VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO DEMONSTRACI METODY HISTOGRAM OF ORIENTED GRADIENTS PRO DETEKCI OBJEKTŮ

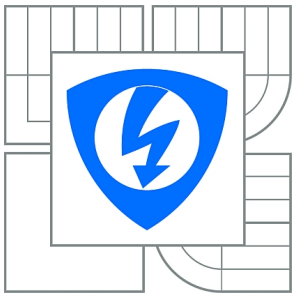
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ZDENĚK MRÁZEK
AUTHOR

BRNO 2014

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

APLIKACE PRO DEMONSTRACI METODY HISTOGRAM OF ORIENTED GRADIENTS PRO DETEKCI OBJEKTŮ

APPLICATION FOR A DEMONSTRATION OF THE HISTOGRAM OF ORIENTED GRADIENTS METHOD FOR OBJECT DETECTION

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

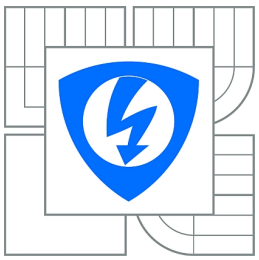
AUTOR PRÁCE — ZDENĚK MRÁZEK

AUTHOR

VEDOUCÍ PRÁCE — Ing. KAMIL ŘÍHA, Ph.D.

SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Zdeněk Mrázek ***ID:*** 146910
Ročník: 3 ***Akademický rok:*** 2013/2014

NÁZEV TÉMATU:

Aplikace pro demonstraci metody Histogram of Oriented Gradients pro detekci objektů

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte teoretický princip metody detekce objektů s pomocí příznakového deskriptoru založeného na metodě histogramu orientovaných gradientů. Naprogramujte aplikaci pro demonstraci vlastností tohoto deskriptoru. Aplikace by měla umožnit demonstraci detailů výpočtu orientovaných gradientů včetně vizualizace vektorů pro konkrétní snímky a detekci libovolných objektů. Doporučený nástroj pro implementaci: knihovny OpenCV a MS Visual C++.

DOPORUČENÁ LITERATURA:

[1] GONZALEZ R. C., WOODS R. E.: Digital Image Processing, Prentice Hall, New Jersey, 2002.
[2] BRADSKI G., KAEHLER A.: Learning OpenCV: Computer Vision with the OpenCV Library, O'Reilly Media, Inc. USA 2008, ISBN: 978-0-596-51613-0.
[3] LAGANI

Termín zadání: 10.2.2014 ***Termín odevzdání:*** 4.6.2014

Vedoucí práce: Ing. Kamil Říha, Ph.D.
Konzultanti bakalářské práce:

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

UPOZORNĚNÍ:

ABSTRAKT

Cílem této práce je shrnutí teorie, ze které vychází metoda Histogram of oriented gradients a následné zpracování vlastního algoritmu pro demonstraci výpočtu a vizualizace vektorů příznakového deskriptoru HOG, natrénování algoritmu SVM a následná detekce objektu. Jako prostředí pro práci bylo zvoleno MS Visual Studio 2012 s použitím objektově orientovaného jazyka C++ a s použitím knihovny OpenCV.

KLÍČOVÁ SLOVA

HOG, Histogram orientovaných gradientů, C++, OpenCV, gradient, histogram, deskriptor, SVM, Support vector machines, trénování, detekce

ABSTRACT

The target of this thesis is summarize the theory of method Histogram of oriented gradients and process algorithm for demonstration and visualization HOG descriptor, train SVM algorithm and subsequent detection of the object. For the work environment was selected MS Visual Studio 2012 using the object-orientedin C++ language with using OpenCV library.

KEYWORDS

HOG, Histogram of oriented gradients, C++, OpenCV, gradient, histogram, descriptor, SVM, Support vector machines, training, detection

MRÁZEK, Zdeněk *Aplikace pro demonstraci metody Histogram of Oriented Gradients pro detekci objektů*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2014. 41 s. Vedoucí práce byl Ing. Kamil Říha, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Aplikace pro demonstraci metody Histogram of Oriented Gradients pro detekci objektů“ jsem vypracoval samostatně pod 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 vytvořením 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 diplomové práce panu Ing. Kamilu Říhovi, Ph.D. za vstřícnost, konzultace, trpělivost a cenné rady k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

HOG	Histogram orientovaných gradientů – Histogram of oriented gradients
$\nabla f(x, y)$	Gradient obrazové funkce v bodě x, y
$M(x, y)$	Velikost vektoru gradientu v bodě x, y
$\alpha(x, y)$	Směr gradientu v bodě x, y
RGB	Barevný model – červená-zelená-modrá
SVM	Support Vector Machines
v	Normalizovaný vektor
$\|\|v\|\|_k$	k-norma normalizovaného vektoru pro k = 1,2

ÚVOD

Detekce objektů v obraze je velmi náročný úkol vzhledem k variabilitě předmětů a jejich specifickému tvaru. Algoritmů existuje mnoho na různých principech. Deskriptor Histogram of Oriented Gradients - Histogram Orientovaných Gradientů (HOG) je považován za vynikající metodu pro detekci objektů, především postav, protože dosahuje velmi dobrých výsledků.

V této práci bude popsáno, jak pracuje metoda HOG krok po kroku, od načtení vstupního obrazu po výpočet příznakového deskriptoru a jeho následnou vizualizaci. Dalším krokem bude natrénování databáze pomocí SVM a posouzení výsledné úspěšnosti detektoru. První kapitola poskytuje teoretický základ pro studium problematiky, na jejíž základě pracuje metoda Histogram Orientovaných Gradientů. Druhá kapitola poskytuje co nejdetailnější popis této metody. Poslední kapitola bude věnována popisu programu, jeho funkcím a výsledné úspěšnosti detektoru.

Demonstrace bude provedena formou samostatné aplikace, v prostředí MS Visual Studio 2012 pomocí objektově orientovaného jazyka C++ a s použitím knihovny OpenCV.

1 TEORETICKÝ ROZBOR

1.1 Histogram

Histogram obrazu s úrovněmi intenzit v rozsahu $[0, L-1]$ je diskrétní funkce

$$h(r)_k = n_\mathrm{k}, \tag{1.1}$$

kde r_k je k-tá úroveň intenzity a n_k je počet pixelů v obraze s intenzitou r_k. Běžně se používá normalizace histogramu pomocí vydělení jednotlivé složky celkovým počtem obrazových bodů v obraze MN, kde M je počet řádků a N počet sloupců v obraze. Normalizovaný histogram je tedy dán

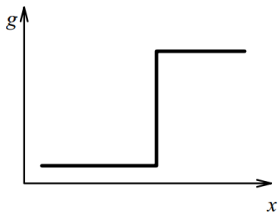
$$p(r_k) = \frac{n_k}{MN}, \tag{1.2}$$

pro $k = [0, 1, 2, \ldots, L-1]$. Volně řečeno, $p(r_k)$ je tedy odhad pravděpodobnosti výskytu intenzity úrovní r_k v obraze. Součet všech prvků normalizovaného histogramu je 1. Histogram se běžně používá v metodách zpracování obrazu a je také použit v metodě histogramy orientovaných gradientů [2].

1.2 Hrana

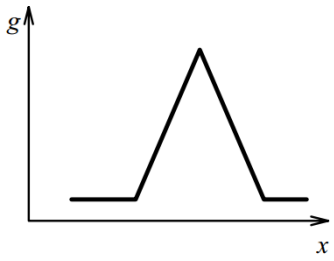
Hrana je oblast, kde dochází ke skokové změně hodnoty jasu. Tyto oblasti s přítomnými hranami obsahují více informace než ostatní místa v obraze. Hrana je dána vlastnostmi obrazového bodu a jeho okolí. Popisuje rychlost změny a směr největšího růstu obrazové funkce $f(x, y)$. Podle průběhu jasové změny je možné hrany rozdělit do jasových profilů.

- Skokový profil



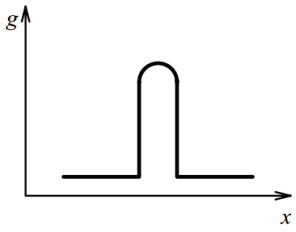
Obr. 1.1: Skokový profil

- Střechový profil



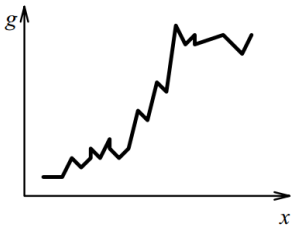
Obr. 1.2: Střechový profil

- Linkový profil



Obr. 1.3: Linkový profil

- Zašuměný profil



Obr. 1.4: Zašuměný profil

Skokový, středový a linkový profil představují ideální hrany. V reálném obraze se setkáváme většinou s profilem zašuměným. [4]

1.3 Výpočet gradientu pro detekci hrany

Gradient se v oblasti zpracování obrazu používá pro detekci hran objektů a směru v místě (x, y). Gradient f, v bodě (x, y) označený ∇f, je definován jako vektor

$$\nabla f = \text{grad}(f) = \begin{bmatrix} g_{\text{x}} \\ g_{\text{y}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}. \tag{1.3}$$

Tento vektor má důležitou vlastnost. Jeho směr udává největší změnu intenzity f v bodu (x, y). Velikost vektoru ∇f, označeného jako $M(x, y)$, kde

$$M(x, y) = |\nabla f| = \sqrt{{g_{\text{x}}}^2 + {g_{\text{y}}}^2} \tag{1.4}$$

je velikost změny ve směru gradientu vektoru. Směr gradientu je dán úhlem podle

$$\alpha(x, y) = \tan^{-1}\left[\frac{g_{\text{y}}}{g_{\text{x}}}\right]. \tag{1.5}$$

Prvky gradientního operátoru jsou lineární operátory, avšak velikost tohoto vektoru není lineární operátor, z důvodu operací umocňování a odmocňování. Výslednou velikost gradientního vektoru můžeme označit jako izotropní (mající ve všech směrech stejné vlastnosti, tedy nezávislé na směru). Avšak jednotlivé parciální derivace izotropní nejsou. Pro zjednodušení terminologie bude velikost gradientního vektoru označována jedním slovem jako gradient. Při výpočtu gradientního vektoru musíme vypočítat velikost vektoru pro každý pixel, proto si tuto operaci můžeme zjednodušit na

$$M(x, y) \approx |g_{\text{x}}| + |g_{\text{y}}|. \tag{1.6}$$

Tento vztah je mnohem jednodušší na implementaci, ale na úkor ztráty izotropnosti. Získání gradientu obrazu tedy vyžaduje výpočet parciálních derivací $\partial f/\partial x$ a $\partial f/\partial y$. Nyní je zapotřebí aproximovat parciální derivace přes sousední pixely bodu, ze kterého chceme získat gradient. Víme, že rovnice

$$g_{\text{x}} = \frac{\partial f(x, y)}{\partial x} = f(x + 1, y) - f(x, y) \tag{1.7}$$

a

$$g_{\text{y}} = \frac{\partial f(x, y)}{\partial y} = f(x, y + 1) - f(x, y) \tag{1.8}$$

mohou být použity pro všechny relevantní hodnoty x a y filtrace $f(x, y)$ s jednorozměrnou (1-D) maskou . Naznačeny jsou na obr. 1.5. Pokud chceme brát v potaz při výpočtu gradientu i pixely které sousedí diagonálně, je potřeba použít 2-D masku.

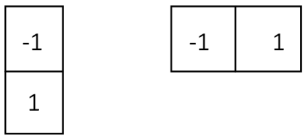
První pokus o 2-D masku byl prezentován L. G. Robertsem, jedná se o tzv. křížové diference. Robertsovy diference jsou založeny na diagonálních diferencích, podle

$$g_{\text{x}} = \frac{\partial f}{\partial x} = (a_9 - a_5) \tag{1.9}$$

a

$$g_y = \frac{\partial f}{\partial y} = (a_8 - a_6). \tag{1.10}$$

Přestože vychází z 3×3 bodové masky, používá pouze 2×2 bodovou masku a výpočet gradientu proto není symetrický podle středového pixelu (viz obr. 1.6 a 1.7). Kvůli své velikosti (sudý počet řádků a sloupců, nemají tedy středový pixel) se špatně realizuje.



Obr. 1.5: 1-D maska

a_1	a_2	a_3
a_4	a_5	a_6
a_7	a_8	a_9

Obr. 1.6: Obecná maska

-1	0
0	1

0	-1
1	0

Obr. 1.7: Robertsovy křížové operátory

Nejjednodušší řešení aproximace s maskou 3×3 je aproximace dle Prewittové (viz obr. 1.8), kde

$$g_{\mathrm{x}} = \frac{\partial f}{\partial x} = (a_7 + a_8 + a_9) - (a_1 + a_2 + a_3) \tag{1.11}$$

a

$$g_{\mathrm{y}} = \frac{\partial f}{\partial y} = (a_3 + a_6 + a_9) - (a_1 + a_4 + a_7). \tag{1.12}$$

Zde je rozdíl třetího a prvního řádku derivace ve směru x a rozdíl mezi třetím a

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Obr. 1.8: Prewittové gradientní operátory

prvním sloupcem je derivace ve směru y. Předpokládá se, že tato aproximace bude daleko přesnější, než při použití Robertsových operátorů. Mírně změněná variace Prewittových operátorů je maska s dvojnásobnou hodnotou středního koeficientu:

$$g_{\mathrm{x}} = \frac{\partial f}{\partial x} = (a_7 + 2a_8 + a_9) - (a_1 + 2a_2 + a_3) \tag{1.13}$$

a

$$g_{\mathrm{y}} = \frac{\partial f}{\partial y} = (a_3 + 2a_6 + a_9) - (a_1 + 2a_4 + a_7), \tag{1.14}$$

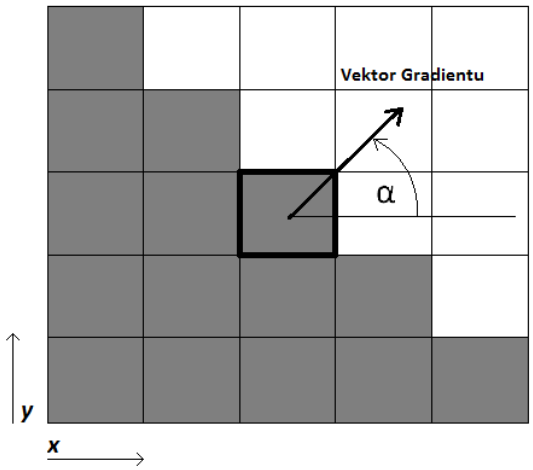
kterou můžeme vidět na obr. 1.9. Nazývá se Sobelův operátor. Prewittové maska je jednodušší pro implementaci než Sobelova maska. Protože Sobelova maska používá dvojnásobnou hodntou středního koeficientu, dochází u ní k většímu vyhlazení obrazu a proto je více preferována.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Obr. 1.9: Sobelovy gradientní operátory

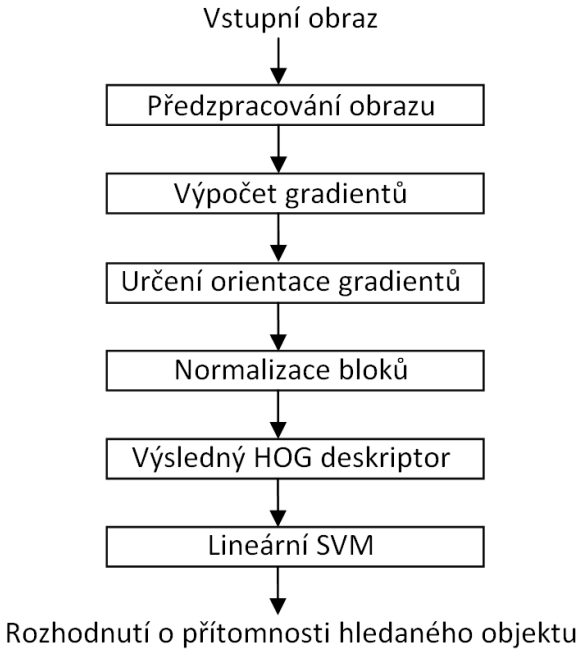
Na obr. 1.10 můžeme vidět přiblížený výřez obrazu, kde se nachází přímá ideální hrana. Každý čtverec zobrazuje jeden pixel. Pixely s šedou barvou mají hodnotu 0 a bílé pixely hodnotu 1. Nyní se musí vypočítat jednotlivé parciální derivace ve směru osy x a y. Toho docílíme použitím jedné z výše uvedených derivačních masek. Výsledný úhel α získáme výpočtem dle vzorce 1.5 a velikost vektoru dle 1.4. Úhel a velikost vektoru budou různé, právě podle derivační masky určené pro výpočet [2] [5].



Obr. 1.10: Výpočet gradientu pro určení směru hrany v bodě.

2 METODA HISTOGRAM OF ORIENTED GRADIENTS

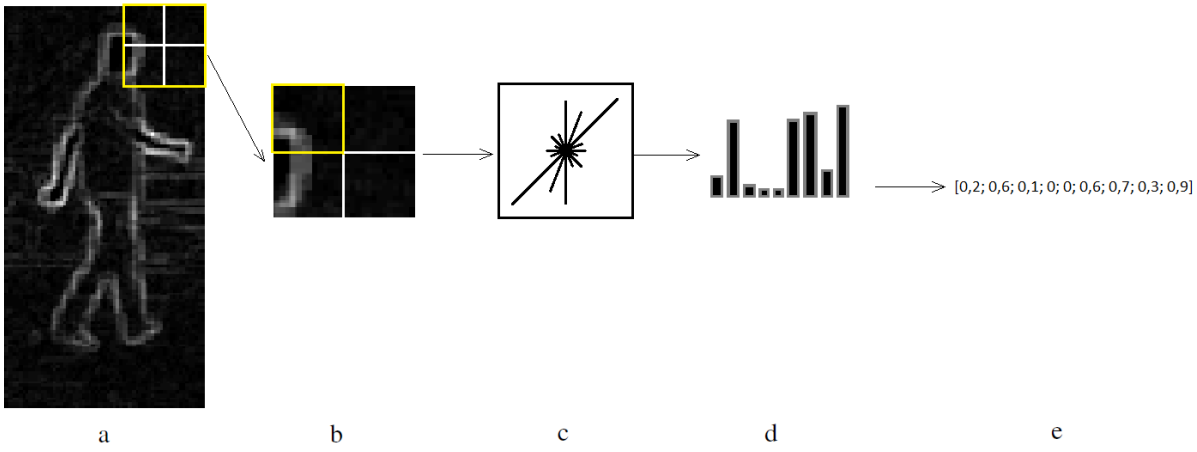
Metoda je založena na histogramech orientovaných gradientů. Základní myšlenkou je, že objekt v obraze může být pomocí vzhledu a tvaru charakterizován pomocí intenzity gradientů, i přestože neznáme jejich přesnou polohu v obraze. Obraz se rozdělí na malé prostorové oblasti (buňky) a pro každou buňku se vypočítá 1-D histogram, který je vypočítán ze všech pixelů z buňky. Je vhodné obraz před započetím výpočtů normalizovat, například kontrastní normalizací, nebo normalizací osvětlení. Toho lze docílit shromaždováním informací do histogramu nejen z jedné konkrétní buňky, ale z větší oblasti z okolních buněk. Těchto několik buněk dá dohromady tzv. blok. Histogram orientovaných gradientů, je tedy buňka normalizovaná blokem, který tvoří i okolní buňky. Algoritmus je znázorněn na obr. 2.1, který byl inspirován v [2].



Obr. 2.1: Algoritmus metody HOG.

Celý obraz je tedy rozdělen do bloků o stejné velikosti a každý z těchto bloků je reprezentován pomocí několika buněk. Tyto buňky jsou hlavním nositelem informace. Každá jednotlivá buňka zachycuje určitou oblast snímku, například 8×8 pixelů. V rámci každé buňky se určí orientace gradientu. Výsledné gradienty by měly vymezovat hledaný objekt v obraze. Pro detekci gradientů slouží některá z funkcí, která dokáže detekovat významné hrany v obraze, protože na nich dochází ke skokovým změnám jasu v obraze. Detekci hran je potřeba provést jak v horizontálním,

tak i ve vertikálním směru. Nad takto zpracovaným obrazem je rozvinuta síť buněk. V každé jednotlivé buňce se určí směr gradientu. Orientace významných hran má rozsah 0°-360°. Kdyby velikost jednotlivých buněk byla příliš velká, detekovala by se příliš velká množina směrů jednotlivých hran. Proto bychom tento rozsah mohli rozdělit do několika kanálů (zásobníků). Například při počtu 9 kanálů má každý kanál rozsah 40°. Z těchto kanálů se vytvoří lokální histogram, který obsahuje informace o zastoupení jednotlivých směrů gradientů. Jednotlivé lokální histogramy mají velikost odpovídající počtu kanálů. Výsledný vektor se tedy vytvoří složením informací ze všech histogramů jednotlivých buněk, které jsou normalizovány dle buněk v jejich bezprostřední blízkosti. Názorná ukázka kroků metody v obrázku 2.2. [1] Metoda je dále popsána detailně.



Obr. 2.2: Ukázka metody Histogram orientovaných gradientů: vypočítané gradienty v obraze (a), jeden blok, obsahující 4 buňky (b), orientace gradientů v buňce(c), určení histogramu orentovaných gradientů v buňce (d), výsledný příznakový vektor (e).

2.1 Předzpracování obrazu

Před výpočtem je vhodné použít proces na vyrovnání vlastností obrazu. Autoři v [1] zkoušeli různé úpravy vstupních obrazů. Například redukci barevného modelu RGB, nebo použití filtru pro úpravu nerovnoměrného osvětlení a jasu. Redukce barevného modelu RGB je provedena převedením obrazu do stupňů šedi. Jak autoři zjistili, tyto

úpravy však mají pouze velmi malý vliv na výslednou úspěšnost detekce, protože výsledky, které získali použitím takto upravených deskriptorů, se skoro nelišily.

2.2 Výpočet gradientů

Výpočet gradientů je stěžejní proces v metodě histogram orientovaných gradientů, protože hledaný objekt je charakterizován právě pomocí gradientních vektorů. Pro určení hledaného vektoru je nejprve potřeba najít významné hrany v obraze. Detekce hran spočívá ve vyhledání lokálních změn v intenzitě sousedních pixelů v obraze. Toho docílíme výpočtem gradientu obrazu. Výkon detektoru je citlivý na způsob, jakým jsou gradienty vypočítány. Testované byly masky centrované, necentrovaná, kubické, Sobelova maska, Prewittové maska atd. Jako nejlepší řešení, se nakonec ukázalo být to nejjednodušší. Autoři v článku [1] zjistili, že nejlepší je použití jednoduché derivační masky $[-1, 0, 1]$. Výpočet je popsán v kapitole 1.3. Pro výpočet gradientu se použije maska v horizontálním i vertikálním směru zvlášť. Protože barevná redukce z barevného obrazu na stupně šedi se ukázala pro výkon nepodstatná, pro barevné obrazy se vypočítají gradienty jednotlivě pro každou složku RGB. Z těchto dílčích gradientů se vybere ten, který dosahuje největší intenzity v bodě, pro který jsou vektory počítány, a pouze s tímto je posléze počítáno.

2.3 Rozdělení obrazu na buňky a orientace gradientů

Nyní musíme obraz rozdělit na hustou mříž buněk o určité velikosti. Autoři v [1] dosáhli nejlepších výsledku s použitím buněk o velikosti 8×8 pixelů. Buňky mohou být čtvercové nebo kruhové. Nyní tedy máme rozdělený obraz na buňky 8×8 pixelů a máme spočítaný gradient pro všechny pixely, které obraz obsahuje. Dále je potřeba určit zvlášť pro každou buňku histogram jejích gradientů. Každý jednotlivý pixel nese informaci o směru a velikosti pro výsledný histogram orientovaných gradientů. Kanály histogramu uvnitř každé buňky jsou určeny směrem gradientu. Počet kanálů (zásobníků) na které je buňka rozdělena je libovolný. V [1] je buňka rozdělena na 9 kanálů v rozsahu 0°- 180°, nebo 0°- 360°. Rozsah 0°- 180° se jeví jako optimální možnost. Velikost kanálů histogramu je určena velikostí příslušných gradientů. Nyní se musí prozkoumat všechny gradienty v buňce a po zjištění orientace gradientu se zařadí do jednotlivých kanálů. Součet stejně orientovaných gradientů náležících do jednotlivých kanálů nám dohromady dávají výslednou velikost kanálu. Takto nám vznikne pro každou buňku histogram orientovaných gradientů.

2.4 Normalizace bloků

Velikost gradientů se významně liší v rámci celého obrazu. Rozdíly jsou způsobeny nerovnoměrným osvětlením a různým kontrastem popředí a pozadí. Pro lepší výsledky je tedy potřeba tyto nežádoucí vlivy nějakým způsobem odstranit. V článku [1] autoři zkoušeli různá normalizační schémata. Většina z nich je založena na seskupení jednotlivých buněk do větších bloků a na normalizování každého bloku odděleně. Výsledný deskriptor je vektor vytvořený ze všech částí normalizovaných buněk. Jednotlivé bloky v obraze se překrývají, a proto mohou být buňky zpracovány i vícekrát. To se může zdát nadbytečné, ale kvalitní normalizace je stěžejní bodem pro dobré výsledky a překrývání jednotlivých bloků je proto žádoucí. Normalizace může být provedena čtvercová, nebo kruhová. Zabývat se budeme normalizací čtvercovou. Bloky se skládají z $n \times n$ buněk a každá buňka se skládá z $m \times m$ pixelů a každá buňka obsahuje p kanálů. Testy ukázaly, že nejvýhodnější je používat bloky velikosti 2×2, nebo 3×3 buněk. V těchto blocích se bude provádět normalizace histogramů jednotlivých buněk. Pro normalizaci je možné použít více metod, zde si uvedeme jednu. Řekněme, že v je nenormalizovaný vektor, obsahující všechny histogramy v určitém bloku a ϵ je malá konstanta. Pak $||v||_k$ je jeho k-norma pro $k = 1, 2$. Autoři testovali různá normalizační schémata. Jsou to $L2 - norm$:

$$v \longrightarrow \frac{v}{\sqrt{||v||_2^2 + \epsilon^2}}, \tag{2.1}$$

$L2 - Hys$, které je stejné jako L2-norm, pouze limituje velikost v na $0, 2$. Dále $L1 - norm$:

$$v \longrightarrow \frac{v}{||v||_1 + \epsilon}, \tag{2.2}$$

a $L1 - sqrt$:

$$v \longrightarrow \frac{v}{\sqrt{||v||_1 + \epsilon}}. \tag{2.3}$$

Schémata $L2-norm$, $L2-Hys$ a $L1-sqrt$ dávají podobně dobré výsledky, schéma $L1 - norm$ dává výsledky mírně horší.

2.5 Detekční okno

Detekční okno o velikosti 64×128 obsahuje 16 pixelů na každé straně, které představují okraj hledaného objektu (např. postavy). Snížení pixelů na každé straně na 8 pixelů (detekční okno o velikosti 48×112) zhoršilo výsledky. Zachování velikosti okna 64×128 a zvětšování postavy při současném zmenšování okraje, také zhoršilo výsledky. 16 pixelů tedy vychází jako optimální velikost. Příklad okraje s velikostí 16 pixelů můžeme vidět na obr. 2.3.

Obr. 2.3: Ukázka detekčního okna s postavou se znázorněnými okraji o velikosti 16 pixelů.

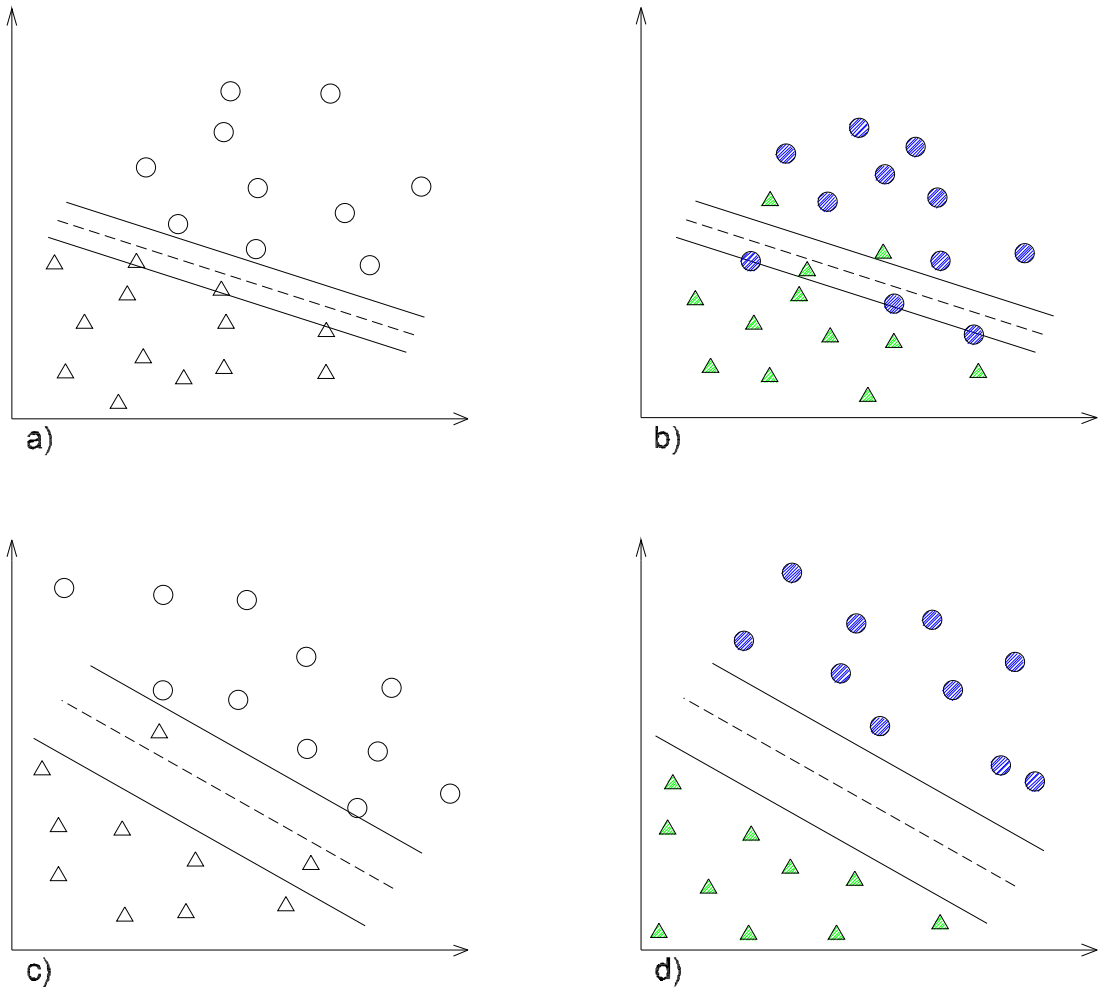
2.6 Deskriptor

Vektor normalizovaných histogramů pro jeden blok nazýváme deskriptor. Z každého bloku tak získáme jeden deskriptor. Posledním krokem podle [1] je předání deskriptorů nějakému klasifikátoru. Podrobný rozbor principu klasifikačních algoritmů je uveden v následující části.

2.7 Klasifikace

Klasifikátor je algoritmus, který při vhodně zvolené množině znalostí, rozděluje vstupní data pomocí příznaků (v metodě HOG pomocí příznakových vektorů), do předem zvolených tříd. V metodě HOG je pro klasifikaci použito nástroje SVM, konkrétněji lineární SVM – lineární Support Vektor Machine algoritmus. Jedná se o metodu strojového učení, což je nauka o získávání a zpracování znalostí. Lineární SVM klasifikátor je jednoduchá klasifikační metoda založena na rozdělení prostoru příznaků na lineární úseky.

Vstupní data musíme rozdělit na trénovací a testovací. Trénovací data jsou použita pro „naučení se“ klasifikátoru, testovací data jsou požita k následnému testování vytvořeného klasifikátoru. Mějme skupinu dat, která popisuje postavu v obraze. Na druhé straně máme data, kde se postava v obraze nevyskytuje. Nástroj SVM by měl tato data vyhodnotit a určit hranici mezi těmito daty. Pokud následně předložíme klsifikátoru jakýkoliv obraz, měl by být schopen určit, zda se v obraze vyskytuje postava či nikoliv. [6] [7]



Obr. 2.4: Ukázka požití nástroje SVM. Na obrázku a) a b) vidíme hranici mezi různými daty, na obrázcích c) a d) vidíme nalezení lepší hranice mezi daty. Obrázek byl inspirován v návodu ke knihovně SVM [7]. Použití klasifikátoru na trénovací data (a), Použití klasifikátoru na testovací data (b), Použití lepšího klasifikátoru na trénovací data (c), Použití lepšího klasifikátoru na testovací data (d).

3 NÁVRH A IMPLEMENTACE

3.1 Použitý software

3.1.1 MS Visual Studio 2012

Pro aplikaci demonstrace metody histogram orientovaných gradientů jsme zvolili jazyk C/C++ v prostředí Microsoft Visual Studio 2012, protože pro jazyk C/C++ je zde již vestavěný použitím Visual C++.

3.1.2 OpenCV 2.4.8

OpenCV[1] je volně šiřitelná, multiplatformní knihovna pro zpracování obrazu a videa. Je určena pro akademické i komerční využití. Zaměřena je především na počítačové vidění a zpracování obrazu v reálném čase. Původně ji vyvíjela společnost Intel. Má C++, C, Python a Java rozhraní. Knihovna je psaná v jazyku C/C++ schopná pracovat pod operačními systémy Windows, Linux a Mac OS. Je distribuována pod BSD licencí. Byla použita verze 2.4.8 OpenCV. Obsahuje následující knihovny:

- Core: modul definující základní datové struktury, pole Mat a základní funkce požadované všemi ostatními moduly.
- Imgproc: modul zpracování obrazu obsahující lineární a nelineární filtrace obrazu, geometrické transformace (změna velikosti atd.), barevné konverze, histogramy atd.
- Video: analýza videa, obsahuje odhad pohybu, odečítání pozadí a algoritmy sledování objektu.
- Calib3d: základní algoritmy více-pohledové geometrie, single a stereo kalibraci kamery, odhadování pozice a prvky 3D rekonstrukce.
- Features2D: detektory charakteristických rysů, deskriptory a vyhodnocování deskriptorů.
- Objdetect: detekce objektů a instance předdefinovaných tříd (např. tváře, oči, hrnky, lidé, auta atd.).
- Highgui: snadno použitelné rozhraní pro záznam videa, obrazu a video kodeků, stejně jako jednoduché uživatelské rozhraní.
- GPU: urychlovací algoritmy z různých OpenCV modulů.

[1] Dostupná z http://opencv.org/

3.1.3 Qt 5.2.1

Qt[1] je multiplatformní aplikace a prostředí pro práci v jazyce C/C++. Pro program jsme použili tuto knihovnu pro vylepšení zobrazovacích funkcí OpenCV. Použitá verze 5.2.1 byla do použité knihovny OpenCV implementována pomocí softwaru Cmake.

3.1.4 CMake 2.1.12

Překlad knihovny Qt pro použití s OpenCV byl proveden pomocí CMake[2] verze 2.1.12. CMake je multiplatformní volně šiřitelný software pro řízení procesu kompilace. Poskytuje automatizaci překladu programu, pro vytvoření adresářové struktury a přípravu zdrojových souborů, pro použití s konkrétními překladači.

3.2 Návrh programu

Algoritmus programu je navržen dle [1]. Parametry, se kterými dosahovali autoři nejlepších výsledků, však nejsou dodrženy, ale jsou zvoleny takové, aby byla demonstrace co nejzřetelnější. Jde nám hlavně o představení algoritmu a ukázky výpočtů jednotlivých kroků metody. Program, který jsem pojmenoval HOG-test, má dvě na sobě nezávislé (vyjma některých společných funkcí) části, které si nyní popíšeme. Zjednodušený vývojový diagram programu HOG-test lze vidět na obr. 3.1. V první části si popíšeme tu část programu, která pracuje se vstupním obrazem pojmenovaným `vstup.png`.

Následující kroky, od načtení vstupního obrazu `vstup.png` až po konečné vykreslení příznakového deskriptoru, si popíšeme každý zvlášť:

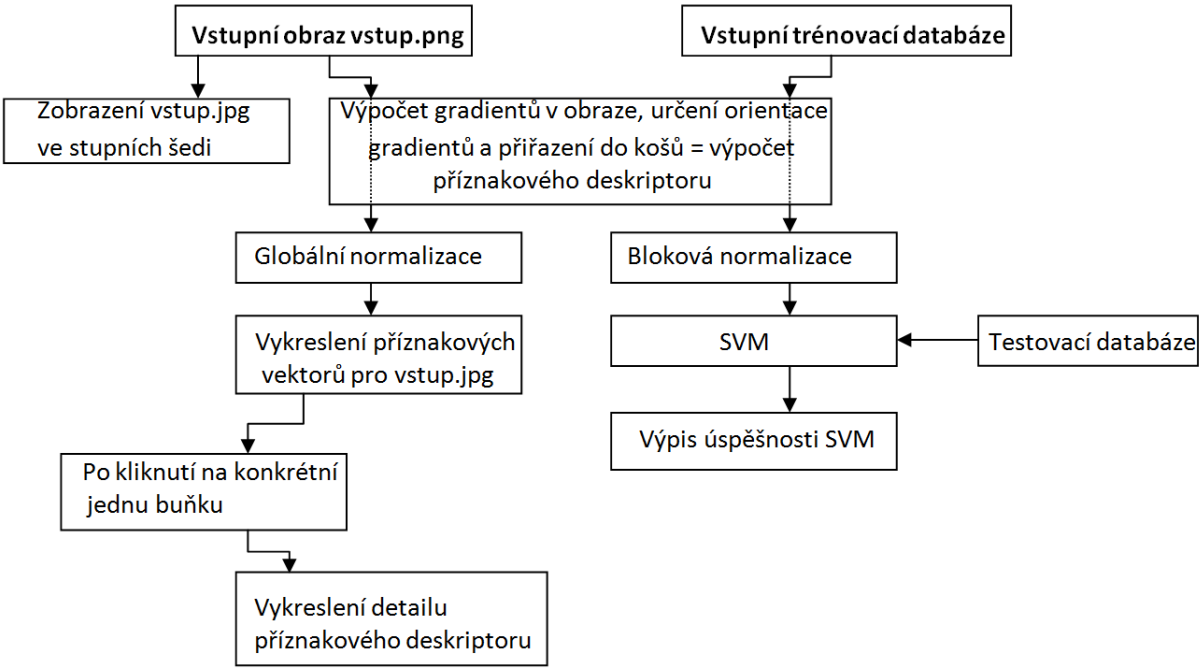
- Vstupní obraz
- Předzpracování obrazu
- Výpočet gradientů
- Určení orientace gradientů a přiřazení do kanálů
- Normalizace obrazu
- Výsledné vykreslení vektorů - deskriptoru
- Detailní vykreslení vektorů v jedné buňce

3.2.1 Vstupní obraz a předzpracování

Vstupní obraz může být barevný nebo černobílý. Protože barevnost/nebarevnost obrazu nemá na výsledný výpočet vliv, je vstupní obraz načtený a rovnou převeden

[1] Dostupná z http://qt-project.org/

[2] Dostupný z http://www.cmake.org/



Obr. 3.1: Zjednodušený vývojový diagram popisující jednotlivé kroky programu HOG-test

do stupňů šedi integrovanou funkcí OpenCV:

```
cv::imread("vstup.png", CV_LOAD_IMAGE_GRAYSCALE);
```

Žádné další úpravy obrazu (jako redukce modelu RGB nebo filtr pro úpravu nerovnoměrného osvětlení) neprobíhají z důvodu zjednodušení algoritmu.

Nyní si obraz rozdělíme na jednotlivé buňky – oblasti o určité velikosti. V teoretické části bylo uvedeno, že se požívají buňky o velikosti 8×8 pixelů. Tato velikost byla zvolena i pro můj algoritmus. Větší velikost buňky, např. 32×32 pixelů, byla vyzkoušena pro detailnější vykreslení deskriptorů, které si můžeme prohlédnout na obr. 3.8. Jedná se o vykrelsení jednoduchých geometrických tvarů.

3.2.2 Výpočet gradientů

Výpočet gradientů je stěžejní proces algoritmu. Výpočet gradientu se provede pro horizontální a vertikální směr zvlášť pro každý pixel v obraze, podle jednoduché derivační masky $[-1, 0, 1]$. Následně, se dle vzorce 1.4, uvedeného v kapitole 1.3, zjistí velikost vektoru gradientu a dle vzorce 1.5 zjistíme směr gradientu. Směr bude dán úhlem vzhledem k ose x. V programu jsem vyzkoušel i sobelovu derivační masku, která pro výpočet používá i diagonálně sousedící pixely. Různé derivační masky nám dávají přibližně stejné výsledky v metodě HOG, a proto je použita jednoduchá maska $[-1, 0, 1]$, která je výpočetně a matematicky nejjednodušší a proto i nejrychlejší.

Obr. 3.2: Převod barevného obrazu (a) na černobílý (b)

3.2.3 Určení orientace gradientů a přiřazení do košů

Po výpočtu gradientů ze vzorce 1.5 zjistíme úhel, který gradient svírá s osou x. Tento úhel zjistím pro každý pixel v buňce obrazu. Pro každou buňku nyní musíme získat histogram orientovaných gradientů, tzn. úhel pro každý pixel musíme zařadit do kanálu histogramu (koše). Orientace úhlů má rozsah 0-180° na obě strany. Tento rozsah rozdělíme na 9 kanálů po 20°. Tzn. první kanál má rozsah 0-20° atd. Takto zařadíme všechny gradienty pixelů do kanálů.

Pro každý kanál sečteme velikost gradientů pixelů, které do něj náleží, a tím získáme výslednou velikost vektoru pro daný kanál. Velikost je vypočítána dle vzorce 1.6. Uvažujeme pouze výpočet pro jednoduchý černobílý obraz, protože načtený obraz je vždy převeden do odpovídajících stupňů šedi. Pokud bychom pracovali s barevným obrazem, musela by se velikost gradientu vypočítat pro všechny složky barevného obrazu (RGB), poté určit který z nich má největší hodnotu, a ten použít pro výsledný výpočet. Postup aplikujeme na každý kanál histogramu. Ve výsledku dostaneme 9 vektorů (pro každý kanál) a jeho velikost, jak je vidět na obr. 3.5.

Úhel vektoru pro jednotlivé kanály bude vždy ve středu rozsahu tohoto kanálu, tedy:

- pro 0-20° bude vektor pod úhlem 10°
- pro 20-40° bude vektor pod úhlem 30°
- pro 40-60° bude vektor pod úhlem 50°
- ...
- pro 160-180° bude vektor pod úhlem 170°.

Obr. 3.3: Jednoduchá derivační maska [-1,0,1] - součet gradientů ve směru osy x a y.

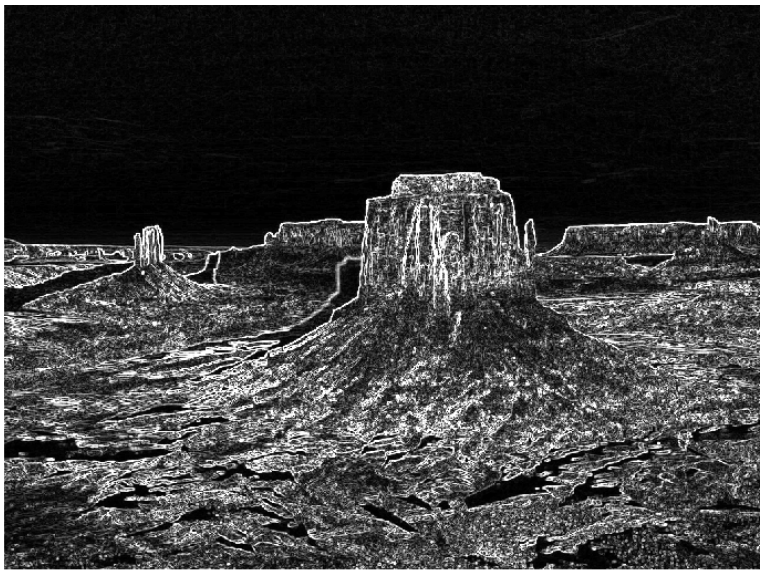
3.2.4 Normalizace

Nyní je zapotřebí normalizovat velikost vektorů pro jednotlivé buňky. Kdyby normalizace neproběhla, rozdíly velikostí vektorů by byly příliš velké. Tyto rozdíly jsou způsobeny nerovnoměrným osvětlením, různým kontrastem popředí a pozadí a především jinou hodnotou gradientů v částech obrazů kde se nachází/nenachází hrana. V metodě HOG nejlépe vychází rozdělení obrazu na tzv. bloky skládající se z $n \times n$ buněk, kdy vektory v každé buňce jsou normalizovány dle sousedních buněk. Na návrh vedoucího práce jsem pro demonstrační metodu vykreslení příznakových deskriptorů použil normalizaci globální pro celý obraz. Globální normalizace je provedena následovně:

- zjistí se velikost všech vektorů v kanálech histogramů v každé buňce obrazu
- vybere se největší z nich
- největší vektor se normalizuje tak, aby jeho velikost nepřesahovala maximální velikost své buňky
- všechny ostatní vektory jsou vyděleny zjištěným poměrem

Tímto postupem je zajištěno, že největší vektor v obraze bude mít maximální velikost v dané buňce, a ostatní vektory budou mít velikost menší, ve zjištěném poměru. Nicméně vektory s mnohonásobně menší velikosti oproti největšímu, tzn. vektoru, dle kterého se normalizuje, budou mít po normalizaci tak malou velikost, že po vykreslení nebudou pravděpodobně patrné.

Normalizované vektory v jednotlivých buňkách jsou výsledné příznakové deskriptory.



Obr. 3.4: Sobelova derivační maska - součet gradientů ve směru osy x a y.

3.2.5 Vykreslení vektorů - příznakových deskriptorů

Pro vykreslení výsledných vektorů jsem použil funkci pro vykreslení jednoduché čáry. Počáteční souřadnici přímky víme, v každé buňce jde o středový pixel. Protože známe velikost vektoru *vect* a úhel α pro daný vektor, souřadnici pro druhý bod přímky získáme pomocí

$$x = |vect| \times \cos\alpha \tag{3.1}$$

a

$$y = |vect| \times \sin\alpha. \tag{3.2}$$

Po vykreslení vektoru pro každý kanál se navíc vykreslí inverzní vektory na opačnou stranu. Vykreslení je provedeno pomocí dvou funkcí:
`cv::Point(xstred,ystred)`
a
`cv::line(test2,stred,bod0,CV_RGB(255,255,255),2,8,0 );`.

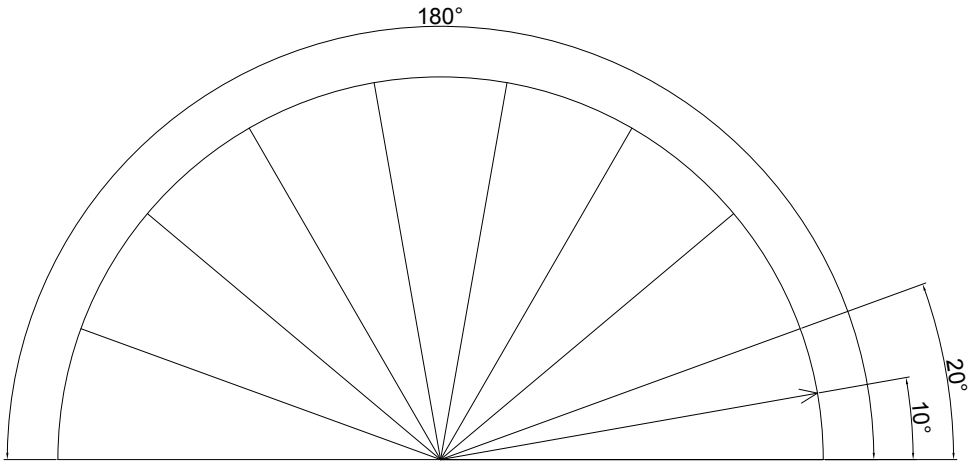
3.2.6 Výstup programu

Jako výstup programu jsou zobrazena dvě okna:

- `vstup`
- `deskriptor`

pomocí funkce z knihovny openCV:

```
cv::namedWindow("vstup",1);
cv::imshow("vstup", obraz);
```



Obr. 3.5: Rozdělení kanálů 0-180°, v rozsahu po 20°, se zobrazeným výsledným vektorem.

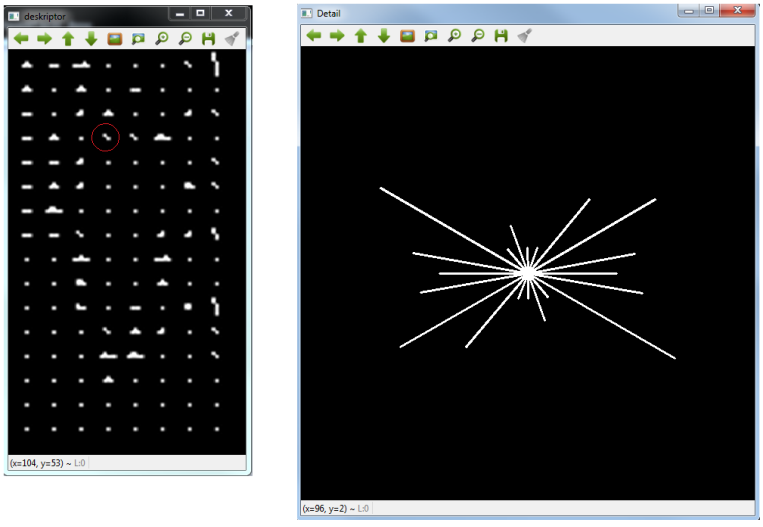
V hlavičce okna se nachází volitelný parametr "1", který znamená, že okno se otevře v automaticky zvolené velikosti dle velikosti vstupního obrazu, přičemž velikost nelze upravovat. V okně `vstup`, se zobrazí `vstup.png` ve stupních šedi. V okně `deskriptor` se zobrazí výsledný vykreslený deskriptor pro celý obraz. Protože je provedena globální normalizace, vektory nejsou v celém obraze s konstantní velikostí. Je to způsobeno velkým rozdílem hodnot gradientů. Po kliknutí levým tlačítkem myši na kterýkoli deskriptor v okně, se použitím funkce:

```
cv::setMouseCallback("deskriptor", CallBackFunc, NULL);
if ( event == cv::EVENT_LBUTTONDOWN );
```

otevře další okno s názvem `detail`, kde se zobrazí detailní zobrazení tohoto konkrétního deskriptoru.

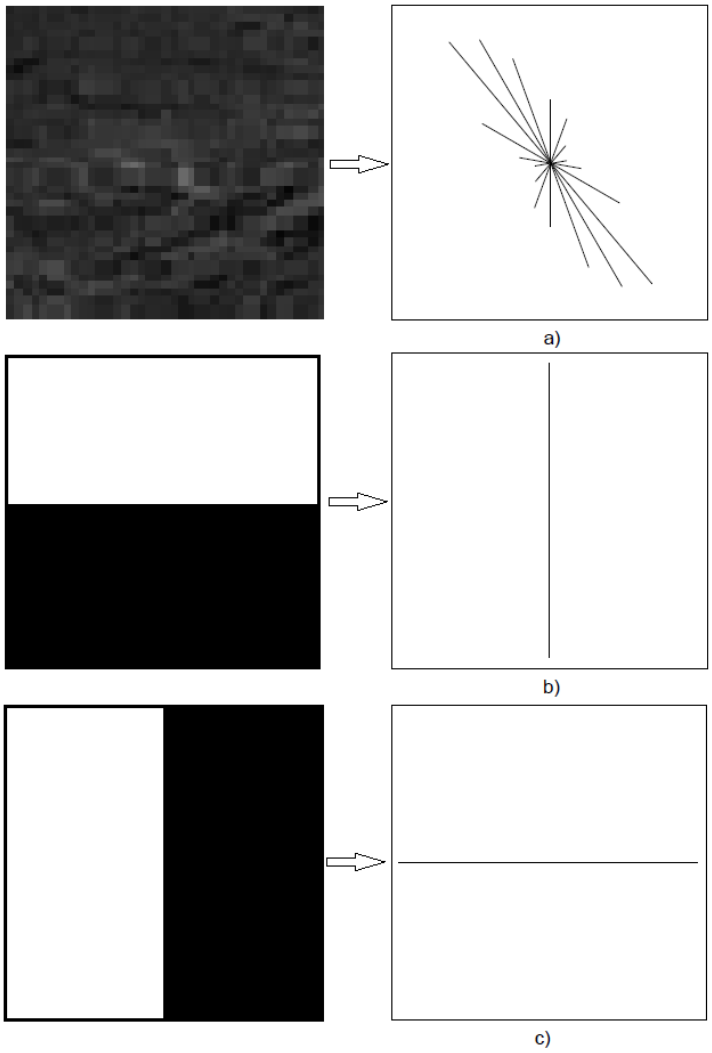
Výsledná okna s vykreslenými deskriptory samotným programem můžeme vidět na obr 3.6

Nyní se podíváme, jak by měly vypadat výsledné vykreslené vektory pro reálný obraz a pro syntetické hrany v jedné buňce. Na obr. 3.7 vidíme vykreslení vektorů v buňce pro reálný obraz a syntetické hrany. V reálném obraze se většinou vykreslí více vektorů, protože obraz obsahuje více než jeden úhel gradientů. U syntetických hran obsahuje pouze gradient ve směru kolmém na hranu, tedy 0° nebo 90°.

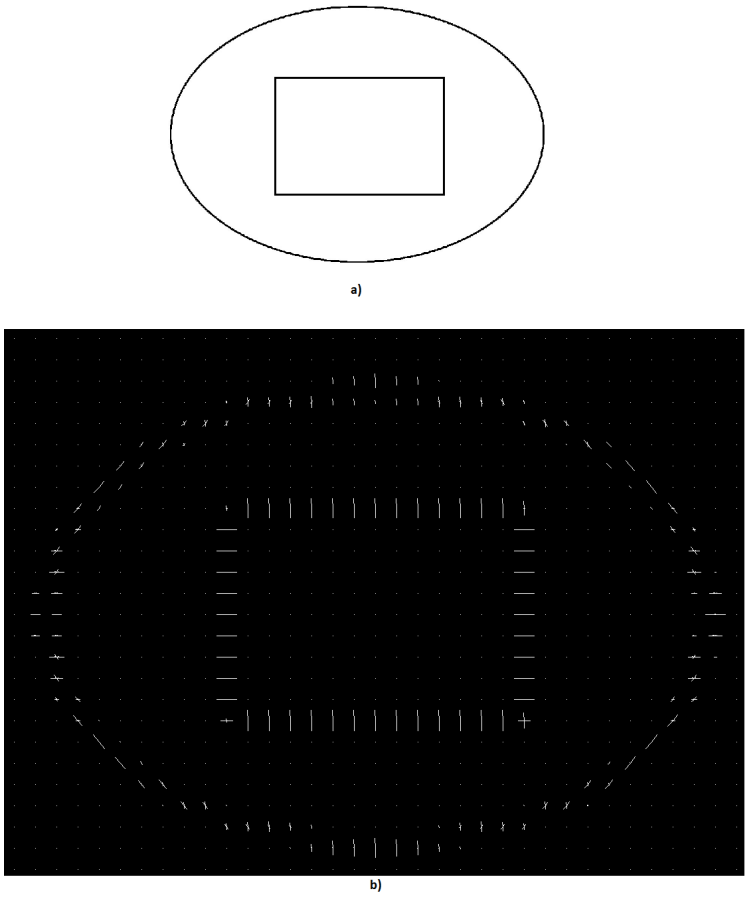


Obr. 3.6: Vykreslené deskriptory pro celý obraz (a), detail deskriptoru jedné buňky - označené červeně (b).

Na obr. 3.8 vidíme vykreslený deskriptor pro jednoduché geometrické tvary. Pro toto vykreslení byly použity buňky o velikosti 32×32 pixelů pro lepší viditelnost. Zde je velmi zřetelně vidět, jak se výsledné příznakové deskriptory vykreslují okolo syntetických hran, s kterými se ovšem v reálném obraze setkáme stěží.



Obr. 3.7: Vykreslené vektory v buňce pro reálný obraz (a), pro syntetickou hranu v ose x (b) a pro syntetickou hranu v ose y (c).



Obr. 3.8: Vstupní obraz (a), výsledné vektory – deskriptor(b).

3.3 Detekční část programu

Druhá část programu HOG-test, jak můžeme vidět na obr. 3.1, slouží pro natrénovaní SVM a následné detekci. S první částí programu, určenou k vykreslení příznakových deskriptorů, má společné některé funkce. Jednotlivé kroky si opět přehledně popíšeme:

- Načtení vstupních obrazů - trénovací databáze
- Předzpracování obrazu
- Výpočet gradientů
- Určení orientace gradientů a přiřazení do košů
- Bloková normalizace
- Natrénování SVM
- Testovací databáze
- Výsledky detekce

3.3.1 Načtení trénovací databáze

Prvním krokem v programu je načtení trénovací databáze. Pro tuto práci byla použita databáze INRIA (INRIA person dataset)[1], kterou vytvořili autoři metody HOG [1]. Databáze obsahuje dohromady několik tisíc pozitivních a negativních snímků postav pro trénování i testování. Několik pozitivních vzorků z této sady můžeme vidět na obr. 3.9. Negativní vzorky jsou různé fotografie krajin, měst, automobilů apod. Pro demonstraci jsem zvolil velikost trénovací databáze 100 vzorků,

Obr. 3.9: Příklad několika pozitivních vzorků z databáze INRIA.

50 pozitivních a 50 negativních. Tyto vzorky musí být umístěny v kořenovém adresáři programu ve složce `trenovani`. Vzorky musí být pojmenovány `trenX.png`, kde `X` je index vzorku. Indexování začíná od 0. Prvních 50 vzorků (s indexy 0-49) jsou pozitivní vzorky, tzn. je na nich postava. Dalších 50 (indexy 50-99) jsou vzorky bez postavy.

3.3.2 Předzpracování obrazu

Všechny načtené vzorky jsou opět přímo převedeny do odstínů šedi a jejich velikost je pomocí funkce OpenCV
`cv::resize(vystup,vystup, cv::Size(64,128), 0, 0, 1);`
změněna na 64×128 pixelů. Pozitivní vzorky v databázi INRIA mají velikost 64×128 pixelů, ale negativní vzorky mají různé velikosti a proto musí být změna velikosti provedena. Proč musí mít všechny vzorky pro náš program stejnou velikost si uvedeme v další části.

3.3.3 Výpočet gradientů

Výpočet gradientů pro všechny vstupní trénovací vzorky je proveden stejně, jak je uvedeno v předchozí části, tzn. pomocí jednoduché derivační masky $[-1, 0, 1]$.

[1] Dostupná z http://pascal.inrialpes.fr/data/human/

3.3.4 Určení orientace gradientů a přiřazení do košů

Nyní máme vypočítané gradienty pro vzorek z trénovací databáze. Ještě před tím, než se dostaneme k orientaci gradientů a přiřazování do košů, je třeba zmínit jednu souvislost. Protože SVM klasifikátor z knihovny OpenCV, který je použit, přijímá jako jeden vzorek pole číselných hodnot, je třeba výsledný příznakový deskriptor upravit na pole o velikosti $1 \times x$ čísel. Protože vzorky se kterými pracujeme, mají velikost 64×128 pixelů a buňky deskriptorů 8×8 pixelů, lze snadno vypočítat, že počet buněk v obraze je 128 (Pro osu x $64/8 = 8$, pro y $128/8 = 16$, celkem $8 \times 16 = 128$). Určení orientace gradientů a přiřazení do košů je provedeno stejně, jak je uvedeno v kapitole 3.2.3, tzn. vektory jsou rozděleny do 9 kanálů po 20° (obr. 3.5). Z každé buňky tedy získáme 9 číselných hodnot, z každého kanálu jednu. Jeden vzorek pro SVM klasifikaci tedy bude mít 128 buněk $\times$ 9 kanálů = 1152 hodnot. Pro každý vzorek je vytvořeno dynamicky nové pole o této velikosti:

```
float* vyslednyvektor = new float[1152];
```

Do tohoto pole se postupně načítají vypočítané hodnoty. Pole je poté předáno SVM klasifikátoru, což bude blíže popsáno v následující části.

3.3.5 Bloková normalizace

Jako normalizační schéma bylo použito normalizační schéma dle vzorce 2.2, uvedené v teoretickém úvodu. Každá buňka je normalizovaná pomocí okolních buněk = okolního bloku. Proto normalizace bloková. Protože se bloky překrývají, v buňkách proběhne normalizace více než jednou. To je pro dobré výsledky žádoucí. Velikost bloků je zvolena 2×2 buňky = každá buňka se normalizuje pomocí 3 okolních.

3.3.6 Natrénování SVM

Jak bylo uvedeno v teoretické části, lineární SVM je klasifikační algoritmus, který rozdělí data do dvou částí a určí mezi nimi lineární hranici. V programu jsem použil SVM z knihovny OpenCV. Její základní parametry, jako například lineární jádro se určí takto:

```
CvSVMParams params;
params.svm_type = CvSVM::C_SVC;
params.kernel_type = CvSVM::LINEAR;
params.term_crit = cvTermCriteria(CV_TERMCRIT_ITER, 100, 1e-6);
```

Do proměnné `trainingData` uložíme pole o velikosti 1152 (počet vektorů z jednoho vzorku) $\times$ 100 (počet vzorků pro trénování, 50 pozitivních a 50 negativních). V proměnné `labels` je SVM určeno, které vzorky jsou pozitivní a které negativní.

Samotné natrénování probíhá pomocí funkcí:

```
CvSVM SVM;
SVM.train(trainingData, labels, cv::Mat(), cv::Mat(), params);
```

Po natrénování databáze přichází krok testování.

3.3.7 Testovací databáze

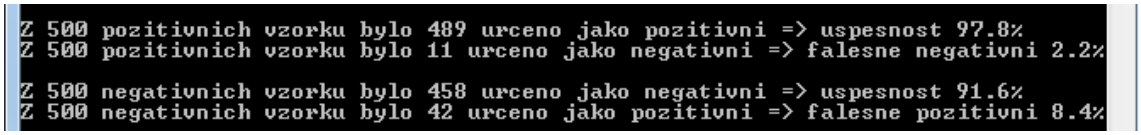
Pro testování bylo zvoleno celkově 1000 vzorků z výše zmiňované databáze INRIA. V našem programu jde pouze o demonstraci, nicméně pokud bychom chtěli dosáhnout co nejlepších výsledků, bylo by potřeba použít mnohem více vzorků jak pro testování, tak především pro trénování detektoru. Pokud bychom chtěli co nejkvalitněji natrénovaný detektor, vhodné by také bylo použití křížové validace. Princip křížové validace spočívá v rozdělení sady dat na podmnožiny dat této sady. Jedna podmnožina by byla použita jako testovací, ostatní jako trénovací. Tento postup by se několikrát opakoval, pokaždé s jinou podmnožinou testovacích dat. Na konci bychom dostali robustní detektor. Protože se však v naší práci zabýváme pouze demonstrací, jednoduché trénovaní jedné skupiny dat bude dostačující. Vzorky pro testování jsou programem zpracovány stejně jako vzorky pro trénování. Pro ověření na které straně hranice dat, rozdělené pomocí algoritmu SVM, se testovaný vzorek nachází, je použita funkce:

```
float response = SVM.predict(sampleMat);
```

Pokud je do proměnné `response` uložena hodnota 1, znamená to, že obraz je určen jako pozitivní (je na něm hledaný objekt). Pokud detektor vyhodnotí vzorek jako negativní, do `response` je uložena hodnota 0.

3.3.8 Výsledky detekce

Na obr. 3.10 můžeme vidět konzolový výpis úspěšnosti detekce deskriptoru. Hodnoty



Obr. 3.10: Konzolový výpis úspěšnosti detekce

jsou přehledně uvedeny také v tab. č. 3.1. Pro shrnutí si uvedeme ještě jednou nejdůležitější parametry:

- Trénovací sada 50 pozitivních a 50 negativních snímků
- Buňka velikost 8×8 pixelů
- Normalizace bloková 2×2 buňky

- Lineární SVM
- Testovací sada 500 pozitivních a 500 negativních snímků

Tab. 3.1: Testování detekce

Vzorků	Pozitvní	Detekováno	%	Negativní	Detekováno	%	Celkem
1000	500	489	97,8	500	458	91,6	94,7%

Celková úspěšnost 94,7% je velmi dobrým výsledkem. Nicméně jedná se o detekci, která je provedena na snímcích, které mají všechny stejnou velikost a postava je na nich vždy ve stejném poměru.

Posledním krokem, který už v našem programu není implementován, protože přesahuje rámec této práce, by tedy byla detekce objektu v rámci celého obrazu. Provedla by se pomocí detekčního okna o velikosti 64×128 pixelů, jak je uvedeno v teoretickém rozboru v kapitole 2.5. Toto okno postupně mění svoji pozici v obraze. Protože postava v obraze může být v různém poměru, detekční okno musí také měnit svoji velikost dle předpokládané velikosti objektu. Výsledná postava (nebo postavy) by se detekovala a v obraze se zvýraznila např. pomocí rámečku. Tato část již přesahuje rámec práce a mohla by být řesena v případné diplomové práci.

4 ZÁVĚR

Cílem této práce bylo seznámit se s metodou Histogram Orientovaných Gradientů pro detekci objektů v obraze a následně naprogramovat aplikaci pro demonstraci výpočtu a vizualizaci konkrétních obrazů s vykreslením příznakových vektorů - deskriptorů a detekci objektů.

Aplikace pracuje dle předpokladů, v obraze se vykreslí deskriptory pro 8 × 16 buněk, celkově tedy 128 deskriptorů. Vykreslení vektorů pracuje správně, v každé buňce se vykreslí jeden, nebo více vektorů. Velikost daného vektoru závisí na velikosti tohoto kanálu. Naprogramovaná aplikace je pouze konzolová a nelze během spuštění jakkoli měnit parametry s kterými program pracuje. Načtený obraz pojmenovaný `vstup.jpg` se načítá z kořenové složky programu. Trénovací i testovací databáze se taktéž načítají z kořenového adresáře programu ze složek `testovani` a `trenovani`. Výsledné vykreslené příznakové deskriptory se vykreslí v samostatném okně a po kliknutí na daný deskriptor se vykreslí detail této buňky. Vykreslování je interaktivní, tzn. po kliknutí na jinou buňku se automaticky zobrazí detail nové buňky.

Výslednou úspěšnost detekce postavy, s dosaženou úspěšností 94,7%, můžeme považovat za velmi dobrý výsledek. I když v této práci pracujeme pouze s detekcí lidské postavy v obraze, metodu HOG by bylo možné použít i pro detekci jakéhokoliv typu objektu. Pokud bychom chtěli program použít např. pro detekci automobilů, museli bychom si předem připravit databázi vzorků, na kterých se vyskytuje automobil a ty by byly použity pro natrénování SVM. Výsledek by ovšem s největší pravděpodobností nedosahoval dobrých výsledků, protože metoda Histogram Orientovaných Gradientů je primárně určena pro detekci lidských postav a v tomto ohledu dosahuje vynikajících výsledků. Právě z tohoto důvodu jsem pro demonstraci zvolil detekci postav. Pro testování byla použita databáze INRIA, kterou vytvořili autoři metody HOG.

V detekční části programu se zabýváme pouze fixní velikostí obrazu, tzn. není řešeno škálování. Škálování, tedy detekce objektů různých poměrů v různých velikostech obrazů, by mohlo být námětem pro budoucí diplomovou práci.

Navržený algoritmus byl implementován v prostředí MS Visual Studio 2012 pomocí objektově orientovaného jazyka C++ s použitím knihovny OpenCV a je součástí přílohy. Pro správné spuštění programu musí být v systému dostupné dynamické knihovny OpenCV příslušné verze.

LITERATURA

[1] Dalal, N.,Triggs, B.: *Histograms of Oriented Gradients for Human Detection*. International Conference on Computer Vision and Pattern Recognition, volume 2, pages 886–893, 2005. Dostupné z WWW: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1467360`

[2] Gonzales, R. Woods, R., 3. vydání. *Digital Image Procesing*. 3. vyd. New Jersey: Pearson Education, 2008. ISBN 978-0-13-168728-8.

[3] Bertozzi, M., Broggi, A., Rose, M., Felisa, M., Rakotomamonjy, A., Suard, F.: *A Pedestrian Detector Using Histograms of Oriented Gradients and a Support Vector Machine Classifier*. Intelligent Transportation Systems Conference, pages 143–148, 2007. Dostupné z WWW: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4357692`

[4] Zeman,V., *Hledání hran* [online]. Publikováno 21.10.2013 [cit. 2013-11-15]. Dostupné z WWW: `http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/22EdgeDetectionCz.pdf`

[5] Říha, K., *Pokročilé techniky zpracování obrazu*. Vysoké učení technické v Brně, 2007

[6] Hsu C.-W., Chang, C.-C., Lin, C.-J.: *LIBSVM: A Library for Support Vector Machines*. 2001, Department of Computer Science, National Taiwan University, Taipei, Taiwan, Dostupné z WWW: `http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf`

[7] Chang, C.-C., Lin, C.-J.: *A Practical Guide to Support Vector Classification*. 2003, Department of Computer Science, National Taiwan University, Taipei 106, Taiwan, Dostupné z WWW: `http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf`

A OBSAH PŘILOŽENÉHO DVD

- Elektronická verze práce
- Zdrojové kódy k aplikaci, projekt pro MS Visual studio 2012
- Sada trénovacích a testovacích obrázků

  Pro správné spuštění exe souboru musí být v systému dostupné dynamické knihovny OpenCV. Jinak se mohou vyskytnout problémy s kompatibilitou. Jejich seznam (verze 2.4.8) je zde:

  `opencv_calib3d248d.lib`
  `opencv_contrib248d.lib`
  `opencv_core248d.lib`
  `opencv_features2d248d.lib`
  `opencv_flann248d.lib`
  `opencv_gpu248d.lib`
  `opencv_highgui248d.lib`
  `opencv_imgproc248d.lib`
  `opencv_legacy248d.lib`
  `opencv_ml248d.lib`
  `opencv_nonfree248d.lib`
  `opencv_objdetect248d.lib`
  `opencv_photo248d.lib`
  `opencv_stitching248d.lib`
  `opencv_ts248d.lib`
  `opencv_video248d.lib`
  `opencv_videostab248d.lib`