

Robustní detekce laserových stop

Robust Detection Of Laser Blobs

Roman Hala

Bakalářská práce
2011



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2010/2011

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: Roman HALA
Osobní číslo: A08219
Studijní program: B 3902 Inženýrská informatika
Studijní obor: Informační a řídicí technologie

Téma práce: Robustní detekce laserových stop

Zásady pro vypracování:

1. Popište problematiku detekce laserových stop.
2. Popište vhodné algoritmy.
3. Analyzujte možnosti implementace.
4. Navrhněte vlastní implementaci.
5. Demonstrujte ukázkovou aplikaci.

Rozsah bakalářské práce:

Rozsah příloh:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. PRATA, Stephen. *Mistrovství v C++*. Computer Press, 2004. 1006 s. ISBN 80-251-0098-7.
2. SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. *Image Processing, Analysis, and Machine Vision*. PWS Publishing at Brooks/Cole Publishing Company, 1998. 770 s. ISBN 0-534-95393-X.
3. SNYDER, Wesley, E.; QI, Hairong. *Machine Vision*. Cambridge University Press, 2004. 433 s. ISBN 0 521 83046 X.
4. DAVIES, E., R. *Machine Vision*. Morgan Kaufmann Publishers, 2005. 934 s. ISBN-13: 978-0-12-206093-9, ISBN-10: 0-12-206093-8.
5. *Open Source Computer Vision Library Reference Manual*, Intel Corporation, 1999-2001. 417 s. Order Number: 123456-001. Dostupné z WWW: http://hawaiiilibrary.net/eBooks/Give-Away/Technical_eBooks/OpenCVReferenceManual.pdf.
6. BRADSKI, Gary; KAEHLER, Adrian. *Learning OpenCV*. O'Reilly Media, 2008. 555 s. ISBN: 978-0-596-51613-0.

Vedoucí bakalářské práce:

Ing. Erik Král

Ústav bezpečnostního inženýrství

Datum zadání bakalářské práce:

25. února 2011

Termín odevzdání bakalářské práce:

7. června 2011

Ve Zlíně dne 25. února 2011

prof. Ing. Vladimír Vašek, CSc.
děkan



prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

ABSTRAKT

Tato bakalářská práce se zabývá detekcí laserových stop. Návrh řešení vychází z metod knihovny OpenCV, které jsou schopny pracovat s obrázky a videem. Cílem je navrhnout program, jenž detekuje laserový paprsek tvaru kříže v reálném čase. Program je optimalizován pro práci s kamerou CMUcam3, která podporuje programy napsané v jazyce C.

Klíčová slova: Detekce přímek, OpenCV, počítačové vidění, detekce laserových stop, C++, C

ABSTRACT

This thesis deals with the detection of laser blobs. The proposed solution is based on OpenCV library methods that are able to work pictures pictures and video. The goal is to propose a program that is able to detect the cross-shaped laser beam in real time. The program is optimized to work with the camera CMUcam3 that supports programs written in C.

Keywords: Line detection, OpenCV, computer vision, laser blob detection, C++, C

Chtěl bych poděkovat vedoucímu bakalářské práce panu Ing. et Ing. Eriku Královi za jeho odbornou pomoc a rady při psaní této práce a návrhu programů.

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

OBSAH

ÚVOD.....	9
I TEORETICKÁ ČÁST.....	10
1 POČÍTAČOVÉ VIDĚNÍ	11
2 PROBLEMATIKA DETEKCE LASEROVÝCH STOP	12
2.1 BAREVNÝ MODEL	13
2.1.1 RGB model.....	13
2.2 PRAHOVÁNÍ.....	14
2.2.1 Základní prahování.....	15
2.2.1.1 Algoritmus základního prahování.....	15
2.3 MOMENTY	15
2.4 VYHLAZOVÁNÍ OBRAZU	16
2.4.1 Průměrné vyhlazování.....	16
2.5 DETEKCE HRAN	16
2.5.1 Canny	17
2.5.1.1 Algoritmus Canny.....	17
2.6 ALGORITMY PRO DETEKCI PŘÍMEK	18
2.6.1 Houghova transformace	18
2.6.1.1 Algoritmus Houghovy transformace pro detekci přímek	20
2.6.2 Ramer – Douglas – Peucker.....	21
2.6.2.1 Algoritmus Ramer - Douglas – Peucker.....	21
2.6.3 RANSAC.....	22
2.6.3.1 Algoritmus RANSAC.....	23
3 OPENCV	24
3.1 CO JE OPENCV?.....	24
3.2 STRUKTURA A OBSAH OPENCV.....	24
II PRAKTICKÁ ČÁST	26
4 INSTALACE OPENCV PRO VISUAL STUDIO 2010.....	27
4.1 INSTALACE OPENCV 2.2.0	27
4.2 INSTALACE CMAKE	33
4.2.1 Překlad knihovny OpenCV pomocí CMake	33
4.3 NASTAVENÍ PROJEKTU VE VISUAL STUDIO 2010.....	35
5 DETEKCE LASEROVÝCH STOP POMOCÍ KNIHOVNY OPENCV.....	38
5.1 NAČTENÍ OBRÁZKU	39
5.1.1 Laserová tečka.....	39
5.1.2 Laserový kříž.....	39

5.2	EXTRAKCE ČERVENÉ SLOŽKY	40
5.3	PRAHOVÁNÍ.....	41
5.4	VYHLAZENÍ PRO ODSTRANĚNÍ CHROMATICKÉ ABERACE	42
5.5	DETEKCE HRAN	43
5.6	OBRYSY PRO LASEROVOU TEČKU	43
5.7	HOUGHOVA TRANSFORMACE PRO LASEROVÝ KŘÍŽ.....	43
5.8	ALGORITMUS RAMER - DOUGLAS - PEUCKER PRO LASEROVÝ KŘÍŽ.....	44
5.9	VÝPOČET POZICE STŘEDU LASEROVÝCH STOP	44
5.9.1	Laserová tečka.....	44
5.9.2	Laserový kříž – Houghova Transformace.....	44
5.9.3	Laserový kříž – Algoritmus Ramer - Douglas – Peucker	45
5.10	VÝSLEDEK.....	45
5.10.1	Laserová tečka.....	45
5.10.2	Houghova transformace	46
5.10.3	Ramer - Douglas – Peucker.....	47
6	VLASTNÍ IMPLEMENTACE ALGORITMŮ PRO DETEKCI PŘÍMEK V JAZYKU C.....	49
6.1	RAMER - DOUGLAS – PEUCKER.....	49
6.2	RANSAC.....	50
6.3	HOUGHOVA TRANSFORMACE.....	51
	ZÁVĚR	52
	ZÁVĚR V ANGLIČTINĚ.....	53
	SEZNAM POUŽITÉ LITERATURY.....	54
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	57
	SEZNAM OBRÁZKŮ	59
	SEZNAM TABULEK.....	61
	SEZNAM PŘÍLOH.....	62

ÚVOD

Detekce laserových stop má široké uplatnění jak pro civilní potřeby, tak i pro armádu. Používá se v robotice, měření vzdálenosti a rychlosti, pro bezpečnostní zařízení, atd.

K těmto účelům můžeme použít knihovnu OpenCV s potřebnými metodami k získání informací, které potřebujeme.

Cílem práce je pomocí této knihovny otestovat různé metody a algoritmy v praxi, vybrat z nich nejvhodnější typ pro detekci laserového paprsku ve tvaru kříže a ten naprogramovat v jazyce C tak, aby byl kompatibilní s mikropočítačem obsaženým v kameře CMUcam3.

Teoretická část práce je zaměřena na problematiku detekce laserových stop a popis algoritmů, které se k tomu nejčastěji používají.

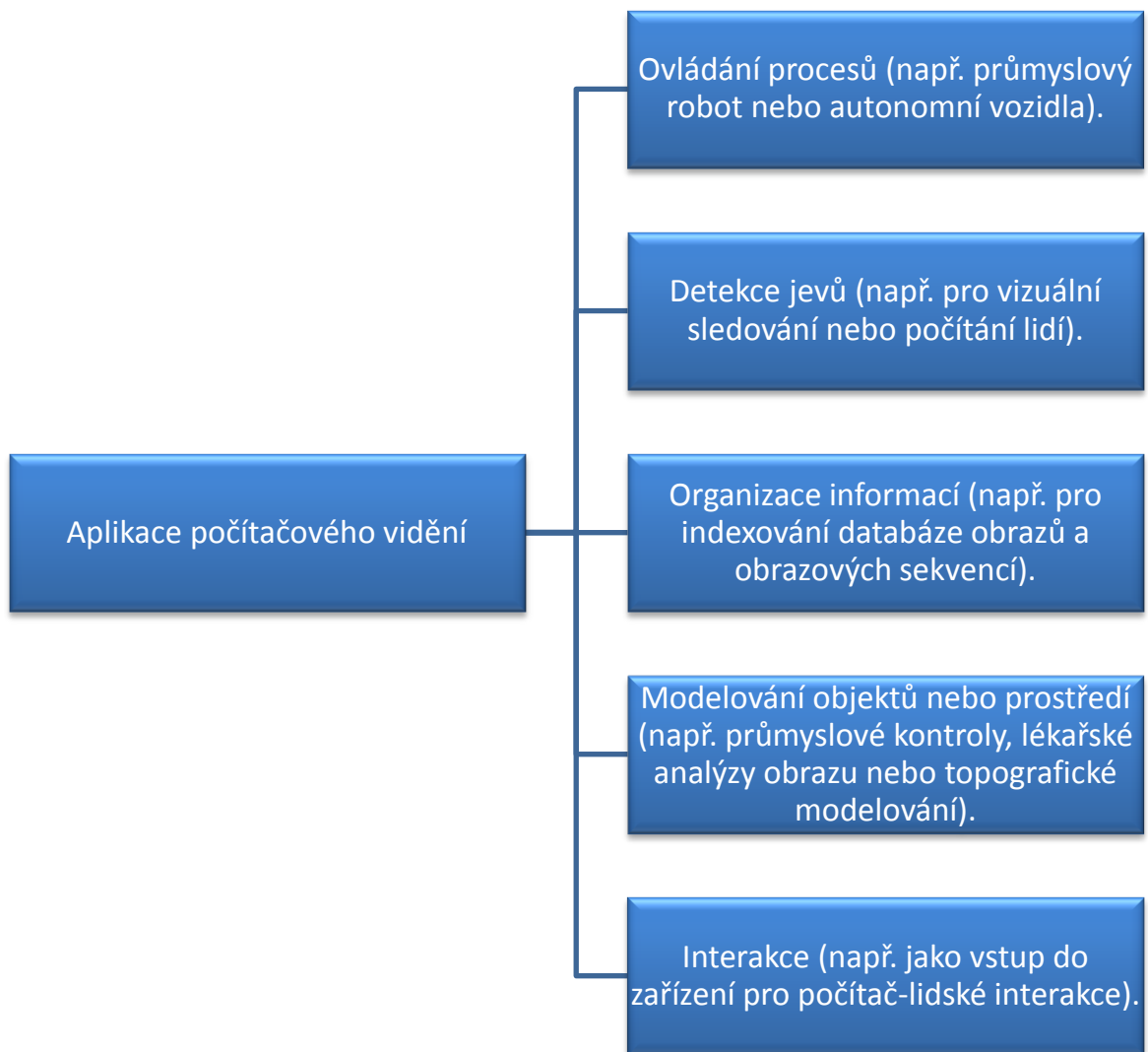
Praktická část je věnována instalaci knihovny OpenCV, v níž posléze testuji jednotlivé algoritmy. Poté je popsána vlastní implementace vhodná pro kameru CMUcam3.

I. TEORETICKÁ ČÁST

1 POČÍTAČOVÉ VIDĚNÍ

Počítačové vidění je věda a technologie strojů, u kterých je schopnost vidění podmíněna získáním informací z obrázku, jež jsou nezbytné k řešení určitého úkolu. Jako vědecká disciplína se počítačové vidění zabývá teorií umělých systémů extrahujících informace z obrazu. Obrazová data mohou mít mnoho forem, jako jsou videosekvence, pohledy z více kamer, nebo multidimenzionální data z lékařského skeneru.

Mezi podoblasti počítačového vidění patří rekonstrukce obrazu, detekce události, rozpoznávání objektů, učení, indexování, odhad pohybu, atd. [1]



Obr. 1. Aplikace počítačového vidění

2 PROBLEMATIKA DETEKCE LASEROVÝCH STOP

Při detekci laserových stop se nehledá laser jako takový. Můžeme pouze najít stopy červené (nebo jiné barvy v závislosti na použitém laseru) a tvar, který na osvětleném povrchu laser zanechá. V praxi si ze zdrojového obrázku vybereme libovolnou barvu, prahováním se zbavíme falešných stop a poté hledáme tvar stopy, kterou laser zanechal. Hledání těchto tvarů je z celého našeho problému matematicky nejsložitější. Algoritmů je pro tento problém spousta. Jejich výběr záleží na tvaru stopy a požadavcích na rychlost a kvalitu detekce. Abychom byli schopni detekovat stopy laseru, převedeme zdrojový obraz do binárního tvaru pomocí techniky prahování, k níž můžeme připojit jeden z algoritmů pro detekci hran. Pak už můžeme použít jeden z algoritmů pro detekci přímek, například z (Tab. 1).

Tab. 1. Experimentální výsledky algoritmů pro detekci přímek [2]

Algoritmus	Výpočetní obtížnost	Rychlost [Hz]	Počet nalezených čar	Správnost		Přesnost	
				Správné pozitivita [%]	Falešné pozitivita [%]	σ_{Δ_r} [cm]	σ_{Δ_α} [°]
Split-Merge + Clustering	$N \cdot \log N$	1470	641	86	8,9	1,95	0,74
Přírůstkový	$S \cdot N^2$	344	561	77,8	5,9	2,04	0,72
Přírůstkový + Clustering	$S \cdot N^2$	617	567	79,2	5,1	2,04	0,76
Lineární regrese	$N \cdot N_f$	364	577	76,4	10,1	1,99	0,8
LR + Clustering	$N \cdot N_f$	384	562	75,8	8,4	1,97	0,79
RANSAC	$S \cdot N \cdot N.Trials$	29	749	75,6	31,5	1,68	0,77
RANSAC + Clustering	$S \cdot N \cdot N.Trials$	93	547	70,7	12,2	1,37	0,7
Houghova Transformace	$S \cdot N \cdot NC + S \cdot NR \cdot NC$	8	825	82	32,5	1,63	0,76
HT + Clustering	$S \cdot N \cdot NC + S \cdot NR \cdot NC$	9	600	79,5	10	1,51	0,67
Expectation-Maximization	$S \cdot N1 \cdot N2 \cdot N$	0,6	1153	78,6	53,7	2,09	0,97
E-M + Clustering	$S \cdot N1 \cdot N2 \cdot N$	0,7	709	80,3	23,1	1,58	0,73

Legenda k (Tab. 1):

N	Počet vstupních bodů
S	Počet získaných segmentů čar (7 v průměru, záleží na algoritmu)
N_f	Posuvná velikost okna pro lineární regresi (9)
$N.Trials$	Počet pokusů o RANSAC (1000)
NC, NR	Počet sloupců a řádků, případně pro HT akumulátor pole ($NC = 401$, $NR = 671$ pro rozlišení $r_{res} = 1$ cm, $\alpha_{res} = 0,9^\circ$)
$N1, N2$	Počet pokusů a konvergence iterací, respektive, pro EM ($N1 = 50$, $N2 = 200$).

2.1 Barevný model

Barevný model je abstraktní matematický model popisující způsob zobrazení barev. Barvy mohou být reprezentovány jako n -tice čísel, nejčastěji o třech, nebo čtyřech hodnotách barevné složky. Pokud je tento model spojen s přesným popisem, jak jsou složky interpretovány (světelné podmínky, atd.), tak se výsledná množina barev nazývá barevný prostor. [21]

2.1.1 RGB model

Média, která vyzařují světlo (např. televize) používají aditivního míchání barev červené, zelené a modré, z nichž každá stimuluje oční receptory jedné barvy a zároveň co nejméně ovlivňuje ostatní. Směsi těchto tří základních barev pokrývají velkou část lidského barevného spektra. Bohužel neexistuje žádný diagram barevnosti, jakou by barvy měly mít, takže stejné hodnoty RGB mohou mít na různých obrazovkách odlišné barvy. [21]

R	G	B	barva	barva
0	0	0	černá	černá
255	0	0	červená	červená
0	255	0	zelená	zelená
0	0	255	modrá	modrá
255	255	0	žlutá	žlutá
255	0	255	purpurová	purpurová
0	255	255	azurová	azurová
255	255	255	bílá	bílá

Obr. 2. Ukázka míchání základních barevných směsí pomocí RGB modelu [22]

2.2 Prahování

Prahování v úrovni šedé je nejjednodušší proces segmentace. Mnoho objektů nebo obrázků je charakterizováno konstantní odrazivostí nebo absorpcí světla na jejich povrchu. Konstantní jas nebo prahové hodnoty mohou být nastaveny pro část objektů a pozadí. Prahování je výpočetně nenáročné a rychlé - je to nejstarší způsob segmentace a stále se používá v jednoduchých aplikacích. Prahování lze snadno provádět v reálném čase i pomocí specializovaného hardwaru.

Prahování je transformace vstupního obrazu f na výstup (segmentaci) binárního obrazu g pomocí vztahu

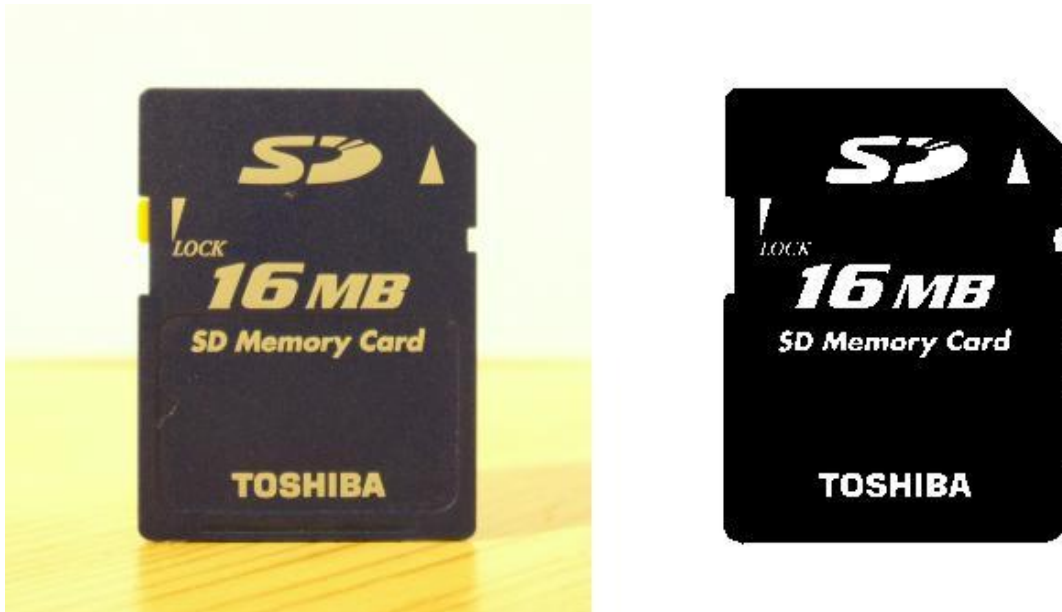
$$\begin{aligned}
 g(i,j) &= 1 && \text{for } f(i,j) \geq T \\
 g(i,j) &= 0 && \text{for } f(i,j) < T
 \end{aligned}
 \tag{1}$$

kde T je práh, $g(i,j) = 1$ pro obrazové prvky objektů, a $g(i,j) = 0$ pro obrazové prvky z pozadí (nebo naopak). [3]

2.2.1 Základní prahování

2.2.1.1 Algoritmus základního prahování

1. Najdi všechny body $f(i, j)$ v obraze f .
2. Obrazový prvek $g(i, j)$ segmentovaného obrazu je bod objektu pokud $f(i, j) \geq T$ zbytek bodů je pozadí. [3]



Obr. 3. Ukázka základního prahování [13]

2.3 Momenty

Objekt v obrazu se dá popsat pomocí plochy, polohy těžiště, náklonu, atd. K extrakci těchto informací nám slouží statistické momenty obrazu. Tyto momenty se dělí na obecné a centrální.

Obecný moment se spočítá pomocí vztahu

$$m_{p,q} = \sum_x \sum_y I(x,y) x^p y^q \quad (2)$$

x a y nám označují souřadnice pixelů, p a q stupeň momentu a $I(x,y)$ je intenzita bodu. Když v binárním obrazu spočítáme obecný moment pro p a $q = 0$, tak získáme plochu objektu. Pro výpočet těžiště nám stačí obecné momenty zahrnuté ve vztahu

$$(\bar{x}, \bar{y}) = \left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}} \right) \quad (3)$$

\bar{x} a \bar{y} jsou v tomto případě souřadnicemi těžiště.

Centrální moment se spočítá pomocí vztahu

$$\mu_{p,q} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I_{(x,y)} \quad (4)$$

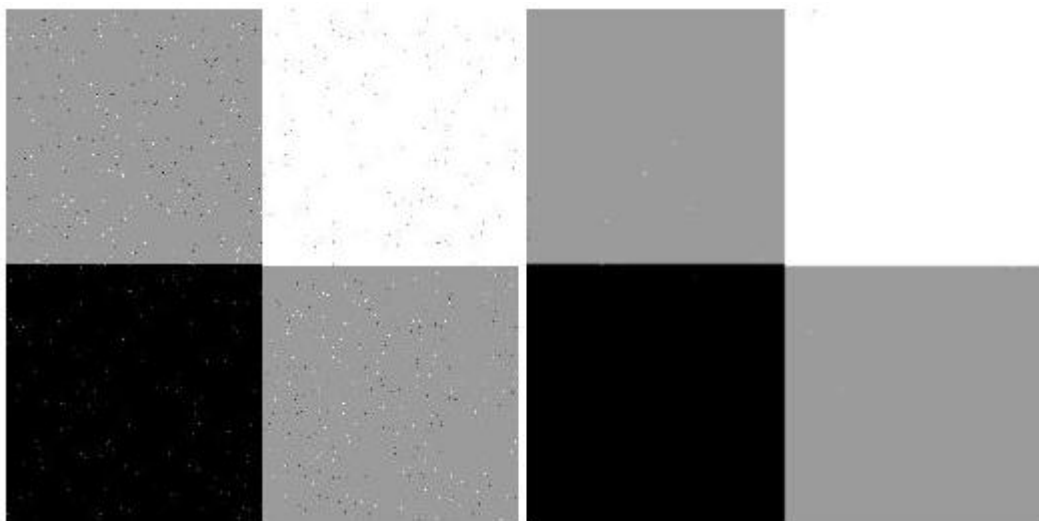
x a y jsou tady souřadnicemi pixelů, $p + q =$ stupeň momentu s , \bar{x} a \bar{y} jsou souřadnicemi těžiště. [24]

2.4 Vyhlazování obrazu

Snahou vyhlazování obrazu je zachytit co nejvíce důležitých dat ze zašuměného zdroje. [12]

2.4.1 Průměrné vyhlazování

Průměrné vyhlazování se používá k odstranění šumu z obrazu. Pixel s jinou intenzitou je nahrazen pixelem s průměrnou hodnotou v rámci okolí nahrazovaného pixelu. Pokud se část okolí nachází mimo obraz, použijí se hodnoty, které jsou uvnitř obrazu. Kruhové okolí se používá k vyhlazování, jež je nezávislé na orientaci obrázku. [14]



Obr. 4. Ukázka vyhlazování obrazu průměrným vyhlazováním o radiusu 1 pixel [14]

2.5 Detekce hran

Detekce hran je základní nástroj pro zpracování obrazu v počítačovém vidění, zejména v oblasti detekce a extrahování vlastností, které se zaměřují na hledání bodů v obrazu, v němž se prudce mění jas. [15]

Účelem detekce hran je podstatně zmenšit množství dat v obrázku a zachovat konstrukční vlastnosti, které se využijí pro další zpracování obrazu. [16]

2.5.1 Canny

Tento algoritmus byl představen Johnem F. Cannym již v roce 1986 a stále se ve výzkumu používá. Cílem Cannyho bylo vyvinout algoritmus, který bude odpovídat následujícím kritériím:

1. Detekce: Pravděpodobnost detekce reálné hrany by měla být maximalizována, zatímco pravděpodobnost falešné detekce by měla být minimalizována. To odpovídá maximalizaci koeficientu signál-šum.
2. Lokalizace: Detekované hrany by měly ležet co nejbližší k reálným hranám.
3. Minimální odezva: Jedna reálná hrana nesmí být detekována více jak jednou. [17]

2.5.1.1 Algoritmus Canny

1. Vyhlazení: Rozmazání obrazu pro odstranění šumu.
2. Hledání gradientů: Hrany by mělo být označeny tam, kde gradienty obrázku mají velké hodnoty.
3. Potlačení hodnot, které nejsou maximální: Za hrany by měly být označeny pouze lokální maxima.
4. Dvojitě prahování: Potenciální hrany jsou určeny prahováním.
5. Detekování hran hysterezí: Konečné hrany jsou určeny potlačením všech hran, které nejsou spojeny k velmi silné hraně. [16]



Obr. 5. Ukázka použití algoritmu Canny [18]

2.6 Algoritmy pro detekci přímek

2.6.1 Houghova transformace

Houghova transformace je technika, která může být použita k izolaci prvku určitého tvaru v rámci obrázku. Protože je potřeba, aby požadované prvky byly zadány v nějaké parametrické podobě, tak se klasická Houghova transformace nejvíce používá pro detekci pravidelných křivek, jako jsou přímky, kruhy, elipsy, atd. Všeobecnou Houghovu transformaci lze využít v aplikacích, kde jednoduchý analytický popis funkce není možný. I přes svá omezení si klasická Houghova transformace najde mnoho druhů uplatnění, jako je většina vyráběných dílů (a mnoho anatomických částí vyšetřovaných v lékařských snímcích) obsahujících funkci ohraničení, které mohou být popsány pravidelnými křivkami. Hlavní výhodou metody Houghovy transformace je, že je tolerantní k mezerám v popisu funkce ohraničení a má relativně malý šum v obraze. [4]

Lineární transformace pro detekci přímek je nejjednodušší případ Houghovy transformace. V obraze (2D – x, y) můžeme přímku popsat rovnicí

$$y = mx + b \quad (5)$$

a graficky ji zobrazit pomocí dvou bodů v obraze. V Houghově transformaci je hlavní myšlenkou nebrat přímku jako soustavu obrazových bodů (x_1, y_1) , (x_2, y_2) , atd., ale místo toho ji charakterizovat pomocí parametrů m (směrnice přímky) a b (průsečík s osou y). Na základě těchto skutečností můžeme přímku $y = mx + b$ interpretovat jako bod (b, m)

v parametrickém prostoru. Nicméně toto řešení má jeden problém s vertikálními přímkami, které způsobí, že parametry m a b budou mít nekonečné hodnoty. Z výpočetních důvodů je proto lepší použít jiné parametry označené r a θ . Parametr r představuje vzdálenost mezi přímkou a počátkem a θ je úhel od počátku k přímce. Pomocí této parametrizace lze rovnici zapsat jako

$$y = \left(-\frac{\cos \theta}{\sin \theta}\right)x + \left(\frac{r}{\sin \theta}\right) \quad (6)$$

která může být upravena na tvar

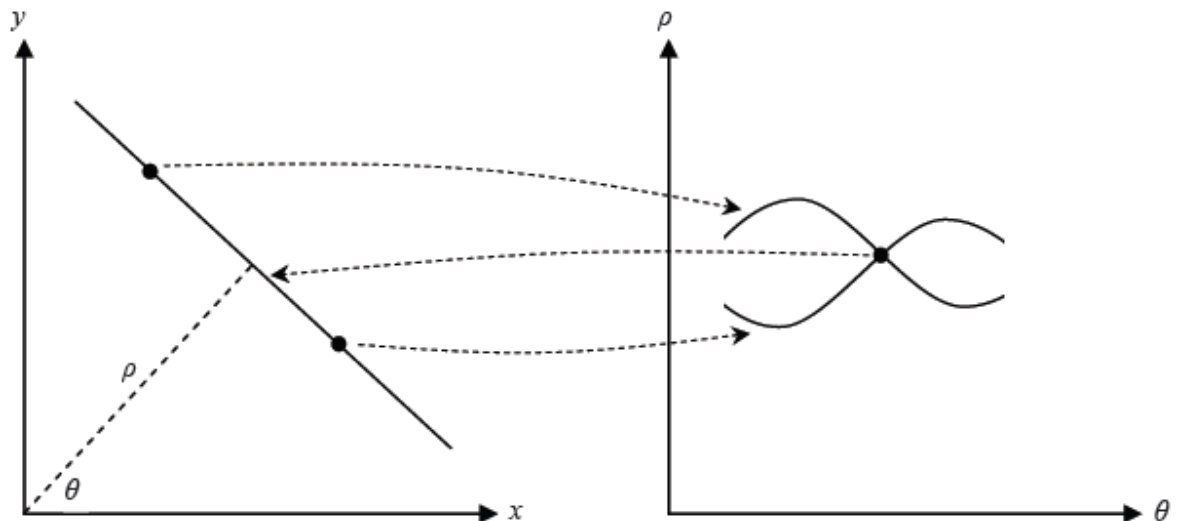
$$r = x \cos \theta + y \sin \theta \quad (7)$$

Je tedy možné přiřadit každé přímce pár (r, θ) , který je jedinečný, pokud $\theta \in [0, \pi)$ a $r \in \mathbb{R}$, nebo pokud $\theta \in [0, 2\pi)$ a $r \geq 0$. Rovina (r, θ) je někdy označována jako Houghův prostor pro množinu přímek ve dvou rozměrech. Pro libovolný bod na obrázku se souřadnicemi (x_0, y_0) platí, že každá přímka, která jím prochází, má rovnici

$$r(\theta) = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta \quad (8)$$

kde r (vzdálenost mezi přímkou a počátkem) je označen jako θ .

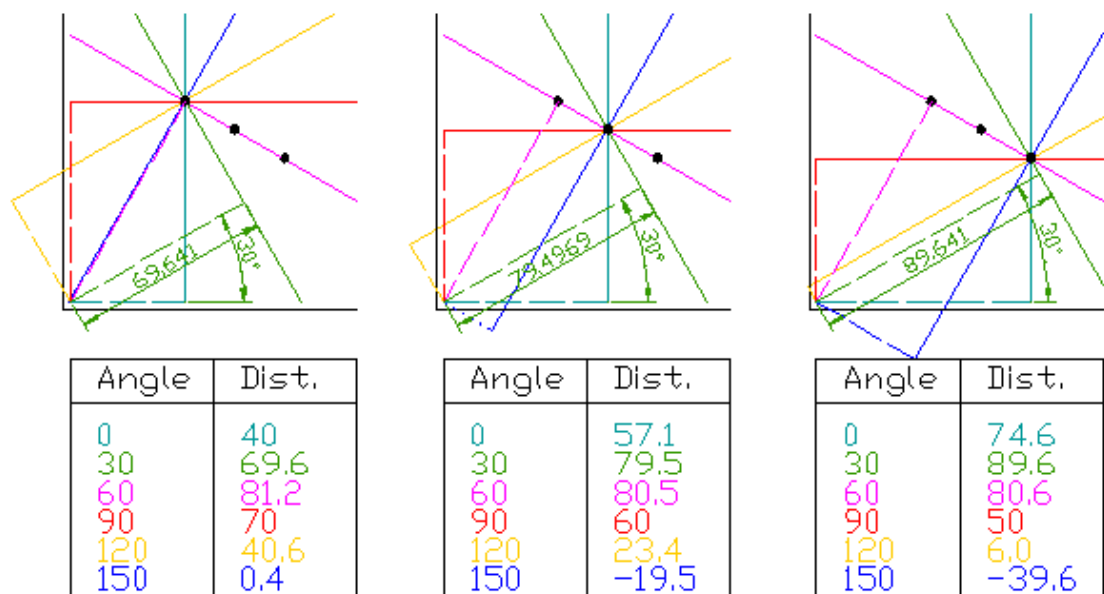
Tato rovnice odpovídá sinusoidě v rovině (r, θ) , která je unikátní pro každý bod. Pokud se křivky odpovídající dvěma bodům protínají, tak místo (v Houghově prostoru), kde se protnou, odpovídá přímce (v původním obrazovém prostoru), procházející oběma body. Obecněji řečeno, množina bodů, z nichž je přímka složena, tvoří sinusoidy protínající se v parametrech přímky. Tak se může problém hledání kolineárních bodů převést na hledání souběžných křivek. [19]



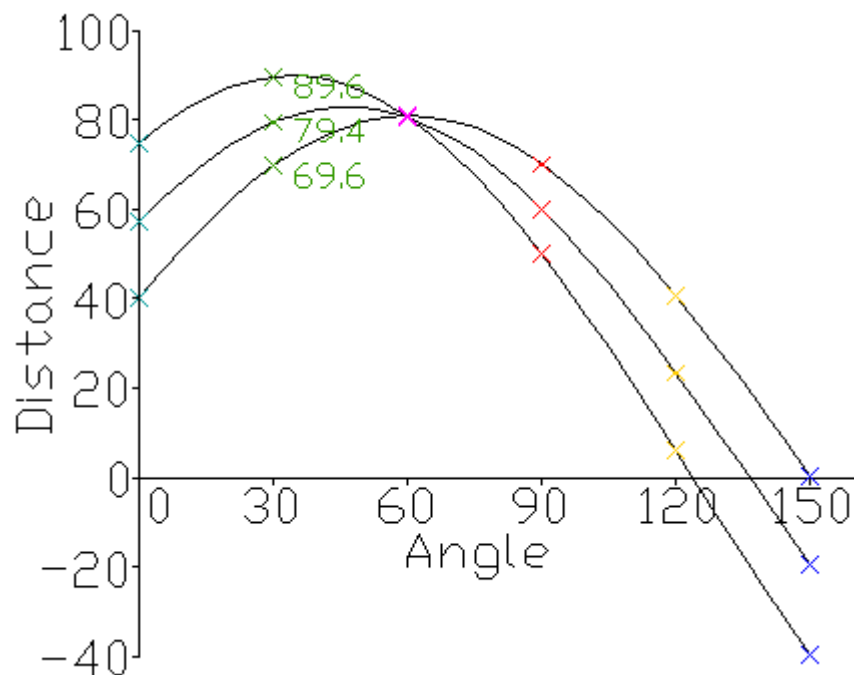
Obr. 6. Ukázka převodu přímky z (x, y) prostoru do Houghova (r, θ) prostoru (v obrázku vyznačeno (ρ, θ)) [20]

2.6.1.1 Algoritmus Houghovy transformace pro detekci přímek

1. Každý datový bod proloží řadou přímek, každou pod jiným úhlem.
2. Pro každou přímku si spočítá vzdálenost od rovnoběžky, která prochází počátkem.
3. Vzdálenost rovnoběžek a úhel přímky od osy x si uloží.
4. Toto se opakuje pro všechny body a pak se za přímku prohlásí soustava bodů, které mají největší počet stejných (v určených mezích) vzdáleností v jednom z úhlů.



Obr. 7. Ukázka proložení třech bodů přímkami po 30° a výpočtu jejich vzdálenosti od rovnoběžné přímky procházející počátkem. [19]



Obr. 8. Ukázka zobrazení v Houghově prostoru proložení třech bodů přímkami po 30° a vyznačení jejich vzdáleností od rovnoběžné přímky procházející počátkem. [19]

2.6.2 Ramer – Douglas – Peucker

Algoritmus Ramer – Douglas - Peucker je algoritmus pro snížení počtu bodů v křivku aproximovanou množinou bodů. Počáteční forma algoritmu byla nezávisle navržena v roce 1972 Urs Ramerem a v roce 1973 Davidem Douglasem a Thomasem Peuckerem. Tento algoritmus je znám také pod těmito názvy: Ramer – Douglas - Peucker algoritmus, The Iterative End-point Fit algoritmus nebo Split and Merge algoritmus.

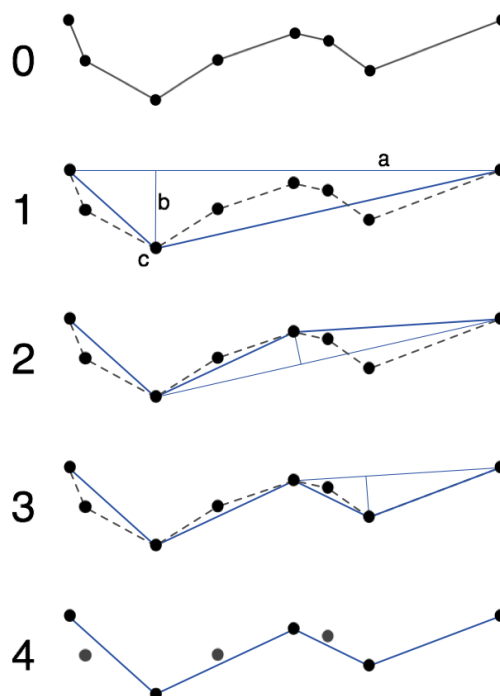
Cílem tohoto algoritmu je, vzhledem k tomu, že je křivka složená z mnoha úseček, najít podobnou křivku s méně body. Zjednodušená křivka se skládá z podmnožiny bodů, které definovaly původní křivku.

Výchozí křivka je uložena jako množina bodů nebo přímek a rozsahem vzdáleností $\epsilon > 0$. Algoritmus rekurzivně dělí křivku. [6]

2.6.2.1 Algoritmus Ramer - Douglas – Peucker

1. Algoritmus si sám označí první a poslední bod, aby se nesmazaly.

2. Pak hledá bod, který je nejvzdálenější od linky tvořené prvním a posledním bodem.
3. Pokud je bod blíže, než je zadané ϵ , pak všechny body, které nejsou zrovna označené, mohou být vyřazeny, aniž by vyhlazená křivka byla horší než ϵ , které je zadané.
4. Pokud je bod dál, než je zadané ϵ , pak se přímka procházející prvním a posledním bodem rozdělí na dvě přímky a vzdálený bod bude taky jedním z bodů, kterými budou přímky procházet. [6]



Obr. 9. Ukázka zjednodušení aproximace metodou Ramer – Douglas – Peucker [6]

2.6.3 RANSAC

RANSAC je zkratka pro „RANdom SAMple Consensus“ (souhlas náhodného vzorku). Je to iterační metoda k odhadu parametrů matematického modelu z množiny bodů, která obsahuje body nepatřící do modelu (šum). Je to nedeterministický algoritmus ve smyslu produkce rozumného výsledku pouze s určitou pravděpodobností, která se zvyšuje s počtem povolených iterací. Tento algoritmus byl poprvé publikován Fischlerem a Bollesem v roce 1981.

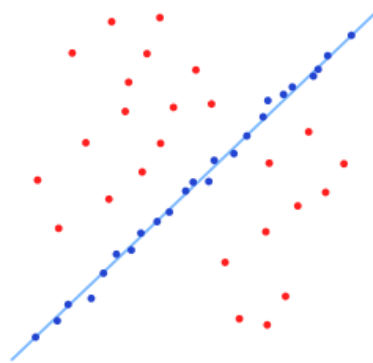
Ve 2D prostoru se přímky hledají takto. Nejprve máme množinu bodů, mezi nimiž některé patří do modelu a některé ne. RANSAC vytvoří model, který se počítá pouze z bodů, které do modelu patří pod podmínkou, že je vysoká možnost jejich výběru. Samozřejmě není žádná záruka, že tato situace nastane, ale je spousta parametrů algoritmu, které musí být voleny opatrně, aby se udržela pravděpodobnost na vysoké úrovni. [8]

2.6.3.1 Algoritmus RANSAC

RANSAC dosáhne svého cíle iteračním výběrem určité části originálních dat. Tyto data jsou potenciální přilehlé body a tato možnost se testuje takto:

1. Odhadnutý model je sestaven z bodů, které hypoteticky patří modelu, tzn., že všechny parametry modelu jsou vytvořeny z bodů, které patří modelu.
2. Všechny ostatní body jsou pak testovány vůči modelu, a pokud bod dobře zapadá do odhadnutého modelu, tak je rovněž považován za bod patřící modelu.
3. Odhadnutý model je dostatečně dobrý, pokud je dost bodů, které byly prohlášeny za patřící k modelu.
4. Model je znova odhadnut ze všech bodů, které hypoteticky patří modelu, protože byl odhadnut pouze ze vstupní množiny potenciálních bodů modelu.
5. A naposled je model vyhodnocen odhadem chyby bodů modelu vzhledem k modelu.

Tento postup se opakuje několikrát (počet je pevně zadán). Pokaždé se vyskytne nějaký model, který je odmítnut pro nízký počet bodů prohlášených za přilehlé, nebo vybraný model odpovídá chybě měření. Ve druhém případě, pokud je jeho chyba nižší než posledního uloženého modelu, model necháme. [8]



Obr. 10. Ukázka proložení přímky
algoritmem RANSAC. [8]

3 OPENCV

3.1 Co je OpenCV?

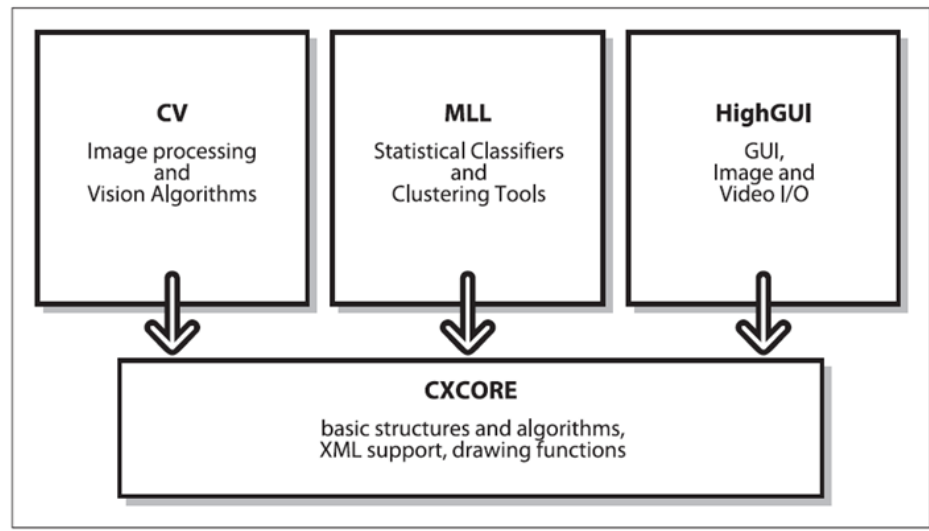
OpenCV je opensource knihovna počítačového vidění. Knihovna je napsaná v jazycích C a C++ a běží pod operačními systémy Linux, Windows a Mac OS X. Probíhá aktivní vývoj pro Python, Ruby, Matlab, a jiné jazyky.

OpenCV byla navržena pro výpočetní efektivitu se zaměřením na realtime aplikace. OpenCV je napsána v optimalizovaném jazyce C a díky tomu může využít výhod vícejádrových procesorů. Pokud se rozhodnete pro automatickou optimalizaci, můžete si koupit od Intelu Integrated Performance Primitives (IPP) knihovny, které se skládají z nízkoúrovňových výpočtů optimalizovaných v mnoha oblastech algoritmizace. OpenCV IPP knihovnu, pokud je nainstalována, automaticky použije.

Jedním z cílů OpenCV je poskytovat uživatelsky jednoduchou infrastrukturu počítačového vidění, která uživatelům pomáhá tvořit v krátkém čase poměrně sofistikované aplikace. Knihovna OpenCV obsahuje více než 500 funkcí počítačového vidění, které zahrnují kontrolu výrobků, zobrazování v lékařství, bezpečnost, uživatelské rozhraní, kalibraci kamer, stereo vidění a robotiku. Jelikož počítačové vidění a učení strojů jdou často ruku v ruce, OpenCV také obsahuje celou, všeobecně zaměřenou knihovnu učení strojů (MLL). Tato knihovna je zaměřena na statistické rozpoznávání a clustering. MLL je hodně užitečná pro úkoly počítačového vidění, které jsou jádrem OpenCV, ale jsou dostatečně obecné, aby mohly být použity i pro problém učení strojů. [9]

3.2 Struktura a obsah OpenCV

OpenCV je široce strukturovaná do pěti hlavních částí, z nichž čtyři jsou uvedeny v obrázku. Oddíl CV obsahuje základní zpracování obrazu a vysokoúrovňové algoritmy počítačového vidění. MLL je knihovna učení strojů, která zahrnuje mnoho statistických třídících ERS a nástrojů pro clustering. HighGUI obsahuje vstupní a výstupní funkce pro ukládání a načítání videa a obrázků a CXCore obsahuje základní datové struktury a obsah.



Obr. 11. Struktura a obsah OpenCV [9]

(Obr. 11) nezahrnuje CvAux, která obsahuje dvě zaniklé oblasti (vestavěné HMM rozpoznání obličejů) a experimentální algoritmy (segmentaci pozadí / popředí). CvAux není moc dobře zdokumentována na Wiki, ani v podsložce `.../opencv/docs`. CvAux obsahuje:

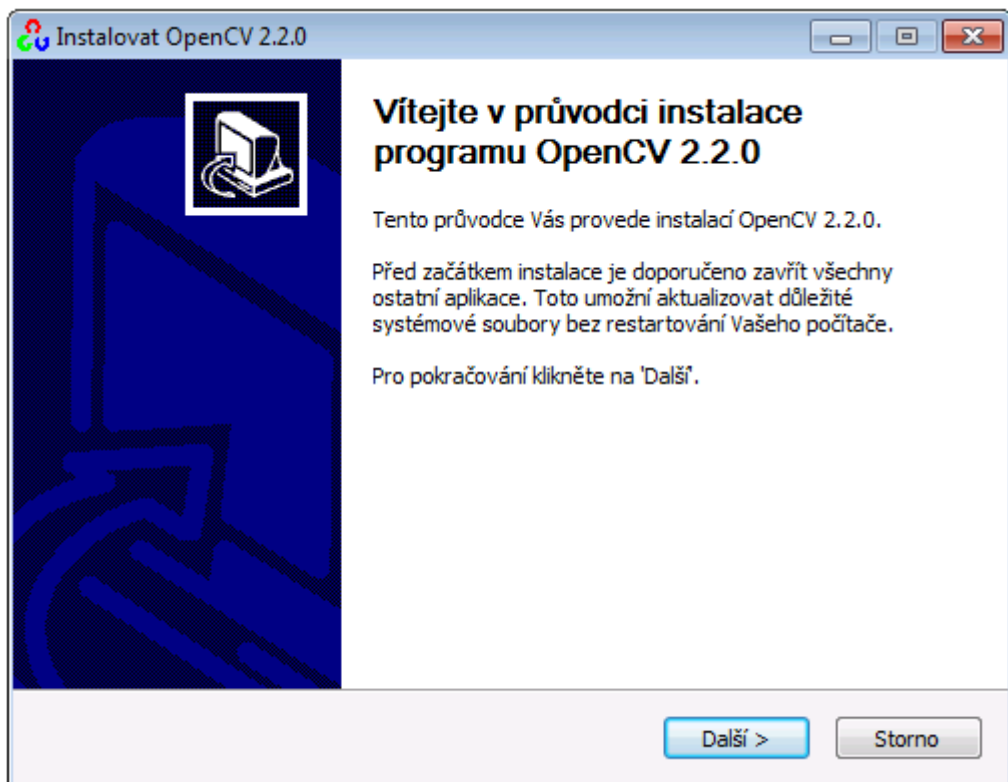
- Vlastní objekty, výpočetně efektivní rozpoznávací technika, která je v podstatě šablonou shody procesu.
- 1D a 2D skryté Markovovy modely, statistické metody rozpoznávání řešené dynamickým programováním.
- Vestavěné HMM (Vyjádření rodiče HMM jsou HMM samy o sobě)
- Rozeznání gest ze stereo vidění
- Rozšíření Delaunayovy triangulace, sekvencí, atd.
- Stereo vidění
- Rozpoznávání tvaru s konturami regionu
- Deskriptory textur
- Sledování očí a úst
- 3D sledování
- Nalezení kostry objektů (centrální linie) v obrázku
- Deformace středních pohledů dvou kamer
- Segmentace pozadí / popředí
- Video sledování
- Kalibrace kamery pomocí tříd C++ (funkce psané v C a jádro jsou v oddílu CV) [9]

II. PRAKTICKÁ ČÁST

4 INSTALACE OPENCV PRO VISUAL STUDIO 2010

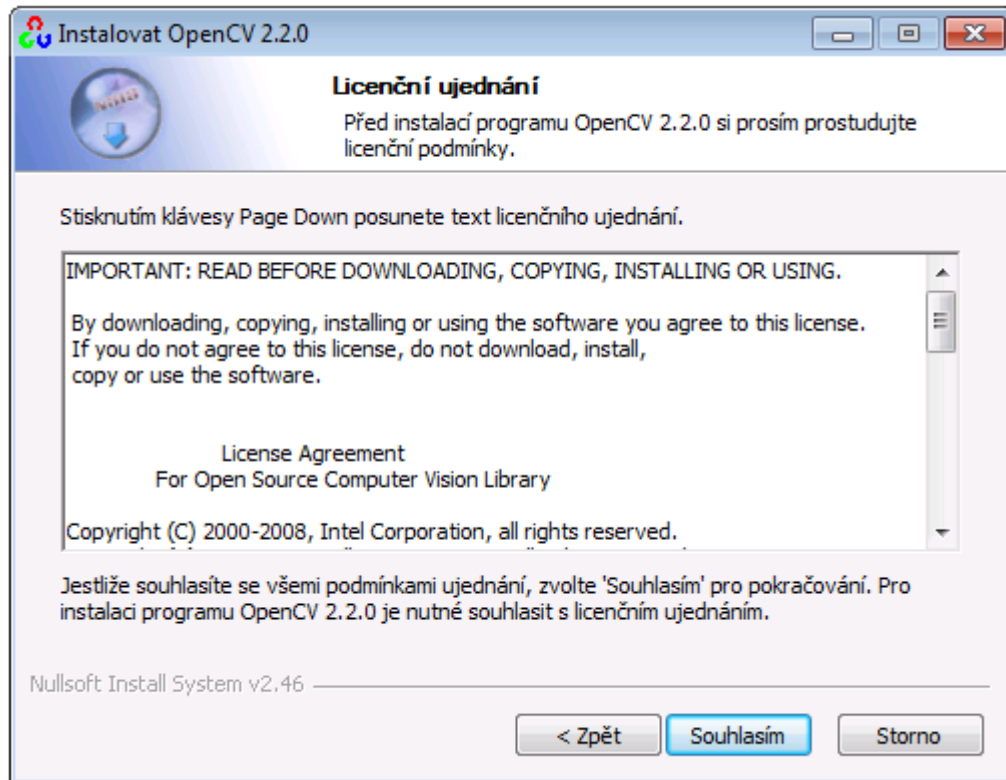
4.1 Instalace OpenCV 2.2.0

Nejprve je potřeba si stáhnout knihovnu OpenCV ze stránek tvůrců (zde). Po stáhnutí si spusťte instalaci a pokračujte podle uvedených kroků:



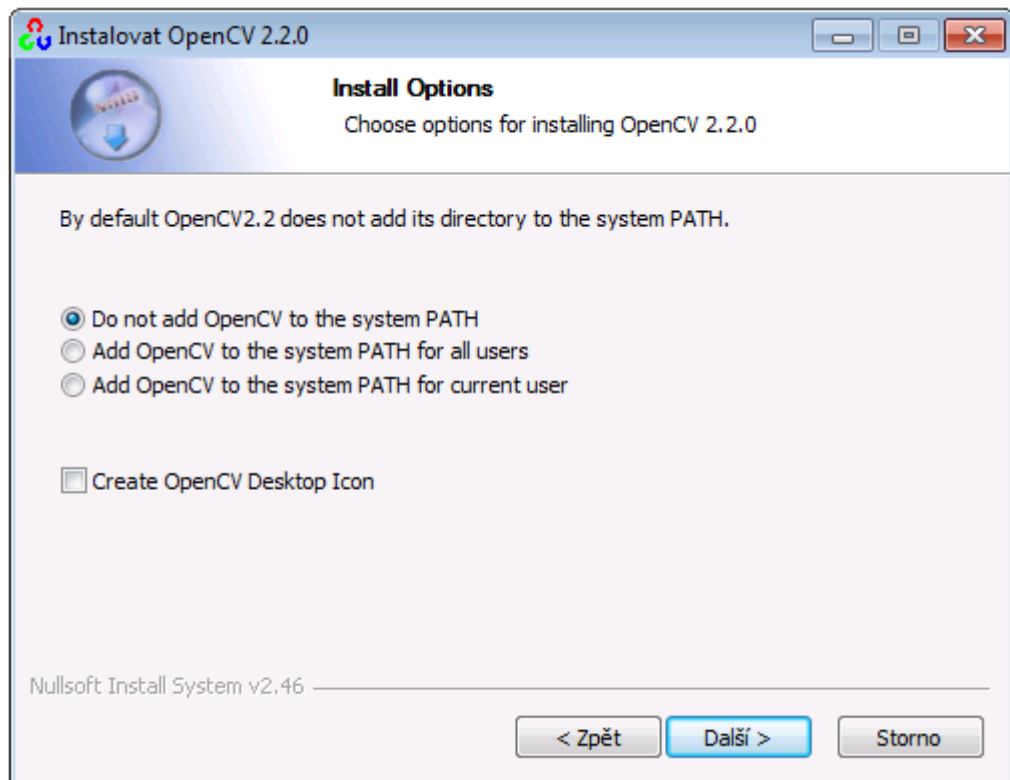
Obr. 12. Instalace OpenCV – Úvodní okno

Zvolte tlačítko **Další >**,



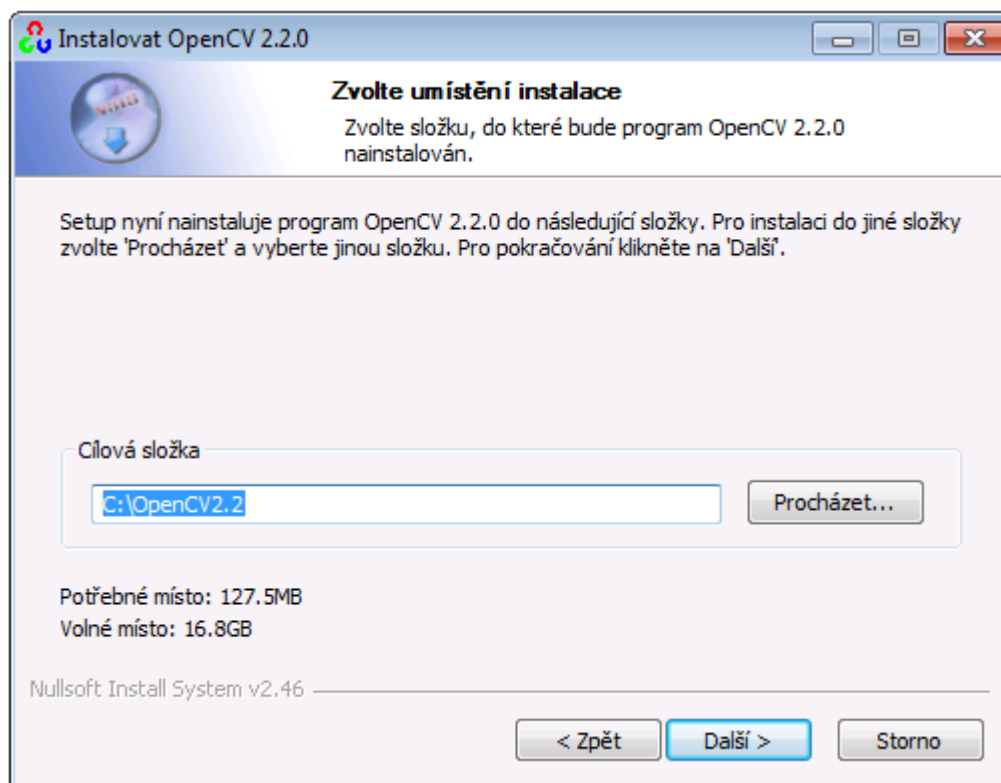
Obr. 13. Instalace OpenCV – Licenční ujednání

Zvolte tlačítko **Souhlasím**,



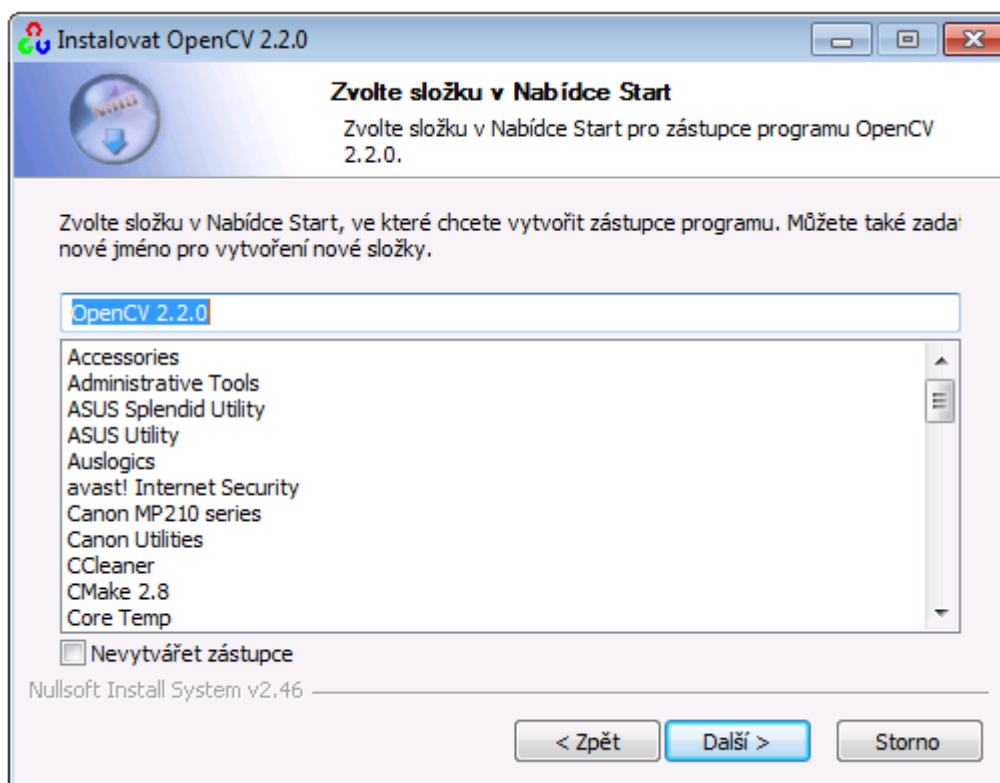
Obr. 14. Instalace OpenCV – Možnosti instalace

Tady je výběr přidání OpenCV do systémové cesty. Vyberte si možnost, kterou chcete a pak zvolte tlačítko **Další >**



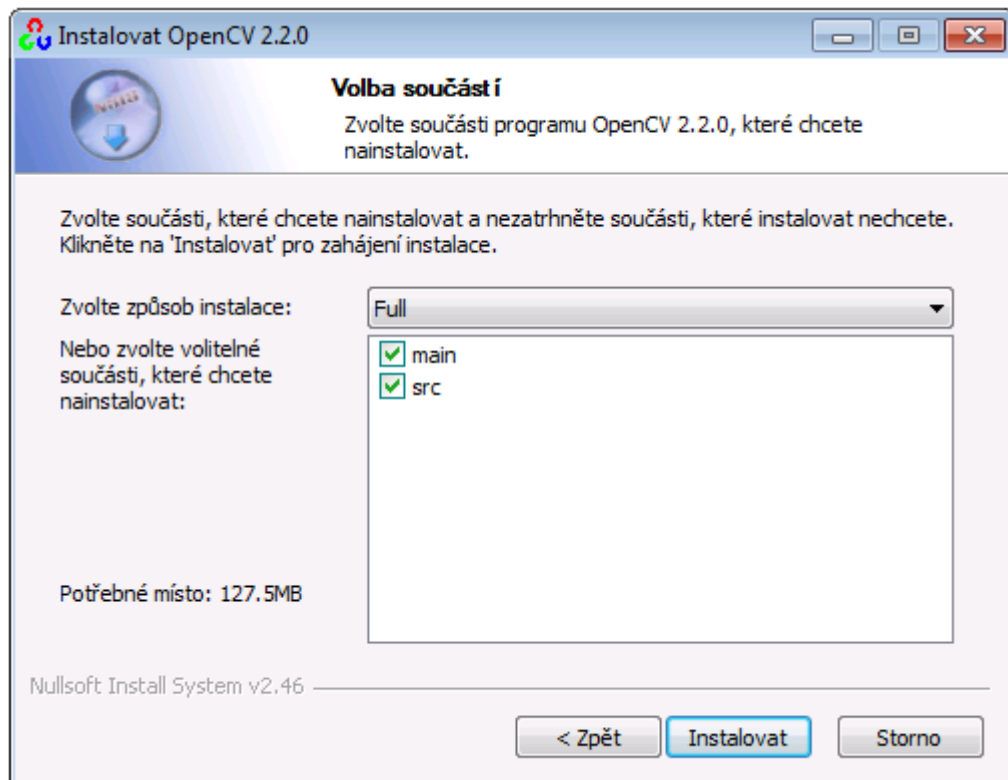
Obr. 15. Instalace OpenCV – Nastavení složky, do které bude OpenCV nainstalováno

Zadejte **cílovou složku**, do které chcete nainstalovat soubory OpenCV (Návod bude navazovat na defaultní nastavení). Poté zvolte tlačítko **Další >**,



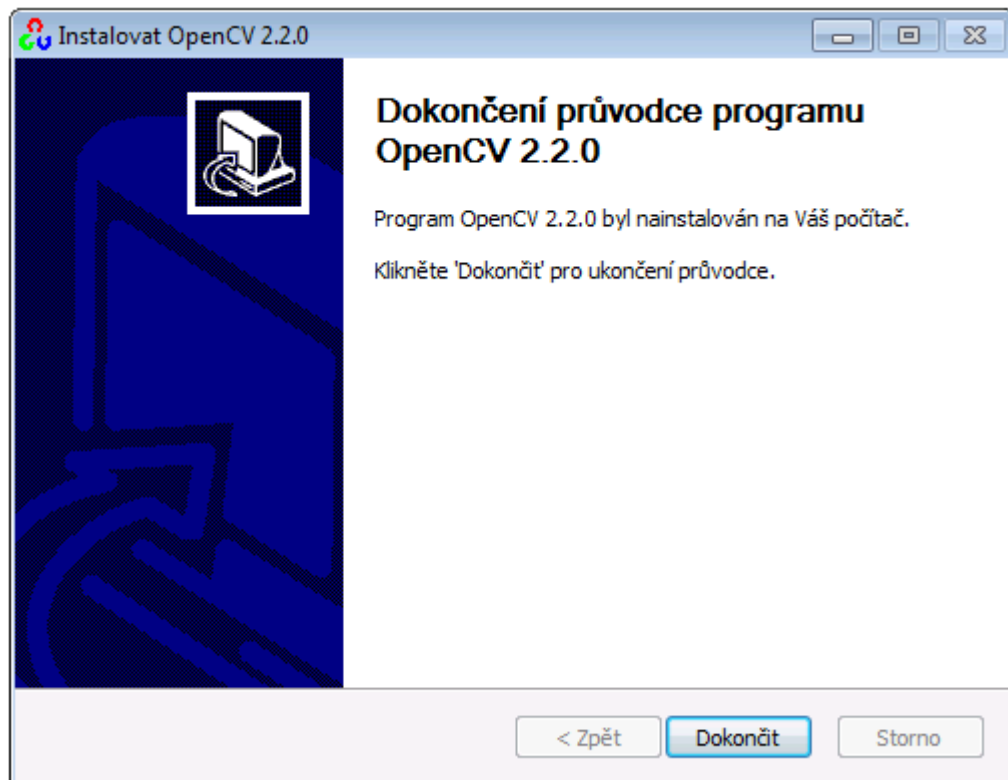
Obr. 16. Instalace OpenCV – Nastavení názvu složky v nabídce Start, ve které najdeme zástupce knihovny OpenCV

Zadejte název složky v nabídce Start, ve které budou uloženi zástupci OpenCV. Pak zvolte tlačítko **Další >**,



Obr. 17. Instalace OpenCV – Volba způsobu instalace

Vyberte způsob instalace **Full** a pak zvolte tlačítko **Instalovat**. Následně se vám knihovna nainstaluje do systému.



Obr. 18. Instalace OpenCV – Dokončení instalace

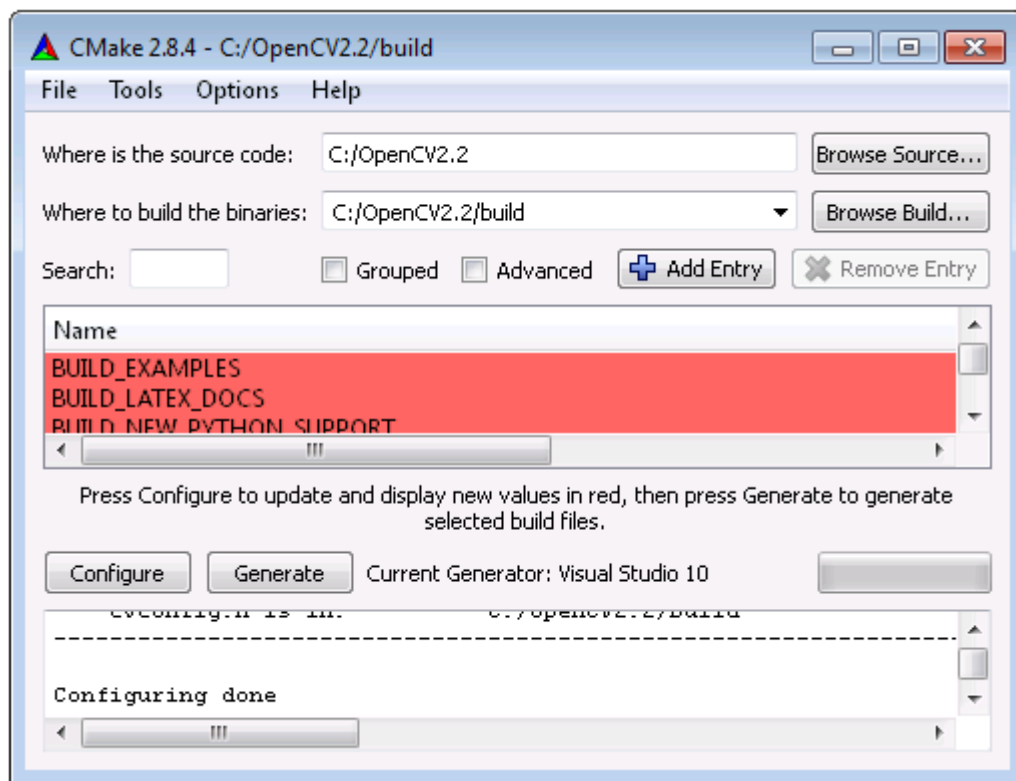
Pro ukončení instalačního procesu zvolte tlačítko **Dokončit**.

4.2 Instalace CMake

CMake si stáhněte ze stránky www.cmake.org (zde). Instalace je stejná jako pro knihovnu OpenCV, takže se můžete řídit pokyny pro instalaci knihovny OpenCV.

4.2.1 Překlad knihovny OpenCV pomocí CMake

Zapněte CMake pomocí **CMake (cmake-gui)** v nabídce Start.

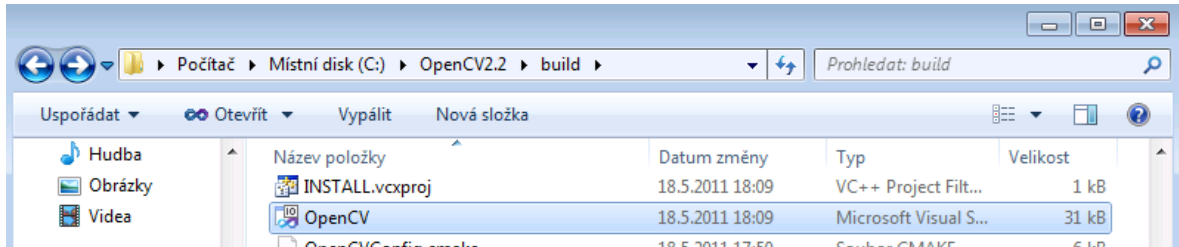


Obr. 19. CMake – Hlavní okno

Vyberte složku, ve které je uložen zdrojový kód (**Where is the source code**) a složku, do níž se nahrají vytvořené soubory (**Where to build the binaries**). Pak stiskněte tlačítko **Configure** a vyberte Visual Studio 2010. Vložte parametry, pokud chcete vytvořit přiřazení k Pythonu, dokumentaci, sdíleným knihovnám, použít TBB, CUDA, knihovny Eigen2 (Poz.: TBB, 64-bit a CUDA mohou být stále ve fázi vývoje, a proto by mohly vyházovat chyby). Dále opakovaně mačkejte **Configure**, dokud nezmizí červené řádky. Nakonec stiskněte **Generate** a překlad je hotov.

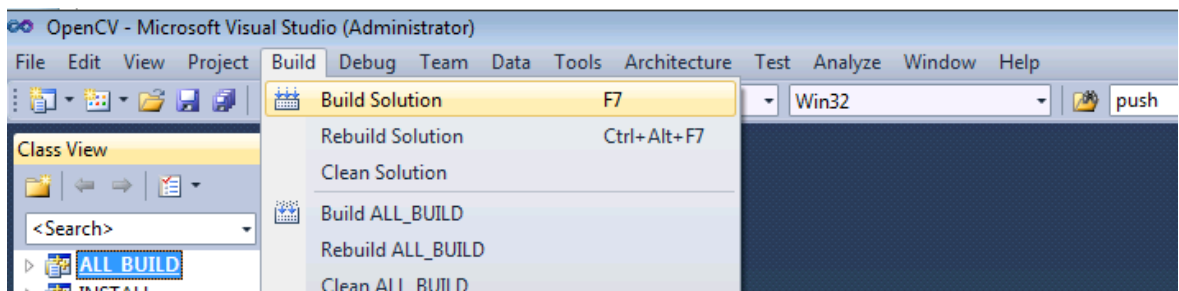
4.3 Nastavení projektu ve Visual Studiu 2010

Otevřete si složku build (defaultně `C:\OpenCV2.2\build`) a najděte soubor `OpenCV.sln`.



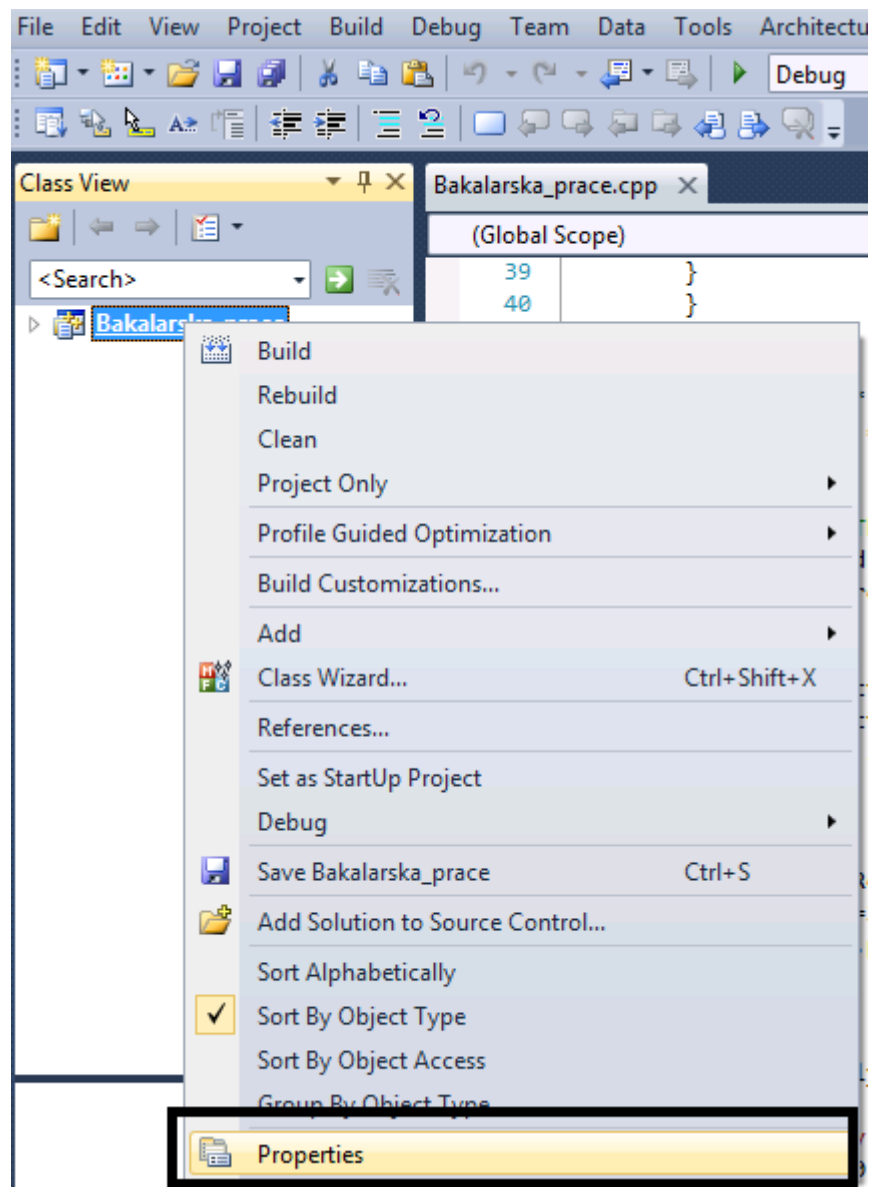
Obr. 20. Cesta se souborem `OpenCV.sln`

Ten otevřete ve Visual Studiu 2010 a zvolte **Build->Build Solution**.



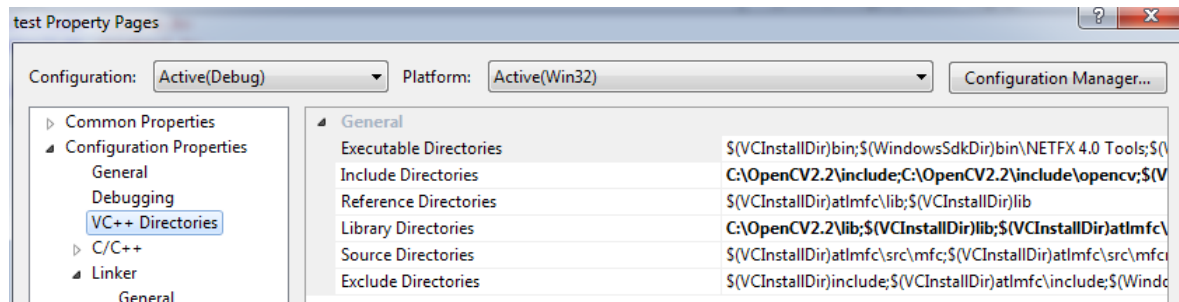
Obr. 21. Visual Studio 2010 – Překlad projektu (*Build Solution*)

Následně ve Visual Studiu 2010 vytvořte C/C++ projekt. Pak klikněte pravým tlačítkem na projekt a vyberte záložku **Properties**. Nyní budu předpokládat, že jste nechali instalační cestu defaultně nastavenou.



Obr. 22. Visual Studio 2010 – Vlastnosti projektu (Properties)

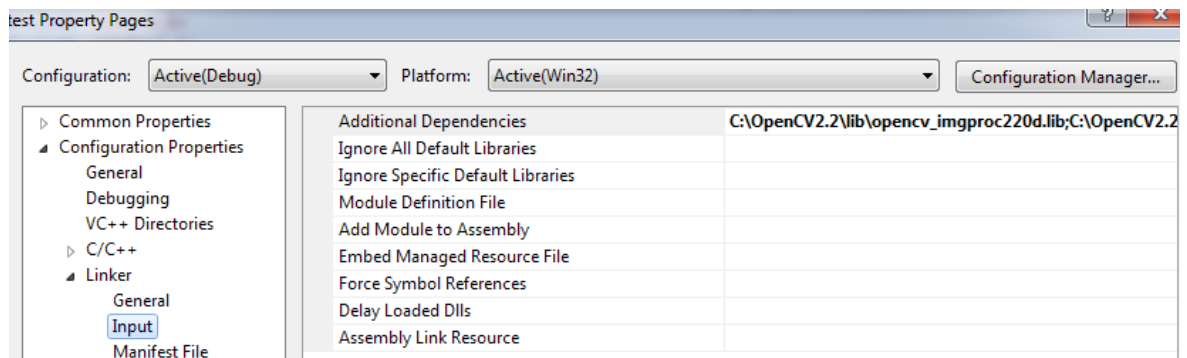
Běžte do záložky **VC++ Directories**, tam přidejte dvě nové Include Directories (`C:\OpenCV2.2\include`; `C:\Program Files\OpenCV2.2\include\opencv`) a jedno Library Directory (`C:\OpenCV2.2\lib`).



Obr. 23. Visual Studio 2010 – Properties → VC++ Directories

Pak běžte do záložky **Linker** a vyberte možnost **Input**. Tady přidejte do Additional Dependencies

(C:\OpenCV2.2\lib\opencv_imgproc220d.lib;C:\OpenCV2.2\lib\opencv_legacy220d.lib;C:\OpenCV2.2\lib\opencv_ml220d.lib;C:\OpenCV2.2\lib\opencv_core220d.lib;C:\OpenCV2.2\lib\opencv_highgui220d.lib;C:\OpenCV2.2\lib\opencv_video220d.lib;).



Obr. 24. Visual Studio 2010 – Properties → Linker → Input

Následně zvolte tlačítko **Ok** a instalace OpenCV do Visual Studia 2010 je hotova.

5 DETEKCE LASEROVÝCH STOP POMOCÍ KNIHOVNY OPENCV

Jako první jsem detekoval laserové ukazovátko, které mělo čočku zaostřenou na tečku. Celkem jsem postupoval ve třech krocích. Nejdříve jsem si ze zdrojového obrázku extrahoval jen odstíny červené, poté jsem si prahováním obrázků rozdělil na černou a bílou barvu a nakonec jsem si našel střed bílých skvrn.



Obr. 25. Fotka laserové tečky, která sloužila pro testování algoritmu pro hledání laserových stop ve tvaru tečky

Dále jsem detekoval laserový kříž dvěma algoritmy – pomocí Houghovy transformace pro detekci přímek a algoritmu Ramer - Douglas - Peucker. U těchto řešení bylo potřeba více kroků. Nejprve jsem opět extrahoval červenou, pak jsem použil prahování (u Houghovy transformace nebylo potřeba), vyhlazení, detekci hran a pak algoritmy pro detekci přímek. Z dat, které mi tyto algoritmy vrátily, jsem si spočítal hrany laserových paprsků a jejich střed.



Obr. 26. Fotka laserového kříže, která sloužila pro testování algoritmu pro hledání laserových stop ve tvaru kříže

5.1 Načtení obrázku

5.1.1 Laserová tečka

K načtení obrázku se používá funkce `cvLoadImage`, která má v prvním případě, kdy hledáme laserovou stopu ve tvaru tečky, pouze jeden parametr a to cestu k obrázku s jeho názvem a příponou. To znamená, že se obrázek načte s jeho hloubkou barev. Načtení obrázku je ještě ošetřeno podmínkou, pokud by se hledaný obrázek nemohl otevřít.

```
IplImage *img = cvLoadImage("laserdot3.jpg");
if (!img) {
    printf("Error: Couldn't open the image file.\n");
    getchar();
    exit(0);
}
```

5.1.2 Laserový kříž

Pro hledání červeného laserového paprsku ve tvaru kříže se musel načítat obrázek pouze v jednom barevném kanálu a 8 bitech, protože při normálním načtení nám červená barva blízko středu zašumí obraz takovým způsobem, že znemožní detekci laserových paprsků.

```
IplImage *img = cvLoadImage("laser5.jpg",CV_8UC1);
if(img==NULL){
    printf("Error: Couldn't open the image file.\n");
    getchar();
    exit(0);
}
```

5.2 Extrakce červené složky

Postupně jsem procházel pixel po pixelu originálního obrázku, extrahoval z něj jas červené a uložil v odstínech šedé do nového obrázku `IplImage* RED`.

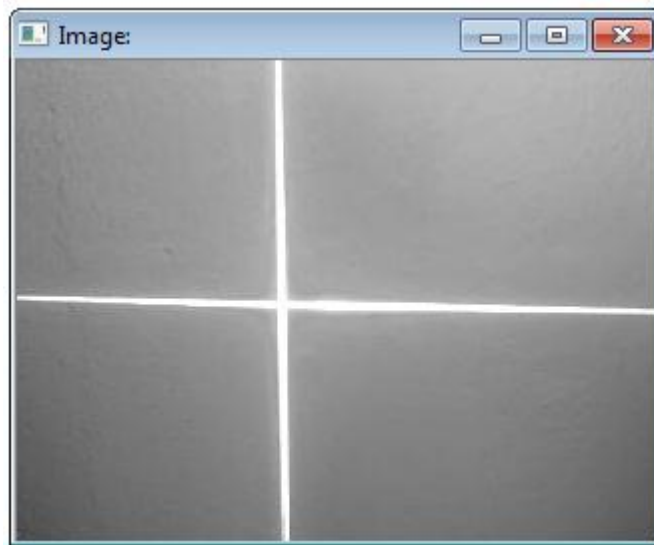
```
int h, w, step, channels, i, j, k;
uchar *data, *trans;
h = img->height;
w = img->width;
step = img->widthStep;
channels = img->nChannels;
data = (uchar *)img->imageData;

IplImage* RED = cvCreateImage(cvGetSize(img), 8, 1);
trans = (uchar *)RED->imageData;
int trans_step;
trans_step = RED->widthStep;

//extracting red color from RGB model
int col = 2;
for(i=0; i<h; i++){
    for(j=0; j<w; j++){
        trans[i*trans_step+j] = data[i*step+j*channels+col];
    }
}
```



Obr. 27. Extrahované odstíny červené
(laserová tečka)

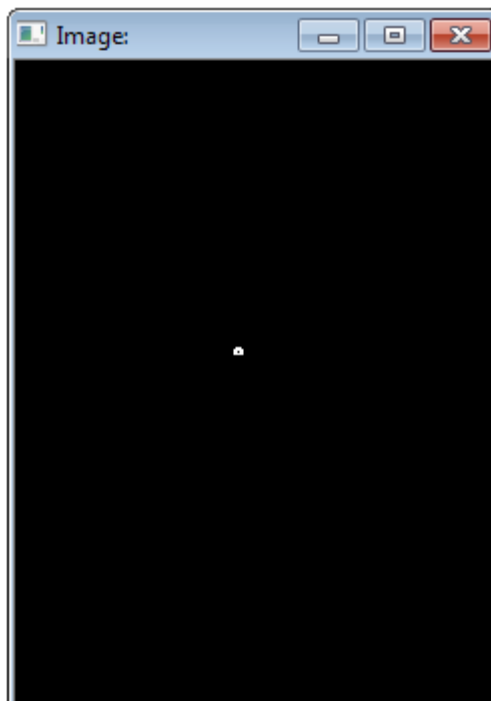


Obr. 28. Extrahované odstíny červené (laserový kříž)

5.3 Prahování

Pro tento testovací příklad jsem použil jen jednoduché prahování, jež všechny pixely mající jas menší, než je zadaný limit, bere jako pozadí (černé) a zbytek jako body zájmu (bílé). V mém případě byla hranice jasu 240, pozadí pak dostalo hodnotu 0 a body zájmu 255. `cvThreshold` jsem využil jen pro detekci laserové tečky a u algoritmu Ramer - Douglas - Peucker.

```
cvThreshold( RGB, RGB, 240, 0, CV_THRESH_TOZERO );
```

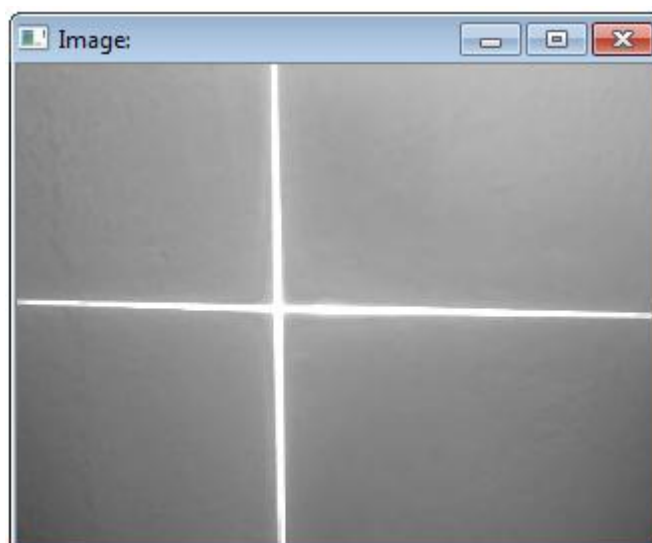


Obr. 29. Prahováním oddělená stopa paprsku (laserová tečka)

5.4 Vyhlazení pro odstranění chromatické aberace

K vyhlazení obrázku laserového paprsku ve tvaru kříže, který měl hrany nerovnoměrně zarovnané, jsem použil průměrné vyhlazení o radiusu 3 pixely.

```
cvSmooth( thresh, thresh, CV_MEDIAN, 3, 3);
```

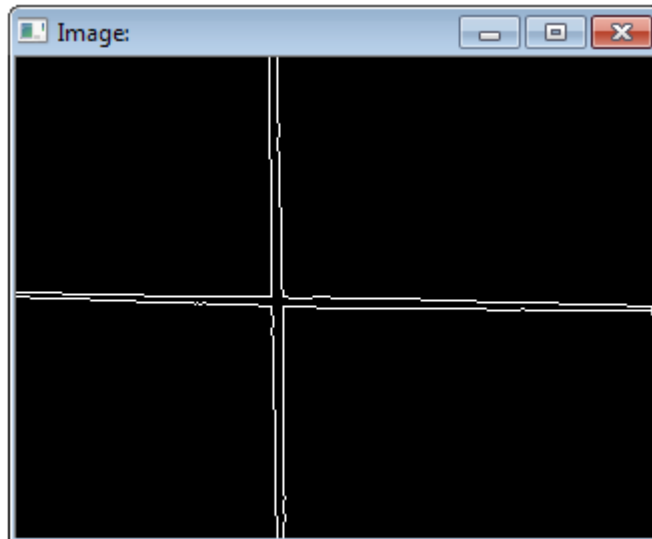


Obr. 30. Vyhlazený obrázek (laserový kříž)

5.5 Detekce hran

Pro detekci hran jsem použil algoritmus Canny, který je implementován v metodě cvCanny. Canny jsem použil jen pro detekci laserové stopy ve tvaru kříže.

```
cvCanny(thresh,thresh,240,0);
```



Obr. 31. Oddělené hrany laserových paprsků algoritmem Canny (laserový kříž)

5.6 Obrysy pro laserovou tečku

Pro zjištění polohy laseru jsem použil dvě funkce. cvGetSpatialMoment spočítala obrys kontury (naší laserové skvrny) a cvGetCentralMoment spočítala obsah obrysu. Vzájemným podělením výsledků jsem získal souřadnice X (šířka) a Y (výška).

```
CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
cvMoments(RED, moments, 1);

double moment10 = cvGetSpatialMoment(moments, 1, 0);
double moment01 = cvGetSpatialMoment(moments, 0, 1);
double area = cvGetCentralMoment(moments, 0, 0);
```

5.7 Houghova transformace pro laserový kříž

Pro detekování přímk pomocí Houghovy transformace jsem využil metodu cvHoughLines2.

```
CvMemStorage* storage = cvCreateMemStorage(0);
CvSeq* lines = 0;
lines = cvHoughLines2(RED,storage,CV_HOUGH_STANDARD,2,0.175,100,0,0)
```

5.8 Algoritmus Ramer - Douglas - Peucker pro laserový kříž

Pro tento algoritmus jsem využil metodu approxPolyDP.

```
int color;
uchar *pData = ( uchar* )thresh->imageData;
std::vector<cv::Point> contour;
std::vector<cv::Point> aprox;
for( i = 0 ; i < thresh->height ; i++ ) {
    for( j = 0 ; j < thresh->width ; j++ ) {
        color = pData[i*thresh->widthStep + j*thresh->nChannels + 2];
        if(color==255){
            contour.push_back(cv::Point2f(j, i));
        }
    }
}
cv::approxPolyDP(Mat(contour), aprox, 120, false);
```

5.9 Výpočet pozice středu laserových stop

5.9.1 Laserová tečka

Podělením detekovaných dat získáme souřadnice X a Y.

```
int posX = moment10/area;
int posY = moment01/area;
```

5.9.2 Laserový kříž – Houghova Transformace

Algoritmus Houghovy transformace vrací přímky ve tvaru ρ a θ , takže pro zjištění souřadnic bodu X a Y je potřeba přepočít z Houghova prostoru na normální (x, y) .

```
for( i = 0; i < MIN(lines->total,4); i++ ){
    float* line = (float*)cvGetSeqElem(lines,i);
    float rho = line[0];
    float theta = line[1];
    CvPoint pt1, pt2;
    double a = cos(theta), b = sin(theta);
    double x0 = a*rho, y0 = b*rho;
    pt1.x = cvRound(x0 + 1000*(-b));
    pt1.y = cvRound(y0 + 1000*(a));
    pt2.x = cvRound(x0 - 1000*(-b));
    pt2.y = cvRound(y0 - 1000*(a));
    pole1x[i]=pt1.x;
    pole1y[i]=pt1.y;
    pole2x[i]=pt2.x;
    pole2y[i]=pt2.y;
}
int stredx = pole1x[1];
int x1 = (pole1x[3]-pole1x[1])/2 + pole1x[1];
int x2 = (pole2x[3]-pole2x[1])/2 + pole2x[1];
int y1 = (pole1y[2]-pole1y[0])/2 + pole1y[0];
int y2 = (pole2y[2]-pole2y[0])/2 + pole2y[0];
int x = (x2 - x1)/2 + x1;
int y = (y2 - y1)/2 + y1;
```

5.9.3 Laserový kříž – Algoritmus Ramer - Douglas – Peucker

Algoritmus uložil nalezené úsečky do pole bodů `aprox`. Pro souřadnici šířky jsem z tohoto pole vybral první a poslední bod a jejich hodnoty x jsem zprůměroval. Pro souřadnici výšky jsem si vzal prostřední dva body a jejich hodnoty y jsem taktéž zprůměroval. Body jsem si musel vzít takto nepravidelně, jelikož algoritmus ukládá úsečky postupně od nejmenší šířky a výšky po největší.

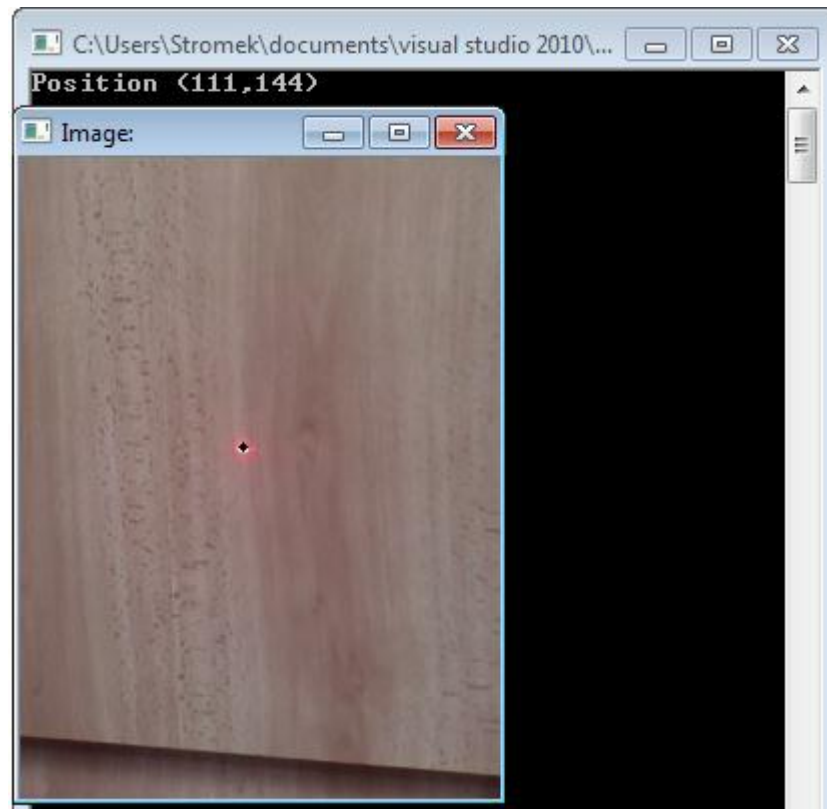
```
int averagex = ((aprox[0].x+aprox[3].x)/2);  
int averagey = ((aprox[1].y+aprox[2].y)/2);
```

5.10 Výsledek

5.10.1 Laserová tečka

Výsledek jsem vykreslil do obrázku černou tečkou a souřadnice vypsal na konzoli. Po stisknutí libovolného tlačítka se okna zavřou, uvolní se paměť a program se ukončí.

```
printf("Position (%d,%d)\n", posX, posY);  
  
cvNamedWindow("Image:", CV_WINDOW_AUTOSIZE);  
cvShowImage("Image:",img);  
  
cvWaitKey();  
  
cvDestroyWindow("Image:");  
  
return 0;
```



Obr. 32. Výstup programu pro detekci laserových skvrn

5.10.2 Houghova transformace

Z vypočítaných dat jsem si do obrázku vykreslil obrysy hran zelenými přímkami. Na konzoli jsem vypsal střed laserového kříže, jenž jsem pak vykreslil do obrázku červenou tečkou. Po stisknutí libovolného tlačítka se okna zavřou, uvolní se paměť a program se ukončí.

```
for( i = 0; i < MIN(lines->total,4); i++ ){
    cvLine( img2, pt1, pt2, CV_RGB(0,255,0), 1, 4 );
}

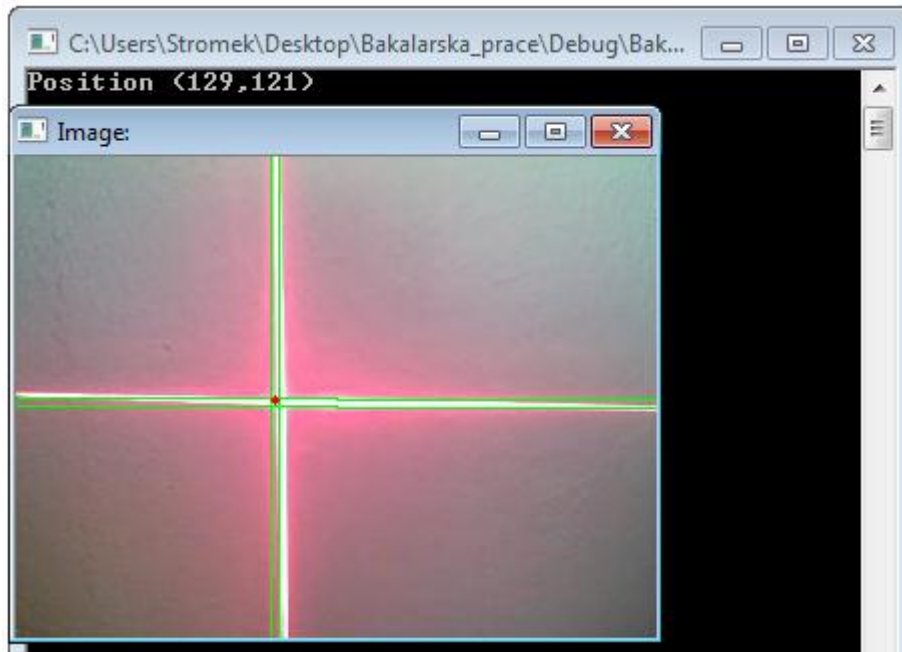
printf("Position (%d,%d)\n", x, y);

cvNamedWindow("Image:", CV_WINDOW_AUTOSIZE);
cvShowImage("Image:",img2);

cvWaitKey();

cvDestroyWindow("Image:");
cvReleaseImage(&img);

return 0;
```



Obr. 33. Výstup programu pro detekci laserového kříže (pomocí Houghovy transformace)

5.10.3 Ramer - Douglas – Peucker

Na konzoli jsem vypsal všechny body, které mi uložila metoda `approxPolyDP` do pole `aprox`. Tyto body jsem pak vykreslil do obrázku zelenou barvou a spojil je úsečkami. Dále jsem si vypsal na konzoli střed laserového kříže a vykreslil jsem ho do obrázku červenou tečkou. Po stisknutí libovolného tlačítka se okna zavřou, uvolní se paměť a program se ukončí.

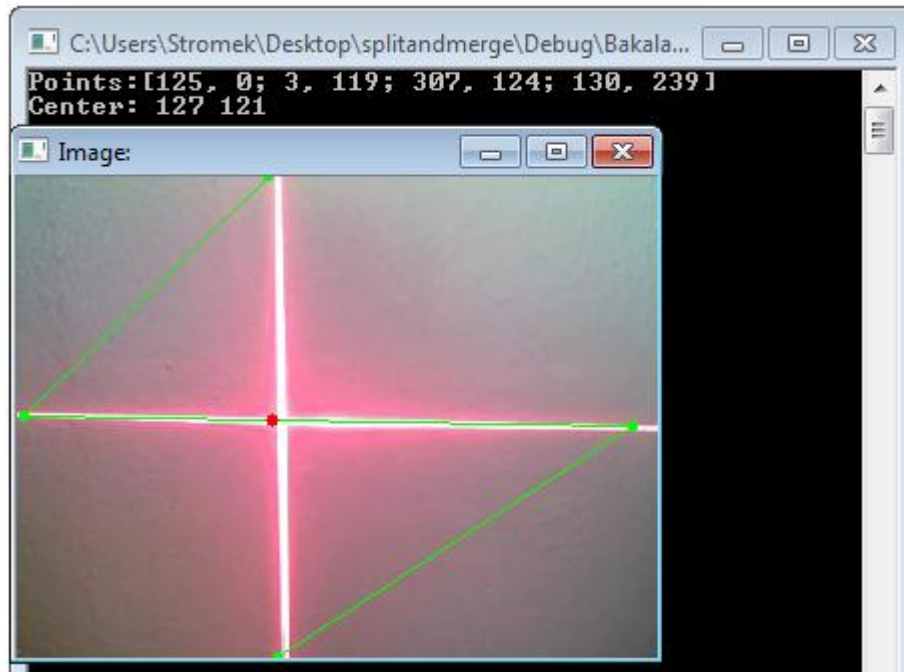
```
cout << "Points:" << aprox << endl;
for(int i = 0; i < aprox.size(); i++){
    cvCircle(img2,aprox[i],1,cvScalar(0,255,0),2,1,0);
}
for(int i = 0; (i+1) < aprox.size(); i++){
    cvLine(img2,aprox[i],aprox[i+1],cvScalar(0,255,0), 1, 8);
}
printf("Center: %d %d",averagex,averagey);

cv::Point center = cv::Point(averagex,averagey);
cvLine(img2,center,center,cvScalar(0,0,255), 5, 8);

cvNamedWindow("Image:", CV_WINDOW_AUTOSIZE);
cvShowImage("Image:",img2);

cvWaitKey();

cvDestroyWindow("Image:");
cvReleaseImage(&img);
```

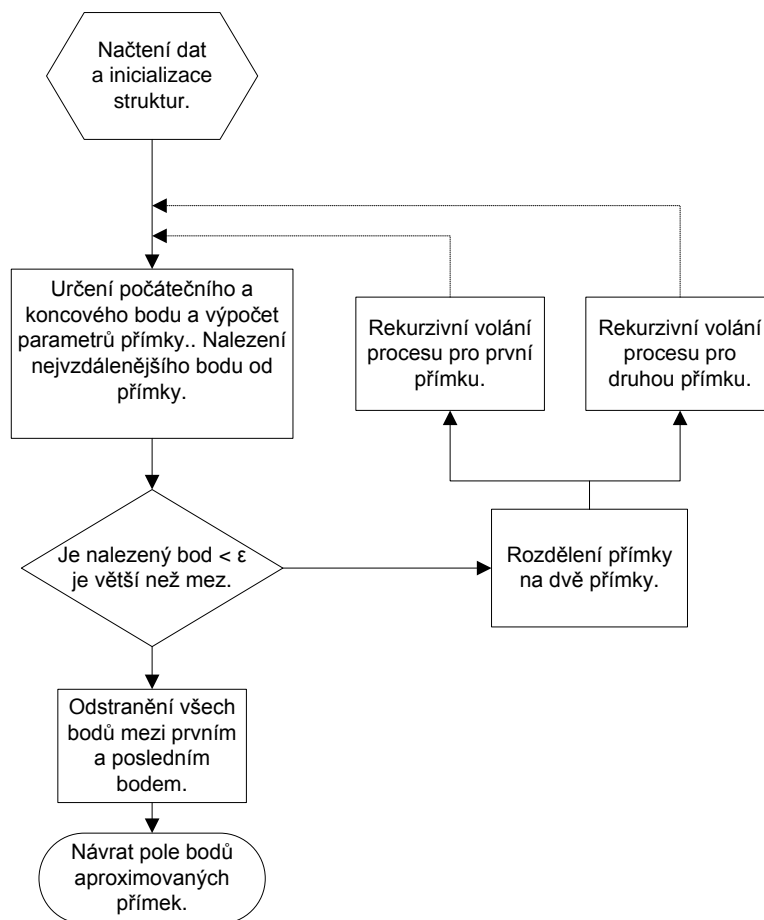


Obr. 34. Výstup programu pro detekci laserového kříže (pomocí algoritmu Ramer - Douglas - Peucker)

6 VLASTNÍ IMPLEMENTACE ALGORITMŮ PRO DETEKCI PŘÍMEK V JAZYKU C

U všech programů je pole bodů předepsané do struktury `PointList` `poleBodu[]`. Jednotlivé body jsou pak jednotlivě uloženy ve struktuře `Point`. Zdrojové kódy jsou přiloženy k příloze P I.

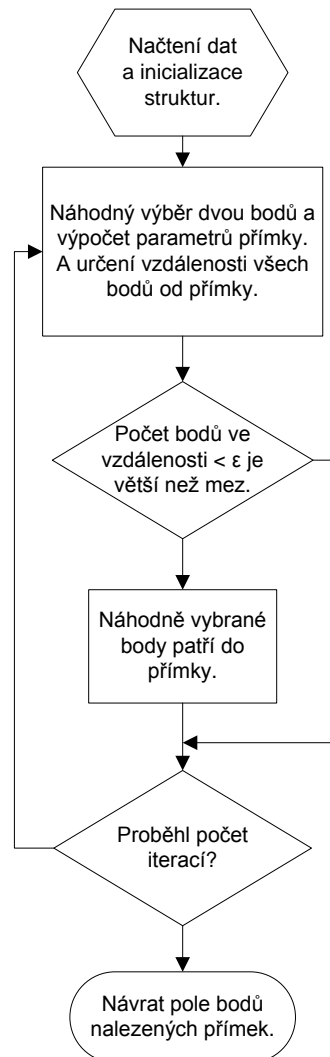
6.1 Ramer - Douglas – Peucker



Obr. 35. Vývojový diagram implementace algoritmu Ramer - Douglas – Peucker

Tento algoritmus je vhodný pro zjednodušení křivky proložením úseček vybranými body. Využit k detekci kříže by sice šel, ale program by nebyl dostatečně robustní.

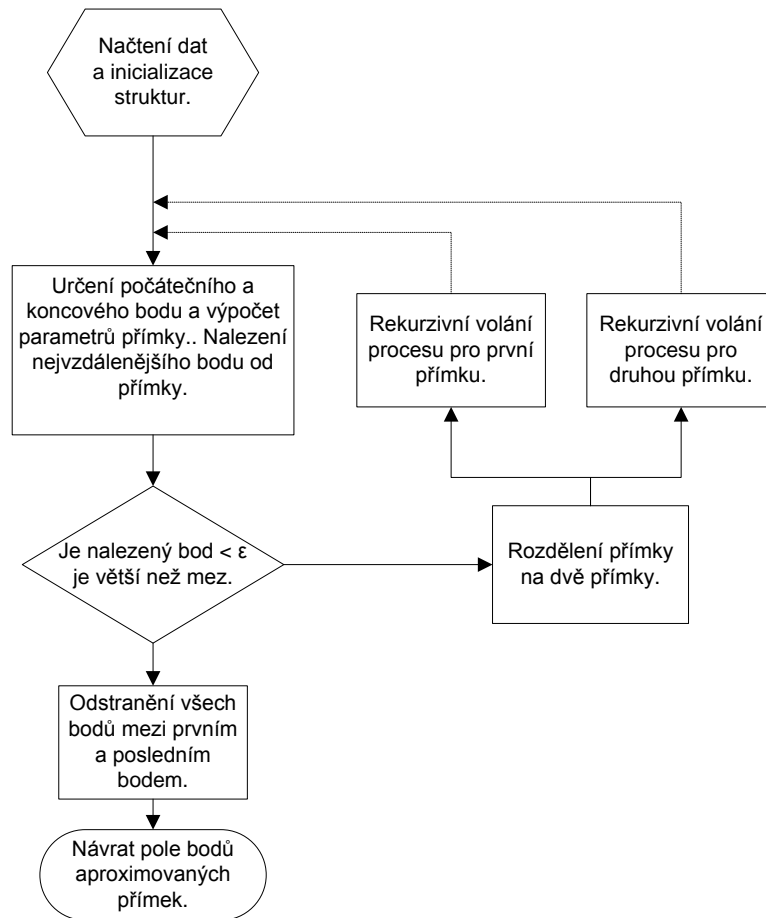
6.2 RANSAC



Obr. 36. Vývojový diagram implementace algoritmu RANSAC

Navržená implementace RANSACu je upravená pro detekci více přímek, takže se výborně hodí pro náš problém detekce laserového kříže pomocí CMUcam3.

6.3 Houghova transformace



Obr. 37. Vývojový diagram implementace algoritmu Houghovy transformace pro detekci přímek

Houghova transformace je sice vhodná pro detekci více přímek, bohužel je ale tento algoritmus příliš matematicky složitý, takže ho pro detekci kříže pomocí CMUcam3 nevyužívám.

ZÁVĚR

Teoretická část práce je zaměřena na problematiku detekce laserových stop a popis algoritmů, jichž se využívá nejčastěji. Velký díl je věnován algoritmům pro detekci přímek, jelikož by bez nich nebylo možné laserový paprsek tvaru kříže detekovat. Dále je popsána knihovna OpenCV, která je v praktické části využita pro testování jednotlivých algoritmů pro detekci přímek a jejich implementaci v problematice detekce laserových stop.

V praktické části jsem se věnoval detekci laserových paprsků. Nejdříve jsem pomocí knihovny OpenCV v jazyku C++ detekoval laserovou tečku z obrázku. Z něj si program extrahoval červenou barvu, použil prahování a pomocí momentů si vypočítal střed paprsku. Následně jsem se věnoval detekci laserového kříže, k čemuž jsem využil algoritmů pro detekci přímek. Z nejpoužívanějších jsem si vybral dva (Houghovu transformaci a Ramer - Douglas - Peucker), které jsem pomocí knihovny OpenCV otestoval na testovacím obrázku s laserovým křížem. Programy použijí metody, které potřebujeme k získání bodů zájmu před tím, než použijí algoritmy pro detekci přímek. Pak detekují přímky, vrátí nám jejich hodnoty a vypočítají střed laserového kříže. Nakonec se obrázek vykreslí s vyznačenými obrysy a středem laserového kříže. Poté jsem naprogramoval v jazyce C tyto dva algoritmy a algoritmus RANSAC, jenž je po úpravě vhodný k řešení našeho problému a není matematicky složitý, takže se dá využít v realtime aplikacích. Tyto programy umí v poli bodů najít podle zadaných podmínek přímky a vrátit jejich hodnoty uživateli, aby s nimi mohl dále pracovat. Algoritmus RANSAC byl na kameře CMUcam3 otestován, čímž bylo zjištěno, že je funkční.

Práce by se dala rozšířit o vlastní implementaci detekce hran, která by se v kameře dala taktéž využít k problému detekce laserových stop nebo pro jiné účely.

ZÁVĚR V ANGLIČTINĚ

The theoretical part is focused on detection of laser blobs and a description of algorithms, which are used most often. The most dedicated is theoretical part to algorithms for detecting lines, because without them can not be cross-shaped laser beam detected. Furthermore, I described the OpenCV library, which I used for the practical testing of algorithms for detecting lines and their implementation in the issue of detection of laser tracks.

The practical part is devoted to the detection of laser beams. At first I used OpenCV library with programming language C++ to detect a laser dot on the image. From it, the program extracted the red color, used threshold, and using moments calculated the center of laser beam. Then I dealt with the detection laser cross, where I used algorithms for detecting lines. I chosed two from the most popular (Hough transform and Ramer - Douglas - Peucker), that I am using OpenCV library to test them on a image with a laser cross. Programs are using the methods that we need to gain points, which we want before the algorithms are used to detecting lines. Then they detect the lines, returns the values and calculate the center of the laser cross. Finally, the image is drawn with marked lines and the center of the laser beam. Then I programmed in C, these two algorithms and the RANSAC algorithm, which is adjusted to a solution of our problem, and is mathematically complex, so it can be used in realtime applications. These programs can find lines in the array under the given conditions and return the values to the user so that they are able to work with them. RANSAC algorithm was tested CMUcam3 camera, which was found to be functional.

The work could be extended to its own implementation of edge detection, which would put the camera also used to solve the problem of laser detection or other purposes.

SEZNAM POUŽITÉ LITERATURY

- [1] Computer vision. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 25 October 2001, last modified on 14 May 2011 [cit. 2011-05-22]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Computer_vision>.
- [2] NGUYEN, Viet, et al. A Comparison of Line Extraction Algorithms using 2D Laser Rangefinder for Indoor Mobile Robotics. *Autonomous Robots* [online]. 2005, Number 2, [cit. 2011-05-22]. Dostupný z WWW: <<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.149.7666&rep=rep1&type=pdf>>.
- [3] SONKA, Milan; HLAVAC, Vaclav; BOYLE, Roger. *Image Processing, Analysis, and Machine Vision*. Second edition. USA : Brooks/Cole publishing company, 1999. 770 s. ISBN 0-534-95393-X.
- [4] *Image Transforms - Hough Transform* [online]. 2003 [cit. 2011-05-22]. Homepages.inf.ed.ac.uk. Dostupné z WWW: <<http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm>>.
- [5] DAVIES, E.R. *Machine Vision*. 3rd edition. USA : Morgan Kaufmann Publishers, 2005. 934 s. ISBN 978-0-12-206093-9.
- [6] Ramer–Douglas–Peucker algorithm. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 24 June 2008, last modified on 20 May 2011 [cit. 2011-05-22]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm>.
- [7] CHUM, Ondřej. *Two-View Geometry Estimation by Random Sample and Consensus*. Praha, 2005. 90 s. Dizertační práce. Czech Technical University in Prague. Dostupné z WWW: <<http://cmp.felk.cvut.cz/~chum/Teze/Chum-PhD.pdf>>.
- [8] RANSAC. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 21 October 2004, last modified on 23 April 2011 [cit. 2011-05-22]. Dostupné z WWW: <<http://en.wikipedia.org/wiki/RANSAC>>.
- [9] BRADSKI, Gary; KAEHLER, Adrian. *Learning OpenCV* [online]. First edition. USA : O'Reilly Media, 2008 [cit. 2011-05-22]. Dostupné z WWW: <<http://www.cse.iitk.ac.in/users/vision/dipakmj/papers/OReilly%20Learning%20OpenCV.pdf>>. ISBN 978-0-596-51613-0.

- [10] PRATA, Stephen. *Mistrovství v C++*. 2. aktualizované vydání. Brno : Computer press, 2004. 1006 s. ISBN 80-251-0098-7.
- [11] SNYDER, Wesley E.; QI, Hairong. *Machine Vision*. Cambridge : Cambridge University Press, 2004. 433 s. ISBN 0-521-83046-X.
- [12] Smoothing an Image. In *Image Processing : Contrasting and Filtering* [online]. [s.l.] : [s.n.], 2007 [cit. 2011-05-28]. Dostupné z WWW: <http://idlastro.gsfc.nasa.gov/idl_html_help/Smoothing_an_Image.html>.
- [13] Prahování. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 7. 2. 2006, last modified on 12. 1. 2011 [cit. 2011-05-28]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Prahov%C3%A1n%C3%AD>
- [14] Image Smoothing. In *Image Registration and Fusion Systems* [online]. [s.l.] : [s.n.], 2010 [cit. 2011-05-28]. Dostupné z WWW: <<http://www.imgfsr.com/index.html>>.
- [15] Edge detection. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 30 September 2003 , last modified on 24 May 2011 [cit. 2011-05-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Edge_detection>.
- [16] Canny Edge Detection. In *Canny Edge Detection* [online]. [s.l.] : [s.n.], March 23, 2009 [cit. 2011-05-28]. Dostupné z WWW: <http://www.cvmt.dk/education/teaching/f09/VGIS8/AIP/canny_09gr820.pdf>.
- [17] Canny edge detector. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 19 February 2004 , last modified on 28 March 2011 [cit. 2011-05-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Canny_edge_detector>.
- [18] *Basic Image Processing Demos* [online]. 6 April 1996 [cit. 2011-05-28]. Canny edge detector demos. Dostupné z WWW: <<http://robotics.eecs.berkeley.edu/~sastry/ee20/cademo.html>>.
- [19] Hough transform. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 18 January 2004, last modified on 3 April 2011 [cit. 2011-05-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Hough_transform>.

- [20] ANTOLOVIC, Danko. *Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection* [online]. [s.l.], 200*. 11 s. Referát. Indiana University. Dostupné z WWW: <[Review of the Hough Transform Method, With an Implementation of the Fast Hough Variant for Line Detection](#)>.
- [21] Color model. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 19 December 2002, last modified on 14 March 2011 [cit. 2011-05-28]. Dostupné z WWW: <http://en.wikipedia.org/wiki/Color_model>.
- [22] RGB. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 7. 12. 2004, last modified on 2. 5. 2011 [cit. 2011-05-28]. Dostupné z WWW: <<http://cs.wikipedia.org/wiki/RGB>>.
- [23] SINHA, Utkarsh . *Aishack* [online]. 200* [cit. 2011-05-28]. Aishack. Dostupné z WWW: <<http://www.aishack.in/>>.
- [24] KARTOUS, Petr. *Hra s ovládáním pomocí gest ruky* [online]. [s.l.], 2008. 31 s. Bakalářská práce. VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ. Dostupné z WWW: <<http://www.fit.vutbr.cz/study/DP/rpfile.php?id=6676>>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

C++	Programovací jazyk
C	Programovací jazyk
RGB	Barevný model - červená, zelená, modrá
N	Počet vstupních bodů
S	Počet získaných segmentů čar (7 v průměru, záleží na algoritmu)
N_f	Posuvná velikost okna pro lineární regresi (9)
N.Trials	Počet pokusů o RANSAC (1000)
NC, NR	Počet sloupců a řádků, případně pro HT akumulátor pole (NC = 401, NR = 671 pro rozlišení $r_{res} = 1$ cm, $\alpha_{res} = 0,9^\circ$)
N1, N2	Počet pokusů a konvergence iterací, respektive, pro EM (N1 = 50, N2 = 200).
HT	Houghova transformace
EM	Expectation – Maximization (Očekávání - Maximalizace)
f	Výstupní obraz
g	Vstupní obraz
T	Práh
m	Obecný moment
p	Stupeň momentu
q	Stupeň momentu
I	Intenzita
r	Vzdálenost mezi přímkou a počátkem
θ	Úhel mezi počátkem a přímkou
\bar{x}	Souřadnice těžiště objektu x
\bar{y}	Souřadnice těžiště objektu y
μ	Centrální moment
RANSAC	RANdom SAMple Consensus (souhlas náhodného vzorku)

CV	Computer Vision
IPP	Integrated Performance Primitives
MLL	Machine Learning Library
HMM	Hidden Markov Models

SEZNAM OBRÁZKŮ

Obr. 1. Aplikace počítačového vidění	11
Obr. 2. Ukázka míchání základních barevných směsí pomocí RGB modelu [22]	14
Obr. 3. Ukázka základního prahování [13]	15
Obr. 4. Ukázka vyhlazování obrazu průměrným vyhlazováním o radiusu 1 pixel [14]	16
Obr. 5. Ukázka použití algoritmu Canny [18]	18
Obr. 6. Ukázka převodu přímky z (x, y) prostoru do Houghova (r, θ) prostoru (v obrázku vyznačeno (ρ , θ)) [20]	20
Obr. 7. Ukázka proložení třech bodů přímkami po 30° a výpočtu jejich vzdálenosti od rovnoběžné přímky procházející počátkem. [19]	20
Obr. 8. Ukázka zobrazení v Houghově prostoru proložení třech	21
Obr. 9. Ukázka zjednodušení aproximace metodou Ramer – Douglas – Peucker [6]	22
Obr. 10. Ukázka proložení přímky	23
Obr. 11. Struktura a obsah OpenCV [9]	25
Obr. 12. Instalace OpenCV – Úvodní okno	27
Obr. 13. Instalace OpenCV – Licenční ujednání	28
Obr. 14. Instalace OpenCV – Možnosti instalace	29
Obr. 15. Instalace OpenCV – Nastavení složky, do které bude OpenCV nainstalováno	30
Obr. 16. Instalace OpenCV – Nastavení názvu složky v nabídce Start, ve které najdeme zástupce knihovny OpenCV	31
Obr. 17. Instalace OpenCV – Volba způsobu instalace	32
Obr. 18. Instalace OpenCV – Dokončení instalace	33
Obr. 19. CMake – Hlavní okno	34
Obr. 20. Cesta se souborem OpenCV.sln	35
Obr. 21. Visual Studio 2010 – Překlad projektu (Build Solution)	35
Obr. 22. Visual Studio 2010 – Vlastnosti projektu (Properties)	36
Obr. 23. Visual Studio 2010 – Properties → VC++ Directories	37
Obr. 24. Visual Studio 2010 – Properties → Linker → Input	37
Obr. 25. Fotka laserové tečky, která sloužila pro testování algoritmu pro hledání laserových stop ve tvaru tečky	38
Obr. 26. Fotka laserového kříže, která sloužila pro testování algoritmu pro hledání laserových stop ve tvaru kříže	39
Obr. 27. Extrahované odstíny červené (laserová tečka)	40

Obr. 28. Extrahované odstíny červené (laserový kříž)	41
Obr. 29. Prahováním oddělená stopa paprsku (laserová tečka)	42
Obr. 30. Vyhlazený obrázek (laserový kříž)	42
Obr. 31. Oddělené hrany laserových paprsků algoritmem Canny (laserový kříž)	43
Obr. 32. Výstup programu pro detekci laserových skvrn	46
Obr. 33. Výstup programu pro detekci laserového kříže (pomocí Houghovy transformace)	47
Obr. 34. Výstup programu pro detekci laserového kříže (pomocí algoritmu Ramer - Douglas - Peucker)	48
Obr. 35. Vývojový diagram implementace algoritmu Ramer - Douglas - Peucker	49
Obr. 36. Vývojový diagram implementace algoritmu RANSAC	50
Obr. 37. Vývojový diagram implementace algoritmu Houghovy transformace pro detekci přímk	51

SEZNAM TABULEK

Tab. 1. Experimentální výsledky algoritmů pro detekci přímek [2].....	12
---	----

SEZNAM PŘÍLOH

PI CD - ROM