TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Algoritmy pro kreslení obrázků pomocí průmyslového robota

Algorithms for drawing pictures using an industrial robot

Bakalářská práce

Autor: Jan Tachecí

Vedoucí práce: Ing. Miroslav Holada, Ph.D.

Konzultant: Ing. Josef Chaloupka

V Liberci dne 12. 5. 2011

Prohlášení

Byl jsem seznámen s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 o právu autorském, zejména § 60 (školní dílo).

Beru na vědomí, že TUL má právo na uzavření licenční smlouvy o užití mé bakalářské práce a prohlašuji, že **s o u h l a s í m** s případným užitím mé bakalářské práce (prodej, zapůjčení apod.).

Jsem si vědom toho, že užít své bakalářské práce či poskytnout licenci k jejímu využití mohu jen se souhlasem TUL, která má právo ode mne požadovat přiměřený příspěvek na úhradu nákladů, vynaložených univerzitou na vytvoření díla (až do jejich skutečné výše).

Bakalářskou práci jsem vypracoval samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce a konzultantem.

Datum

Podpis

Poděkování

Chtěl bych poděkovat Ing. Miroslavu Holadovi, Ph.D. za vstřícný přístup při řešení technických problémů, konzultace k bakalářskému projektu, poskytnutí nezbytných informací a za čas, který mi věnoval. Dále pak panu Ing. Janu Strnadovi, který mi umožnil vstup do učebny robotů a praktické vyzkoušení mého algoritmu. Také bych mu rád poděkoval za výpomoc při konstrukci kreslícího zařízení.

Abstrakt

Předmětem bakalářské práce bylo předzpracování počítačových obrazových dat, aby je bylo možné vykreslit průmyslovým robotem. Na začátku jsou zmíněny některé možné metody vykreslování. Ty jsou dále porovnány a popsány. Stručně jsou zmíněny také teorie detekce hran a popis jednotlivých hranových detektorů. Podrobněji se soustředí na Cannyho hranový detektor a implementaci v programovacím jazyce C++/CLI, který je popsán a vysvětlen. Je zde také stručně popsána reprezentace grafiky v PC.

Dále se práce důkladně zabývá předáváním předzpracovaných obrazových dat (výstup) pro ovládací rozhraní robota. V neposlední řadě jsou vysvětleny všechny funkce vyvinutého programu a stručný návod na ovládání. Nedílnou součástí projektu je též výroba kreslícího zařízení a příprava průmyslového robota. V závěrečné části je představena ukázka vykreslování.

Klíčová slova

Vykreslování robotem, C++/CLI, Canny, hranový detektor, předzpracování obrazu.

Abstract

The subject of this thesis was to prepare visual data for depicting them by an industrial robot. At the beginning there are mentioned some possible methods of depicting. These methods are compared and described thereafter. The theories of edges detection and the description of individual edge detectors are explained briefly also. Then the thesis is focused on Canny's edge detector and on the implementation in programming language C++/CLI. This language is described and explained as well. The representation of graphics in PC is shortly demonstrated further.

The thesis examines thoroughly the transfer of prepared visual data for controlling interface of the robot. Not least all functions of the developed program and the short service instructions are accounted. The integral part of the project is also the producing of the drawing machinery and the preparation of the industrial robot. At the conclusion there is an exhibition of depicting.

Keywords

Depicting by a robot, C++/CLI, Canny, edge detector, preparation of the image.

Obsah

Prohlášení **3**
Poděkování **4**
Abstrakt **5**
Klíčová slova **5**
Abstract **6**
Keywords **6**
Obsah **7**
Seznam obrázků **9**
1 Úvod **10**
2 Teoretický rozbor vykreslování **11**
2.1 Metoda „jehličková tiskárna“ 11
2.2 Metoda „inkoustová tiskárna“ 11
2.3 Metoda „krátké linky“ 11
2.4 Metoda „plotter“ 11
3 Hranové detektory **12**
3.1 Hrana 12
3.2 Gradient funkce 13
3.3 Detektor hran využívající první derivaci 13
3.3.1. Diskrétní 2D konvoluce 13
3.4 Detektor hran využívající druhou derivaci 15
4 Cannyho hranový detektor **17**
4.1 Převod obrazu do odstínů šedi 17
4.2 Vyhlazení Gaussiánovým filtrem 17
4.3 Určení gradientu (derivace) 19
4.4 Ztenčení (detekce lokálních maxim) 19
4.5 Prahování 20
5 Vlastní vývoj programu **21**
5.1 .NET 22
5.2 C++/CLI 22
5.3 Grafika v PC 22
5.3.1. Rastrová grafika 23
5.3.2. Vektorová grafika 24
5.4 Práce s bitmapou v C++/CLI 25
5.5 Reprezentace bitmapy v C++/CLI 27
5.6 Implementace Cannyho detektoru v C++/CLI 27

5.7 Generování dat z programu pro rozhraní robota 29
5.7.1. Princip algoritmu „krátké linky“ 30
5.7.2. Princip algoritmu „Plotter“ 30
5.8 Popis programu 31
6 Vykreslování 32
6.1 Použití programu, postup 33
7 Závěr 34
Seznam použité literatury 36
Příloha A – Ukázka vykresleného obrázku 1
Příloha B – Ukázka vykresleného obrázku 2
Příloha C – Kompaktní disk

Seznam obrázků

Obr.: 1 – Jasové profily (skoková, střechová, liniová) 12
Obr.: 2 – Reálná hrana (zašuměná) 12
Obr.: 3 – Derivace signálu u 1D signálu 13
Obr. 3: Grafické znázornění konvoluce dvou matic 15
Obr. 4: Výsledek 2D konvoluce pomocí Sobelova operátoru 15
Obr. 5: Ukázka druhé derivace skokové změny 16
Obr. 6: Výsledek 2D konvoluce pomocí LoG operátoru 16
Obr. 7: 3D grafické znázornění Gaussova rozložení 18
Obr. 9: Zvětšená hrana získaná derivací (a = hrana, b = ztenčená hrana) 19
Obr. 10: Finální rozložení Windows form 21
Obr. 11: Ukázka zvětšení rastrového obrázku (kruhu) 23
Obr. 12: Příklad zvětšení vektorového obrázku 24
Obr. 13: Základní struktura uzamčeného pole bitmapy 25
Obr. 14: Část algoritmu používající přímý přístup do paměti 26
Obr. 15: Část algoritmu používající metody GDI+ setPixel a getPixel 27
Obr. 16: Vylepšená metoda na převod do odstínu šedi 27
Obr. 17: Rozdělení úhlu do jednotlivých sekcí 29
Obr. 18: Vývojový diagram algoritmu 30
Obr. 19: Kreslící zařízení (efektor) na robotickou ruku 33
Obr. 20 – Ovládací rozhraní robota 33
Obr. 21 –Vlevo použití metody „Krátké linky“. Vpravo „Plotter“ 34

1 Úvod

V dnešní době jde automatizace stále kupředu a čím dál více lidských prací nahrazují stroje a roboti. Důvodů je hned několik – ulehčení výroby, rychlost, finance, lepší využití prostředků. Tyto výhody s sebou nesou i jisté problémy. Stroj, robot nemůže nikdy nahradit 100% člověka. To, co člověk považuje za jednoduchou záležitost, může být pro robota, resp. jeho implementaci velice náročné, ne-li nemožné.

Smyslem celého snažení bylo naučit robota kreslit jednoduché obrazce nebo text. Uplatnění by se našlo např. v automatizovaných průmyslových linkách nebo pro demonstrativní účely a prezentace průmyslových robotů. Software by mohl být také základem pro různé aplikace. Dal by se využít pro vyřezávání součástí z obrazové předlohy, kde by každá hrana znamenala trasu řezání. Obdobný postup by se mohl použít také pro gravírování.

Předpokládalo se, že vyvíjený program bude komunikovat s ovládacím rozhraním robota. Tudíž by se soustředil výhradně na předzpracování obrazových souborů. Vstupními daty je myšlen jakýkoliv obrazový soubor typu BMP nebo JPG a výstupem textový soubor, který je pak dále zpracováván. Jedním z problémů byla optimalizace obrazových souborů, aby je robot dokázal vykreslit. Výsledkem by měla být tzv. monochromatická perokresba. A podle toho byly použity patřičné metody. Jednou z možností bylo použití detektorů hran a jejich implementace.

Z praktického hlediska bylo potřeba průmyslového robota (ABB) předem připravit na vykreslování a obstarat kreslící zařízení.

2 Teoretický rozbor vykreslování

Bylo potřeba řešit, jak bude robot vykreslovat zpracovaná data. Přitom se vycházelo z faktu, že výsledný obraz měl být monochromatický. Bylo navrženo několik způsobů. Jako nejvýhodnější metoda se jevila metoda „plotter“.

2.1 Metoda „jehličková tiskárna“

Nejjednodušší metoda, která je založena na vykreslování jednotlivých bodů. Pixel představuje bod a každý je vykreslen samostatně. Nevýhodou tohoto postupu je příliš velká režie. Robot by po každém nakresleném bodu musel od plochy zvednout své kreslící zařízení, přesunout na jinou pozici a nakreslit další bod. Nepraktické, pomalé a neefektivní.

2.2 Metoda „inkoustová tiskárna“

Jiná metoda spočívá ve vykreslování po řádcích. Kreslící zařízení analogicky vykonává pohyby inkoustové tiskárny ze strany na stranu a potřebná místa by se vykreslovala jako linky. Výsledek by nebyl nejvhodnější.

2.3 Metoda „krátké linky“

Obraz je složen z krátkých linek, které jsou identifikovány v různých úhlech (45°, 90°, 135°, 180°). Tyto linky dohromady dávají celkový obraz. Výsledný postup vykreslování je rychlejší než předchozí metody, ale stále není vhodný.

2.4 Metoda „plotter“

Nejoptimálnější metoda. Obraz je složen z křivek. Robot tyto křivky kreslí kontinuálně. Postup vykreslování lze přirovnat ke klasickému plotteru, kde se pohybuje jen kreslící nástroj. Tato metoda však vyžaduje dokonalejší předzpracování obrazu, nicméně je v tomto projektu použita (viz. 5.7.1).

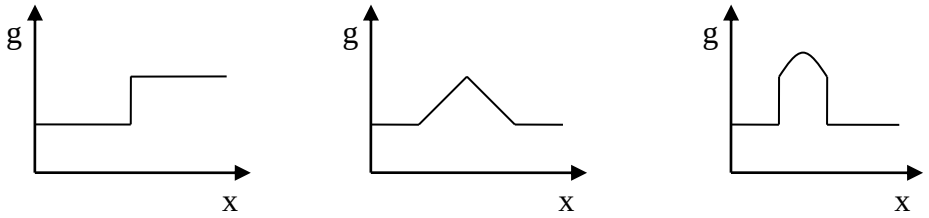
3 Hranové detektory

Protože výsledek musel odpovídat monochromatické perokresbě, bylo nutné aplikovat nějakou metodu, která by vytvořila binární obraz a zároveň zajistila dostatečnou identifikovatelnost objektu v obraze. Pro tyto účely byl zvolen hranový detektor. Hrany jsou důležitou součástí v obraze, určují velikost objektů, tvar a detaily. Hranových detektorů existuje celá řada. Lze je rozdělit do několika skupin:

- Detektory využívající první derivaci
- Detektory využívající druhou derivaci
- Detektory aproximující obrazovou funkci parametrickým modelem

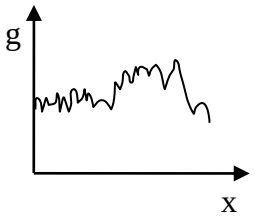
3.1 Hrana

Hrana, resp. hranový bod je místo, které odpovídá náhlé jasové změně. Může být nalezena na hranici objektů, rozhraní světla a stínu nebo v místech trojrozměrných hran objektů. Existuje několik teoretických typů hran. Rozlišují se podle jasových profilů.



Obr.: 1 – Jasové profily (skoková, střechová, liniová)

Ve skutečných podmínkách však hrany nebývají jednoznačně detekovatelné – jsou zašuměné. Obraz je tedy vhodné zpracovat ještě před hledáním hran.



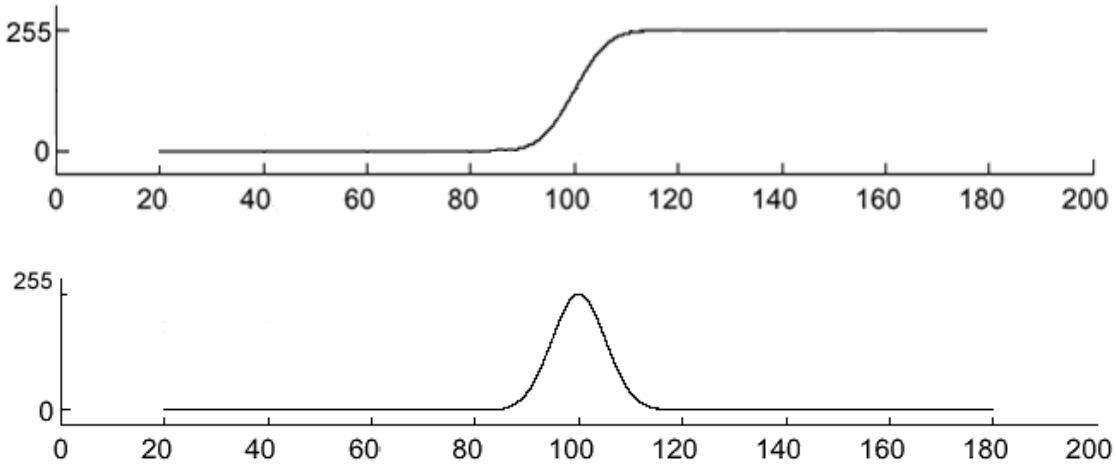
Obr.: 2 – Reálná hrana (zašuměná)

3.2 Gradient funkce

Gradient je diferenciální operátor, jehož výsledkem je vektor hodnot. U 1D signálů je to jen obyčejná derivace v bodě, tedy velikost změny. U 2D signálů je výsledkem vektor 2 hodnot – velikost síly gradientu a směr gradientu.

3.3 Detektor hran využívající první derivaci

Tento detektor je založen na hledání maxim prvních derivací. Využívá funkce, které aproximují první derivaci a snaží se náhlou změnu jasu detekovat. Derivace v bodě vyjadřuje, jak rychle se mění průběh funkce v daném bodě. Toho se využívá právě k detekování hran. Před samotným derivováním je potřeba ještě vyhladit průběh vyhlazovací funkcí.



Obr.: 3 – Derivace signálu u 1D signálu

U 2D signálů se pro detekci hran používá výpočet gradientu (využití parciálních derivací) každého bodu. Hranové detektory se zpravidla liší tím, jaké body použijí pro výpočet gradientu. Ty jsou určeny tzv. konvolučním jádrem.

3.3.1. Diskrétní 2D konvoluce

Konvolucí se rozumí operace zpracovávající 2 funkce. V tomto případě konvoluční jádro a zpracovávaný obraz. Princip počítání je v průchodu konvolučního jádra obrazem.

V případě gradientních operátorů (aproximující jednosměrnou derivaci) prochází ve více směrech. Nicméně postačí jen rovnoběžně s osou x a poté rov-

noběžně s osou y, protože podle Pythagorovy věty je možné velikost síly gradientu vypočítat dle vztahu:

$$|\nabla f(x,y)| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$ *(Vzorec 1: Velikost síly gradientu)*

Pro každý bod je pomocí konvolučního jádra počítán středový bod (viz obr. 3). Je užitečné vědět i úhel hrany – ten je kolmý na úhel gradientu:

$$\theta(x,y) = \arctan\frac{\left(\frac{\partial f}{\partial y}\right)}{\left(\frac{\partial f}{\partial x}\right)}$$ *(Vzorec 2: Směr gradientu)*

Z matematického hlediska je diskrétní 2D konvoluce dána vztahem:

$$y[m][n] = h[m][n] \times x[m][n] = \frac{1}{M \cdot N}\sum_{k=0}^{M-1}\left(\sum_{l=0}^{N-1}\left(h[m-k][n-l] \cdot x[k][l]\right)\right)$$

kde x[m][n] je vstupní signál (v našem případě obrazový soubor, tedy matice pixelů) a h[m][n] je matice konvolučního jádra.

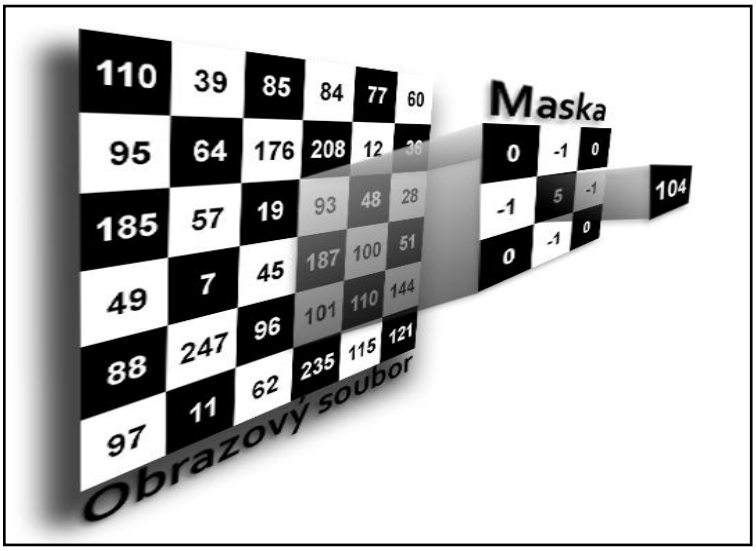
Pro detekci hran se používá několik typů konvolučních masek, které se liší především tím, jaké body používají pro počítání.

Typy konvolučních jader:

- Velikost 2×2
  - Robertsův operátor $x \Rightarrow h = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ $y \Rightarrow h = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ (citlivé na šum)
- Velikost 3×3
  - Prewittův operátor $x \Rightarrow h = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$ $y \Rightarrow h = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
  - Sobelův operátor $x \Rightarrow h = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$ $y \Rightarrow h = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$

- Kirschův operátor $x \Rightarrow h = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}$ $y \Rightarrow h = \begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$

Nutno dodat, že vyobrazená konvoluční jádra jsou jen výčtem z mnoha. Pro lepší detekci je možné jádra lehkou modifikací zvětšit např. na 5×5.

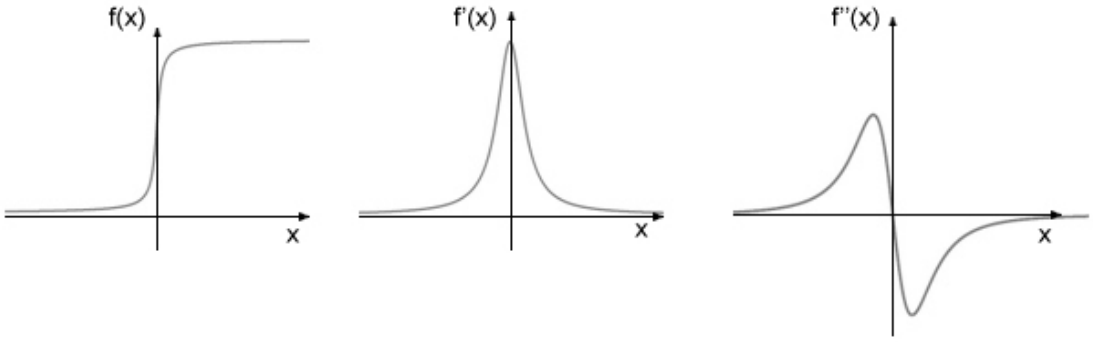


Obr. 3: Grafické znázornění konvoluce dvou matic

Obr. 4: Výsledek 2D konvoluce pomocí Sobelova operátoru

3.4 Detektor hran využívající druhou derivaci

Tento detektor je založen na hledání průchodu nulou. To znamená, že v místě, kde měla první derivace lokální maximum, bude druhá derivace procházet nulou. Zde je citlivost na šum větší než u gradientních detektorů, proto je nutné zahrnout vhodný filtr. Další nevýhodou je dvojitá odezva na hrany, které odpovídají tenkým linkám v obraze.



Obr. 5: Ukázka druhé derivace skokové změny

Podle Maarovy teorie je mnohem spolehlivější detekovat hranu v místě průchodu nuly než na plochém maximu první derivace. Pro filtraci je použit Gaussiánův filtr a pro odhad druhé derivace všesměrový Laplaceův operátor. Oba operátory mohou být spojeny do jediného (vznik LoG – Laplacian of Gaussian). Dvojitým zderivováním Gaussovy funkce dostáváme LoG. Maska je tzv. všesměrová.

Vztah definující hodnoty konvoluční masky:

$$\nabla^2 f(x,y) = \frac{1}{\pi\sigma^4}\left(\frac{x^2+y^2}{2\sigma^4}-1\right)e^{-\frac{x^2+y^2}{2\sigma^2}}$$ *(Vzorec 3: Laplacián Gaussiánu)*

Konvoluční jádro:

$$h = \begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

Obr. 6: Výsledek 2D konvoluce pomocí LoG operátoru

Obecně platí, že čím větší konvoluční maska, tím odolnější na šum v obrázku. Je potřeba zvážit, zda je nutné nasazení většího jádra.

4 Cannyho hranový detektor

Jedná se o optimální hranový detektor, který byl publikován Johnem F. Cannym v roce 1983. Do jisté míry lze říci, že tento detektor byl završením hledání optimálního detektoru hran. J. F. Canny využil doposud nasbírané znalosti a k nalezení detektoru přistoupil jako k optimalizační úloze. Detektor splňuje 3 základní požadavky:

- Jednoznačná detekce (každá hrana by měla být detekována jen jednou)
- Spolehlivost detekce – minimální počet chyb (musí být detekovány všechny hrany a musí být co největší poměr mezi hranami a šumem)
- Přesnost (každá poloha nalezené hrany musí být detekována přesně)

Pro co nejlepší detekci používá několik typů filtrů, které se aplikují současně.

4.1 Převod obrazu do odstínů šedi

V Cannyho detektoru není použit, nicméně v tomto projektu je implementován z důvodu, aby bylo možné pracovat jen s jednou hodnotou jasu každého pixelu.

4.2 Vyhlazení Gaussiánovým filtrem

Nejjednodušší metoda vyhlazení je průměrovací filtr. Používá konvoluční masku, která je tzv. všesměrová (tzn., že není nutné obraz procházet ve více směrech), kde jsou všechny prvky = 1. Součet jasových hodnot se potom vydělí součtem všech prvků, které se v tomto případě rovnají počtu prvků, resp. provede se obyčejný aritmetický průměr.

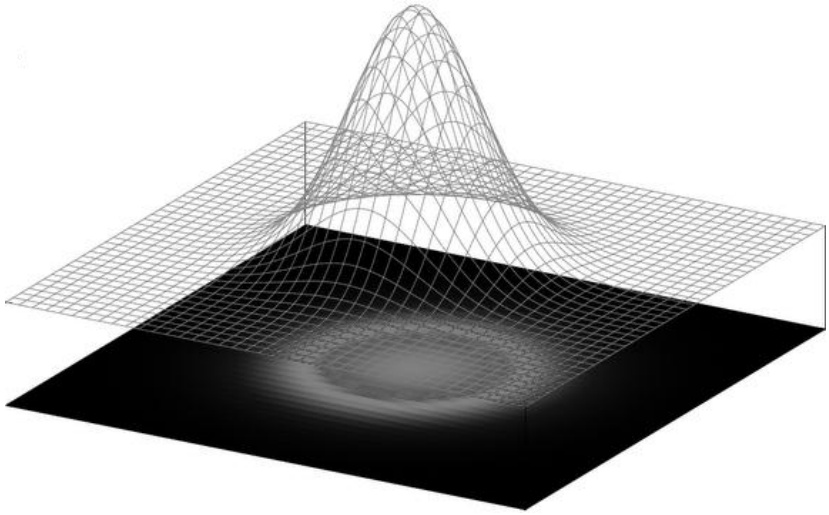
Vyhlazení Gaussiánovým filtrem je obdobné. Využívá vhodné konvoluční masky, kde již nejsou všechny prvky = 1. Gaussiánův filtr se jmenuje proto, že pro výpočet jednotlivých prvků používá Gaussiánův vzorec, resp. jeho rozlože-

ní ve 2D. Výsledkem tohoto filtru je snížení vysokofrekvenčních složek v obraze.

Pro 1D signál je tento vzorec definován takto: $G(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-x^2}{2\sigma^2}}$

Pro 2D signál, je tento vzorec obdobný: $G(x, y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}}$

Jediným parametrem v tomto vzorci je σ, tzv. směrodatná odchylka, která určuje, jak moc se bude obraz rozostřovat, resp. jak velké bude rozostření v pixelech. 3D znázornění Gaussova rozložení je na obrázku (obr. 7). Výsledek konvoluce je pak vážený průměr všech okolních prvků z konvoluční matice.



Obr. 7: 3D grafické znázornění Gaussova rozložení

Výsledné konvoluční jádro (5×5) může vypadat takto: (při použití σ =1.0 a 1.4)

$$h_1 = \frac{1}{620} \cdot \begin{bmatrix} 2 & 8 & 14 & 8 & 2 \\ 8 & 37 & 61 & 37 & 8 \\ 14 & 61 & 100 & 61 & 14 \\ 8 & 37 & 61 & 37 & 8 \\ 2 & 8 & 14 & 8 & 2 \end{bmatrix} \quad h_2 = \frac{1}{1070} \cdot \begin{bmatrix} 13 & 28 & 36 & 28 & 13 \\ 28 & 60 & 77 & 60 & 28 \\ 36 & 77 & 100 & 77 & 36 \\ 28 & 60 & 77 & 60 & 28 \\ 13 & 28 & 36 & 28 & 13 \end{bmatrix}$$

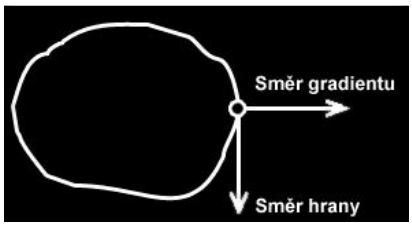
Vzorec 3: Konvoluční masky s rozdílnými směrodatnými odchylkami

4.3 Určení gradientu (derivace)

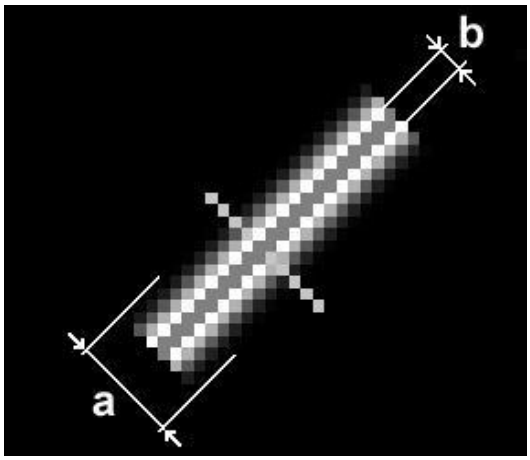
Jedná se o klasickou metodu hledání hran, kdy se použije jeden z gradientních operátorů. Čím větší, tím méně náchylný na šum. V případě Canny detektoru je použit Sobelův operátor, díky menší citlivosti na šum, ale je možné použít i jiný. U 2D signálu je vypočítána velikost změny gradientu (viz vzorec 1) a směr gradientu (viz vzorec 2).

4.4 Ztenčení (detekce lokálních maxim)

Výsledkem předchozí metody je obraz, ve kterém jsou body s největší jasovou změnou vyjádřeny světlejším pixelem. Čím větší jasová změna, tím širší a světlejší místo. Metoda je aplikována ve směru gradientu, tedy kolmo na směr hrany (obr. 8). Ta má za úkol prohledat a vymazat body, které neodpovídají maximu. Na obrázku (obr. 9) je vidět zvětšená hrana (o velikosti *a)* po derivaci proložená ztenčenou (žlutou) hranou (o velikosti *b)*. Modrá linka znázorňuje, v jakém směru se aplikuje detekce lokálního maxima.



Obr. 8: Směr gradientu je kolmý na směr hrany



Obr. 9: Zvětšená hrana získaná derivací (a = hrana, b = ztenčená hrana)

4.5 Prahování

Výstupem předchozí metody je šedotónový obrázek, který však není žádoucí. Závěrečná metoda tedy převede obraz do monochromatické podoby. Exituje více druhů metod prahování. Fungují na principu stanovení prahu, který určuje, která část z obrazu se bude vykreslovat.

- **Globální prahování –** Nad obrázkem se zvolí jediný práh (běžně střední hodnota). A ten se již dále nemění.

$$T > Práh \Rightarrow T = 255$$
$$T < Práh \Rightarrow T = 0 \qquad \textit{(Vzorec: 4)}$$

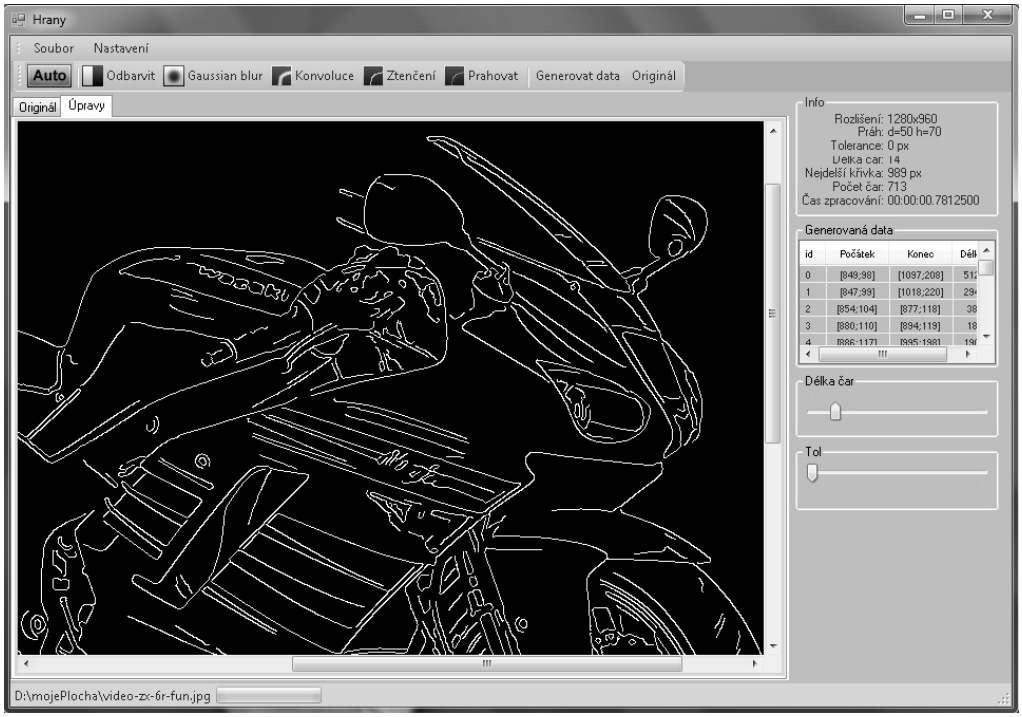
kde *T* je hodnota jasu pixelu.

- **Adaptivní prahování –** Určuje dynamicky hodnotu prahu. Kritérií pro určení prahové hodnoty je několik, proto se rozlišují podle metod na:
    - **Prahování s hysterezí –** na začátku se zvolí 2 prahy, horní a dolní. Všechny pixely přesahující horní práh jasové hodnoty jsou vykresleny. Dále jsou vykresleny pixely, které jsem v intervalu mezi dolním a horním prahem, jestliže jsou nalezeny bezprostředně u pixelů, které přesahují horní práh. V tomto projektu a v Cannyho detektoru je tato metoda použita z důvodu nejvěrnější detekce.
    - **Metoda Chow a kanenko –** Princip tohoto algoritmu spočívá v rozdělení obrazu do překrývajících podobrazů. Na každém podobrazu se vytvoří práh. Ten je určen interpolací jednotlivých podobrazů.
    - **Metoda optimálního prahu –** Algoritmus předpokládá, že obrázek, který se prahuje, obsahuje dvě třídy pixelů (např. popředí a pozadí). Hledá se práh, kde vzdálenosti středních hodnot obou tříd jsou maximální, dochází tak k optimální separaci tříd.
    - **Lokální prahování –** V této metodě se hledá prahová hodnota způsobem, který prohledává okolí pixelu a zjišťuje střední hodnotu. Jiný způsob provedení této metody spočívá v nalezení maximální a minimální hodnoty jasu v okolí bodu.

5 Vlastní vývoj programu

Realizace se prováděla v programovacím jazyce C++, konkrétně revizi C++/CLI (standardizovanou jako ECMA-372), která pracuje s platformou .NET podobně jako C#. Program na zpracování obrazu měl být rychlý, přehledný, což C++/CLI splňuje, protože používá jak managed, tak i unmanaged kód. V programu je pracováno s GDI knihovnou, pro kterou má .NET dostatek funkcí, a tak zjednodušuje práci.

Jednotlivé metody předzpracování obrazu bylo nutné samostatně implementovat v programu, aby je uživatel mohl nezávisle na sobě měnit a tím dosáhl co nejlepšího výsledku. Dále měl program za úkol uživatele vizuálně informovat o stavu a výsledek pak v nějakém tvaru zapisovat do textového souboru. V jakém formátu budou data v souboru, nebylo zpočátku jasné. Prvně se tedy muselo vytvořit rozložení formuláře. Jeho konečná úprava je vidět na ilustraci (viz obr. 10).



Obr. 10: Finální rozložení Windows form

Následovala aplikace algoritmů na zpracování obrazu. Zde došlo k prvním problémům. Metody pro práci s obrazem (getPixel() a setPixel()), které v .NET jsou, byly velmi pomalé a překreslování obrázků trvalo mnohdy desítky vteřin. Muselo se tedy najít nějaké řešení, které by práci s obrazovými soubory zrychlilo.

5.1 .NET

.NET je název pro soubor technologií v softwarových produktech tvořící celou platformu, která je dostupná nejen pro Web, Windows i Pocket PC. Základní komponentou je Microsoft .NET Framework, který je potřeba pro běh aplikací. C++/CLI této komponenty využívá a zprostředkovává tím veškeré knihovny, které .NET nabízí.

5.2 C++/CLI

C++/CLI je jakési rozšíření jazyka C++ (které je obyčejně čistě unmanaged) od společnosti Microsoft, které přidává různé funkce, jazykové konstrukce a jiné věci k tomu, aby bylo možné pracovat s .NET framework. Tím tedy zpřístupňuje 2 knihovny – C++ a .NET. Je nutné říci, že tímto rozšířením je možné používat 2 druhy kódů v jednom projektu – managed i unmanaged. Což vede k nepřehlednosti, ale zároveň se do rukou programátora dostává téměř dokonalý jazyk.

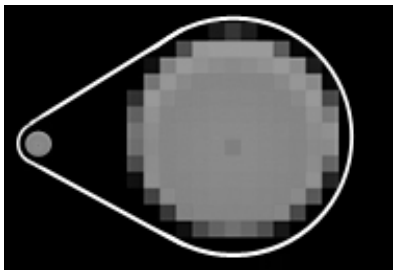
5.3 Grafika v PC

Nejprve je nutné vysvětlit, jakým způsobem se reprezentuje grafika v počítači. Je potřeba myslet na to, že vše kolem nás je spojité. Počítače však pracují diskrétně, tzn. že spíše aproximují spojitý „svět“ pomocí diskrétních signálů. Dochází ke ztrátám, které mohou být zaregistrovány. Nejinak je tomu i v grafice – nelze vytvořit kopii obrazu, který vidíme, naprosto beze ztrát na počítači. Řešením je napodobení reálného obrazu do takové míry, aby ho lidské oko nedokázalo rozlišit od původního. Je potvrzeno, že více než 2^{32} barev již není oko schopné rozeznat, takže v počítači není nutné zobrazovat větším množstvím. Pokud je velikost pixelů dostatečně malá, lidské oko rovněž nepo-

zná rozdíly původního obrazu od reálného. Takovýmto způsobem lze oklamat lidské smysly. V PC se grafika může vyskytovat ve dvou variantách.

5.3.1. Rastrová grafika

V rastrové (bitmapové) grafice je obraz definován maticí bodů (pixelů), kterou si lze představit jako mřížku. Každý pixel má svou polohu a barvu. Pokud se obrázek v rastrové grafice zvětší, pak se zvětší i pixely, ze kterých je složen, tzn. že obrázek bude čtverečkovaný (viz. obr. 11). Kvalita rastrového obrázku je dána barevnou hloubkou (mono, 8 bit, 16 bit, 24 bit) a počtem pixelů, které definují obrázek. Čím větší barevná hloubka a rozlišení, tím kvalitnější je obrázek. Tím ale také nabývá na velikosti. Jednou ze základních jednotek je DPI (Dots per inch) – určuje, kolik pixelů se vejde do jednoho palce. Pro jednoduchost předpokládejme barevný model RGB – tedy aditivní míchání barev. Každý pixel nese informace o intenzitě jednotlivých barev (R – červená, G – zelená, B – modrá).



Obr. 11: Ukázka zvětšení rastrového obrázku (kruhu)

Výhody rastrové grafiky:

- Vysoká realističnost obrazu
- Snadné pořízení obrazu (fotoaparát, skener)
- Široká podpora
- Nezávislost na obsahu obrázku

Nevýhody rastrové grafiky:

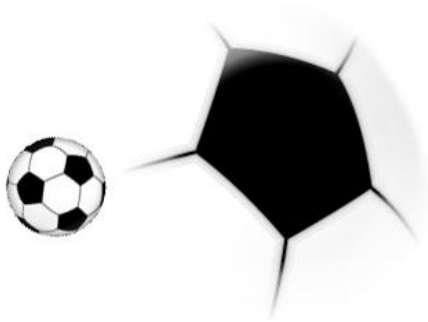
- Paměťová náročnost (každý pixel musí nést informace o svých barvách)
- Při zvětšování dochází ke snížení kvality

Rastrová grafika kvůli velké paměťové náročnosti používá několik druhů komprese. Ta je dále dělena na ztrátovou a bezeztrátovou.

- **Windows bitmap (BMP) –** nepoužívá kompresi. Barevná hloubka 1 - 24 bitů
- **Graphics Interchange Format (GIF) –** používá bezeztrátovou kompresi LZW84. Vhodný na čárovou grafiku. Umožňuje animace. Barevná hloubka je jen do 8 bitů.
- **Portable Network Graphic (PNG) –** podobně jako GIF používá bezeztrátovou kompresi dat. Vylepšení GIFu. Barevná hloubka až 24 bitů. Obsahuje i tzv. alfa kanál, který definuje průhlednost. Nepodporuje animace.
- **JPEG File Interchange Format –** Používá ztrátovou kompresi. Je určen na fotografie a publikování na webu. Barevná hloubka až 24 bitů.

5.3.2. Vektorová grafika

Na rozdíl od rastrové grafiky je vektorová grafika popsána geometrickými útvary (obdélníky, trojúhelníky, kružnice, křivky). Největší výhodou je zvětšování obrazu beze ztráty kvality (viz obr. 12). To je způsobeno tím, že celý obraz je složen z matematicky vypočítaných křivek a objektů, které se mohou snadno přizpůsobovat aktuálnímu zvětšení. Nevýhodou je složitější pořízení obrázku, které spočívá zpravidla v převodu z rastrové grafiky. Další nevýhodou je náročnost na výkon procesoru a paměti, pokud je grafický objekt příliš složitý. Mezi základní obrazové formáty patří PDF, SVG, PS.



Obr. 12: Příklad zvětšení vektorového obrázku

5.4 Práce s bitmapou v C++/CLI

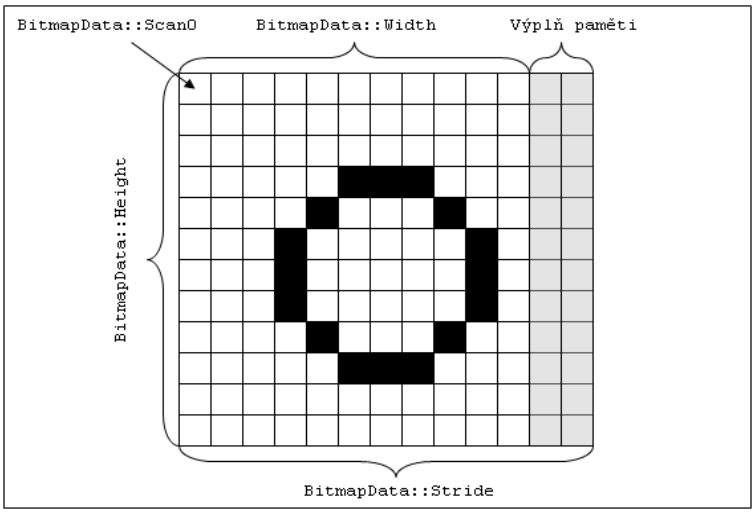
V projektu je pracováno jen s rastrovými obrazy a jak již bylo zmíněno, práce s bitmapou v .NET, resp. GDI+ pomocí metod GetPixel a SetPixel byla velice pomalá. Bylo třeba naprogramovat algoritmus, který pracuje na úrovni pointerů a má přímý přístup k pixelům do paměti. Třída, která na takové úrovni pracuje je BitmapData.

Nejdůležitější vlastnosti třídy BitmapData (viz obr. 11).

- Scan0 – čte nebo nastavuje adresu prvního pixelu v uzamčené bitmapě pomocí metody LockBits.
- Stride – šířka pole bitmapy. Jedná se o násobek šířky obrázku + výplň. (Násobek je určen počtem bytů na jednotlivý pixel)
- PixelFormat – počet bitů, které jsou alokované v paměti pro každý pixel.
- Výplň – přírůstek, který je přičten k Stride, aby byl zaokrouhlený k 4 bytům.

Metoda, která vrací třídu BitmapData, je LockBits. Její parametry jsou:

- Rectangle – velikost bitmapy určené třídou rectangle
- ImageLockMode – mód přístupu do paměti (Read, Write a ReadWrite). Ve svém programu používám mód ReadWrite.
- PixelFormat – počet bitů, které jsou alokované v paměti pro každý pixel.



Obr. 13: Základní struktura uzamčeného pole bitmapy

Pro představu rychlosti používaného algoritmu byl vytvořen testovací program, který pomocí 2 různých metod obrázek převedl do odstínů šedi.

První metoda používala přímý přístup do paměti (viz obr. 14). Druhá metoda používala metody getPixel a setPixel (viz obr. 15). Testovací fotografie měla 24 MPixelů. Čas rychlejší metody (viz obr. 14) byl 9.5 vteřiny. Druhá metoda (viz obr. 15) trvala 73.2 vteřin. Je tedy patrný značný rozdíl. Rychlejší algoritmus je použitý pro další operace.

Oba algoritmy mají své pro i proti. Největší nevýhodou unmanaged metody je snad jen menší čitelnost a délka kódu. U managed metody je největší negativum příliš dlouhá doba provádění.

```
Rectangle r = Rectangle(0,0,pracovni->Width,pracovni->Height);//čtverec
int pf = pracovni->PixelFormat;
BitmapData^ bmpData;
bmpData = pracovni->LockBits(r,ImageLockMode::ReadWrite,pf);
// zamknutí bitů velikosti „r" v módu ReadWrite
int pixelBytes = (Image::GetPixelFormatSize(pracovni->PixelFormat))/8;
// zjištění počtu bytů z obrázku
int size= bmpData->Stride * pracovni->Height, index, Y, stride;
// size = celková velikost pole hodnot RGB
double r,g,b;
stride=bmpData->Stride;
IntPtr ptr = bmpData->Scan0;
// nastavení pointru na začátek paměti alokované paměti
array<Byte>^pixels = gcnew array<Byte>(size);
System::Runtime::InteropServices::Marshal::Copy(ptr,pixels,0,size);
// zkopíruje data z unmanaged pointru do managed pole o velikosti „pixels".
for (int row=0;row<pracovni->Height;row++){
      for (int col=0;col<pracovni->Width;col++){
            index = (row * stride) + (col * pixelBytes);
            // projdem všechny pixely od začátku
            b = 0.114*pixels[index+2];
            g = 0.587*pixels[index+1];
            r = 0.299* pixels[index];
            Y=Math::Round(r+g+b);
            pixels[index + 2] = Y; // index+2 = modrá
            pixels[index + 1] = Y; // index+1 = žlutá
            pixels[index + 0] = Y; // index = červená
      }
}
System::Runtime::InteropServices::Marshal::Copy(pixels, 0, ptr, size);
// zkopíruje pole „pixels" do unmanaged memory pointer
pracovni->UnlockBits(bmpData); // uvolní paměť
```

Obr. 14: Část algoritmu používající přímý přístup do paměti

```
Bitmap^ obrazek =(Bitmap^) Bitmap::FromFile(jmenoSouboru);
// jmenoSouboru je kompletní cesta k bitmapě resp. jpeg obrázku
for (int row=0;row<obrazek->Height;row++){
      for (int col=0;col<pracovni->Width;col++){
            // procházíme jednotlivé pixely
            Color c=obrazek->GetPixel(col,row);
            // do c se načtou RGB informace o pixelu
            int seda = (int)(c.R*0.3 + c.G*0.59+ c.B*0.11);
            // do seda se přiřadí stupeň šedi
            obrazek->SetPixel(col,row,Color::FromArgb(seda,seda,seda));
            // do bitmapy do souřadnic [col,row] se přiřadí stupně šedi
      }
}
```

Obr. 15: Část algoritmu používající metody GDI+ setPixel a getPixel

5.5 Reprezentace bitmapy v C++/CLI

Právě z důvodu nepřehlednosti a délky algoritmu přistupující přímo do paměti byla vytvořena třída, která tuto nevýhodu odstraňuje. Implementuje 2 statické metody *loadBitmap* a *saveBitmap*. Metoda *loadBitmap* má jako parametr odkaz na objekt *Bitmap* a vrací dvoudimenzionální pole typu *unsigned char*, které má reprezentovat obrazovou matici složenou z jasových hodnot pixelů. Ve skutečnosti jsou řádky pole 3× delší než šířka obrázku, protože každý pixel má trojici jasových hodnot (je počítáno s barevným modelem RGB). S tímto polem lze poté provádět jakékoliv operace (převod do odstínu šedi, počítání gradientu atd.) Takto zpracované pole je pak potřeba uložit do bitmapy přes metodu *saveBitmap*. Ta má jako parametr odkaz na bitmapu a nové dvoudimenzionální pole. Rychlé, efektivní a přehledné.

```
unsigned char **polePuvodni = bitmap::LoadBitmap(bitmapa);
// puvodni matice hodnot bitmapy
unsigned char **poleZmenene = c->RGB2GRAY(polePuvodni);
// metoda rgb2gray prevede obraz do odstinu sedi
bitmap::SaveBitmap(bitmapa, poleZmenene);
// tato metoda ulozi zmenene pole do bitmapy
```

Obr. 16: Vylepšená metoda na převod do odstínu šedi

5.6 Implementace Cannyho detektoru v C++/CLI

Jednotlivé kroky Cannyho detektoru byly v programu implementovány jako samostatné metody, které reagují na stisk tlačítek. Tyto metody volají me-

tody třídy *CPoVi* pro zpracování obrazu. Toto oddělení od hlavního programu je z důvodu zapouzdření. Třída *CPoVi* je základní třída pracující s obrazem. Definuje všechny metody zpracování a kvůli rychlejšímu zpracování jsou využity funkce a jazykové konstrukce jazyka C++ namísto .NET Frameworku. V programu je vytvořena automatická detekce hran, která všechny metody aplikuje současně. Hodnoty, které jsou potřeba, načte ze třídy *Konfigurace,* která je statická a její hodnoty jsou nastaveny již před spuštěním programu. V záložce nastavení lze tyto hodnoty měnit.

- **Převod do odstínů šedi**

Jednoduchá metoda, která je zavolána po stisku tlačítka „Odbarvit". V těle metody je zavolána metoda *RGB2GRAY*. Obecně se převádí na odstín šedi tak, že se sečtou jednotlivé poměrné jasové hodnoty všech barev v pixelu a tato hodnota se pak přiřadí všem jasovým hodnotám pixelu. Největší váha je dávána zelené barvě. Výpočet je prováděn podle tohoto vzorce:

$$PixelJas = 0.2989 \cdot R + 0.5870 \cdot G + 0.1140 \cdot B \qquad \textit{(Vzorec 5: odbarvení)}$$

- **Gaussiánské rozostření**

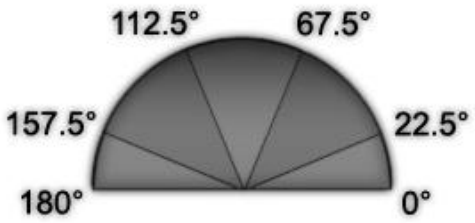
Implicitně je v programu nastavena směrodatná odchylka 1.4 a rozměr jádra 5×5. Tyto hodnoty jsou načteny ze třídy *konfigurace*. Nejprve je spočítáno jádro, resp. jeho hodnoty a poté se toto jádro aplikuje na obraz (konvolucí).

Poznámka: U metod používajících konvoluční jádro je jedno omezení. Konvoluční jádro není aplikováno bezprostředně od kraje obrazu. To je dáno tím, že výsledná hodnota je vypočítávána pro střed jádra a krajní pixely se tedy berou pouze jako informace. Čím větší jádro, tím větší prostor od kraje není zahrnut do výpočtu. Tento nedostatek by se dal odstranit použitím jiného jádra pro každou stranu zvlášť.

- **Určení gradientu**

V programu jsou informace o každém pixelu ukládány do struktury *DataPixel,* konkrétně velikost gradientu a směr gradientu. Tyto údaje jsou počítány pomocí konvoluce a použití Sobelova konvolučního jádra (lze změnit). Velikost

gradientu je počítána dle vzorce (viz vzorec 1) a směr gradientu dle vzorce (viz vzorec 2). Směr hrany je kolmý na směr gradientu a pro zjednodušení je úhel hrany zařazen mezi 4 úhlové oblasti (viz obr. 17). Každý pixel, který je hranový, obdrží jednu z oblastí.



Obr. 17: Rozdělení úhlu do jednotlivých sekcí

- **Ztenčení (hledání lokálního maxima)**

Výsledek předchozí operace není optimální, jelikož jsou hrany mírně rozostřené. Metodou hledání lokálního maxima je docíleno požadovaného ztenčení. Z předchozí operace jsou získána data pro každý pixel a z informace o úhlu lze aplikovat hledání největší jasové hodnoty kolmo na směr hrany. Tuto jasovou hodnotu je potřeba ponechat a všechny ostatní smazat. Vznikne obraz, kde jsou z hran vybrána jen místa s největší jasovou hodnotou.

- **Prahování s hysterezí**

Protože není vhodné vykreslovat i méně významné hrany, je použita metoda na převod do monochromatického formátu. V programu je implementované klasické prahování s hysterezí a pro přehlednost jsou jednotlivé hrany zbarveny do barvy odpovídající jejich úhlu.

5.7 Generování dat z programu pro rozhraní robota

Posledním krokem před samotným vykreslováním je generování dat. Z předzpracovaného obrazu se hrany přepíší do souřadnic jednotlivých pixelů a uloží se do textového souboru (pro přehlednost). Tento soubor je poté načten samotným programem ovládající robota. Podle souřadnic se pak ruka robota pohybuje a kreslí na plochu, na kterou je kalibrována.

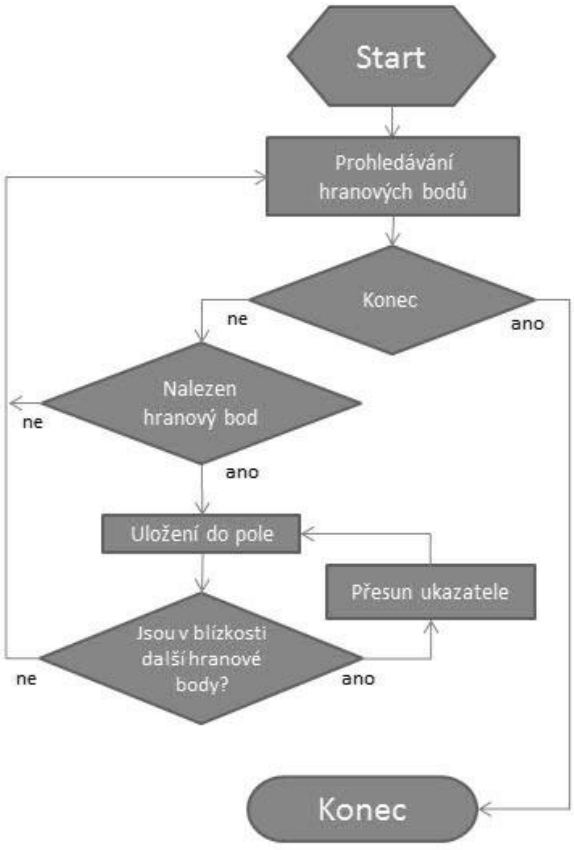
5.7.1. Princip algoritmu „krátké linky“

První algoritmus, který byl implementován, pracuje na principu rozdělení hran na krátké linky (v programu lze nastavit jejich minimální délku). Ty jsou skládány za sebou, a tvoří tak výsledný obraz. Linky se liší podle úhlu, které svírají (4 typy). Nevýhoda je v tom, že linky navzájem na sebe nenavazují, a je tedy potřeba seřazovat. Seřazování však trvá dlouho, a to i u obrázků, které mají méně hran. Přesto, že je obraz seřazen z velmi krátkých linek, výsledný nákres není věrohodný a jsou vidět přechody mezi jednotlivými linkami.

5.7.2. Princip algoritmu „Plotter“

Algoritmus vychází z toho, že vstup je binární obraz, kde jsou všechny hrany reprezentovány linkami o šířce 1 px. Aby se robotická ruka nezvedala příliš často od plochy a kreslila kontinuálně, je algoritmus realizován tak, že jsou procházeny všechny pixely, a když je nalezen hranový pixel, hledá v jeho blízkosti další hranový pixel. Tím je vytvořena křivka, která se skládá z jednotlivých pixelů. Algoritmus pokračuje až do té doby, kdy se již v blízkosti nenachází žádné hranové body. Algoritmus skončí, když jsou nalezeny všechny hranové body. Tyto body jsou ukládány do pole a jednotlivá pole představují křivky (tedy sled bodů, které na sebe vzájemně navazují).

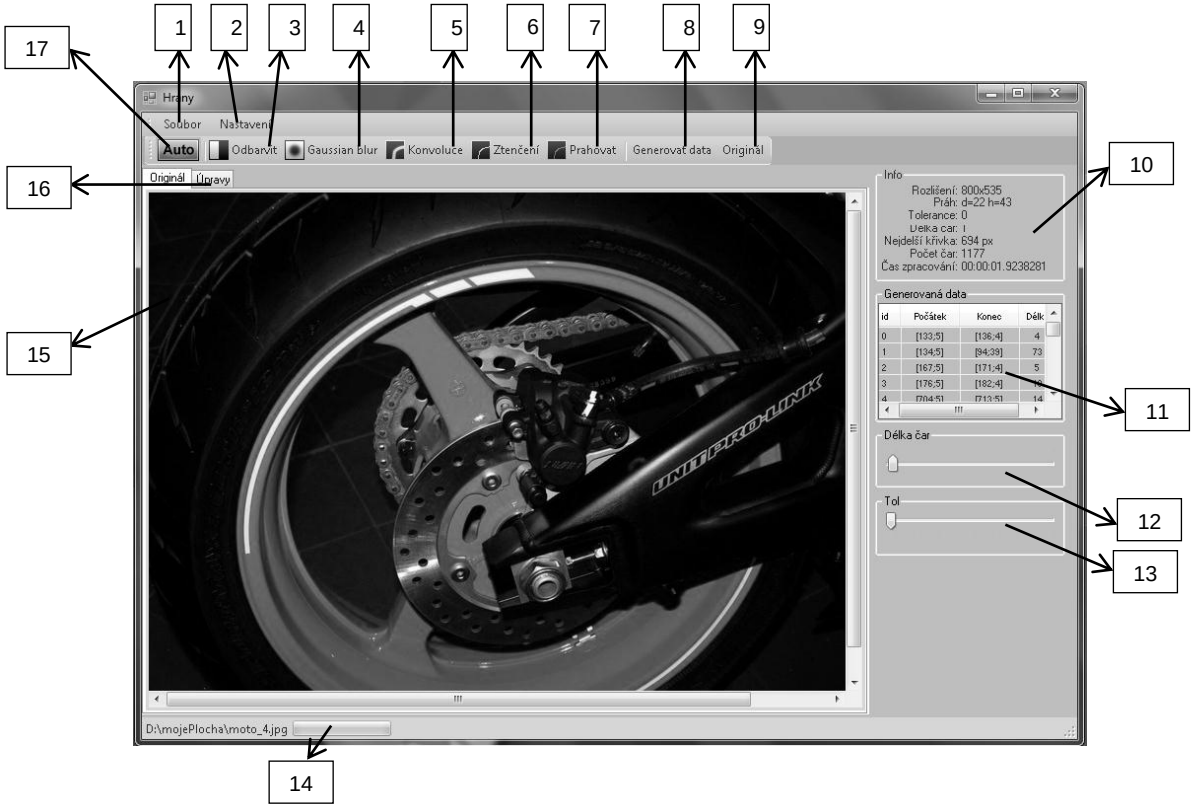
Celý algoritmus je lépe znázorněn na vývojovém diagramu (viz. obr. 19).



Obr. 18: Vývojový diagram algoritmu

5.8 Popis programu

Následující popis odpovídá finální verzi programu.

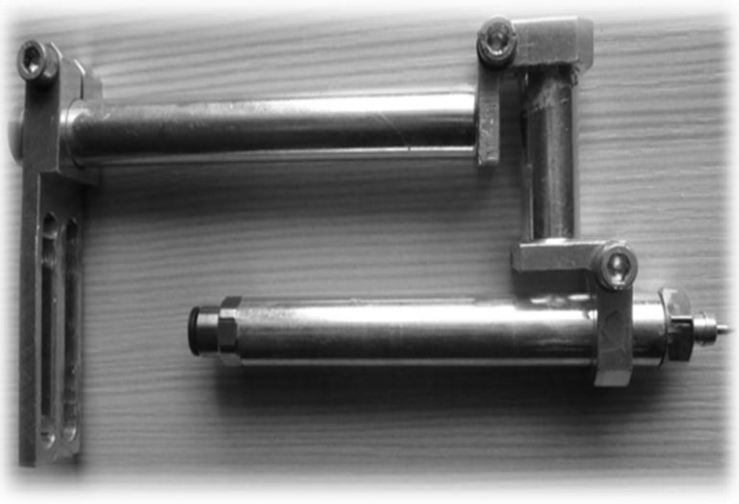


1. Nabídka Soubor – Obsahuje otevření obrazového souboru (*.jpg,*.bmp), otevření textových dat obrazu, uložení obrazu a uložení do textového formátu.
2. Nabídka Nastavení – Formulář se základním nastavení všech parametrů pro Cannyho detektor.
3. Tlačítko odbarvit – Samostatná metoda odbarvení.
4. Tlačítko Gaussianské rozzostření – Zobrazí se dialog s minináhledem, nastavením a ukázkou rozmazání.
5. Tlačítko konvoluce – Zobrazí se dialog s minináhledem, nastavením konvoluční masky (výchozí maska je Sobel).
6. Tlačítko ztenčení – Z předchozí metody se použijí data o hranách a ztenčí se.
7. Tlačítko prahovat – Zobrazí se dialog s minináhledem, nastavením obou prahů (pro prahování s hysterezí) a ukázkou.

8. Tlačítko generovat data – Generuje data po úspěšně provedené detekci pro rozhraní robota.
9. Tlačítko originál – Zobrazí originál obrázku pro další použití.
10. GroupBox Info – Zobrazuje aktuální informace o obrázku, nastavení a výsledku.
11. ListView seznam – Tabulka, zobrazující všechna data, která jsou určena pro robota. Poklepáním na jednotlivé řádky lze zobrazit vybrané hrany.
12. Posuvník délka hran – V závěrečné fázi lze tímto posuvníkem nastavit práh, který určuje, jak dlouhé čáry se mají ještě vykreslovat.
13. Posuvník tolerance – Čím vyšší je práh, tím méněkrát se robot zvedne od kreslicí plochy. Ideál by byl nakreslit celý obrázek jednou čarou. Na druhou stranu se obrázek mírně deformuje. Je třeba zvážit, jak tento posuvník nastavit.
14. Bar – vyobrazuje stav provedení detekce.
15. Hlavní okno – v tomto oknu je zobrazen samotný obrazový soubor.
16. Záložky – Slouží k přepínání mezi originálem a upravovaným obrazem (k porovnávání).
17. Tlačítko Auto – po stisku jsou provedeny všechny potřebné operace k vytvoření textového souboru pro robota.

6 Vykreslování

Vykreslování a zkoušení algoritmu probíhalo na robotech ABB v učebně robotů na Technické univerzitě v Liberci. Bohužel roboti nebyli vybaveni správným efektorem, a proto bylo nutno vyrobit kreslící zařízení (viz obr. 18). To je složené z hliníkových částí a dutého válce, kde je zastrčena tuha z propisky. Výhoda tohoto zařízení je v pružném hrotu, které vyrovnává nerovnosti kreslicí plochy. Pro použití vykreslování byly poskytnuty knihovny ovládající robota, takže bylo jednoduché naprogramovat si své ovládací rozhraní (viz obr. 19).



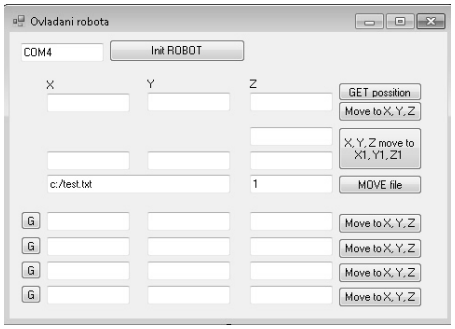
Obr. 19: Kreslící zařízení (efektor) na robotickou ruku

Efektor byl použit u všech nákresů a fungoval výborně. Náplně jsou snadno vyměnitelné a snadno dostupné (z propisek).

6.1 Použití programu, postup

Doporučený postup pro vykreslení obrazu, resp. hran z obrazu:

1. Použije se kvalitní fotografie nebo obraz sejmutý ze skeneru a vloží se do programu.
2. Tlačítkem „Auto“ nebo postupným použití všech metod Cannyho detektoru je vytvořen monochromatický hranový obraz a v menu Soubor se zvolí uložení dat.
3. Dále se spustí ovládací rozhraní robota a zkalibruje se výchozí poloha.
4. Načte se uložený soubor z programu pomocí tlačítka „MOV file“ robot začne vykreslovat.



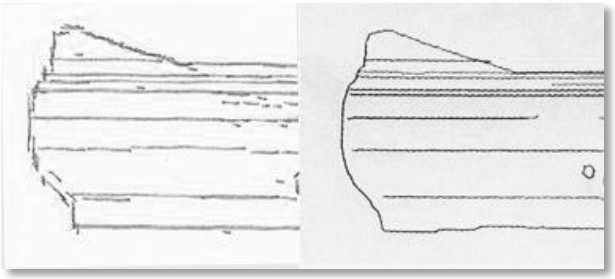
Obr. 20 – Ovládací rozhraní robota

7 Závěr

V průběhu celé práce bylo použito několik způsobu a metod. Nejdelší dobu zabralo právě hledání optimální metody. Vyskytlo se mnoho úskalí a nejednou bylo nutné celý program předělat. Nicméně zadání této práce bylo úspěšně splněno.

Původně bylo hledání hran jednoduché, přes LoG operátor s následným globálním prahováním. Výsledný obraz nebyl špatný, ale vykreslování bylo možné jen metodou „Krátké linky“ (viz. 2.3). Hrany totiž nebyly spojité a ani jednoznačné. Byly široké i několik pixelů, a proto je bylo nutné překreslovat několikrát. Robot tedy musel často zvedat kreslící zařízení od plochy. Částečným řešením tohoto problému bylo nasazení seřazení linek navazujících na sebe. Bohužel linek bylo opravdu velké množství a seřazovaly se struktury dat, proto celý proces trval velmi dlouho. Výsledný produkt tedy nebyl optimální. Ani metoda na zbavení se nežádoucích pixelů a hran, které do obrazu nepatřily, problém neodstranila (hlavně z důvodu nejisté detekce nežádoucích pixelů).

Zásadní a zároveň poslední změnou bylo předělání celého projektu na detekci hran dle Cannyho. Na obrázku (viz obr. 20) je vidět rozdíl mezi současnou a minulou implementací programu. Touto metodou se vyřešilo mnoho chyb a také se celý proces zrychlil. Seřazení bylo mnohem rychlejší. Prakticky šlo jen o hledání navzájem navazujících pixelů.



Obr. 21 –Vlevo použití metody „Krátké linky“. Vpravo „Plotter“.

Největším problémem celé práce byla délka samotného vykreslování. U první implementace byl nejpravděpodobnější důvod příliš častý a dlouhý přesun kreslícího zařízení z místa na místo. Délka přesunu byla mnohem delší než

délka vykreslované linky. Zejména proto se zdál tento algoritmus neefektivní. Naopak u druhé implementace kreslící zařízení nemuselo tolikrát opustit kreslicí plochu, ale vyskytl se jiný problém, který se již dále nepodařilo odstranit. Ovládání robota je totiž koncipováno na mírně odlišné aplikace. Není problém kreslit dlouhé, táhlé čáry. Více miniaturních přesunů je však problémem, protože po každém vykonaném pohybu ovládací rozhraní čeká a dotazuje se, zda již byla dokončena operace, a až poté mohou být vykonávány další akce. Tím pádem je vykreslování pomalé. Mnohem rychlejší by bylo, pokud by mohl program všechny souřadnice nahrát a robot by těmito body jen projel. Bohužel toto se nepodařilo realizovat. Nicméně v budoucnu by se urychlení dalo aplikovat a celý proces vykreslování urychlit.

Seznam použité literatury

[1] Ján Hanák – C++/CLI – Začínáme programovat 1.vydání, Artax a.s. 2009, ISBN 978-80-87017-04-3

[2] BruXyho home page [online]. 1. verze. 2006 [cit. 2010-05-23]. Diskrétní 2D konvoluce. Dostupné z WWW: <http://bruxy.regnet.cz/fel/36ACS/konvoluce.pdf>.

[3] W. POWELL, Robert. Using the LockBits method to access image data [online]. 2003 [cit. 2010-05-23]. Using the LockBits method. Dostupné z WWW: <http://www.bobpowell.net/lockingbits.htm>.

[4] .NET Framework Class Library [online]. 2010 [cit. 2010-05-23]. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms229335%28v=VS.100%29. aspx>.

[5] GENČŮR, Martin. NÁSTROJ NA ZPRACOVÁNÍ FOTOGRAFOVANÉHO TEXTU [online]. [s.l.], 2007. 29 s. Diplomová práce. Vysoké učení technické v Brně. Dostupné z WWW: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=4521&y=0>.

[6] MIKŠÍK, Ondřej. *Praktické využití metod digitálního zpracování obrazu* [online]. [s.l.], 2007. 43 s. Středoškolská odborná činnost. Gymnázium Kroměřiž. Dostupné z WWW: <http://soc.nidm.cz/data/2007/01-2.pdf>.

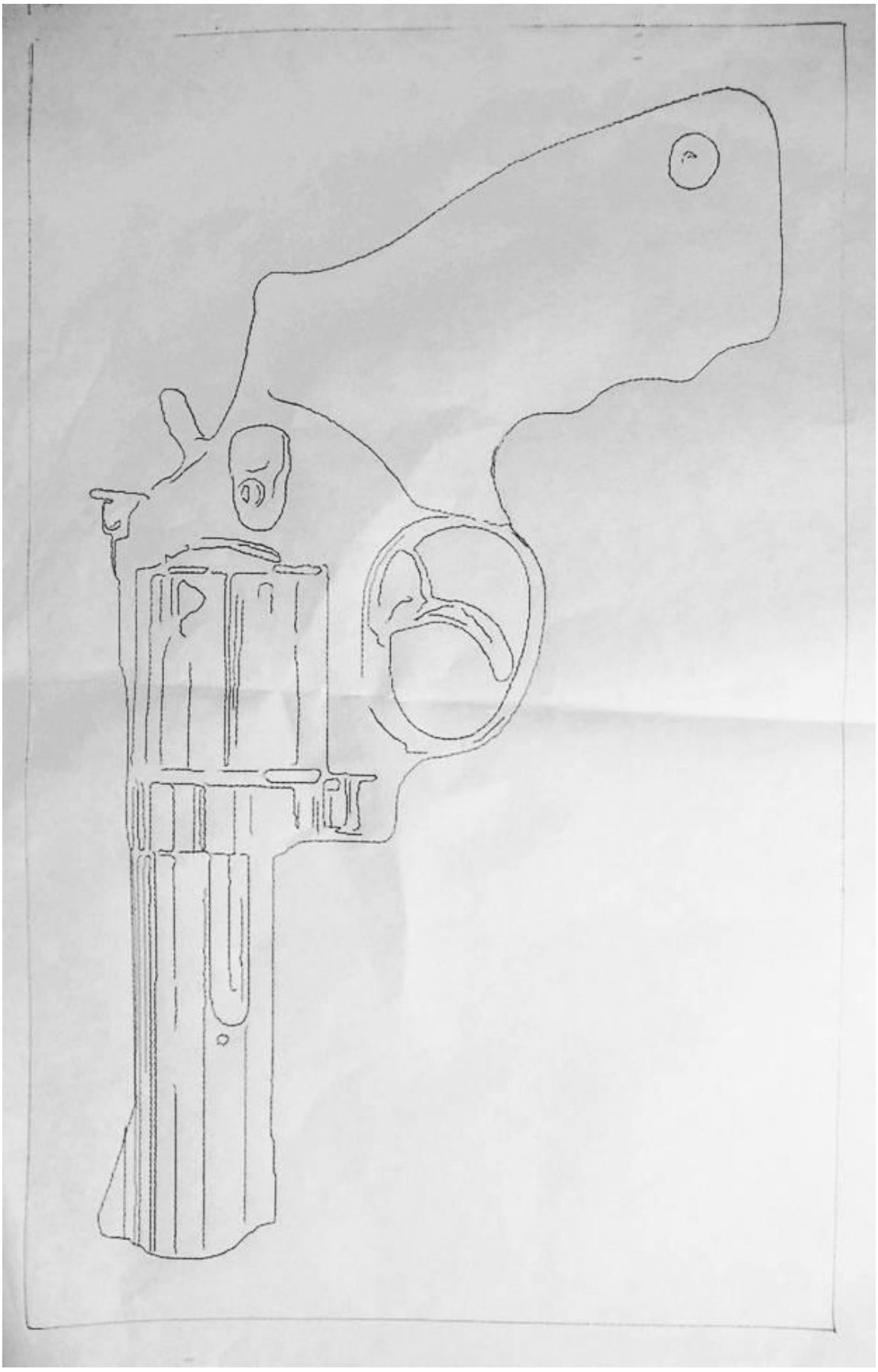
[7] SOJKA, Eduard. *Digitální zpracování obrazu* [online]. [s.l.], 2000. 133 s. Diplomová práce. Vysoká škola báňská - Technická univerzita Ostrava . Dostupné z WWW: <http://mrl.cs.vsb.cz/people/sojka/digitalni_zpracovani_obrazu.pdf>.

[8] GREEN, Bill. *Canny Edge Detection Tutorial* [online]. 2002 [cit. 2011-04-17]. Canny Edge Detection Tutorial. Dostupné z WWW: <http://www.pages.drexel.edu/~weg22/can_tut.html>.

[9] LEE, Tzu-Heng Henry. *Edge Detection Analysis* [online]. [s.l.], 2007. 25 s. Diplomová práce. National Taiwan University. Dostupné z WWW: <http://disp.ee.ntu.edu.tw/henry/edge_detection.pdf>.

[10] *Moserware: Computing Your Skill* [online]. THURSDAY, MARCH 18, 2010 [cit. 2011-05-12]. Computing Your Skill. Dostupné z WWW: <http://www.moserware.com/2010/03/computing-your-skill.html>.

Příloha A – Ukázka vykresleného obrázku 1



Příloha B – Ukázka vykresleného obrázku 2

Příloha C –Kompaktní disk

Obsahuje elektronickou formu práce, program s testovacími obrázky a zdrojový kód.