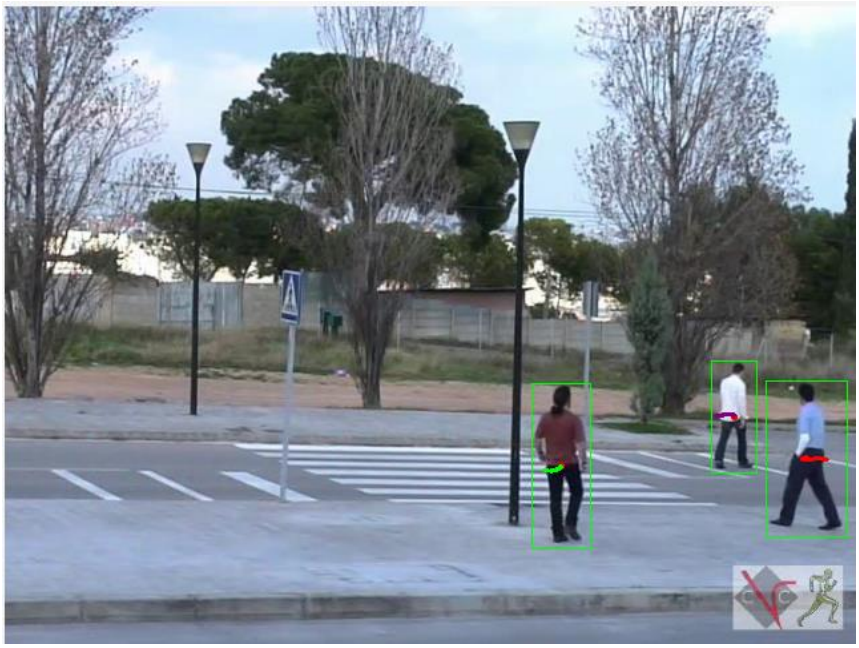
V této kapitole je na více příkladech ukázána funkčnost aplikace, co se týče detekce pohybujících se objektů ve videu. Jsou k tomu použity nejlepší možné nastavení v záložce Advanced Settings týkající se odečítání pozadí a samotné detekce, tzv. nastavení Kálmánova filtru bude vždy stejné. Všechny ukázky jsou prováděný v módu Detect everywhere jelikož ostatní módy závisí na schopnosti aplikace detekovat pohyb

a následně na něj reagovat vypisováním hlášek do konzole. Každé video bude zastaveno v okamžiku kdy na obraze je největší množství objektů k detekci.

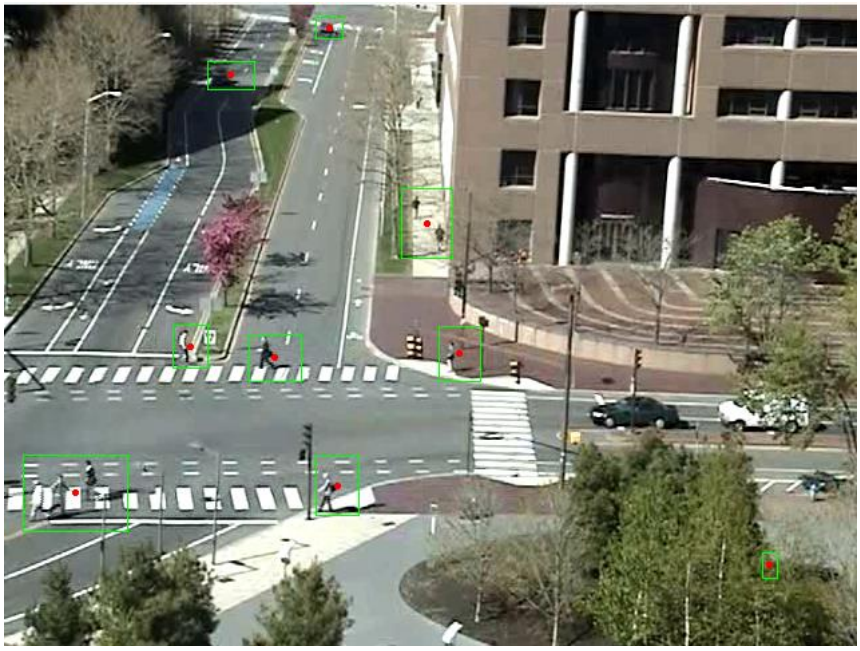
První ukázkové video je nazvané „atrium“. Na Obr. 16 je možné vidět dva úspěšně detekované lidi. Toto video bylo nejvíce využito při vývoji aplikace, a tak je úspěšná detekce očekávaná. Na Obr. 17 jsou v obraze celkem 3 objekty (auta) ale pouze dvě jsou aplikací zaznamenány. Tato chyba je pravděpodobně způsobena rychlostí pohybu objektů a jejich různé velikosti díky vzdálenosti na kamery. Na Obr. 18 je vidět jeden obrázek z videa s názvem „přechod“. Jsou na něm vidět tři lidé, kteří jsou všichni aplikací detekovaní. Nedostatky aplikace jsou vidět při videích s názvy „ulice“ a „tanec“. Na Obr. 19 je vidět mnoho špatně detekovaných objektů. Jsou to především chyby spojování dvou objektů do jednoho nebo úplná falešná detekce objektu ve stromu v pravo dole, tato chyba může být způsobena nízkým rozlišením videa. Na Obr. 20 jde vidět mnohonásobná falešná detekce jednoho objektu. Na tomto videu má aplikace nejhorší výslekdy v důsledku toho, že lidé se na videu příliš nehýbou, pouze tančí na místě a tak jsou zachyceny ty části, které se zrovna hýbou a ne celá postava. Poslední ukázkové video s názvem „škola“ (Obr. 21) ukazuje schopnost aplikace detekovat velké množství objektů najednou. Detekce není ovšem dokonalá co se týče vícenásobné detekce na jednom objektu nebo zbytečně velké oblasti detekce.

Obr. 16 Ukázka detekce pohybu ve videu s názvem atrium

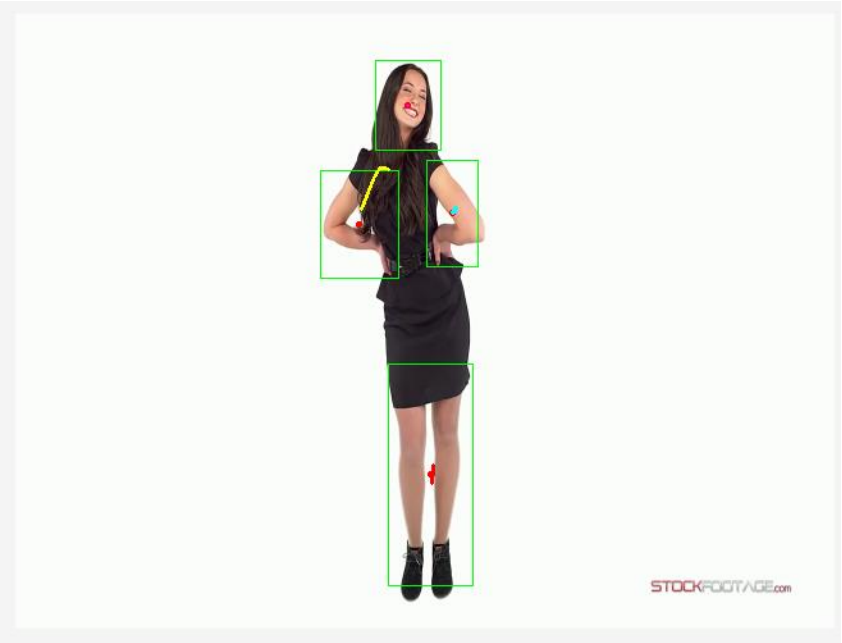
Obr. 17 Ukázka detekce pohybu ve videu s názvem dálnice



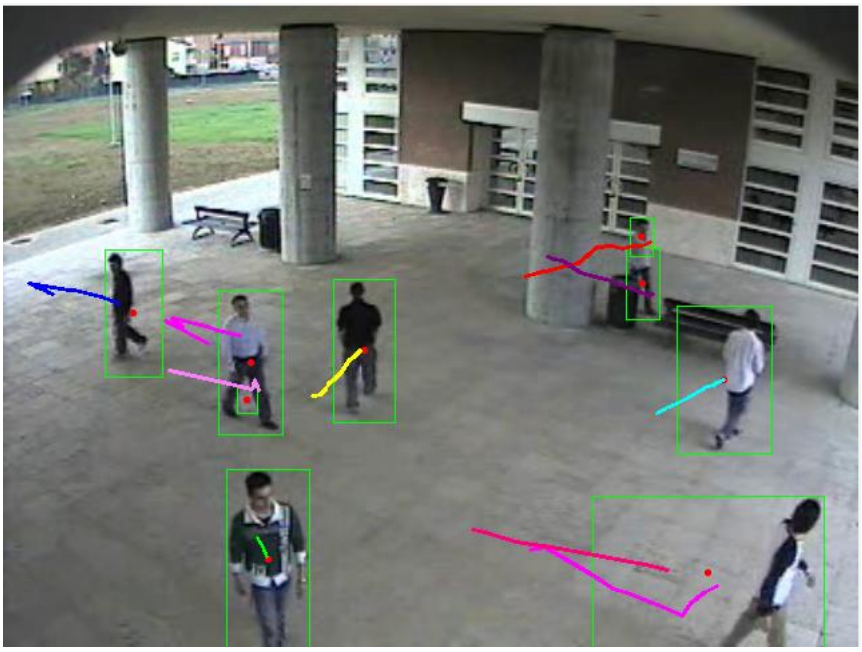
Obr. 18 Ukázka detekce pohybu ve videu s názvem přechod



Obr. 19 Ukázka detekce pohybu ve videu s názvem ulice



Obr. 20 Ukázka detekce pohybu ve videu s názvem tanec



Obr. 21 Ukázka detekce pohybu ve videu s názvem škola

4 Závěr

Cílem této bakalářské práce bylo prostudovat knihovnu pro počítačové vidění OpenCV, seznámit se s metodami pro odečítání pozadí z obrazu a následně vytvořit aplikaci a vhodně demonstrovat detekci pohybujících se objektů ve videu.

V práci byla popsána volně dostupná knihovna OpenCV, která disponuje metodami pro detekci pohybu. Jednou z těchto metod je metoda na odečítání pozadí, která byla teoreticky popsána a také použita v práci. Dále byly popsány algoritmy jako Kálmánův filtr, maďarská metoda nebo algoritmus BFS, které jsou potřebné pro funkčnost vytvořené aplikace.

V kapitole Implementace řešení je popsána vlastní implementace metod a algoritmů. V další části této kapitoly jsou popsány problémy, které provázely vytvoření této aplikace a jejich řešení.

V kapitole Výsledky je vytvořená aplikace prezentována a na více příkladech předvedena její funkčnost. Jednotlivé módy jsou zde vysvětleny jak z hlediska funkce, tak i z hlediska ovládání v grafickém uživatelském rozhraní. Jsou zde zmíněny i nedostatky těchto módu jako například občasná neschopnost aplikace rozeznat jeden objekt na druhého. Následně byla funkčnost aplikace otestována a ověřena na šesti různých videích. Tyto videa byla situována do různých prostředí, aby bylo možné ukázat schopnost aplikace detekovat objekty v různých situacích.

Výstupem této bakalářské práce je aplikace, která je schopná nejen detekovat pohybující se objekty ale i pracovat v různých módech na tuto detekci. Uživatel má možnost nakreslit jakoukoliv čáru, přes kterou je aplikace schopna detekovat pohyb a tento jev zaznamenat do konzole. Uživatel má také možnost nakreslit do okna aplikace libovolně velkou a tvarovanou oblast zájmu, ve které bude detekce prováděna.

Vytvořená aplikace a navržené metody by mohly být v praxi využity při zabezpečování prostorů v různých institucích, soukromých objektech nebo v dopravě. Aplikace se díky jednomu z implementovaných módu dá využít na počítání lidí, aut a dalších objektů, které překročí uživatelem nakreslenou linii.

Literatura

[1] BRADSKI, G., KAEHLER, A. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[2] BURGET, R. Teoretická informatika. Brno: Vysoké učení technické v Brně, 2012. s. 1-198. ISBN: 978-80-214-4897–1.

[3] ČÍKA, P., ZUKAL, M. Multimediální služby – počítačová cvičení. Brno, 2013. s. 1-67. ISBN 978-80-214-4721-9

[4] GitHub – Franciscodesign/Moving-Target-Tracking-with-OpenCV. The world's leading software development platform · GitHub [online]. Copyright © 2017 [cit. 02.06.2017]. Dostupné z: https://github.com/Franciscodesign/Moving-Target-Tracking-with-OpenCV

[5] KUHN, Harold W. The Hungarian method for the assignment problem. Naval research logistics quarterly, 1955, 2.1-2: 83-97.

[6] MAYBECK, Peter S., et al. Stochastics Models, Estimation, and Control: Introduction. 1979.

[7] Open Source Initiative. OpenSource.org. [online]. 10.12.2016 [cit. 2016-12-10]. Dostupné z: https://opensource.org/licenses/BSD-3-Clause

[8] PICCARDI, M. Background subtraction techniques: a review. In: Systems, man and cybernetics, 2004 IEEE international conference on. IEEE, 2004. p. 3099-3104.

[9] SEN-CHING, S. Cheung; KAMATH, Chandrika. Robust techniques for background subtraction in urban traffic video. In: Electronic Imaging 2004. International Society for Optics and Photonics, 2004. p. 881-892.

[10] ŠUBRT, Tomáš et al. Ekonomicko-matematické metody. Plzeň : Aleš Čeněk, 2011. s. 351 ISBN 978-80-7380-345-2.

[11] WELCH, Greg; BISHOP, Gary. An introduction to the Kalman filter. 1995.

[12] YILMAZ, JAVED, SHAH. Object Tracking: A survey. [online]. ACM Computing Surveys.

[13] ZIVKOVIC, Zoran. Improved adaptive Gaussian mixture model for background subtraction. In: Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on. IEEE, 2004. p. 28-31.

Seznam zkratek

BFS	Breadth-first search – prohledávání do šířky
BSD	Berkeley Software Distribution – licence pro svobodný software [OpenSource.org]
CV	Computer Vision – Počítačové vidění
MLL	Machine Learning Library – Knihovna strojového učení
HIGHGUI	High-Level Graphical User Interface – Grafické Uživatelské Rozhraní vyšších řádů
I/O	Input/Output – Vstup a výstup
CXCore	Jádro knihovny
RGB	Red, Green, Blue – Červená, Zelená, Modrá
HSV	Hue, Saturation, Value – ve významu Tón barvy, Sytost barvy, Hodnota jasu
BMP	BitMaP
MoG	Mix gaussiánů

Seznam příloh

A Obsah přiloženého CD

Přiložené CD obsahuje elektronickou verzi bakalářské práce. Ve složce „Dokumentace“ se nachází hlavní dokument pod názvem „xhanek01.pdf“. Ve složce „Zdrojové kódy“ se nachází vyexportovaný projekt z vývojového prostředí Eclipse. Ve složce „OpenCV“ se nachází knihovna OpenCV 3.2.0, která byla použita k vytvoření této aplikace. Ve složce s názvem „Testovací videa“ se nachází videa, na kterých byla aplikace testována.