VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Porovnání rychlostí Pythonových implementací vybraných algoritmů zpracování obrazu

Benchmark of Selected Digital Image Processing Algorithms Implemented in Python

2017 Marianna Mikulecká

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Marianna Mikulecká**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Porovnání rychlostí Pythonových implementací vybraných algoritmů zpracování obrazu
Benchmark of Selected Digital Image Processing Algorithms Implemented in Python

Jazyk vypracování: čeština

Zásady pro vypracování:

Algoritmy zpracování obrazu se postupně dostávají k uživatelům prostřednictvím internetových prohlížečů. Cílem práce je ověřit reálnou použitelnost takových algoritmů implementovaných v jazyce Python a porovnat rychlost jejich běhu s referenční implementací v tradičních prostředích jako je C++.

Ve své práci proveďte:
1. Nastudujte vybrané algoritmy zpracování obrazu a popište je.
2. Proveďte jejich implementaci v C++ a Pythonu. Použijte také knihovny Numpy a speciálního překladače Cython a PyPy.
3. Srovnejte rychlost implementací v různých prostředích (nativní běh, CPython, Cython, PyPy).
4. Naměřené výsledky v práci řádně zdokumentujte.

Seznam doporučené odborné literatury:

[1] Perona, P., Malik, J.: Scale-space and Edge Detection using Anisotropic Diffusion. IEEE Transactions on Pattern Analysis and Machine Intelligence, 12 (7): 629-639
[2] Sojka, E.: Digitální zpracování a analýza obrazů. VŠB - Technická univerzita Ostrava, ISBN: 80-7078-746-5, 2000

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Jan Gaura, Ph.D.**

Datum zadání: 01.09.2014

Datum odevzdání: 28.04.2017

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně. Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

V Ostravě 24. dubna 2017

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 24. dubna 2017

Chci poděkovat mému vedoucímu bakalářské práce Ing. Janu Gaurovi, Ph.D. za jeho vedení, konzultace a odbornou pomoc. Dále své rodině a přátelům za jejich podporu a pomoc.

Abstrakt

Tato bakalářská práce se zabývá digitálním zpracováním obrazu v Pythonu, popisem vybraných algoritmů, jejich implementací a použitou technologií. Porovnává rychlosti běhu vybraných algoritmů na zpracování obrazu v C++ a Pythonu, potom Pyhonu s využitím speciální knihovny Numpy a mezi překladači PyPy a Cython.

Klíčová slova: algoritmy zpracování obrazu, Anisotropní filtrace, mixtura Gaussiánů, Python, C++

Abstract

This Bachelor thesis deals with the digital image processing in Python, description of selected algorithms their implementation and used technology. Specifically, it is focused on comparison of speed among selected algorithms, implemented in languages C++ and Python, Python with NumPy library and compilers PyPy and Cython.

Key Words: image processing algorithms, Anisotropic filtering, mixture of Gaussians, Python, C++

Obsah

Seznam použitých zkratek a symbolů

JIT	–	just-in-time compilation
PIL	–	Python Imaging Library
HW	–	Hardware
SW	–	Software
RAM	–	random-access memory, paměť s přímým přístupem
DDR3	–	Double-Data-Rate 3 SDRAM, typ operační paměti
JSON	–	JavaScript Object Notation, formát pro přenos dat, lehce čitelný
RGB	–	Red, Green, Blue
CCD	–	Charge-Coupled Device, elektronická součástka na snímání obrazu
UML	–	Unified Modeling Language, grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systémů
GUI	–	Graphical User Interface
HTML	–	HyperText Markup Language

Seznam obrázků

Seznam tabulek

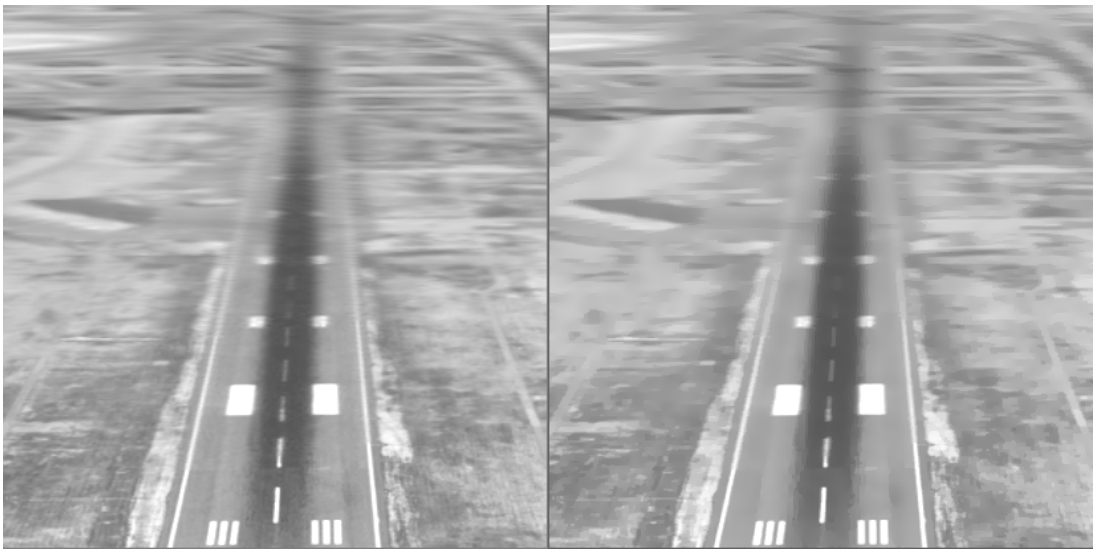
Seznam výpisů zdrojového kódu

1 Úvod

Cílem této bakalářské práce je implementovat algoritmy vybrané mým vedoucím práce a ověřit použitelnost jednotlivých implementací. Implementace bude v C++, Pythonu, Pythonu s využitím speciálních modulů jako je NumPy a OpenCV a dále použití speciálního překladače PyPy a Cython. V práci jsou nejdříve popsány vybrané algoritmy. Jedná se o Anisotropní filtraci a mixturu Gaussiánů. Následně je práce zaměřena na jednotlivé jazyky, speciální překladače a moduly včetně rozdílů jednotlivých implementací. V závěru je srovnání naměřených rychlostí implementací z různých prostředí.

Anisotropní filtrace je především využívána v počítačové grafice, hlavně tedy ve hrách kde je použita k dosažení dokonalého vzhledu s cílem minimalizace výpočetních nároků, které se provádí přímo na grafické kartě. Jedná se o vylepšení obrazu kdy nedochází k rozmazávání hran a je zachováno více detailů. Můžeme to vidět na obrázku 1.

Metoda mixtury Gaussiánů má největší využití při analýze obrazu z pevně nainstalovaných kamer. Je to jeden z nejkvalitnějších algoritmů, protože dokáže eliminovat vlivy venkovního prostředí. Uplatnění může najít třeba jako součást bezpečnostních systémů, které vyhodnocují pohyb osob na pozemcích nebo vozidel v dopravě. Jako vstupní data se používá sekvence snímků, výstupem je pak dvoubarevný obrázek označující „pixely v pohybu“. Nejde tedy o vyhodnocování tvarů ani seskupování pixelů do objektů. Takové algoritmy mohou následovat až za ním.



Obrázek 1: Vlevo bez Anisotropní filtrace, vpravo s filtrací.

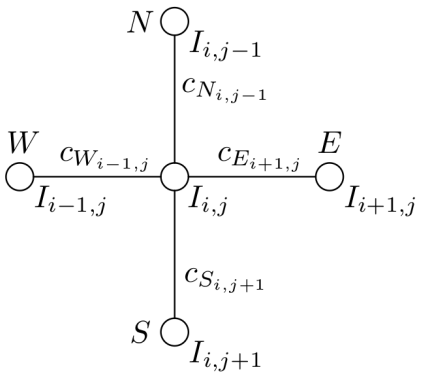
2 Algoritmy analýzy obrazu

Následující dvě podkapitoly se zaměřují na popis použitých algoritmů pro zpracování obrazu. Jak již bylo řečeno v úvodu, jedná se o dva algoritmy. Bude také vysvětlen jejich postup zpracování a budou uvedeny použité vzorce potřebné kvýpočtům.

2.1 Anisotropní filtrace

Anisotropní filtrování je metoda zaměřená na zlepšení kvality zobrazovaných textur na površích promítnutých pod šikmým úhlem pohledu. Obrazový signál obsahuje množství „ostrých“ hran (jasových skoků), které se pomocí obrazové filtrace odstraňují. Anisotropní filtrace je pouze jedna z možných metod, avšak její hlavní výhodou je, že při ní nedochází k rozmazání, jako například u „Gaussova rozostření“. To sice kostrbaté hrany také maskuje a je rychlejší na výpočet, ale celkově zhoršuje kvalitu obrazu, protože je určen jako filtr šumu nebo také „dolnofrekvenční propust“ obrazového signálu.

Na rozdíl od Gaussova rozostření nám Anisotropní filtrace [1] dává na výstupu obraz, který stále obsahuje ostré hrany. Filtrace je založena na jednoduchém fyzikálním jevu šíření energie z míst s vyšší koncentrací do míst s nižší koncentrací. Koncentraci energie si můžeme představit například jako hodnotu jasu každého pixelu. Všechny obrazové body, neboli pixely, jsou umístěny v mřížce, které tvoří síť. Sousedící pixely jsou propojeny mezi sebou vazbami, které závisí na míře jejich podobnosti. Filtrace je rozložena v čase, kdy v každém kroku přeteče část energie jasu mezi sousedními pixely. Po provedení zadaného počtu kroků se iterace zastaví. Model sousedících pixelů je zobrazen na obrázku 2.



Obrázek 2: Model pixelu na pozici *i, j* se sousedy na *severu (N)*, *jihu (S)*, *západě (W)* a *východě (E)*.

Pro výpočet vazeb (vodivostí) mezi jednotlivými pixely se použijí následující vztahy:

$$c^t_{N_{i,j}} = g(\| \nabla_N I^t_{i,j} \|) \tag{1}$$

$$c^t_{S_{i,j}} = g(\parallel \nabla_S I^t_{i,j} \parallel) \tag{2}$$

$$c^t_{E_{i,j}} = g(\parallel \nabla_E I^t_{i,j} \parallel) \tag{3}$$

$$c^t_{W_{i,j}} = g(\parallel \nabla_W I^t_{i,j} \parallel) \tag{4}$$

Kde g je definováno následovně:

$$g(\nabla I) = e^{\left(-\frac{|\nabla I|^2}{\sigma^2}\right)} \tag{5}$$

a $\nabla_N I_{i,j} = I_{i-1,j} - I_{i,j}$. Pro ostatní směry je logika výpočtu stejná.
Nová hodnota se v každém kroku iterace počítá takto:

$$I^{t+1}_{i,j} = I^t_{i,j} + \lambda[c_N \cdot \nabla_N I + c_S \cdot \nabla_S I + c_E \cdot \nabla_E I + c_W \cdot \nabla_W I]^t_{i,j} \tag{6}$$

$$= I^t_{i,j}\left(1 - \lambda(c_N + c_S + c_E + c_W)^t_{i,j}\right)\lambda(c_N I_N + c_S I_S + c_E I_E + c_W I_W)^t_{i,j}, \tag{7}$$

kde $I^{t+1}_{i,j}$ je nová hodnota pixelu se souřadnicemi i, j v čase $t+1$ a $I^t_{i,j}$ je jeho původní hodnota na souřadnicích i, j v čase t. Použitím posledního vzorce na každý pixel v čase t, vypočítáme nové hodnoty jasu v čase $t+1$. Nesmíme však zapomenout, že nové hodnoty se musí ukládat do nového obrazu, aby neovlivňovaly výpočty sousedních pixelů.

Doporučuje se [2] nastavit následující konstanty pro výpočet na hodnoty $\sigma = 0{,}015$ a $\lambda = 0{,}1$.

2.2 Mixtura Gaussiánů

Účel této metody je rozlišení prvků statických (okolní prostředí) od prvků dynamických, jak můžeme vidět na obrázku 3. Efektivní bude pouze v případě pevného umístění kamery, neměla by se ani chvět, jinak bude docházet ke zkreslení výsledku a výsledek nebude přesný.

2.2.1 Základní princip detekce pohybu v obrazu

Samotným jádrem počítačového „vidění“ je v současnosti nejpoužívanější CCD kamera. Již od vzniku filmu a první televize bylo pro projekci pohyblivého obrazu nutné používat alespoň 25 snímků za sekundu a to z důvodu setrvačnosti oka, kdy při této frekvenci již přestáváme vnímat, že se rychle za sebou přepínají statické snímky. U digitálního zpracování obrazu určeného pro stroje toto pravidlo neplatí, ale frekvence snímků se volí podle požadavků aplikace, jak často potřebujeme informaci obnovovat a také jak rychle na ní musí stroj reagovat.

Obrázek 3: Ukázka výstupu algoritmu na detekci pohybu.

Před vysvětlením principu detekce pohybu v obraze, je třeba vysvětlit dva základní pojmy. Jsou jimy pozadí a popředí obrazu.

- **Pozadí obrazu** - reprezentuje nám pevnou a neměnnou strukturu. Z hlediska jednoho pixelu představuje konstantní jas.
- **Popředí obrazu** - tvoří pohybující se objekty. Z hlediska jednoho pixelu se jedná o nenadálou změnu jasu, která ovšem trvá minimálně několik po sobě jdoucích snímků. Tím se dá odlišit od nahodilé poruchy, která se vyskytne pouze jednou.

Každý pixel obrazu je nezávislým elementem plochy dvourozměrné matice, jedná-li se o černobílý obraz. V případě barevného obrazu by se jednalo o trojrozměrnou matici (každý pixel by obsahoval tři barvy, RGB). V tomto případě může jeden pixel nabývat pouze jasových hodnot od černé, přes stupnici šedé až k bílé. Cílem je v reálném čase vyhodnocovat nenadálé změny jeho jasu a zároveň eliminovat nahodilé poruchy, jako například šum nebo déšť. Je potřeba vytvořit nějakou referenční úroveň pozadí, od které se budou vyhodnocovat odchylky.

Metod vytváření modelu pozadí je více a můžeme je rozdělit do dvou hlavních skupin:

a) Nerekurzivní – algoritmus využívá posloupnost několika posledních zachycených snímků („Vážená střední hodnota“, „Dočasná střední hodnota“).

b) Rekurzivní – pozadí je modelováno ze všech předchozích snímků jako pravděpodobnost výskytu jasu každého pixelu, přičemž odpadá nutnost snímky zaznamenávat. Hodně se používá metoda tzv. „klouzavého Gaussova průměru“. Mixtura Gaussiánů patří k těm nejnáročnějším rekurzivním technikám.

2.2.2 Metoda modelování pozadí pomocí směsi Gaussiánů

Pro nás je pixel elementární zářivý bod, který skokově změní jas vždy s přijetím dalšího nového snímku z kamery. Protože se jedná o rekurzivní metodu, tak nejprve vytvoříme matematický

model jednoho pixelu, který bude vyjadřovat pravděpodobnost výskytu jeho konkrétního jasu rozloženou v čase. Tuto pravděpodobnost bude reprezentovat Gaussova chybová křivka normálního rozdělení. Budou v ní tedy zahrnuty úrovně jasů daného pixelu i z předchozích snímků. Čím déle bude jas tohoto pixelu konstantní, tím větší pravděpodobnost výskytu stejného jasu bude i v následujícím (ještě neexistujícím) snímku.

Venkovní prostředí často obsahuje objekty, které se například třepetají ve větru – listí, vlajky, déšť, sníh, vodní hladina atd. Snímáním takové scény dostáváme tzv. multimodální obraz, který pomocí klouzavého Gaussova průměru nelze správně vyhodnocovat, protože by se tyto objekty jevily jako trvalý pohyb.

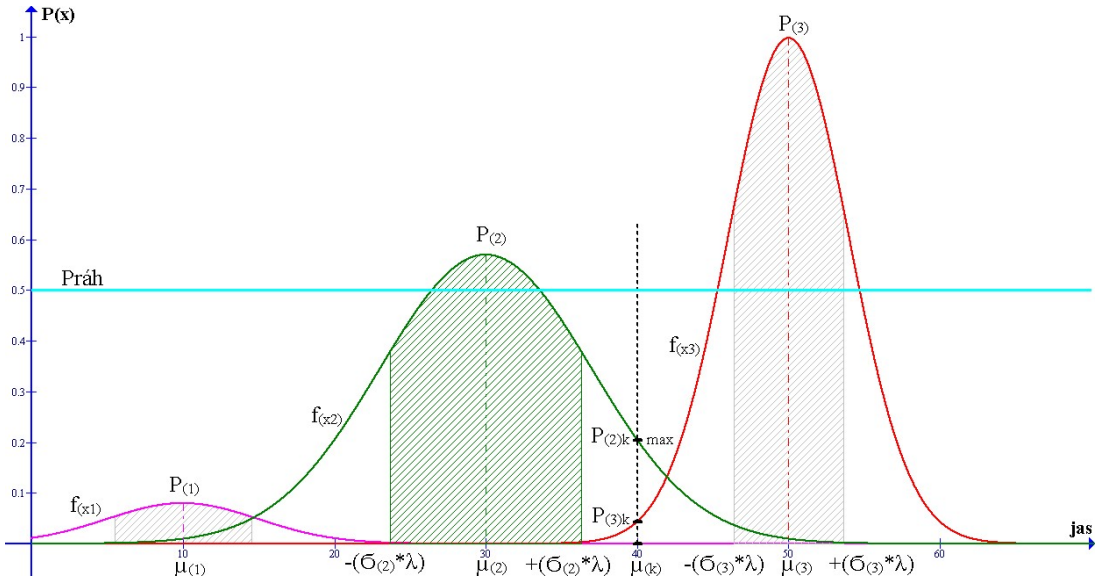
Princip metody směsi Gaussiánů vychází z metody klouzavého Gaussova průměru, ale povyšuje jeho úroveň, protože eliminuje výše zmíněné nedostatky. Pro každý pixel vytvoříme směs, nejčastěji pěti Gaussových křivek (tzv. gaussiánů), přičemž každá se vztahuje k jiné úrovni jasu (střední hodnotě pravděpodobnosti jejího výskytu). Zvětšováním počtu křivek se již reálně nezlepšuje výsledek. Tím docílíme toho, že na rozdíl od předchozí metody může v každém pixelu vzniknout i několik jasů s vyšší pravděpodobností výskytu. Změna jednoho jasu na jiný pak nemusí být ihned interpretována jako pohyb, ale teprve až počet jeho opakování výrazně převýší svou pravděpodobnost výskytu. Můžeme tedy říci, že se program při spuštění musí nejprve naučit, co je pozadí a co popředí.

Poskládáním jednotlivých Gaussových křivek dostaneme jeden výsledný graf – mixturu Gaussiánů pravděpodobností výskytu libovolného jasu daného pixelu. S příchodem každé nové hodnoty jasu se všechny gaussiány vždy znovu aktualizují, čímž se hodnota jasu pozadí zároveň nepostřehnutelně přizpůsobuje malým změnám osvětlení scény.

Když uspořádáme tyto jednotlivé mixtury do jednotné matice rozměrů obrazového pole, dostaneme tzv. „model pozadí“, který na rozdíl od obyčejného obrazu obsahuje množství statistických informací o jednotlivých objektech na scéně, je zbaven všech nahodilých poruch, a zároveň je schopen se přizpůsobovat pomalým změnám. Tento model budeme používat jako výchozí (referenční) předpoklad pro detekci pohybu popředí. Získaný obraz popředí pak zobrazíme pomocí dvou úrovní jasu.

Nevýhodou této metody je však nutný několikanásobně větší výpočetní výkon. Dále pak touto metodou nelze eliminovat chvění kamery. Stejně tak objekty, které se na delší dobu zastaví, se začnou stávat součástí pozadí. Obdobná situace nastane u podlouhlých pohybujících se objektů (dlouho zůstává stejný jas pixelů), kdy se v něm začnou objevovat díry. Také můžeme pozorovat tzv. duchy, například po přejezdu tramvaje, se pozadí za ní chvíli jeví jako pohybující se objekt.

Příklad pravděpodobnosti povrchů pro jeden pixel je zobrazen v grafu na obrázku 4. Jsou zde zobrazeny tři gaussiány. Ten první má příliš malou pravděpodobnost výskytu a protože nedosahuje prahové hodnoty T, není považován za model pozadí. Jedná se tedy o nějaký šum. Zbylé dva mají významnou pravděpodobnost výskytu. Jsou tedy součástí multimodálního pozadí. Po přiřazení aktuálního jasu μ_k ke Gaussiánu s největší velikostí v daném bodě, v tomto případě ten druhý, se musí provést kontrola, jestli se nachází v poli s největší pravděpodobností



Obrázek 4: Pravděpodobnost povrchů pro jeden pixel, tři povrchy.

výskytu, tedy $\mu_2 \pm \sigma_2$ (vyšrafovaná oblast). Jak je vidět, tuto podmínku nesplňuje a proto bude vyhodnocen jako popředí.

2.2.3 Detekce popředí pomocí mixtury Gaussiánů

Abychom mohli detekovat pohyb v popředí, musíme nejprve vytvořit a poté stále aktualizovat model pozadí. Zaměříme se tedy na tvorbu modelu pozadí [3] [4].

Základní vzorec pro výpočet Gaussovy chybové křivky normálního rozdělení je bohužel také časově nejnáročnějším. Zde je uvedený už zjednodušený vzorec pro černobílý obraz s 256-ti stupni šedi

$$f_{X|k}(X|k,\Theta_k) = \frac{1}{\sigma_k\sqrt{2\pi}} e^{\left(-\frac{(X-\mu k)^2}{2\sigma_k^2}\right)}. \tag{8}$$

Pro každý pixel x se musí opakovat k-krát, kde k je zvolený počet povrchů. Vždy po příchodu nového jasu pixelu se musí vypočítat váha aktuální hodnoty x pro každý jeden gaussián. Je to hodnota, kterou vytíná jas pixelu na Gaussově křivce všech povrchů.

Následně se pro daný jas x počítá součet všech vah, které vytíná na každém gaussiánu podle

$$f_X(X|\Phi) = \sum_{k=1}^{K} P(k) f_{X|k}(X|k,\Theta_k). \tag{9}$$

Je to způsob, jak z několika nezávislých gaussiánů dostat jeden společný průběh.

Po tomto kroku je čas na výpočet Bayesova teorému. Jde o pouhý poměr maximální váhy a součtu vah z předcházejícího výpočtu. Bayesův teorém budeme potřebovat k aktualizaci stávající

pravděpodobnosti povrchu, ke kterému bude nyní přiřazen nový jas x. Na rozdíl od gaussiánů je Bayesův teorém pouze jeden pro každý pixel. Vypočítá se následovně:

$$P(k|X,\Phi) = \frac{P(k) f_{X|k}(X|k,\Theta_k)}{f_X(X|\Phi)} \tag{10}$$

Dále uvedené vztahy jsou určeny k aktualizaci modelu pozadí. Proměnné označené stříškou zde znamenají nově aktualizované hodnoty.

Nejdříve se aktualizuje index povrchu, na jehož gaussiánu vytíná jas x největší hodnotu pravděpodobnosti. Ten získáme tak, jak jsme je vypočítali ve vzorci 8

$$\widehat{k} = \arg\max_k P(k) f_{X|k}(X|k,\Theta_k). \tag{11}$$

Tímto přidělíme k hodnotě jasu x index k, který nám dále bude určovat, který povrch se všemi jeho parametry budeme tímto jasem aktualizovat.

Poté se aktualizuje hodnota pravděpodobnosti na pozici toho gaussiánu, ke kterému byl přiřazen aktuální jas x:

$$\widehat{P}(k) = (1-\alpha)P(k) + \alpha P(k|X,\Phi). \tag{12}$$

Kromě výše získaného Bayesova teorému zde ovlivňuje výpočet také koeficient rychlosti adaptace modelu pozadí α . Doba adaptace se používá řádově jednotky až desítky vteřin.

Další se aktualizuje střední hodnota pravděpodobnosti tohoto vybraného povrchu, čímž se v malých krocích přizpůsobuje pozadí změnám osvětlení scény. Použijí se k tomu tyto dva vzorce:

$$\rho_k = \alpha P(k|X,\Phi)/\widehat{P}_k, \tag{13}$$

$$\widehat{\mu} = (1-\rho_k)\mu_k + \rho_k X. \tag{14}$$

Naposledy se aktualizuje střední odchylka pravděpodobnosti vybraného povrchu

$$\widehat{\sigma}_k^2 = (1-\rho_k)\sigma_k^2 + \rho_k((X-\widehat{\mu}_k)\circ(X-\widehat{\mu}_k)). \tag{15}$$

V následujících krocích je třeba na základě zvoleného kritéria vybrat ty pixely x, které zařadíme do „popředí scény“. Budeme postupovat podle následující podmínky a výsledek bude zobrazen jako dvoubarevný obraz.

Jakmile bude odchylka mezi aktuálním jasem x a střední hustotou pravděpodobnosti μ větší jako definovaná mez podle

$$|x_t - \mu_k| > \lambda\sigma_k, \tag{16}$$

bude vyhodnocen jako popředí – čili pohybující se. Tato mez je sice ovlivnitelná předdefinovaným parametrem λ, ale zároveň se odvíjí od statistické odchylky σ. Pokud je tedy v dané oblasti

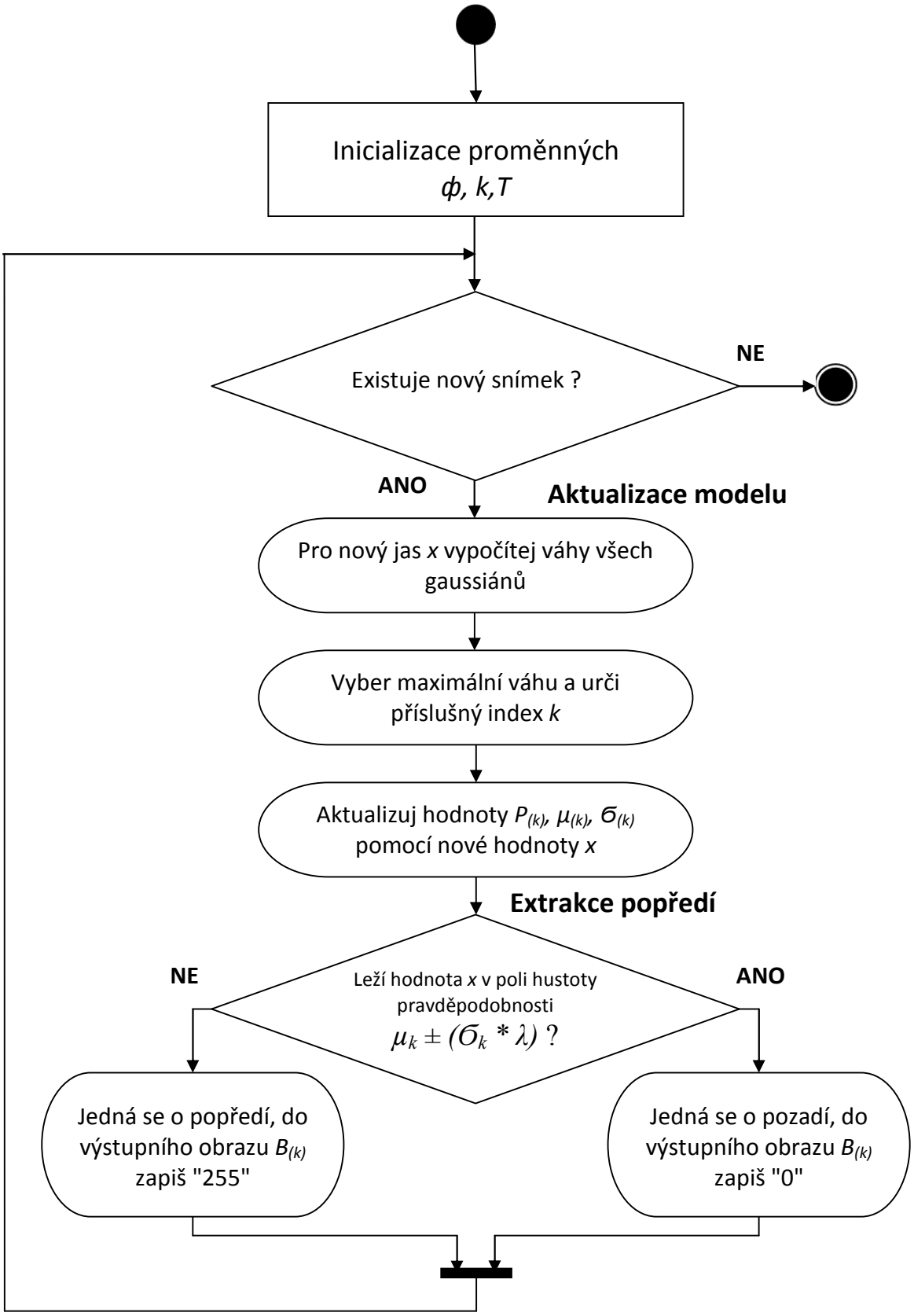
pixelu například chvějící se list stromu, bude v modelu pozadí gaussián s vetší střední odchylkou pravděpodobnosti a hranice potřebná k detekci popředí se tím také zvýší.

Zde je uveden význam použitých symbolů:

μ - střední hodnota pravděpodobnosti
σ - střední odchylka pravděpodobnosti
Θ_k - množina $\{\mu_k, \sigma_k\}$
Φ - množina parametrů Θ_k pro všechny povrchy k
α - ovlivňuje rychlost adaptace modelu na změnu pozadí
T - práh, který rozhodne, jestli se jedná o popředí nebo pozadí
x - jas
k - počet povrchů
$P_{(k)}$ - pravděpodobnost výskyku k-tého povrchu na pozici analyzovaného pixelu, zvaný také „váha“ (Inicializuji všechny na hodnoty *1/k*, protože suma $P_{(k)}$ je vždy = 1)
λ - konstanta pro určení prahové hodnoty při extrakci popředí

2.2.4 Diagram aktivit

Pro lepší znázornění činnosti programu je na následující stránce uveden jeden z UML diagramů, sloužícího ke grafickému znázornění běhu programu. Konkrétně se jedná o diagram aktivit.


Inicializace proměnných
ϕ, k,T
Existuje nový snímek ?
NE
ANO
Aktualizace modelu
Pro nový jas x vypočítej váhy všech gaussiánů
Vyber maximální váhu a urči příslušný index k
Aktualizuj hodnoty $P_{(k)}$, $\mu_{(k)}$, $\sigma_{(k)}$ pomocí nové hodnoty x
Extrakce popředí
Leží hodnota x v poli hustoty pravděpodobnosti
$\mu_k \pm (\sigma_k * \lambda)$?
NE
ANO
Jedná se o popředí, do výstupního obrazu $B_{(k)}$ zapiš "255"
Jedná se o pozadí, do výstupního obrazu $B_{(k)}$ zapiš "0"

3 Python

Následující kapitoly jsou zaměřeny na popis použitých programovacích jazyků. Jako první se seznámíme s Pythonem. To je objektově orientovaný, skriptovací jazyk. Je víceúčelový a platformně nezávislý. Aplikace lze vytvářet mnohem rychleji než v klasických jazycích jako je C++.

Jeho nevýhodou je pomalá rychlost provádění programu. Není totiž plně kompilovaný. Jedná se o interpretovaný jazyk. Program je nejdříve překompilován do interního bajtového kódu, který je pak vykonán interpretem Pythonu.

Takže z jedné strany je jazyk efektně implementován, ale na straně druhé je pomalejší běh programu. Moderní počítače mají dnes tak vysoký výpočetní výkon, že je rychlost jazyka většinou zanedbatelná. Problém zůstává u matematických výpočtů, které jsou značně pomalé. Dobré je, že se program v Pythonu dá jednoduše spojit s moduly psanými v jazyce C/C++ pro náročné výpočetní operace. Dále s využitím překladače PyPy či Cythonu se tyto výpočty dají do určité míry zrychlit. Stejně tak by mělo přinést zrychlení, když použijeme modul NumPy, určený pro práci s maticemi, což obrázky jsou.

Zaměřme se na výhody Pythonu. Těmi jsou:

- Úplně automatické řízení paměti.
  Nemusíme se starat o alokaci a dealokaci paměti.
- Datové typy nejsou spojeny s proměnnými, ale s objekty.
  Typ proměnné nemusí být předem deklarován a můžeme jí přiřadit hodnotu libovolného typu. Stejně tak můžeme seznamu přiřadit objekty mnoha různých typů.
- Velice jednoduchá pravidla syntaxe.
  Python je přibližně 5 krát stručnější než jazyk C, je tedy hodně přímočarý. Například operace s řetězci jsou velice rychle psané. Povinné je odsazování k odlišení jednotlivých bloků kódu a tím je i velice přehledný, jak můžeme vidět v ukázce kódu 1 .

Ukázka syntaxe na Fibonacciho posloupnosti.

```
def fib(n):
    a, b = 0.0, 1.0
    for i in range(n):
        print a
        a, b = a + b, a
```

Výpis 1: Python - Ukázka syntaxe na Fibonacciho poslounosti.

V programu jsou použity následující moduly:
time - práce s funkcemi času, měření rychlosti běhu programu
json - slouží pro získání dat z videa
subprocess - přístup a obsluha externích procesů
Tkinter - vytváření oken, tvorba GUI
PIL - práce s grafickými daty
platform - informace ohledně platformy
math - matematické funkce
copy - vytvoření kopie

3.1 Knihovna NumPy

NumPy, neboli Numerical Python, je knihovna sestávající z objektů vícerozměrných polí a kolekcí rutin potřebných k práci s nimi. Díky NumPy mužeme na polích provádět různé matematické a logické operace, manipulovat s rozměry, jsou zde funkce pro lineární algebru nebo generátor náhodných čísel. Ve spojení s balíčky SciPy (Scientific Python) a Mat-plotlib (plotting library) poslouží dobře jako náhrada za Matlab.

Nejdůležitější objekt je N-rozměrná matice zvaná `ndarray`. Při vytváření pole je určen typ prvků. Pole jsou tedy homogenní a staticky typovaná. Ukážeme si základní práci s touto knihovnou:

```
>>> A=numpy.array([[1,3],[2,4]])
>>> type(A) # Typ objektu
<type 'numpy.ndarray'>
>>> A.dtype # Typ prvků
dtype('int64')
>>> A.shape # Rozměr
(2, 2)
>>> A.size # Počet prvků
4
>>> numpy.zeros((2,3)) # Matice nul. Matici jedniček získáme použitím numpy.ones.
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> numpy.diag([1,2,3]) # Diagonální matice
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
>>> A*A # Násobení matic
array([[ 1,  9],
       [ 4, 16]])
```

V kódu pro mixturu Gaussiánů je použito `numpy.empty(shape, dtype)`. To nám vrátí novou matici daného rozměru a typu, ale bez inicializace záznamů. Je tedy na vytváření rychlejší [5] než kdyby sme měli pole rovnou plnit třeba nulami.

3.2 PyPy

První interpret Pythonu je napsán v jazyce C, nazývá se CPython. PyPy je náhradou za CPython. Tento interpretr je napsaný v Pythonu. Využívá RPython (Restricted Python), což je staticky typovaná a zkompilovaná podmnožina Pythonu a je překládána do jazyka C.

Největší výhodou použití překladače PyPy [6] [7] je jeho rychlost. Té je dosaženo za pomoci Just-in-time (JIT) kompilace, tedy převedení zdrojového kódu z Pythonu do strojového kódu počítače rovnou za běhu programu a tím dojde ke zrychlení jeho běhu.

Vzhledem k tomu, že JIT kompilace zabere nějaký čas, nemůžeme počítat se zrychlením u programů, které běží jen pár vteřin, protože se musí analyzovat často prováděné části programu. Obdobný výsledek bude v programu, který stráví mnoho času v Runtime knihovnách (C funkce), to jsou specifické funkce prováděné za běhu programu, jako třeba správa paměti nebo standartní vstup a výstup z programu. PyPy pracuje s běžnými programy v Pythonu. Není tedy třeba je kvůli tomu nijak upravovat.

Nevýhodou používání PyPy je nedostatek podpory pro rozšiřující moduly CPythonu. Příkladem je třeba knihovna NumPy. Pro spuštění programu stačí zadat v terminálu příkaz:

```
# pypy filtration.py
```

3.3 Cython

Jedná se o programovací jazyk [11], který spojuje Python se staticky typovaným systémem jako má C a C ++. Překladač Cython přeloží zdrojový kód Cythonu do kódu C nebo C++ a ten je následně zkompilován jako rozšiřující modul pro Python.

Síla Cythonu pochází z kombinace Pythonu s lehkým přístupem k Céčku. Vzhledem k příkladu syntaxe Pythonu na Fibonaciho posloupnosti ukázanou ve výpisu kódu 1, si zde ukážeme rozdíl na stejné funkci `fib` ještě mezi Céčkem a Cythonem. Už samotný kód v Pythonu je již funkční Cython kód. Ale aby jsme dosáhli zlepšení výkonu, musíme ho upravit na syntaxi charakteristickou pro Cython. Nejdříve se podíváme na výpis kódu 2 funkci `fib` napsanou v C:

```
double fib(int n)
{
    int i;
    float a = 0.0, b = 1.0, tmp;
    for (i = 0; i < n; ++i)
    {
```

```
        printf("%.f ", a);
        tmp = a;
        a = a + b;
        b = tmp;
    }
}
```

Výpis 2: Jazyk C - Ukázka syntaxe na Fibonacciho poslounosti.

Teď, když smícháme typy proměných z Céčka s kódem v Pythonu získáme staticky typovaný Cython, jak vidíme ve výpisu kódu 3

```
def fib(int n):
    cdef int i
    cdef float a = 0.0, b = 1.0
    for i in range(n):
        print a
        a, b = a + b, a
```

Výpis 3: Cython - Ukázka syntaxe na Fibonacciho poslounosti.

kde pomocí `cdef` deklarujeme statické proměnné.

U kompilovaných jazyků se datové typy ověřují jenom jednou během kompilace. Můžeme zde tedy vidět nevýhodu, proč je dynamicky typovaný Python velice pomalý v cyklech. Výraz `a + b` může znamenat cokoliv (sčítání čísel celých, desetinných nebo řetězců, atd.). I když my víme, že budeme počítat celou dobu jen s čísly typu integer, Python to neví. Proto si musí při každém novém průchodu cyklem zjistit jakého typu je vstupní proměnná `a` a `b`.

Jak již bylo zmíněno, kód Cythonu se potřebuje před spuštěním kompilovat. Abychom to mohli udělat, nejprve si uložíme zdrojový kód do souboru s koncovkou `*.pyx`. Poté musíme vytvořit soubor pojmenovaný `setup.py`, který nám sestaví modul napsaný v Cythonu. Tento skript vypadá následovně:

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Distutils import build_ext

ext_modules = [Extension("fibo",
               ["fib.pyx"],)]
setup(
```

```
    cmdclass={'build_ext': build_ext},
    ext_modules=ext_modules
)
```

Výpis 4: Cython - skript setup.py

a spustíme ho pomocí tohoto příkazu v terminálu:

```
# python setup.py build_ext --inplace
```

Kromě statických proměnných můžeme deklarovat také typ třídy pomocí `class`, nebo funkcí pomocí `cdef`. Velice užitečné je, když spustíme zdrojový kód překladačem `cython` s parametrem `-a`

```
# cython -a fibonacci.pyx
```

Vyprodukuje se nám tím HTML stránka, kde můžeme po rozkliknutí žlutých řádků vidět, čím se jednotlivé části kódu nahradí během převedení z Pythonu do Céčka.

Řádky, které jsou podbarvené slabě žlutou barvou, se dají ignorovat a to i v případě že se opakují často. Nebudou nás stát moc výpočetního času. Jedná se hlavně o věci jako je kontrola hranic polí nebo dělení nulou. Ale pokud se tam často objevují řádky sytě žluté barvy, měli bychom je opravit. Jsou to náročná místa kódu které budou stát hodně výpočetního času (třeba i stonásobky). Příklad bude uveden v kapitole 5.1.2.

Pokud budeme chtít spustit samotný program, tedy modul s naší funkcí *fib*, stačí si spustit Python v terminálu, naimportovat modul fibonacci a zavolat funkci fib().

```
>>> import fiby
>>> fibo.fib(6)
0.0
1.0
1.0
2.0
3.0
5.0
```

4 C++

Céčko je čistě strukturovaný programovací jazyk. C++ je rozšířením jazyka C ve kterém lze programmovat objektově. Je to jeden ze starších jazyků a je velice široce využívaný. Jak již bylo řečeno musí se kompilovat.

Zdrojové kódy z jazyka C++ jsou kompilovány pomocí příkazu:

```
# g++ main.cpp -o main ‘pkg-config --cflags --libs opencv‘
```

Jelikož je v kódech použita knihovna OpeCV, budeme se jí zde věnovat. Knihovna OpenCV, neboli Open Source Computer Vision, je vyvíjena od roku 1999 a je určena pro práci s obrazem [12]. Je to svobodný software a lze ho využít v jazycích C, C++, Python a Java. Podporuje OS Linux, Windows, Mac OS, iOS a Android.

Obsahuje více než 2500 optimalizovaných algoritmů, zahrnující klasické i nejmodernější algoritmy počítačového vidění a strojového učení (například rozpoznávání obličeje, sledování či rozeznávání objektů a další). Dále v ní najdeme nástroje pro práci jak s obrázky tak s videem, transformace obrazu, jednoduchá práce s okny a uživatelským rozhraním, 3D modely, atd.

V algoritmech popsaných v této práci jsou využity hlavně její základní funkce. Jde například o funkce cvNamedWindow která slouží k vytvoření okna, funce cvCaptureFromFile sloužící pro načtení videa a s tím spojená funkce cvQueryFrame, která vezme a vrátí následující snímek videa. Pomocí metody CV _IMAGE_ELEM je možné získat hodnoty z jednotlivých pixelů.

5 Implementace

Tato kapitola je zaměřena na implementaci obou algoritmů a na rozdíly mezi jednotlivými kódy. Implementace v čistém Pythonu, tzn. bez použití knihoven NumPy a OpenCv (které nepodporuje [10] překladač PyPy), bude sloužit jako výchozí kód pro další modifikace. A tedy pro implementaci kódu s knihovnou Numpy a potom pro Cython a PyPy.

5.1 Anisotropní filtrace

Na vstupu programu je barevný obrázek. Nejdůležitější funkcí v kódu je **anisotropie**, která se postará o samotnou anisotropní fitraci obrázku. Jejími parametry jsou: vstupní obrázek, případně jeho rozměr a dvě defaultně nastavené hodnoty. Míra filtrace je závislá na počtu nastavených cyklů. Výstupem programu je zobrazení vstupního obrázku, obrázku převedeného na stupně šedi a nakonec obrázek po zpracování algoritmem filtrace.

Na obrázku 5 můžeme vidět výsledek filtrace v závislosti na zvoleném počtu cyklů, kolikrát byla provedena (vedle vstupního obrázku je tam výsledek po 50 a 500 cyklech, pod nimi jsou priblížené detaily). Můžeme vidět, jak se zvyšujícím se počtem cyklů začínají být přechody velmi výrazné.

5.1.1 Rozdíl implementací pro Python a NumPy

Chceme-li zpracovat obraz, je nutné z něj nejdříve získat hodnoty jednotlivých pixelů spolu s rozměry obrázku. K tomu nám v Pythonu poslouží funkce z použité knihovny PIL(Python Image Library). Jak můžeme vidět ve výpisu kódu 5, nejprve se otevře a identifikuje obrázek, poté si uložíme jeho rozměry a nakonec si uložíme získané hodnoty pixelů. Zatímco v NumPy se jedná o použití knihovny SciPy.

```
# Python
img = Image.open('../lena.png')
shape = img.size
obrazek = img.getdata()

# NumPy
obrazek = misc.imread('../lena.png')
h, w = img.shape
```

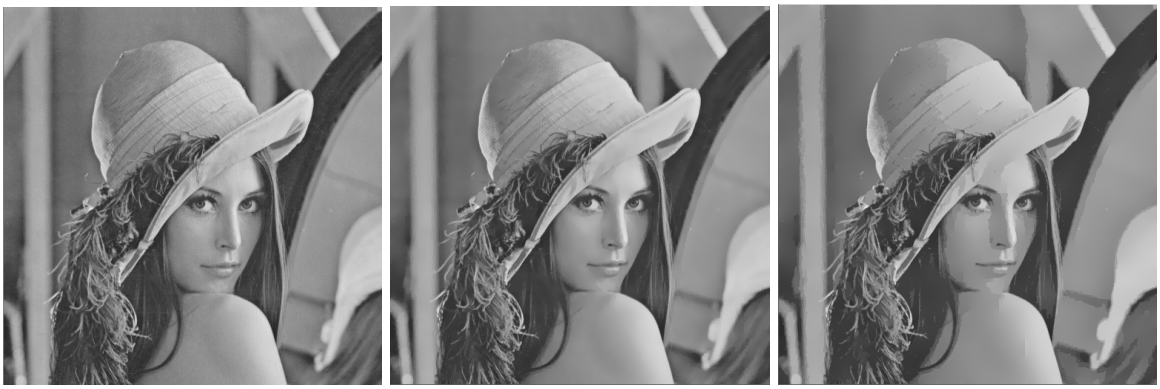
Výpis 5: Python - získání hodnot pixelů.

Pří získávání dat zde vznikná rozdíl. V Pythonu nám funkce getdata vrátí 1D pole entic (tuple), v NumPy dostaneme pole dvourozměrné.

Další důležitou částí je převedení barevného obrázku na černobílý, kterou je vidět ve výpisu 6. Pro převod obrázku se zavolá funkce greyScale a vypadá odlišně pro pole jednorozměrné nebo dvourozměrné.

```
# Python - 1D pole
    for i in range(v*s):
        r, g, b = img[i]
        greyPicture[i] = (r*0.299)+(g*0.587)+(b*0.114)
# NumPy - 2D pole
    for row in range(0, v):
        for column in range(0, s):
            r,g,b = img[row][column]
            grayPicture[row][column] = (r * 0.299) + (g * 0.587) + (b * 0.114)
```

Výpis 6: Python - převod RBG obrázku na černobílý.



(a) Vstupní (b) 50 cyklů (c) 500 cyklů

(d) Výřez (e) 50 cyklů, výřez (f) 500 cyklů, výřez

Obrázek 5: Výstup z anisotropní filtrace.

Před samotnou anisotropní filtrací se každý jeden pixel dělí 255 (hodnota jasu je v rozmezí 0 - 255) a to z toho důvodu, že na výpočet je lepší použít float hodnoty kvůli přesnosti. Po výpočtu algoritmu filtrace se opět jednotlivé pixely vynásobí stejnou hodnotou jakou jsme je dělili.

Zde ve výpisu 7 je vidět rozdíl, že v Pythonu musíme projít každý jeden pixel po pixelu, zatímco s využitím NumPy jde velice jednoduše provést operaci dělení na celé pole. V obou případech vytváříme kopii obrázku.

```
# Python
aniPicture = [greyPicture[i] / 255.0 for i in range(h * w)]

# NumPy
aniPicture = grayPicture.copy() / 255.0
```

Výpis 7: Python - převedení hodnot pixelů na typ float.

5.1.2 Cython

V Cythonu mělo největší přínos statické typování proměnných. Nejdůležitější bylo, aby funkce, kde jsou obsaženy všechny výpočty, byla co nejvíce podobná Céčku. To z toho důvodu, že se v ní stráví nejvíce času, tak aby docházelo k co nejmenšímu zdržování. Při pokusu o definování typů jednotlivých funkcí to nepřineslo žádné zlepšení a proto zůstaly v původním tvaru.

V následujícím výpisu kódu 8 je vidět, že mají všechny použité proměnné své typy. Na HTML výstup z této části kódu se můžeme podívat na obrázku 6. Jde vidět, že vnitřek funkce není nikde vyznačen sytě žlutou barvou a tedy není potřeba, aby překladač musel nahrazovat námi napsaný kód a tím se zdržovat a zpomalovat.

```
def anisotropie(float[:] img, int h, int w, float lambdaValue=0.1, float sigma
    =0.015):
    cdef float north = 0.0, south = 0.0, west = 0.0, east = 0.0, actual = 0.0
    cdef float northGradient = 0.0, southGradient = 0.0, westGradient = 0.0,
        eastGradient = 0.0
    cdef float cN = 0.0, cS = 0.0, cW = 0.0, cE = 0.0
    cdef int nt = 0
    cdef int N = h - 2, M = w - 2
    cdef int column = 0, row = 0, r = 0, c = 0

    # create array
```

```
cdef array.array newPic = array.clone(float_array_template, (h * w), zero=
    False)
# reference to array
cdef float[:] newPicture = newPic

for r in range(N):
    row = r + 1
    for c in range(M):
        column = c + 1
```

Výpis 8: Cython - metoda filtrace.

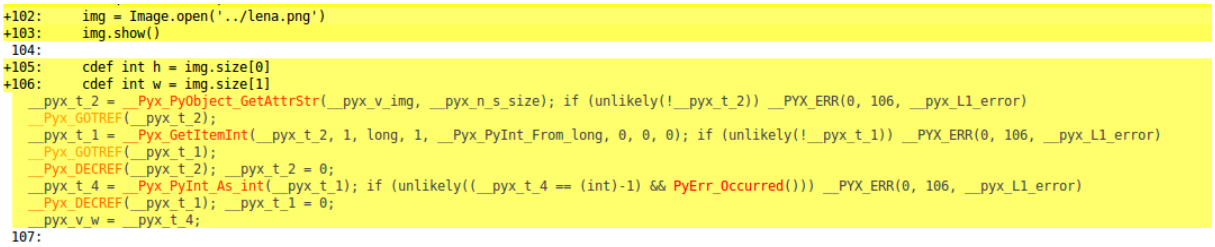
V druhém příkladu si ukážeme opačnou situaci. V této části kódu ve výpisu 9, se na startu programu načte obrázek a je zobrazen. Poté se ukládají jeho získané rozměry výška a šířka.

```
img = Image.open('../lena.png')
img.show()
cdef int h = img.size[0]
cdef int w = img.size[1]
```

Výpis 9: Cython - načtení obrázku a získání jeho rozměrů.

```
+025: def anisotropie(float[:] img, int h, int w, float lambdaValue=0.1, float sigma=0.015):
 026:     """
 027:     Anisotropic filtering (AF), a method of enhancing the image quality of textures
 028:     :param img: greyScale image
 029:     :param h: height of picture
 030:     :param w: width of picture
 031:     :param lambdaValue:
 032:     :param sigma:
 033:     :return: filtered picture
 034:     """
 035:
+036:     cdef float north = 0.0, south = 0.0, west = 0.0, east = 0.0, actual = 0.0
+037:     cdef float northGradient = 0.0, southGradient = 0.0, westGradient = 0.0, eastGradient = 0.0
+038:     cdef float cN = 0.0, cS = 0.0, cW = 0.0, cE = 0.0
+039:     cdef int nt = 0
+040:     cdef int N = h - 2, M = w - 2
+041:     cdef int column = 0, row = 0, r = 0, c = 0
 042:
 043:     # create array
+044:     cdef array.array newPic = array.clone(float_array_template, (h * w), zero=False)
 045:     # reference to array
+046:     cdef float[:] newPicture = newPic
 047:
+048:     for r in range(N):
+049:         row = r + 1
+050:         for c in range(M):
+051:             column = c + 1
 052:
+053:             north = (img[(row - 1) * w + column])
+054:             south = (img[(row + 1) * w + column])
+055:             west = (img[row * w + (column - 1)])
+056:             east = (img[row * w + (column + 1)])
 057:
+058:             actual = float(img[row * w + column])
 059:
+060:             northGradient = north - actual
+061:             southGradient = south - actual
+062:             westGradient = west - actual
+063:             eastGradient = east - actual
```

Obrázek 6: Cython - HTML výstup z kódu metody filtrace.



Obrázek 7: Cython - HTML výstup z kódu načtení obrázku a získání jeho rozměrů.

HTML výstup z této části kódu je na obrázku 7. Vidíme zde, čím se kód musí nahradit při převedení do Céčka. Jelikož se tato funkce volá jen jednou na začátku programu, nemá žádný vliv na jeho celkový běh.

5.2 Mixtura Gaussiánů

Tento program pracuje pouze se vstupním černobílým videem. Implementace algoritmu na detekci popředí je složitější hlavně v tom, že se musí pracovat s video sekvencí, tzn. postupné načítání jednotlivých nových snímků.

Testovací video je dostupné ke stažení na webové adrese `http://mrl.cs.vsb.cz/people/gaura/ano/dt_passat.mpg` s rozlišením 768 × 576 pixelů.

Všechny programy s Pythonem jsou složeny ze tří souborů:

Classes.py - třída pro ukládání obrazu a pomocných proměnných (weight - váha, σ - sigma, μ - mu)

utils.py - zde jsou napsány veškeré funkce sloužící k výpočtům

gaussian.py - hlavní část kódu s načtením videa a vykreslováním

5.2.1 Čistý Python a překladač PyPy

Překladač PyPy neumí zobrazit vstupní a výstupní video, takže výstup z něj je v pravidelných intervalech ukládán jako obrázek. Jinak nebylo třeba žádných úprav v kódu. V čistém Pythonu (kde není použit modul OpenCV) se používá k práci s videem volání externího softwaru ffprobe [8] a FFmpeg [9]. Je nutné je doinstalovat.

Nejdříve potřebujeme vědět rozměr videa. K tomu nám poslouží ffprobe, což je jednoduchý multimediální stream analyzér. Díky němu můžeme zjistit informace o videu jako je doba trvání, rozměr snímku a jeho frekvence, atd. Zvolíme si formát, ve kterém budou data, zde je to JSON. Z toho si už jen stačí vytáhnout výšku a šířku snímku. Potom už zbývá jen otevřít soubor a výstup přesměrovat do Pythonu. To nám zařídí FFmpeg. Ukázka je ve výpisu kódu 10.

```
probe_command = ('ffprobe', '-v', 'quiet', '-print_format', 'json', '-
    show_streams', FILE_PATH)
process = subprocess.Popen(probe_command, stdout=subprocess.PIPE, stderr=None)
```

```
video_info = json.load(process.stdout)
process.stdout.close()
process.wait()
try:
    width, height = video_info['streams'][0]['width'], video_info['streams'
        ][0]['height']
except:
    raise RuntimeError("Error acquiring video dimensions from file '%s'." %
        FILE_PATH)
frame_size = width * height
load_command = ('ffmpeg', '-v', 'quiet', '-i', FILE_PATH, '-f', 'image2pipe', '
    -pix_fmt', 'rgb8','-vcodec', 'rawvideo', '-')
process = subprocess.Popen(load_command, stdout=subprocess.PIPE, bufsize=10**8)
```

Výpis 10: Python - Ukázka práce s videem

Parametry ffprobe:

-print_format json - nastaví formát výstupního tisku na JSON

-v quiet - skryje počáteční informace, které ffprobe ukazuje

-show_streams - poskytne informace o každé stopě (video, audio, atd.), jako například použitý kodek, doba trvání, přenosová rychlost, atd.

Parametry FFmpeg:

-i FILE_PATH - předem definovaný vstupní soubor

image2pipe a "- " na konci říkají, že to bude přes pajpu využívané jiným programem

-pix_fmt rgb8 - nastaví formát pixelu, 8 bit monochromatický formát

Modul subprocess je vhodný pro práci s externími programy. Takže subprocess.Popen nám zavolá příkaz s definovanými parametry. Parametr `stdout=subprocess.PIPE` nám přesměruje výstup do standardního proudu a přitom čeká na ukončení procesu.

5.2.2 Python s využitím NumPy

Vzhledem k předpokladu, že obraz je matice a knihovna NumPy by nám měla značne urychlit běh programu, je tento program naopak nejpomalejší. Ke zpomalení dochází přímo ve funkci `update_model`, která slouží k aktualizaci modelu. Problém začíná již u potřeby zpracovávat obrázek pixel po pixelu, k čemuž nám NumPy nepomůže. Má vlastní formát pro čísla, a tím je v

tomto případě `numpy float 64`. S těmito čísly jsou pomalé aritmetické operace. Velkou roli tam také hraje jejich přesnost. Při přetypování na nativní Python `float` se běh programu zrychlil, ale kvalita vyhodnocení popředí velice kolísala.

Zde 11 je ukázka kódu z třídy pro ukládání hodnot pro proměnné weight, sigma a mu. Práce s NumPy je popsána v kapitole 3.1.

```
self.weight = numpy.empty((num_pixels, K), numpy.float) # weight
for i in xrange(0, num_pixels):
    self.weight[i].fill(1.0 / K)

self.sigma = numpy.empty((num_pixels, K), numpy.float) # sigma
for i in xrange(0, num_pixels):
    self.sigma[i].fill(sigma)

mu_init = numpy.zeros((K,), numpy.float)       # mu
for k in xrange(0, K):
    mu_init[k] = (k * (255 / K) + (255 / K) / 2)

tmp_mu = numpy.empty((num_pixels, K), numpy.float)
for pixel in xrange(0, num_pixels):
    for i in xrange(0, K):
        tmp_mu[pixel][i] = mu_init[i]
self.mu = tmp_mu
```

Výpis 11: Python - Numpy - ze souboru Classes.py - třída polí pro ukládání hodnot.

5.2.3 C++

Mixturu Gaussiánů jsem dostala už naimplementovanou v C++ od svého vedoucího. Mimo to zde uvedu ještě další dvě implementace. Obě dvě jsou obsaženy ve třídách, které jsou součástí knihovny OpenCV[15]. Tyto třídy nejsou úplně shodné s algoritmem popsaným v této práci v kapitole 2.2 - publikovaným Staufferem a Grimsonem, kde metoda také trpí na pomalé učení na začátku, obzvláště v rušném prostředí. Také nerozezná pohybující se objekt od jeho pohybujícího se stínu.

Na rozdíl od této originální varianty algoritmu, jsou tyto dvě vylepšené. První je od P. KaewTraKulPong, R. Bowden popsané v práci „An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection“, a druhá je od Zorana Zivkovice, v jeho práci „Improved Adaptive Gaussian Mixture Model for Background Subtraction“.

- **BackgroundSubtractorMOG** [13] - K modelování každého pixelu pozadí používá metodu směsí Gaussových křivek (od 3 do 5). Jejich váha představuje podíl času, po který tyto barvy zůstávají na scéně. Úpravou rovnic v různých fázích aktualizace modelu pozadí dopomohli k rychlejšímu učení programu, zvýšili přesnost a usnadnili přizpůsobování měnícímu se prostředí.

- **BackgroundSubtractorMOG2** [14] - vyvinul účinný adaptivní algoritmus využívající Gaussovy směsi hustoty pravděpodobnosti. Rekurzivní rovnice jsou použity k neustálému aktualizování parametrů, ale také se současně automaticky vybere potřebný počet Gaussových křivek na každý pixel a tím se zcela přizpůsobí pozorované scéně. Snížila se doba zpracování a také se mírně zlepšila segmentace.

Vzhledem k tomu, že tyto algority jsou vylepšené, tak porovnání těchto implementací s těmi zbývajícími nemá příliš vypovídající hodnotu. Přesto bude pro zajímavost zahrnut do porovnání.

Knihovna `BackgroundSubtractorMOG2` se volá s výchozím nastavením parametrů. Zde je pár nejdůležitějších:

- `static const int defaultHistory2 = 500;` // Learning rate; alpha = 1/defaultHistory2
  - délka historie, výpočet hodnoty α

- `static const int defaultNMixtures2 = 5;` // maximal number of Gaussians in mixture
  - počet povrchů k

- `static const float defaultBackgroundRatio2 = 0.9f;` // threshold sum of weights for background test
  - práh

```
cap = cvCaptureFromFile( "../dt_passat.mpg" );
img = cvQueryFrame( cap );
Ptr<BackgroundSubtractorMOG2> pMOG = createBackgroundSubtractorMOG2();

while(1)
{
   IplImage *frejm = cvQueryFrame( cap );
   Mat imgframe = cv::cvarrToMat(frejm);
   Mat imgfgmask;

   pMOG->apply(imgframe, imgfgmask);
   IplImage appliedFrame = imgfgmask;
```

```
    cvShowImage( "frame", &appliedFrame );
     }
}
    cvReleaseCapture( &cap );
```

Výpis 12: BackgroundSubtractorMOG2 - ukázka z kódu

Po metodě `cvCaptureFromFile`, která slouží jako konstruktor videa, metoda `cvQueryFrame` vezme, dekóduje a vrátí následující snímek. `Ptr<BackgroundSubtractorMOG2>` je pointer na datový typ BackgroundSubtractor a vytvoříme rovnou jeho objekt. Poté stačí jen volat metodu objektu `apply` s parametry nového snímku a obrázku, kde je výsledná maska pohybujících se objektů.

Na obrázcích 8 můžeme vidět rozdílné výstupy hned po spuštění programu. První ukazuje vstupní obraz. Na obrázku *b* je výstup po zpracování originálním algoritmem. Na obrázku *c* je obraz zpracován s využitím knihovny `BackgroundSubtractorMOG` a na posledním obrázku *d* je použita knihovna `BackgroundSubtractorMOG2`. U *b* a *c* je vidět rozdílnost ve zpracování obrazu. První snímek se bere jako pozadí. Na dalších snímcích už se porovnává, kde došlo k pohybu a ten se teprve vykresluje. Proto nejsou vidět ostrůvky zeleně detekované jako pohyb.

(a) Obrazový vstup (b) Výstup

(c) Výstup BackgroundSubtractorMOG (d) Výstup BackgroundSubtractorMOG2

Obrázek 8: Ukázka výstupů z jednotlivých programů - iniciální hodnoty.

6 Naměřené výsledky

Vzhledem k tomu, že algoritmus na detekci popředí je výpočetně náročný, uvedu zde mimo verzí použitého SW také informace o HW na kterém se programy spouštěly. Jedná se o notebook Acer Aspire 5830TG s procesorem Intel Core i5 a frekvencí 2,3 GHz. DDR3 RAM o velikosti 4GB. Operační systém Linux, Ubuntu 15.04, 64-bit s jádrem 3.19.0-59-generic. Co se programů týče, verze GCC (GNU Compiler Collection) je 4.9.2, Python verze 2.7.9, u Cythonu to je verze 0.24 a PyPy 5.4.1.

Naměřené výsledky jsou vždy průměrem určitého počtu měření. Výsledné rychlosti jsou porovnány vzhledem k rychlosti běhu programu v C++ a u každého programu je uvedeno o kolik je daná implementace rychlejší případně pomalejší. Uvedené časy jsou v sekundách.

Zaměříme se nejprve na **Anisotropní filtraci**. Výsledná doba zpracování je průměrná hodnota z deseti měření. Každé jedno měření je doba, za kterou se algoritmus fitrace provede 50×, tedy doba běhu 50 cyklů.

Výsledek této filtrace můžeme vidět na obrázku 5 v kapitole 5.1. Naměřené výsledky jsou v tabulce 1. Pro lepší srovnání jsou výsledky vidět v grafu 9.

Implementace	Doba zpracování 50 cyklů	Zrychlení
C++	2.3205	1
Python	42.77722	0.054
NumPy	71.817192	0.03
PyPy	2.491101	0.93
Cython	0.789335	2.94

Tabulka 1: Naměřené časy rychlosti zpracování Anisotropní filtrace.

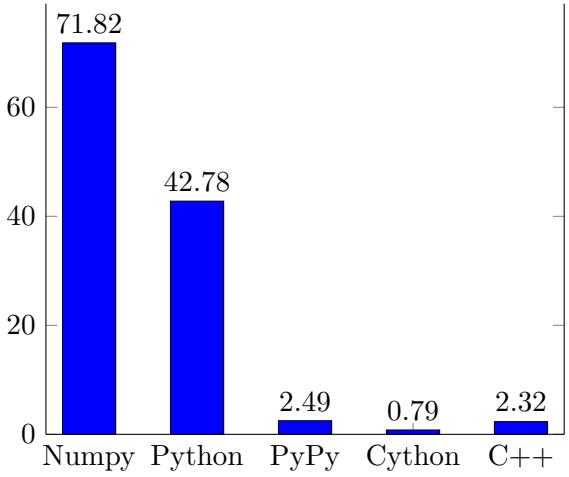
Z naměřených výsledků je vidět, že v tomto případě je Cython dokonce rychlejší než C++, po kterém následuje překladač PyPy, který má značný vliv na zrychlení programu v Pythonu a je jen o necelé dvě desetiny sekundy pomalejší než samotné C++. Python je o dost pomalejší než implementace v C++ což je zapříčiněno jeho dynamickým typováním proměnných. Nejhůře je na tom NumPy, pro který je nevyhovující metoda výpočtů nad jednotlivými pixely.

Obdobných výsledků dosáhneme i při měření algoritmu **mixtury Gaussiánů**. Výsledný čas uvedený v tabulce 2 u každého programu je doba, za kterou se provede výpočet jednoho snímku, tedy jednoho cyklu. Jde o průměr hodnot kdy každý program byl spuštěn 5× a čas je zaznamenán z prvních 50 cyklů běhu programu.

Na grafu 10 je z naměřených hodnot dobře vidět časová rozdílnost jednotlivých programů. Nejrychlejší jsou použité třídy z knihovny OpenCV. Vylepšená verze algoritmu MOG2 od Zorana Zivkovice je až 32 × rychlejší oproti originálnímu algoritmu implementovanému v C++.

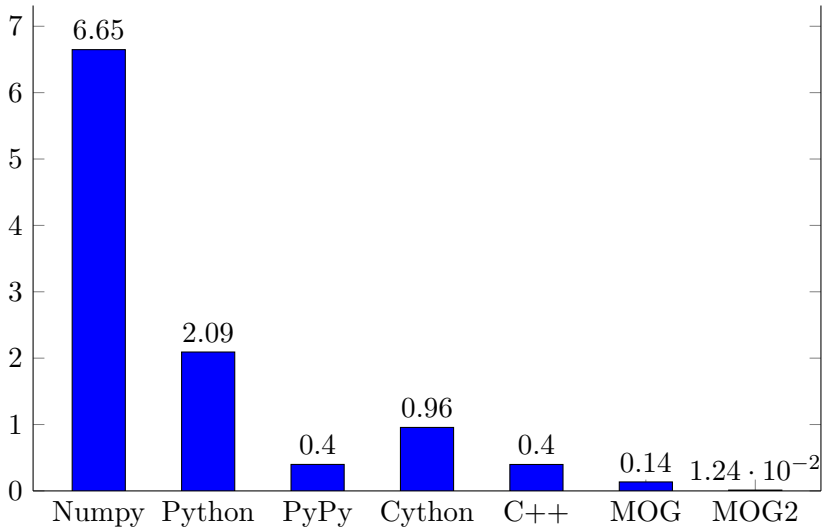
Implementace	Doba zpracování 1 cyklu	Zrychlení
C++	0.39824	1
Python	2.092444	0.19
NumPy	6.64782	0.06
PyPy	0.39901	0.998
Cython	0.956217	0.416
C++ MOG	0.13512	2.95
C++ MOG2	0.01243	32.04

Tabulka 2: Naměřené časy rychlosti zpracování mixtury Gaussiánů.



Obrázek 9: Graf rychlostí zpracování Anisotropní filtrace.

Nebudeme-li počítat tyto dva vylepšené algoritmy, nejrychlejšími programy jsou ty implementované v jazyce C++ a Pythonu s využitím překladače PyPy. Je nutno uvést, že celkový průměr rychlosti překladače PyPy kazí ta skutečnost, že JIT kompilace je v prvních dvou cyklech velice pomalá. Jak již bylo řečeno v kapitole 3.2, musí se nejdříve analyzovat často prováděné části programu. Časy z měření vypadají následovně: 1. cyklus 0.55 s, 2. cyklus 0.46 s, 3. cyklus 0.37. Po třetím cyklu už se čas nezlepšuje. Po dvou cyklech jsme tedy dosáhli zrychlení o desetinu vteřiny. Třetí nejrychlejší je překladač Cython. Samotný Python je skoro o 2 vteřiny pomalejší než C++.



Obrázek 10: Graf rychlostí zpracování mixtury Gaussiánů.

7 Závěr

Digitální zpracování obrazu je velice rozsáhlý obor využívající pokročilou výpočetní techniku a matematiku. V dnešní době je to velice moderní a rychle se rozvíjející oblast. Tato práce se zabývá pouze dvěma algoritmy a má za úkol ověřit a zhodnotit použitelnost jednotlivých implementací. Vzhledem k jejich výpočetní náročnosti je důležité upozornit, že použitý HW nebyl zcela ideální. Ale vzhledem k cílům této práce je to zanedbatelné. Jde nám totiž o porovnání rychlostí zpracování jednotlivých implementací. Na výsledek si počkáme déle, ale výsledné pořadí bude s největší pravděpodobností stejné. Naměřené výsledky z předchozí kapitoly nám dávají jasné výsledky.

Filtrace obrazu je využívána od her přes vylepšování fotografií až k lékařským přístrojům. Neustále se hledají nové metody s cílem dosáhnout méně náročného výpočtu či lepšího výsledku filtrace. Anisotropní filtrace popsaná v této práci, je nejčastěji používaná s cílem dosáhnout reálného vzhledu u objektů zobrazených pod kosými úhly. Při použití na fotografiích nebo obrázcích nám díky technice porovnávání sousedních pixelů odstraní šum a zpřesní okraje objektů na obrázku.

Z výsledků měření můžeme vidět, že se nejvíce osvědčil překladač Cython. Není to příliš obvyklé, aby byl rychlejší než samotné C++. Důvodem může být to, že cython přeložil Python do Céčka a poté ho zkompiloval efektivněji než samotné C++. Nebo samotný program v C++ nebyl dostatečně optimalizovaný. V každém případě nám Cython dal velice působivý výsledek. Stejně tak dobře je vidět nevýhodu dynamičnosti Pythonu a tedy že není vhodný k matematickým operacím.

Druhá použitá metoda mixtury Gaussiánů prochází také vývojem a je snaha algoritmus vylepšit. Dva příklady vylepšení tohoto algoritmu jsou uvedeny v kapitole implementací. Můžeme vidět, že první vylepšená metoda obsažena v knihovně OpenCV je třída názývající se MOG. Dosažené zlepšení je tři desetiny vteřiny. Výborného zlepšení času vylepšeného algoritmu dosahuje třída MOG2, jejíž běh je 32 × rychlejší než naše implementace původního algoritmu v C++.

Metoda detekce popředí najde mnohá využití. Od sledování provozu na silnicích až po hlídání soukromých pozemků. Detekovat pohybující se objekt je základním kamenem pro mnohé další algoritmy, jako je rozpoznávání a sledování objektů nebo detekce přítomnosti člověka.

Měření rychlosti běhu programů ukázalo, že implementace v C++ je stejně rychlá jako překladač PyPy. Jak již bylo zmíněno v předchozí kapitole, naměřené rychlosti PyPy jsou zkreslené pomalým během algoritmu v prvních dvou cyklech za účelem analyzování kódu. Vezmeme-li v úvahu, že pracujeme s videosekvencí, a tedy nepřetržitě pracujeme s novými daty, je na tom překladač PyPy lépe. Také naměřené časy jsou skoro konstantní, zatímco u C++ více kolísaly (někde skoro o desetinu vteřiny).

Oba dva algoritmy jsou v dnešní době důležité a hojně používané i když každý najde své uplatnění v trochu jiném odvětví. Jejich další vývoj je tedy nezbytný a určitě bude zajímavé sledovat, kde tento vývoj bude za dalších pár let.

Literatura

[1] P. Perona, J. Malik *Scale-space and edge detection using anisotropic diffusion* http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=56205

[2] Anisotropic Filtration. *Media Research Lab* [online]. [cit. 2016-10-09]. Dostupné z: http://mrl.cs.vsb.cz/people/gaura/dzo/anisotropic.pdf

[3] Mixtura Gaussiánů. *Media Research Lab* [online]. [cit. 2016-10-09]. Dostupné z: http://mrl.cs.vsb.cz/people/gaura/ano/mog.pdf

[4] Chris Stauffer, W.E.L Grimson: Adaptive background mixture models for real-time tracking. *CSAIL: MIT Computer Science and Artificial Intelligence Laboratory* [online]. [cit. 2016-10-16]. Dostupné z: http://www.ai.mit.edu/projects/vsam/Publications/stauffer_cvpr98_track.pdf

[5] *SciPy* [online]. [cit. 2016-08-20]. Dostupné z: https://docs.scipy.org/doc/numpy/reference/generated/numpy.empty.html

[6] PyPy: What is PyPy? *Pypy.org* [online]. [cit. 2017-01-15]. Dostupné z: http://pypy.org/features.html

[7] Seznamte se: pypy. *Abclinuxu* [online]. [cit. 2017-01-15]. Dostupné z: http://www.abclinuxu.cz/blog/paskma/2007/3/seznamte-se-pypy

[8] FFmpeg: ffprobe Documentation. *Ffmpeg.org* [online]. [cit. 2017-02-21]. Dostupné z: http://ffmpeg.org/ffprobe.html#SEC7

[9] Read and Write Video Frames in Python Using FFMPEG. *Zulko.github.io* [online]. [cit. 2017-02-20]. Dostupné z: http://zulko.github.io/blog/2013/09/27/read-and-write-video-frames-in-python-using-ffmpeg/

[10] PyPy's Python packages compatibility. *Packages.pypy.org* [online]. [cit. 2017-01-02]. Dostupné z: http://packages.pypy.org/#scipy

[11] Kurt W. Smith. *Cython: A Guide for Python Programmers*. United States of America: O'Reilly, 2015. ISBN 978-1-491-90155-7.

[12] *OpenCV* [online]. [cit. 2017-04-10]. Dostupné z: http://opencv.org/

[13] P. KaewTraKulPong, R. Bowden: An Improved Adaptive Background Mixture Model for Real-time Tracking with Shadow Detection [online]. [cit. 2017-02-27]. Dostupné z: http://link.springer.com/chapter/10.1007%2F978-1-4615-0913-4_11#page-2

[14] Zoran, Zivkovic. Improved Adaptive Gaussian Mixture Model for Background Subtraction. *zoranz.net* [online]. 2014 [cit. 2017-03-21]. Dostupné z: `http://www.zoranz.net/Publications/zivkovic2004ICPR.pdf`

[15] Background Subtraction Methods. *OpenCv: Open Source Computer Vision* [online]. [cit. 2016-11-27]. Dostupné z: `http://docs.opencv.org/3.1.0/d1/dc5/tutorial_background_subtraction.html`

[16] Steve Byrnes's Homepage: *Python for scientific computing: Where to start: Python program speed, Numba, and Cython* [online]. [cit. 2017-04-12]. Dostupné z: `http://sjbyrnes.com/python/`

A Přiložené soubory

CD obsahuje dva adresáře, pojmenované podle použitého algoritmu, jejich součástí je použitý testovací obrázek a video. Oba adresáře obsahují podsložky s jednotlivými implementacemi nazvané podle použitého programovacího jazyka či překladače.