VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

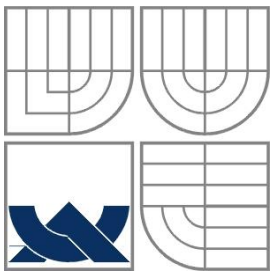
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE KŘIVEK V OBRAZE

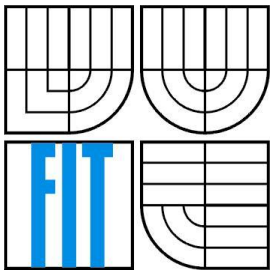
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. TOMÁŠ LABAJ
AUTHOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKCE KŘIVEK V OBRAZE

CURVE DETECTION IN IMAGES

DIPLOMOVÁ PRÁCE
MASTER‘S THESIS

AUTOR PRÁCE — Bc. TOMÁŠ LABAJ
AUTHOR

VEDOUCÍ PRÁCE — Ing. MICHAL ŠPANĚL
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá detekcí křivek v obraze. Shrnuje a popisuje používané metody v této oblasti počítačového vidění. Hlavním cílem je však srovnání metod pro detekci křivek v obraze s parametrizací - tedy Houghovy transformace a metody RANSAC. Tyto metody jsou srovnávány dle několika kritérií, které jsou u hranového detektoru nejdůležitější.

Abstract

This thesis deals with curve detection in images. First, current methods used in this area of image processing are summarized and described. Main topic of this thesis is a comparison of methods of parametric curve detection, such as Hough transformation and RANSAC-based methods. These methods are compared according to several criteria which are the most important for precise edge detection.

Klíčová slova

Detekce hran, Cannyho hranový detektor, detekce křivek, parametrizace křivek, Houghova transformace, RANSAC.

Keywords

Edge detection, Canny edge detector, detection of curves, parameterisation of curves, Hough transform, RANSAC.

Citace

Labaj Tomáš: Detekce křivek v obraze, diplomová práce, Brno, FIT VUT v Brně, 2009

Detekce křivek v obraze

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing.Michala Španěla.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Labaj
24. května 2009

Poděkování

Rád bych poděkoval vedoucímu práce ing. Michalu Španělovi za konstruktivní kritiku a připomínky při vývoji programu i psaní této práce.

Obsah

1 Úvod

S počítačovým viděním se setkáváme velmi často a ani o tom nemusíme vědět. Automatické rozeznání značky u aut, které projedou na červenou, časté použití je v medicíně, také v bezpečnostních systémech či výstupní nebo kontrolní systém v továrně – tady všude se využívá počítačové vidění. Jedná se o složité, stále se rozvíjející odvětví počítačové grafiky. Tato práce se počítačového vidění také týká, jelikož se zabývá detekcí křivek v obraze.

Cílem je vytvoření aplikace, která umí v obraze detekovat a parametrizovat křivky pomocí dvou metod s podobnými vlastnostmi, avšak založené na úplně jiném principu, a tyto metody poté porovnat.

Druhá kapitola pojednává o druzích hranových detektorů a na jakém principu tyto detektory pracují. Podrobněji poté popisuje Cannyho hranový detektor, které je v aplikaci prakticky použit. Třetí kapitola, jejíž jméno se shoduje s názvem této práce, teoreticky popisuje čtyři metody k detekci křivek v obraze. Tyto metody jsou v současném stavu snad jediné, které se pro tuto problematiku používají. Další dvě kapitoly poté postupně podrobněji rozebírají Houghovu transformaci a metodu RANSAC, jakožto dvě metody, které se budou porovnávat. V kapitole 6, která nese titulek Návrh řešení, je uvedeno několik nápadů, jak by aplikace měla fungovat, co by měla obsahovat a jaké výstupy by měla dávat. Kapitola Implementace, jak už napovídá název, se zabývá použitými knihovnami a stručným popisem struktury zdrojových kódů. V předposlední kapitole je potom samotné srovnání Houghovy transformace a metody RANSAC a to v několika kritériích. Kapitola s názvem *Závěr* stručně shrnuje tuto práci.

2 Detekce hran v obraze

V této kapitole jsou popsány základní metody pro detekci. Informace byly čerpány především z *Obrazových segmentačních technik* od Michala Španěla a Vítězslava Berana ([1]).

Jeden z nejdůležitějších úkonů, podle kterého můžeme porozumět obrazu, je detekce hran v obraze. Hranou se rozumí velká změna jasu okolních bodů. Pokud se takováto hrana v obraze nachází, tak se popíše velikostí a jejím směrem.

Hrany se dělí na čtyři typy: skoková (step), náběžná (ramp), impulzní (line) a střechová (roof).

Obrázek 1: Skoková hrana, náběžná hrana, impulzní hrana, střechová hrana (převzato z [1]).

2.1 Hranové detektory

Hranové detektory se dělí do dvou skupin. A to do detekce podle první derivace a podle druhé derivace.

První derivace

Detekce hran probíhá v několika krocích. Vypočteme derivaci (součet součinů hodnoty pixelu s hodnotou masky - konvoluce) pro řádky, poté pro sloupce, vypočteme výsledný gradient podle rovnice 1, popřípadě můžeme vypočtené gradienty aproximovat podle rovnice 2 a můžeme vypočítat orientaci gradientu podle rovnice 3.

$$G(i,j) = \sqrt{G_R(i,j)^2 + G_S(i,j)^2} \tag{1}$$

$$G(i,j) = |G_R(i,j)| + |G_S(i,j)| \tag{2}$$

$$\theta(i,j) = \arctan \frac{G_S(i,j)}{G_R(i,j)} \tag{3}$$

Jednotlivé detektory se od sebe většinou liší jen v konvoluční masce – tedy v matici, podle které se provádí výpočet řádkové a sloupcové derivace. Na tabulce 1 je několik konvolučních masek velikosti 3x3 nejpoužívanějších filtrů zobrazeno. Velikost masky má vliv především na šum, který je při větší masce více potlačen. Proto se používají i větší masky o velikosti 5x5 nebo 7x7. Kdyby se velikost masky ještě více zvětšovala, hrana by už nemusela být nalezena.

Operator	Row gradient	Column gradient
Pixel difference	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Separated pixel difference	$\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & -1 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$
Roberts	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Prewitt	$\frac{1}{3}\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobel	$\frac{1}{4}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{4}\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Frei–Chen	$\frac{1}{2+\sqrt{2}}\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\frac{1}{2+\sqrt{2}}\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

Tabulka 1: Konvoluční masky nejpoužívanějších detektorů (převzato z [1]).

Druhá derivace

Zde detekujeme průchod nulou a nehledáme extrém. Bohužel detektory založené na druhé derivaci mají příliš velké vyhlazení obrazu, zaoblují ostré rohy a mají tendenci k vytváření uzavřených smyček hran.

Reprezentantem detektoru z této kategorie je např. Laplacian. Ten vykazuje stejné vlastnosti ve všech směrech, takže detekce hran je stejná při jakémkoliv natočení obrazu.

Rovnice 4 zobrazuje normalizované jádro konvoluce pro 8-okolí.

$$\boldsymbol{H} = \frac{1}{8}\begin{bmatrix} \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{8} & \mathbf{-1} \\ \mathbf{-1} & \mathbf{-1} & \mathbf{-1} \end{bmatrix} \tag{4}$$

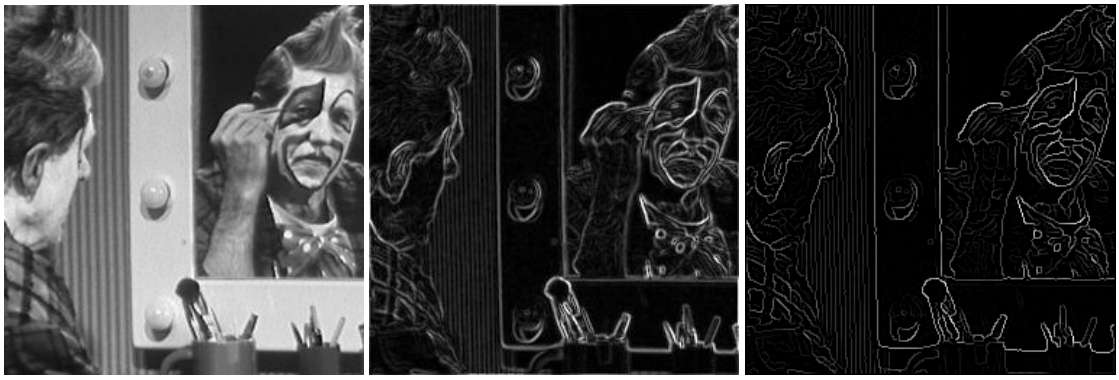
2.2 Cannyho hranový detektor

Cannyho hranový detektor je vůbec nejpoužívanějším hranovým detektorem. Byl navržen tak, aby co nejlépe splňoval následující vlastnosti, které jsou pro každý hranový detektor nejdůležitější: detekce, lokalizace, jedinečná odezva.

Analytická reprezentace bohužel není možná, tak se alespoň pokusím popsat, jak funguje:

- Barevného obrázku na obrázek ve stupních šedi.
- Eliminace šumu – pomocí Gaussovského filtru je obraz rozostřen.

- Provede se standardní detekce hran podle Sobelova operátoru a vypočteme velikost a směr gradientu.
- Nalezení lokálních maxim (thinnig) – v tomto kroku nalezenou hranu zpracujeme tak, že v její tloušťce vezmeme jen ten pixel, kde je odezva největší – hrana tak bude mít tedy tloušťku jednoho pixelu na místě, kde se hrana skutečně nachází.
- Prahování s hysterezí (thresholding) – Cannyho hranový detektor používá dva prahy a to dolní a horní (minimální a maximální). Díky tomu je lepší eliminace ještě možného vyskytujícího šumu. Hodnoty gradientu hrany vyšší než je horní práh, jsou brány jako právoplatné body hrany. Body, které sousedí s právoplatnými body a jejichž gradient je vyšší než dolní práh, jsou uznány také za právoplatné body. A takto postupujeme dále, dokud dochází k nějaké změně uznání. Tímto postupem jsou odstraněny nevýznamné hrany nebo šum, který by byl považován za hranu.



Obrázek 2: zleva – původní obrázek, Sobelův operátor, Cannyho detektor.

3 Detekce křivek v obraze

Tato kapitola obsahuje stručný přehled čtyř nejčastěji používaných metod v současnosti pro detekci křivek v obraze. Každá metoda je krátce popsána a naznačen její způsob řešení. Také jsou uvedeny výhody, nevýhody a použití metody.

Dalo by se říci, že se jedná o dva přístupy k detekci křivek:

1) zjišťování polohy známé křivky
2) detekce libovolné křivky, jejíž parametrizace nemusí být možná.

Do první kategorie spadá metoda Houghovy transformace a metoda RANSAC. Do druhé skupiny se řadí aktivní kontury a level-sets.

3.1 Houghova transformace

Informace byly čerpány z pramenů [2], [5] a [6].

Metoda vyhledává parametricky popsaný objekt, který může být třeba poškozen – jeho hranice nejsou celé, nebo může být zašuměn. Využívá transformaci z Kartézského souřadnicového systému do polárního.

Houghova transformace se nejčastěji používá pro vyhledání přímek, kružnic a občas i elips, ale jelikož potřebujeme znát parametrický popis hledaného objektu, mohli bychom vyhledat například půlkruh, jakkoliv pootočený.

Výhody: Vyhledá objekty jen námi popsané, robustní, vysoká odolnost vůči zašumění, hranice objektu mohou být poškozeny, parametrizace nalezeného objektu (střed, poloměr, …).

Nevýhody: Velká časová i paměťová náročnost, potřeba přesného výstupu z hranového detektoru - což není možné.

Obrázek 3: Detekce kruhového obvodu vlákna hedvábí. Vlevo originální obrázek průřezu vláken a vpravo detekované kružnice (převzato z [2]).

3.2 RANSAC

Opět nejprve slovo k literatuře: Informace zde obsažené jsou převzaty z [3] a [7].

Metoda RANSAC (Random Sample Consesnsus) je založena na opakovaném vybírání náhodného vzorku a testování shody s hledaným tělesem. Musíme tedy znát parametrický popis hledaného tělesa. Toto těleso se potom položí na náhodně vybrané vzorky, kterých je minimální množství pro jednoznačné určení daného tělesa, a pomocí námi definované funkce zjistíme, zda nám daný náhodně vybraný objekt vyhovuje či nikoliv.

Tento algoritmus se používá pro vyhledání geometrických křivek v obraze, při homografii – skládání snímků do panoramat, aj.

Výhody: Robustnost, vyhledání jen námi popsaných objektů, hranice objektu nemusí být uzavřené, parametrizace nalezeného objektu, do určité meze odolné vůči zašumění.

Nevýhody: Možnost nalezení neoptimálního řešení, neschopnost aplikovat na obecné tvary.

Obrázek 4: Detekce elipsy a přímek. Vlevo originální obrázek, vpravo detekována elipsa a přímky (převzato z [3]).

3.3 Aktivní kontury

Jako zdroj informací pro tuto kapitolu posloužily prameny [1], [4] a [5].

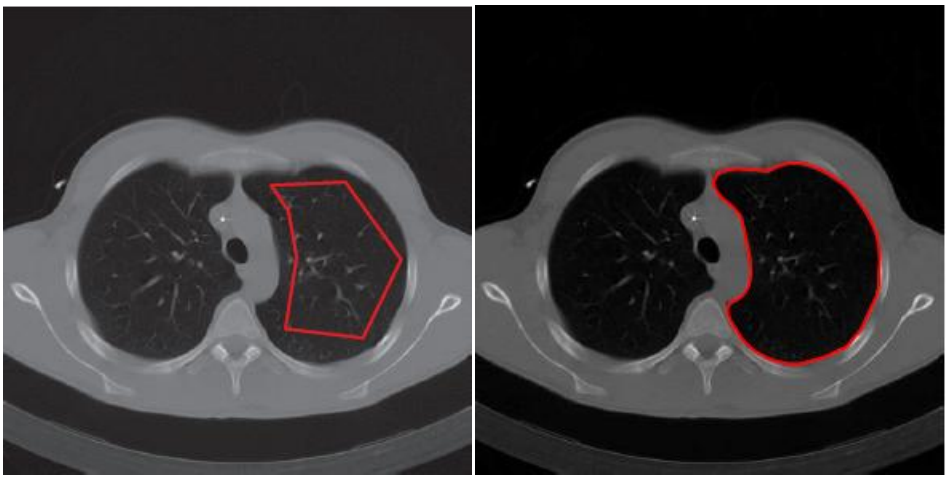
Aktivní kontury (Active Contours / Snakes) nevyžadují žádný popis hledaného tělesa, ale většinou potřebují počáteční inicializaci. Aktivní konturou se totiž rozumí seznam bodů, které spojením tvoří obrys tělesa.

Na začátku tedy označíme přibližnou polohu tělesa, které chceme obejmout, a následně se body kontury pohybují v důsledku působení vnitřních a vnějších sil tak, že je dosaženo požadovaného výsledku. Vnitřní síla ovlivňuje hladkost výsledné kontury, zatímco vnější síla konturu posouvá na obvod tělesa. Celková energie finální kontury je potom minimem energie součtu obou sil.

Tato metoda je často používána v např. medicíně při detekci obvodu buněk, nádorů, ….

Výhody: Detekce parametricky nepopsatelných objektů, nastavitelnost parametrů pro ovlivnění výsledku, výpočetně rychlé, paměťově nenáročné.

Nevýhody: Problém s počáteční inicializací kontury.



Obrázek 5: Demonstrace aktivních kontur na snímku z MRI. Vlevo originální obrázek a počáteční inicializací kontury, vpravo potom výsledek (převzato z [4]).

3.4 Level-sets

Nejprve slovo k literatuře – zdrojem byly prameny [1] a [5].

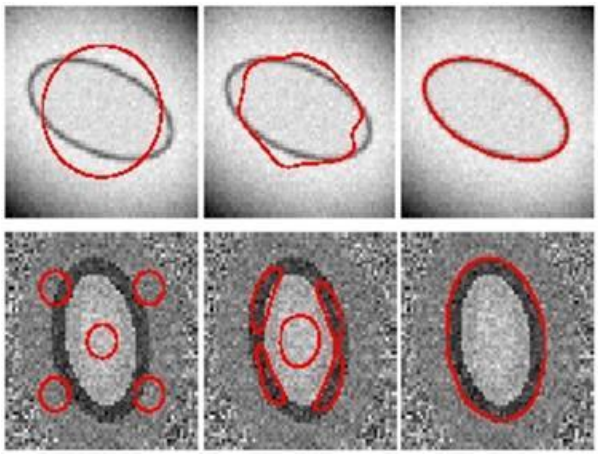
Tato metoda je podobná metodě aktivních kontur. Je také zapotřebí inicializace kontury, která se blíží jejímu výslednému tvaru, a používá se při hledání složitých těles. Její jádro je však poněkud jiné.

Kontura je totiž představována pomocí řezu v tzv. *nulové hladině* určité vícedimenzionální funkce nazvané *level set fiction*. Tato funkce se mění podle potřeby pro obejmutí daného tělesa.

Tato metoda se opět často využívá v medicíně.

Výhody: Detekce parametricky nepopsatelných objektů, možnost nastavení parametru křivosti, možnost implementace znalosti tvaru křivky, výpočetně rychlé, paměťově nenáročné.

Nevýhody: Problém s počáteční inicializací kontury.



Obrázek 6: Ukázka postupu metodou *Level-sets* (převzato z [1]).

4 Houghova transformace

Jak již bylo napsáno v předchozí kapitole, tak informace týkající se Houghovy transformace byly čerpány z literatury: [2], [5] a [6].

Houghova transformace je jednou z nejpoužívanějších metodou pro vyhledávání křivek v obraze a používá se nejčastěji na detekci a parametrizaci přímek, kružnic a elips. Dokáže však vyhledat jakýkoliv definovaný či známý tvar. Její další nespornou výhodou je malá citlivost na šum a také možnost detekování křivky, i když její obvod je z části porušen. Mezi nevýhody této metody patří velká časová a paměťová náročnost, která se exponenciálně zvyšuje se zvyšováním počtu parametrů, které jsou zapotřebí k popisu hledané křivky.

Houghova transformace je algoritmus, který je patentován roku 1962 Paule Houghem. První sloužil k vyhledávání přímek, ale nedalo se dlouho čekat na vyhledávání kružnic a elips. Roku 1972 se vylepšili Houghovu transformaci Richard Duda a Peter Hart a toto vylepšení nazvali *generalized Hough transform*. Jedná se v podstatě o zobecnění Houghovy transformace, pro jakýkoliv známý tvar.

Tento algoritmus obecně funguje na principu, který by se dal nazvat „hrubá síla“. Ve zkratce by se dalo říci, že zjistí všechny možné kombinace jak tvar, který jsme popsali, může být v obrázku umístěn a tu popř. ty varianty, které mají velký úspěch, prohlásí za úspěch. Konkrétněji Houghova transformace pracuje takto: postupně prochází obrázek pro zpracování pixel po pixelu a pokud daný obrazový bod může být součástí naší hledané křivky, tak uvažuje, že tento pixel opravdu dané křivce náleží a zkouší všechny kombinace, jak by objekt mohl být umístěn. Po každém takovémto umístění inkrementuje náležící buňku akumulátoru. Akumulátor je n-rozměrné (tolik rozměrů, kolik je zapotřebí pro popis daného objektu) pole, jehož všechny buňky mají na počátku stejnou hodnotu – obecně nulu. Buňka akumulátoru tedy představuje parametrický popis křivky. Následně zpracováváme další bod obrazu. Pokud je tímto způsobem akumulátor naplněn, tak z něj vybíráme lokální maxima, která jsou nad určitou hladinou, takže by se dalo říci, že akumulátor je prahován. Tato lokální maxima představují hledané křivky. Záleží tedy do určité míry na velikosti prahu, jaké křivky jsou vyhledány. Z tohoto plyne, že objekt nemusí být plně viditelný, aby byl nalezen. A také je malá závislost na šumu. Bohužel zkoušení všech variant je časově velmi náročné[1] a paměťová náročnost je také nezanedbatelná, někdy až nepřiměřená či nemožná.

Abychom zkrátili čas, který je potřeba pro nalezení řešení, musíme omezit počet výpočtů řešení. To můžeme do značné míry ovlivnit vhodnou úpravou obrázku před zpracováním. Jelikož algoritmus počítá varianty řešení z bodů, které mohou náležet k dané křivce, musíme tyto body eliminovat. To můžeme provést například tak, že obrázek převedeme na stupně šedi (jestliže je barevný) a vhodně aplikujeme prahování. Poté je vhodné použít skeletonizaci – tedy ztenčení hran na šířku jednoho pixelu. Nebo můžeme na daný již šedotónový obrázek aplikovat Cannyho hranový detektor. Toto druhé řešení dosahuje lepších výsledků a v řešených projektech se vyskytuje častěji. Takto upravený obrázek, který je již binární, obsahuje důležité hrany s šířkou jednoho pixelu, takže jsou omezeny výpočty hledané křivky.

Optimalizací pro Houghovu transformaci není mnoho. Ale pokud známe přibližnou polohu křivky, můžeme využít např. pravděpodobnostní Houghovu transformaci, která částech, kde se křivka asi nachází, propočítává všechny kombinace a jinde bere s určitou pravděpodobností jen nějaké pixely. Další optimalizací může být např. počítání jen v tomto určeném prostoru popřípadě vnést do

[1] Náročnost záleží na velikosti obrázku a na složitosti křivky, ale řádově se jedná o minuty.

algoritmu jakési náhodné vybírání pixelů. Ani jedna z optimalizací není však nikterak zázračná a spásná a ušetří čas či paměť jen v některých případech (většinou když o obrázku něco víme).

Z principu Houghovy transformace je taky zřejmá paměťová limitace. Akumulátor totiž musí obsahovat prostor pro popis všech variant. A při velkém počtu parametrů nebo při jejich možném velkém rozptylu není možné. I přes tyto nedostatky je Houghova transformace často využívána a to například v lékařství, pro zjištění polohy a velikosti buněk, nádorů, …

4.1 Přímka

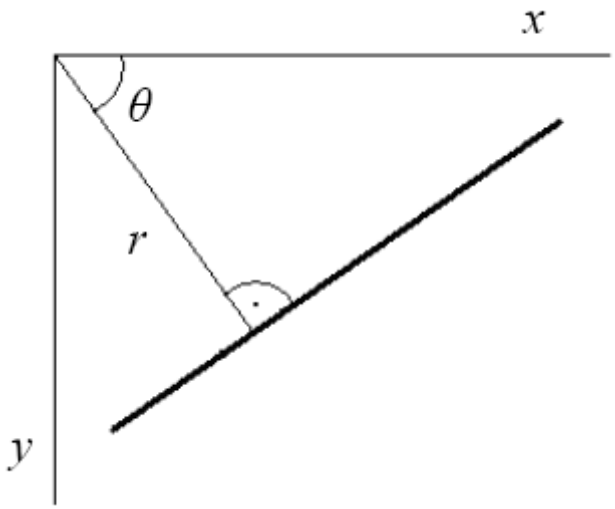
Přímka je dobře popsatelná křivka. Můžeme ji popsat například pomocí směrnicového tvaru přímky, který zobrazuje rovnice 5. Pro Houghovu transformaci tento tvar popisu však není příliš vhodný, jelikož parametr *k* (směrnice přímky) může nabývat jakýchkoliv reálných hodnot. Tento tvar se však využíval od roku 1962.

$$y = k \cdot x + q \tag{5}$$

Od roku 1972 se však díky Richardovi Dudovi a Peterovi Hartovi začal využívat normálový zápis přímky, který můžeme vidět na rovnici 6. Již tedy nebylo zapotřebí vytvářet dva podprostory podle rovnice 5 a její explicitní vyjádření pro *x* a poté tyto podprostory překrýt, ale byl vytvořen jeden akumulátor, který měl jen dva rozměry, dle velikosti parametrů *r* a Θ.

$$r = x \cdot \cos\Theta + y \cdot \sin\Theta \tag{6}$$

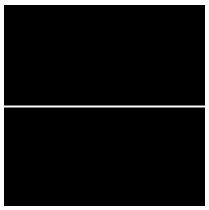
Zde máme opět dvě možnosti, ale už nemáme přístup, který by se častěji používal. Záleží tedy jen na programátorovi, jakou velikost parametrů bude uvažovat. Rovnice 6 totiž popisuje křivku pomocí úhlu, který svírá nejbližší bod přímky s počátkem souřadnic a vzdáleností mezi těmito body. Toto můžeme vidět na Obrázek 7. Úhel Θ tedy může být definován buďto v intervalu <0°; 360°) nebo v intervalu <0°; 180°). Podle toho se poté bude pohybovat parametr *r*. Pokud použijeme první variantu, tak parametr *r* bude nabývat pouze kladných hodno, protože vždy se bude jednat o prostou vzdálenost od počátku souřadnic. Pokud však použijeme druhou variantu, parametr *r* bude nabývat i záporných hodnot, aby bylo zřejmé, na které straně se vlastně přímka nachází. Ani jedna z těchto možností však algoritmus nikterak neovlivňuje.



Obrázek 7: Parametrizace přímky pomoci normálového zápisu.

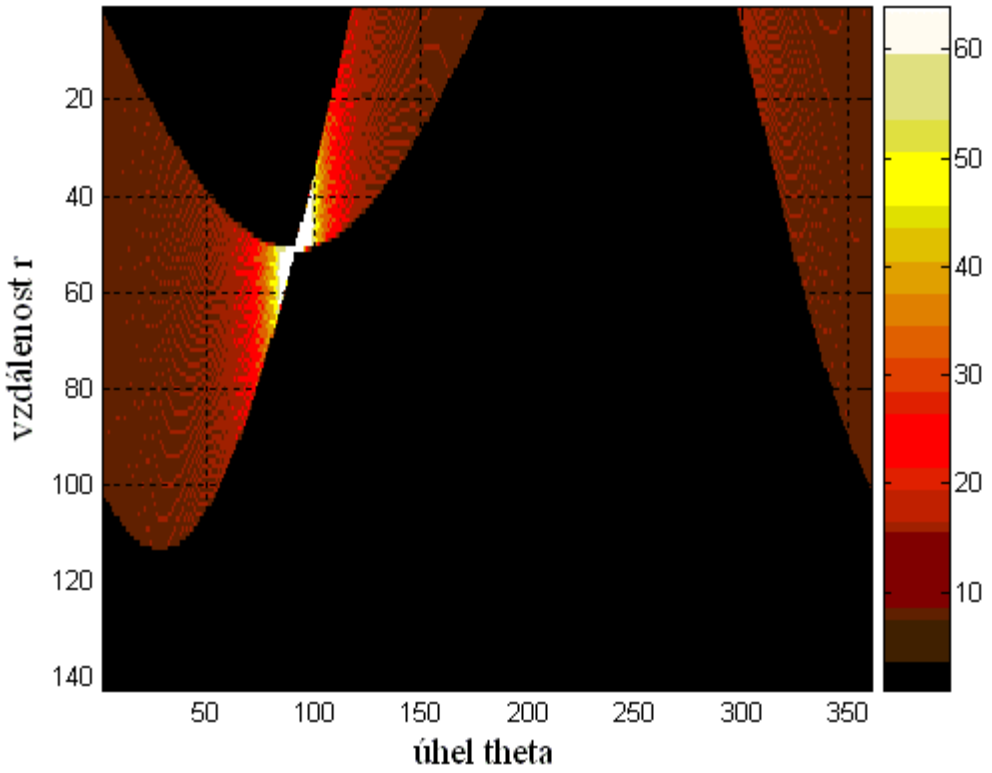
Samotný algoritmus potom nalezne bod, který by mohl náležet přímce a tímto bodem vede přímku pod úhlem 0° a dopočítá vzdálenost. Buňku akumulátoru s těmito parametry inkrementujeme a úhel zvětšíme o jeden stupeň. Takto postupujeme po celém intervalu stupňů úhlu. Tímto nám tedy z jednoho bodu vznikne v akumulátoru několik inkrementovaných buněk, které, kdyby se vykreslily, tak by vykazovaly sinusovou funkci. Z bodu tedy vznikne jistá sinusová funkce. Takto pokrčujeme pro všechny body a každá buňka akumulátoru s lokálním maximem obsahuje parametry přímky. Takto tedy z bodu získáme parametry pro přímku, kterou již snadno dokážeme vykreslit či s ní jinak pracovat.

Celý akumulátor tak, jak by mohl např. vypadat pro Obrázek 8 je zobrazen na Obrázek 9.



Obrázek 8: Vstupní obrázek s přímkou.

Na něm je zřejmá zmíněná sinusová funkce pro každý bod a žlutá až bílá barva značí velkou hodnotu akumulátoru – tedy pravděpodobnou lokalizaci přímky. Bohužel lokální (v tomto případě i globální) maximum nemusí obsahovat pouze jedna buňka v akumulátoru a potom záleží na implementaci, jestli se vykreslí všechny přímky z těchto buněk nebo pouze středová nebo úplně jiná. Toto vzniká tím, že můžeme k přítomné přímce přiložit jinou, mírně otočenou přímku a bude díky malému rozlišení původní téměř a někdy i úplně překrývat.

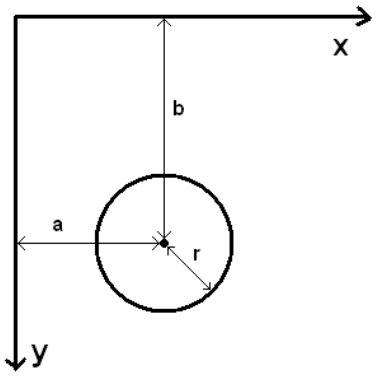


Obrázek 9: Akumulátor přímky k Obrázek 8 (převzato z [2]).

4.2 Kružnice

K hledání kružnic pomocí Houghovy transformace se využívá analytický tvar zápisu rovnice, tak jak lze vidět na rovnici 7. Parametry *a, b* značí posun středu kružnice a parametr *r* poloměr (Obrázek 10).

$$(x-a)^2 + (y-b)^2 = r^2 \tag{7}$$



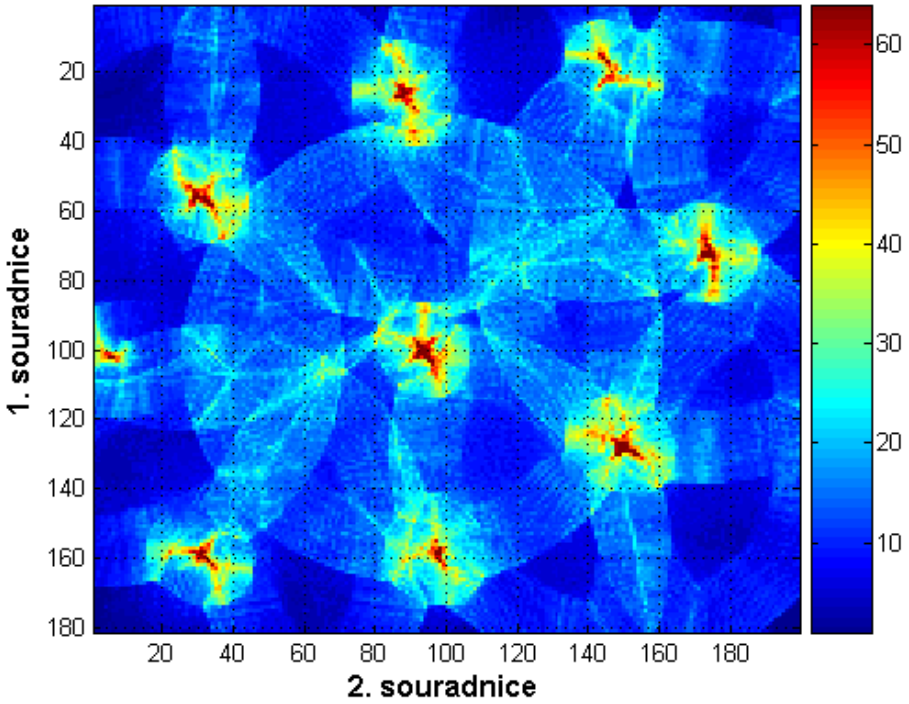
Obrázek 10: Parametry kružnice.

Máme tedy tři parametry, takže již si nevystačíme s dvojrozměrným akumulátorem, ale potřebujeme jeden rozměr přidat – akumulátor bude tedy 3D. Prostor, který budeme tedy potřebovat, nám exponenciálně vzrostl. Pro snížení paměťové složitosti se často požaduje zadání nějaké informace o hledaném poloměru (maximální rozměr, přibližný rozměr, …). Také záleží, jak je prostor definován – zda se předpokládá, že střed leží v obrázku, může být mimo něj nebo zda obrázek musí obsahovat kružnici, která je schopna být v něm celá.

Algoritmus postupuje klasicky, jak již byl popsán. Postupně pro každý vhodný pixel dosazujeme všechny možnosti za 2 parametry a třetí dopočítáváme. Tímto vypočtenou adresu paměťové buňky v akumulátoru vždy inkrementujeme. Každý bod nám tak již nedává sinusovou funkci, jak tomu bylo u hledání přímky, ale dostáváme jistou Gaussovskou funkci.

I kdyby na obrázku byla jen jedna kružnice, tak nemusíme – a často tomu tak bývá – dostat jediný výsledek. Nejvyšší hodnotu v akumulátoru totiž neobsahuje pouze jedna buňka, ale je jich několik. Záleží potom na implementaci, jak si s tímto problémem poradí programátor.

Akumulátor, jak již bylo řečeno, je třídimenzionální, ale pokud bychom omezili jeden parametr, a to konkrétně *r*, tak akumulátor pro obrázek 3 může vypadat jako na obrázku 11. Můžeme tedy vidět pouze velikost hodnot akumulátoru pro parametry středu.



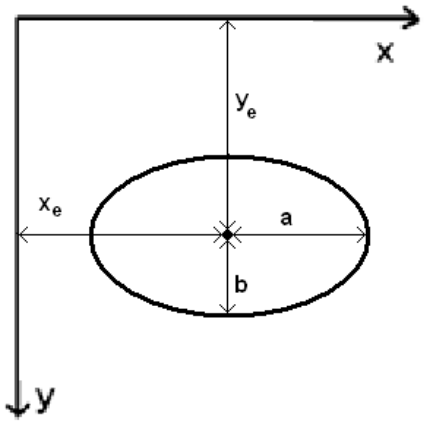
Obrázek 11: Akumulátor s ignorovaným parametrem *r* pro obrázek 3 (převzato z [2]).

4.3 Elipsa

Hledání elipsy by se dalo rozdělit na dvě části. Na elipsy v normální poloze, jejichž osa je rovnoběžná s osou x, a obecnou elipsu, která může být jakkoliv nakloněna.

Nejprve se budeme věnovat elipsám v normální poloze. Její matematický tvar je zobrazen na rovnici 8 a popis jejich parametrů je zřejmý z obrázku 12.

$$\frac{(x-x_e)^2}{a^2}+\frac{(y-y_e)^2}{b^2}=1 \tag{8}$$



Obrázek 12: Parametry elipsy v normální poloze.

Máme tedy čtyři parametry, což je o jeden více než u kružnice. Akumulátor rozšíříme o jeden další prostor, ale jinak vše zůstává stejné. Měníme tři parametry a dopočítáváme čtvrtý. Nyní však již nebude z jednoho bodu vznikat Gaussova křivka, jak tomu bylo u kružnice, ale jakési pravidelné 4D těleso. Vykreslení akumulátoru už také není zcela možné a nebylo by názorné. Pro elipsu platí o to více než pro kružnici, že o hledané křivce musíme vědět něco o její velikosti, protože prohledávání čtyřdimenzionálního prostoru zabere spoustu času a tento paměťový prostor bude velmi velký.

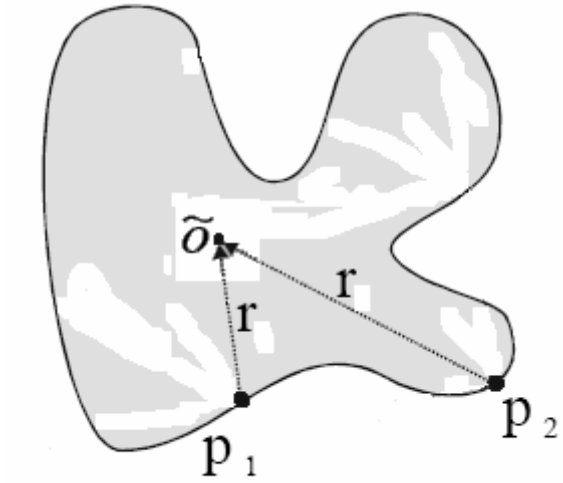
Obecná elipsa popsána rovnicí 9 má parametrů 5. Dalo by se říci, že 4 parametry popíšeme elipsu v normální poloze a pátým parametrem volíme úhel naklonění elipsy. Náročnost (paměťová i výpočetní) tímto opět exponenciálně roste.

$$A \cdot x^2 + B \cdot x \cdot y + C \cdot y^2 + D \cdot x + E \cdot y + F = 0$$

$$za\ podmínky: B^2 - 4 \cdot A \cdot C \geq 0 \tag{9}$$

4.4 Obecné tvary

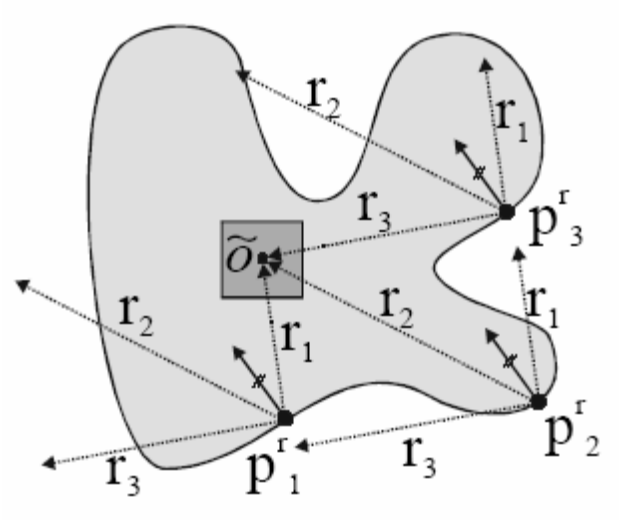
Houghova transformace se dá však použít pro vyhledání křivek, jejichž analytický popis nemusíme znát, ale musíme znát jejich tvar. Pokud víme, jak křivka má vypadat, vypočteme si referenční bod např. těžiště a každý bod křivky popíšeme pomocí úhlu a vzdálenosti (vektor) k referenčnímu bodu (Obrázek 13). Takto vypočtené hodnoty uložíme do LUT – *look up table.*



Obrázek 13: Popis obecného tvaru (převzato z [5]).

Samotný algoritmus postupuje opět „hrubou silou“. Takže prochází obrázek pro zpracování a pro každý bod, který může být bodem křivky, udělá výpočet referenčního bodu pro každou položku v LUT. Pozici takto vypočteného referenčního bodu vždy inkrementujeme v akumulátoru (Obrázek 14). Takže uvažujeme, že každý možný bod křivky v obraze může být jakýmkoliv bodem v LUT a vypočítáváme referenční bod. Akumulátor je v tomto případě dvojrozměrný – pozice referenčního bodu. To však platí jen tehdy, pokud uvažujeme jen prostý posun křivky někde v obraze. Pokud bychom uvažovali i rotaci, museli bychom jeden rozměr přidat. Obdobně je to se změnou měřítka.

Můžeme tedy hledat jakoukoliv křivku o níž víme jak vypadá (ale na obrazu nemusí být celá viditelná) a tato křivka může být jakkoliv zvětšena či zmenšena a otočena.



Obrázek 14: Naznačení postupu při HT u obecných tvarů (převzato z [5]).

Vyhledávání 3D objektů

Houghova transformace se také často používá pro hledání 3D objektů jako jsou např. roviny, válce. Toto uplatnění je především v robotice.

Rovina je popsatelná explicitní rovnicí (Rovnice 1), kde naše hledané neznáme v Houghovu prostoru budou a_x, a_y a d. Výpočet není tak náročný, když se pohybujeme v 3D prostoru, protože se nepředpokládá rapidní změna roviny při dvou za sebou jdoucích snímcích od robota, který se pohybuje. Od něj musíme získávat i vzdálenost objektů, abychom byli schopni počítat v 3D prostoru. To lze zajistit pomocí dvou kamer, nebo jedné, která je vybavena laserem.

$$z = a_x \cdot x + a_y \cdot y + d \tag{10}$$

Rovnice 1: Explicitní rovnice roviny.

5 RANSAC

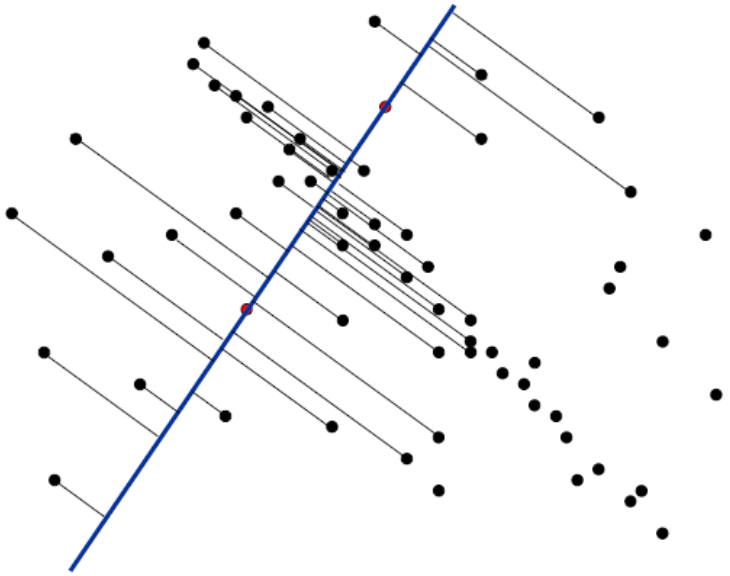
Jak již bylo napsáno v kapitole 3, informace o metodě RANSAC byly čerpány z [3] a [7]. K tomu zde byly ještě použity zdroje [8] a [9].

Jméno této metody je zkratka slov Random Sample Consensus, což v překladu znamená shoda náhodných vzorků. Tato iterativní metoda byla poprvé popsána pány Fischlerem a Bollesem roku 1981 a je nedeterministická. To znamená, že výsledek je akceptovatelný jen při určitém množství opakování algoritmu a nemusíme vždy obdržet stejný výsledek, popř. nenalezení optimálního řešení. RANSAC se často využívá pro vyhledávání křivek, když má obraz mnoho vzorků nebo jeho další aplikací je nalézání homografie v dvou snímcích, tedy skládání panoramat, nalezení umístění kamery v 3D scéně apod. Také se využívá např. pro nalezení roviny a následný pohyb robota.

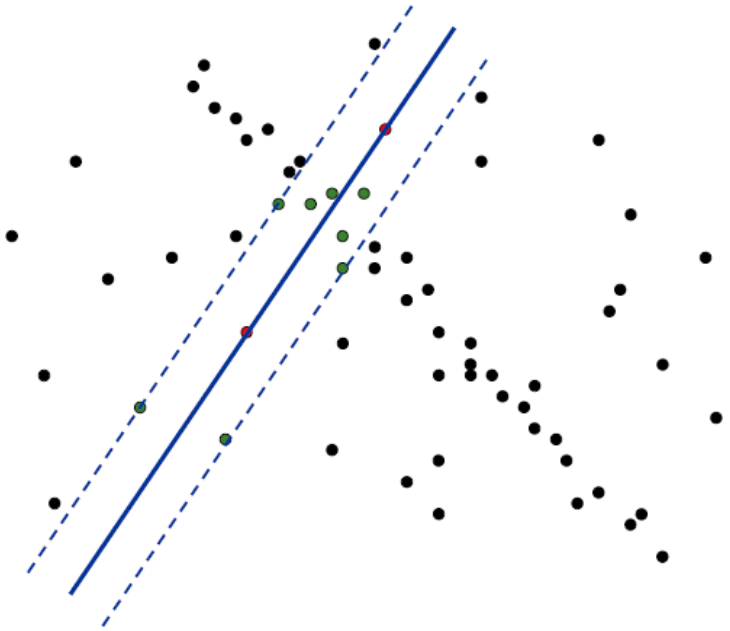
Algoritmus pracuje na principu náhodného vybírání minimálního počtu vzorků pro popis dané křivky a zjistí, kolika bodům takto vytvořené těleso vyhovuje a kolika ne. Abychom však mohli vybírat vzorky, nějaké potřebujeme mít. Za vzorky se v tomto případě považují důležité body – tedy výstup z hranového detektoru nebo z detektoru významných bodů. Při hledání křivek je však mnohem lepší řešení výstup z hranového detektoru a to konkrétně Cannyho hranového detektoru. Ten totiž vykazuje jedny z nejlepších výsledků a šířka odezvané hrany je jeden pixel, takže se jedná i omezení počtu vzorků a nemusíme následně provádět mnohem větší počet iterací. Nyní již z takto nalezených bodů můžeme náhodně vybírat vzorky pro křivku. Pokud těmito body křivku proložíme, tak můžeme pomocí námi definované funkce zjistit, kolik bodů této křivce náleží – tyto body se nazývají *inlieres* a naopak, kolik bodů je mimo křivku tzv. *outliers*. A pamatujeme si nejlepší řešení (může jich být několik). Tato řešení potom prohlásíme za výsledek nalezení.

Funkce, která nám zjistí počet *inliers* může být založena na jakémkoliv principu. Na Obrázek 15 můžeme například vidět naznačení metody nejmenších čtverců. Červené body na obrázku značí ty body, které byly náhodně vybrány pro nalezení přímky. Těmito body je přímka proložena a zjišťujeme vzdálenost každého bodu od této přímky. To řešení, které má sumu vzdáleností nejmenší, je nejlepší.

Častěji než metoda nejmenších čtverců se používá metoda, která je naznačena na Obrázek 16. Ta je založena na tom, že vzorky mohou být určitým způsobem posunuty a proto se může zavést proměnná, která značí míru tolerance odchýlení od původní křivky. Můžeme tedy prohlásit, že body nemusí být součástí křivky – v tomto případě přímky, ale mohou být ve vzdálenosti několika pixelů a stejně je budeme považovat za *inliers*.



Obrázek 15: Metoda nejmenších čtverců (převzato z [3]).



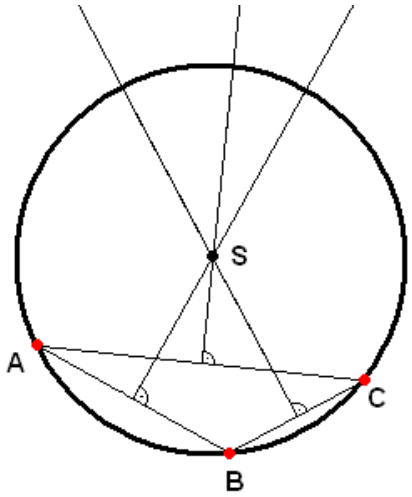
Obrázek 16: Metoda odchylky vzorků (převzato z [3]).

Největší problém však je to kolik iterací má program provést. Pokud to bude málo, tak se zvyšuje šance, že nebude nalezeno optimální řešení. Naopak pokud iterací bude příliš hodně, tak je pravděpodobné, že již máme nejlepší výsledek, ale stále se pokoušíme o jiný, takže vrácení výsledku jen prodlužujeme. Pokud potřebujeme nalézt jen jedno řešení, které by vyhovovalo určitému prahu, tak máme po starostech – vrátíme první, které vyhovuje, a ukončíme vyhledávání. To však většinou nepotřebujeme. Chceme nejlepší řešení a často tomu bývá, že není jen jedno, ale podobných křivek je v obraze více. Proto musíme odhadnout počet iterací. To lze pomocí rovnice 11.

$$počet\ iterací = \frac{\log(1 - pravděpodobnost\ úspěšného\ řešení)}{\log(1 - pravděpodobnost\ vybrání\ inlieru^{počet\ parametrů\ křivky})} \quad \textbf{(11)}$$

Kružnice

Pro kružnici platí vše výše zmíněné. Jak postup RANSACu, tak i metody pro výpočet inliersů. Jen by mohlo být ještě zmíněno, jak se z tří libovolných bodů dosáhne zkonstruování kružnice a tedy její parametrizace. Je několik postupů, ale je jeden oblíbený a poměrně jednoduchý (lze vidět na Obrázek 17). Jak již bylo řečeno, kružnici lze jednoznačně určit třemi body. Ty jsou zobrazeny červeně. Pokud tyto body mezi sebou spojíme a vytvoříme osy jednotlivých úseček, tak v bodě protnutí těchto os se nachází střed kružnice.



Obrázek 17: Zjištění středu kružnice ze známých tří bodů.

Jelikož jsou vzorky vybírány náhodně a iterativně, tak je algoritmus RANSAC odolný vůči šumu. Také z vybírání minimálního počtu vzorků pro křivku plyne to, že křivka může být přerušena, nebo něčím z části překryta a stejně je algoritmem vyhledána. Jelikož však hledá nejvíce *inlierů* a na obrázku může být několik např. kružnic s jiným poloměrem, může prohlásit za výsledek jen jednu s největším počtem bodů – pravděpodobně tu s největším poloměrem, pokud není příliš přerušena.

Elipsa

Pro jednoznačné zkonstruování obecné elipsy je potřeba bodů 5. Těchto 5 bodů však může být prostorově rozloženo tak, že nebudou tvořit elipsu, ale jinou kuželosečku. Pokud se však postupuje pomocí metody nejmenších čtverců, parametry elipsy získáme vždy. Tato metoda zjistí parametry takové elipsy, která má od vybraných pěti bodů nejmenší součet obsahů čtverců. Každý čtverec má stranu tak velikou, jaká je vzdálenost od daného bodu k nejbližšímu bodu elipsy.

Pokud již známe parametry elipsy, můžeme zjistit počet inlierů a případně elipsu vykreslit.

Kubická Bézierova křivka

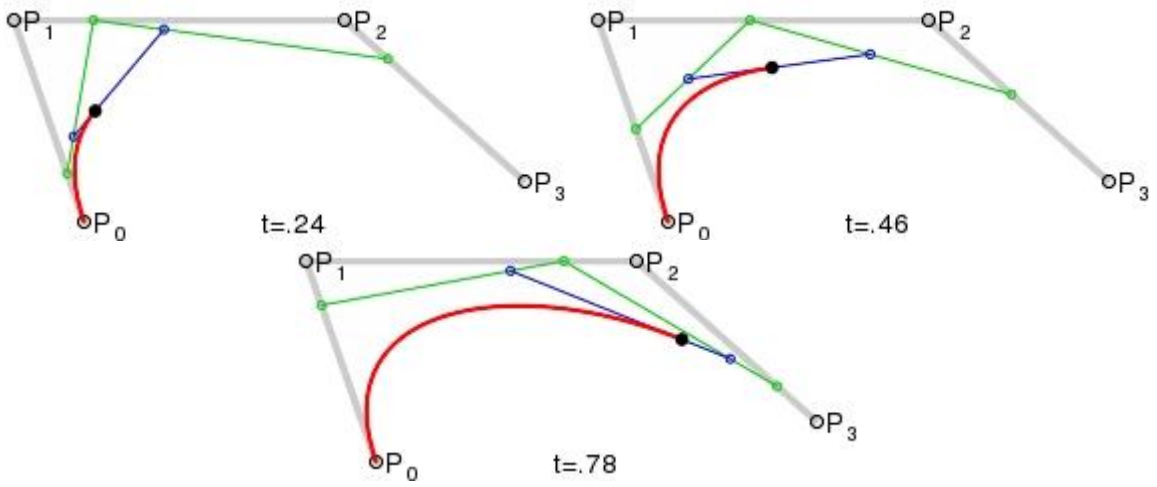
Tento typ Bézierovy křivky má 4 kontrolní body a křivka začíná v prvním a končí ve čtvrtém bodě. Druhým a třetím bodem křivka prochází jen velmi výjimečně a to když všechny 4 body leží na přímce. Tyto 4 body určují řídící polygon a křivka se musí nacházet uvnitř tohoto polygonu. Druhý a třetí bod tedy určují jen směr křivky.

Matematický popis kubické Bézierovy křivky lze vidět na rovnici 12.

$$
\begin{aligned}
C(t) &= \sum_{i=0}^{3} \binom{3}{i} t^i (1-t)^{3-i} P_i \\
&= (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \\
&= (-P_0 + 3P_1 - 3P_2 + P_3)t^3 + (3P_0 - 6P_1 + 3P_2)t^2 + (-3P_0 + 3P_1)t + P_0 \\
&= \underbrace{\begin{pmatrix} t^3 & t^2 & t & 1 \end{pmatrix}}_{radkovy\ vektor} \underbrace{\begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}}_{bazova\ matice} \underbrace{\begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}}_{sloupcovy\ vektor} \quad ,t \in < 0,1 >
\end{aligned}
$$

(12)

Grafické vytváření kubické Bézierovy křivky lze vidět na obrázku 18, kde samotná křivka je zobrazená červeně. Každý obrázek ukazuje situaci, kdy se mění *t* od 0 po danou hodnotu. Takto se postupně křivka tvoří.



Obrázek 18: Grafické znázornění vytváření kubické Bézierovy křivky (převzato z [9]).

Na zkonstruování kubické Bézierovy křivky tedy potřebujeme znát 4 body. 2 body však na této křivce neleží a nejsou to ani body získané z hranového detektoru. Výpočet iterací proto bude muset vypadat jinak, než je tomu klasické vzoreček u této metody (rovnice 11). Těchto iterací musí být mnohonásobně více. Tyto dva body budou vybírány ze všech bodů obrázku.

Poté již budeme vybírat body a zjišťovat kolik inlierů má vybraná křivka.

6 Návrh řešení

Cílem diplomové práce a programu je srovnání metod detekce křivek obrazu, jejichž výstupní křivky můžeme parametrizovat. Jedná se tedy o metodu RANSAC a Houghovu transformaci.

Program by měl být schopen rozpoznat těmito metodami základní parametricky popsatelné křivky jako je přímka, kružnice a elipsa. Také by měl být schopen nám zobrazit alokovanou velikost houghova prostoru a prostoru potřebnou pro metodu RANSAC.

Je tedy potřebné, aby uživatel byl schopen si zvolit metodu vyhledávání a zároveň typy křivek a jejich zobrazovací prahy a další parametry. Pro toto množství parametrů je vhodné aplikaci řešit ne konzolově, ale spíše s GUI. A také by zřejmě bylo vhodné možnost ukládání nastavení pro všechny parametry, aby se urychlilo jejich nastavování pro určité typy obrázků.

Aplikace by také jistě neměla postrádat minimálně dvě pole pro zobrazení obrázků – vstupního a výstupního, tedy grafickou komunikaci s uživatelem, a dále výpis informací, které křivky byly detekovány za jakou dobu a kolik paměťového prostoru k tomu bylo zapotřebí.

6.1 Společná část

Po načtení obrázku do vstupního pole, si uživatel zvolí pomocí check-tlačítek typy hledaných křivek a jejich prahovou hodnotu pro zobrazení a poté může kliknutím na PictureBox dané metody spustit výpočet. Obraz se však vždy převede na šedotónový obraz (grayscale) podle rovnice 13 a na něj se aplikuje Cannyho hranový detektor. Jeho parametry, jelikož velkou měrou ovlivňují výsledek detekce, jsou uživatelem také volitelné. Zde se již další postup liší s každou metodou.

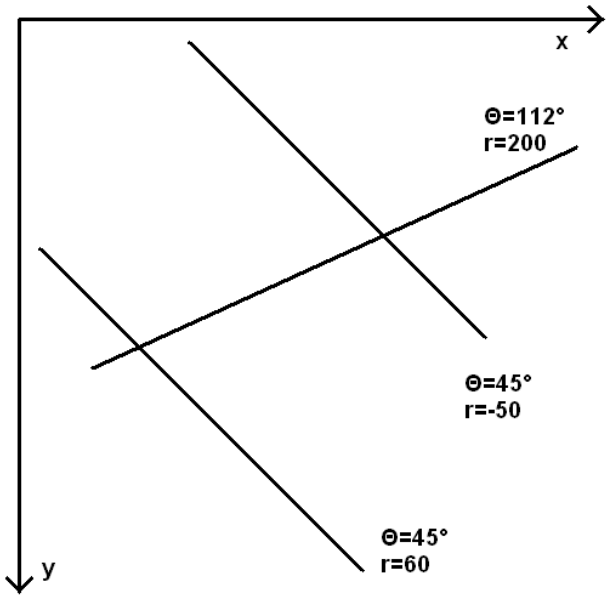
$$jas = 0{,}299 \cdot R + 0{,}587 \cdot G + 0{,}114 \cdot B \tag{13}$$

6.2 Houghova transformace

Přímka

Inicializace akumulátoru

Inicializujeme akumulátor pro přímku. Ten je velký 180x(2xuhlopříčka obrazu ve kterém se bude hledat přímka). Uvažujeme tedy, že úhel Θ se pohybuje v intervalu <0; 180) a minimální vzdálenost přímky od počátku souřadnic je jak kladná, tak i záporná, abychom věděli, kde se přímka přesně nachází.



Obrázek 19: Parametrizace přímek.

Na obrázku 19 je naznačen způsob kladných a záporných vzdáleností od počátku souřadnic.

Plnění akumulátoru

Dále pro každý bod v obraze vytvoříme sinusovou funkci v akumulátoru. Tedy předpokládáme, že tímto bodem vede přímka se sklonem Θ=0° a vypočítáme vzdálenost. Hodnotu takto vypočtené adresy v akumulátoru inkrementujeme a inkrementujeme i úhel Θ. Pokračujeme, dokud Θ < 180°. Toto provedeme pro každý možný bod přímky.

Filtrace akumulátoru a zobrazení

Naplněný akumulátor můžeme nyní filtrovat. Vyhledáme veškeré lokální maxima a ostatní buňky akumulátoru vynulujeme. Takto upravený akumulátor již můžeme procházet, a pokud by hodnota některé z buněk byla vyšší nežli práh, tak křivku popsanou danými parametry zobrazíme.

Paměť

Jelikož detekce přímek není paměťově příliš náročná, tak akumulátor přímky může být i po vyhledání ponechán. Po následné změně prahu – nikoliv však jiných parametrů, které by ovlivnily jiný výsledek v akumulátoru, se tak nemusí akumulátor vypočítávat znovu a jen se zobrazí hodnoty, podle nového zadání.

Kružnice

Inicializace akumulátoru

Inicializujeme akumulátor pro kružnici. Velikost tohoto akumulátoru je závislá na uživatelem zvolené minimální a maximální hodnotě hledaného poloměru a na velikosti obrázku ve kterém budeme hledat kružnici, jelikož nebudeme alokovat větší pole než je zapotřebí. Akumulátor bude mít tedy velikost: (maximální poloměr – minimální poloměr) x šířka x výška oblasti hledání. Předpokládáme tedy, že kružnice má střed v obrázku.

Naplnění akumulátoru

Poté pro každý bod, který by mohl být na kružnici, voláme funkci pro naplnění akumulátoru. Zde zjistíme možné polohy středu. Střed sice může ležet jen v oblasti kružnice s poloměrem r a se středem počítaného bodu, ale pro lepší funkčnost uvažujeme čtverec, ve kterém budeme volit všechny body a filtrovat jako možný střed pomocí vzdálenostní funkce. Takto se tedy do akumulátoru dostanou jen ty body, které mají maximálně povolený poloměr. Aplikace, jak již bylo napsáno, umožňuje zvolení i minimálního poloměru. Toto už nijak výrazně neurychluje výpočet. Minimální poloměr je však implementován ze dvou důvodů: zaprvé když nechceme zobrazovat malé kružnice a zadruhé z paměťových důvodů.

Filtrace akumulátoru a zobrazení

Akumulátor, který je naplněný nyní podrobíme filtraci. Vyhledáme tedy lokální maxima v celém paměťovém prostoru. Tato maxima ponecháme a ostatní buňky vynulujeme. Veškeré buňky – resp. křivky, které jsou nad hranicí prahu, zobrazíme.

Paměť

Veškerý uložený paměťový prostor si ponecháme dokud nedojde ke změně parametrů, které by mohly vést i ke změně tohoto prostoru (změna Cannyho parametrů, zvýšení maximálního poloměru, zmenšení minimálního poloměru). Takto můžeme příště zobrazit křivky bez zbytečných výpočtů.

Elipsa

Tento geometrický útvar už je pro hledání pomocí Houghovy transformace poměrně náročný. Elipsu bychom mohli rozdělit ještě na dvě podskupiny – elipsa v normální poloze a obecnou elipsu. Pro Houghovu transformaci je toto rozdělení poměrně zásadní, jelikož obecná elipsa má o jeden parametr více než elipsa v normální poloze. A pokud uvažujeme obrázek o velikosti 1000x1000 pixelů, tak by k hledání elipsy v normální poloze bylo zapotřebí prostor o velikosti: souřadnice středu x velikost hlavní osy x velikost vedlejší osy x datový typ short= 1000x1000x1000x1000x16 bitů = 1012x2byte = 1863GB. Pro obecnou elipsu by bylo zapotřebí ještě 180x více paměťového prostoru. Nemusím snad příliš zdůrazňovat, že takovýto alokovaný prostor není možný. Proto hledání elips pomocí Houghovy transformace je možné jen po úpravě algoritmu a to zpomalí výkon vyhledání natolik, že hledání obecných elips není příliš aplikovatelné a proto není ani implementované.

Modifikace algoritmu

Jak již bylo napsáno, základní algoritmus musíme určitým způsobem modifikovat, abychom mohli elipsy vyhledávat. To lze udělat několika způsoby – např. převzorkováním vstupního obrazu na menší rozlišení. Tím zmenšíme oblast pro nutnou alokaci a metoda už může být použitelná. Bohužel bychom museli obrázek často velmi zmenšit, což by přineslo jeho velkou obrazovou deformaci a vyhledání elips by potom s velkou pravděpodobností bylo negativní. Uživatel by o této deformaci ani nemusel vědět a ztratila by se výhoda Houghovy transformace – přesnost. Další varianta úpravy algoritmu by mohlo být postupné vyhledání v obraze. Pokud tedy nemůžeme alokovat paměť pro celý obraz najednou, tak alokujeme jen pro určitou část obrazu. V ní se pokusíme elipsu vyhledat. Potom se s tímto výřezem posuneme na další část obrazu, dokud jej neprohledáme celý. Tato metoda by měla však tu nevýhodu, že bychom nemohli hledat větší elipsy. Mohli bychom hledat jen takové,

které by se vlezly do výřezu a to by bylo nedostačující. Další způsob, který je nakonec aplikovaný zachovává ideu hrubé síly Houghovy transformace, tedy prošetření každé varianty.

Aby bylo alokované místo pro vyhledání elipsy únosné algoritmus je tedy nakonec pozměněn takto: Program nevyhledává potencionální bod křivky a nezkouší jej umístit na veškeré možné křivky parametrického popisu, ale uvažujeme, že v obrázku je nějaký bod, který by mohl ležet na hledané elipse, a předpokládáme, že tato elipsa má střed na souřadnicích [0, 0]. Poté dosazujeme všechny možné hodnoty parametru *b* a dopočítáváme *a*. Akumulátor, který si na začátku inicializujeme, je dvojrozměrný a má velikost [maximální hodnota parametru *a;* maximální hodnota parametru *b*]. Tyto hodnoty mohou být maximálních hodnot šířka a výška obrazu. Plníme tedy tento akumulátor hodnotami pro všechny možné elipsy se středem [0, 0] a pokud je některá hodnota buňky rovna prahu, tak tuto elipsu vykreslíme. Pokud už nemůžeme měnit hodnotu parametru *b*, tak akumulátor vynulujeme a uvažujeme střed elipsy na vedlejším pixelu. Tímto způsobem postupujeme přes celý obraz a stačí nám k tomu poměrně malý akumulátor.

Pro příklad popsaný v prvním odstavci elipsy bychom namísto 1863,6 GB potřebovali 1000x1000x16 bitů = 106x2 byte = 1,9 MB. Jak už to však většinou bývá, algoritmus, který je založen na stejném principu a nepotřebuje tolik místa, potřebuje více času. Není tomu jinak ani v tomto případě. Metoda je časově velmi náročná, ale zachovává původní ideu a proto se pro porovnání algoritmů hodí nejlépe. Přeci jen aby bylo možné malinko urychlit vyhledání, tak je možno omezit parametry elipsy – *a,b* – tedy velikost os zmíněného geometrického tělesa. Program především urychlí maximální a minimální hodnota parametru b. Jelikož ten měníme v uživatelem definovaném intervalu a parametr *a* pouze dopočítáváme a porovnáváme, zda výsledná hodnota leží v daných mezích.

Další informace

Pro elipsu je definována jedna funkce, která má za úkol urychlit výpočet. Touto funkcí je vymazání světlých bodů, které jsou součástí elipsy, která již dosáhla potřebného prahu. Tímto se tedy smažou body, které patří již k určité elipse, a nepočítáme s nimi dále, takže nezbrzďujeme výpočet. Takovéto jednání může být někdy kontraproduktivní, ale ve většině výsledků je užitečné.

Elipsa je tedy jediná křivka u Houghovy transformace, která nevyužívá uložení nalezených křivek, protože to není paměťově možné.

6.3 RANSAC

Tato metoda není tolik paměťově náročná, takže si můžeme uchovat veškeré výsledky v paměti a pokud budeme hledat více typů křivek a poté změníme nastavení u jednoho typu, tak přepočteme jen tento jeden typ a ostatní jen zobrazíme z paměti.

Po spuštění funkce pro výpočet pomocí metody RANSAC načteme veškeré hranové body do pole, abychom věděli jejich počet a mohli je rychle vybírat. Poté přecházíme k hledání daných křivek.

Pro přímku a Bézierovu křivku je postup následovný:

- Výpočet iterací a tolikrát provést:
  - Vybrání náhodných bodů.
  - Výpočet křivky.
  - Zjištění počtu inlierů.
  - Vykreslení.

Pro kružnici a oba typy elips by se postup výpočtu dal strukturovat takto:

- Vybrání náhodných bodů.
- Výpočet křivky.
- Počet inlierů.
- Přepočet iterací.
- Vykreslení.

Podrobný popis algoritmů je uveden níže u každého typu křivky zvlášť.

Přímka

Výpočet iterací

Nic nebrání tomu, abychom vypočetli počet potřebných iterací a vyhledali přímky podle idey metody RANSAC. Hodnota neznámé ve vzorci pro výpočet iterací *Pravděpodobnost vybrání inlieru* totiž vypočteme jednoduchým vydělením uživatelem zvolené hodnoty pro přímku s celkovým počtem bodů, které jsme si dali do pole.

Výpočet křivky

Poté již vybíráme x-krát 2 náhodné body, vytvoříme z nich přímku a pro všechny body v poli vypočteme minimální vzdálenost k této přímce. Pokud je tato vzdálenost menší nebo rovna zvolené toleranci, tak tento bod zařadíme mezi inliery.

Paměť

Pokud je dostatečný počet inlierů, tak přímku vykreslíme a zároveň ji zařadíme do seznamu přímek. V tomto seznamu budou dokud nedojde ke změně, která by mohly ovlivnit, že po přepočtu by v tomto seznamu už nebyly nebo by nějaké křivky ještě přibyly – změna parametrů Cannyho hranového detektoru, změna prahu. Oproti Houghově transformaci je zde i změna prahu a to proto, že ikdyž práh snížíme, tak musíme celý výpočet provést znovu, protože nemáme uloženy všechny přímky.

Kružnice

Pokud je RANSAC spuštěn s parametrem hledání kružnic, tak nejprve zkontrolujeme, jestli již nemáme kružnice načteny v paměti, abychom nemuseli vyhledávat. Pokud musíme kružnice vyhledávat, tak nejprve musíme vypočítat počet iterací.

Výpočet iterací

Počet iterací by mohl být vypočítán staticky neboli na začátku a poté tento počet neměnit. To by však mělo za důsledek jediné: počet iterací by byl pro obrázek nevhodný – většinou potom velký. Pokud by tedy vypočtené číslo bylo vysoké – algoritmus by byl proveden mnohonásobně vícekrát než je potřebné a došlo by ke zbytečnému zdržení. Na druhou stranu pokud by počet iterací byl nízký, nemuseli bychom najít některé kružnice nebo dokonce žádné. Proto je vhodnější na začátek dát určitý, nízký, počet iterací a poté dle situace počtu přiřazování bodů k inlierů, počet iterací popřípadě zvyšovat. Popis nejefektivnějšího způsobu výpočtu se ukázal takový, že na začátek přiřadíme počtu iterací určité číslo – např. 50. A pokračujeme výpočtem křivky, z něhož dostaneme kolik procent

inlierů je vzhledem k vypočtené kružnici - obvodu. Nyní vypočteme pomocí váženého průměru hodnotu průměrného procenta, které jsme u tohoto obrázku vybrali a vypočteme pomocí rovnice 11 počet potřebných iterací. Naše vypočtené procento je potom v tomto vzorci hodnota pravděpodobnosti vybrání inlierů. Pokud je tato hodnota vyšší než je současný počet iterací, tak počet iterací zvýšíme. Aby se nám nestalo, že budeme tento počet pořád zvyšovat a cyklus by se tak stal nekonečným, tak je implementována zarážka horní hranice. Více iterací se tedy neprovede.

Tato metoda adaptivního určení počtu iterací se velmi osvědčila. Některé obrázky měly totiž stejný počet vrácených z Cannyho hranového detektoru, ale jejich uskupení mohlo být zcela jiné. Např. jeden obrázek mohl obsahovat několik větších kružnic a druhý mnoho malých. V tom případě potřebuje rozdílný počet iterací. Ale statický výpočet nám určí počet stejný – vysoký, který se hodí jen pro druhý obrázek. Pro první stačí několikanásobně méně iterací. Adaptivního určení počtu iterací tento problém přesně vyřeší.

Výpočet křivky a inlierů

Vybereme náhodně tři body z bodů, které nám Cannyho hranový detektor vrátil jako body hrany. Z těchto náhodných bodů vypočteme kružnici. Výpočet kružnice probíhá tak jak je vysvětleno v teoretické části – tedy přes průsečík kolmic vedených ze středů úseček tvořené z našich tří bodů. V tomto průsečíku se totiž nachází střed hledané kružnice. Výpočet poloměru je potom již vzdálenost vypočteného středu k jakémukoliv, v aplikaci k prvnímu, bodu. Pokud vypočtená kružnice splňuje velikostní podmínky, které požaduje uživatel, tak můžeme vypočítat počet inlierů.

Pro každý světlý obrazový bod z výstupu Cannyho hranového detektoru tedy počítáme vzdálenost ke kružnici. Resp. počítáme vzdálenost od tohoto k středu kružnice. Pokud od této vzdálenosti odečteme poloměr kružnice, získáme tak požadovanou vzdálenost bodu od kružnice. Když je tato hodnota menší, než zadaná tolerance, tak tento bod prohlásíme za jeden z inlierů k této kružnici.

Nyní je potřeba si vypočíst procento bodů ke kružnici přiřazených k celkovému počtu bodů kružnice. Pokud je procento vyšší než je zvolený práh, kružnici si uložíme do pole pro pozdější vykreslení. Kdybychom ji vykreslili hned, museli bychom s každou kružnicí přistoupit k obrázku ke kreslení vykreslit a kreslení uzavřít. Tyto operace jsou časově náročné a proto je vhodnější k němu přistoupit jednou, vykreslit všechny kružnice a poté kreslení uzavřít. Čas se tak radikálně zmenší.

Posledním krokem je již vykreslení kružnice.

Elipsa

Hledání elips není rozděleno podle toho, zda uživatel požaduje obecnou elipsu nebo elipsu v normální poloze. Rozdíl bude popsán níže.

Výpočet křivky

Opět se podíváme, zda již elipsy uloženy nemáme. Pokud ano, tak jen vykreslíme. V opačném případě určíme počet iterací, které budeme provádět. Tento počet však budeme adaptivně měnit dle vývoje výběru bodů. Poté již můžeme vybírat náhodně body a počítat elipsy. Jelikož nebereme v potaz rozdíl mezi typy elips, tak vždy vybíráme 5 náhodných bodů. Tuto pětici bodů dosadíme do funkce, která nám vypočte pomocí metody nejmenších čtverců elipsu. Takto získanou elipsu poté prověříme testu, zda se nejedná o elipsu, která je v normální poloze. Pokud ano, tak si nastavíme o této informaci příznak. Tento příznak využijeme mimo jiné hned v následujícím rozhodnutí. Do bloku výpočtu inlierů totiž vstoupíme pouze tehdy, když vyhledáváme všechny (obecné) elipsy nebo máme

nastaven zmíněný příznak a zároveň k těmto podmínkám musí platit to, že vypočtená elipsa velikostně splňuje uživatelovy kritéria.

Výpočet inlierů

Výpočet inlierů u elipsy je už založen na jiném principu než v předchozích případech a to proto, že přesně nemůžeme analyticky vypočítat vzdálenost bodu k nejbližšímu bodu elipsy. Metody jsou pouze odhadové. Tyto metody jsou jen o trochu rychlejší než metoda implementovaná, ale odchylky mají poměrně výrazné. Proto jsem implementoval metodu, která vypočte vzdálenosti ke všem bodům elipsy a vrátí vzdálenost nejmenší. Je to metoda hrubé síly, ale elipsy nemají tolik bodů, aby docházelo k velkému zdržení a vrácená hodnota je přesná. Tato funkce se však provede pouze tehdy pokud bod leží ve vzdálenosti menší než je úhlopříčka os elipsy. Tím se výpočet ještě více urychlí. Toto provedeme pro veškeré body. Spočítáme tedy vzdálenost a popř. je zařadíme mezi inliery. Po tomto výpočtu se zeptáme, jestli je dostatečný práh pro vykreslení dané elipsy a pokud ano, tak elipsu vykreslíme.

Přepočet iterací

A jako poslední krok přepočteme počet iterací. To se děje obdobně jako u kružnice. Pomocí váženého průměru procent příslušnosti inlierů k obvodu vypočtené elipsy přepočteme počet iterací, které budeme provádět a pokud je tento počet vyšší než je současná hodnota, tak počet iterací zvýšíme. Počet iterací zvyšujeme o jednu i tehdy, když po vybrání náhodných bodů vypočteme elipsu s takovými parametry, které nevyhovují hodnotám zadaných uživatelem. A opět abychom nemohli zvyšovat počet iterací do nekonečného běhu programu. Proto je definována maximální hodnota proveditelných iterací.

Kubické Bézierovy křivky

Počte iterací

Počet iterací by musel být enormní, a proto je iterací naimplementováno méně, takže nemusí dojít k vyhledání nejlepšího řešení. Kubické Bézierovy křivky jsou implementovány spíše jako pokus funkčnosti, který je založen na myšlence pokus-omyl, avšak trošku jiném než samotný RANSAC, protože vybíráme i body, které neleží přímo na křivce.

Výpočet křivky

Po vypočtení počtu potřebných iterací můžeme vyhledávat tyto křivky. Náhodně vybereme 2 body z bodů vrácených Cannyho hranovým detektor jako hrana a další 2 body, které jsou vybrány náhodně z celého obrazu. To, že tyto body dva body jsou vybírány z obrazu, je podstatná informace, protože některé křivky nemusí být nalezeny. Je to způsobeno tím, že řídící polygon Bézierovy křivky je potenciálně tak velký jako obrázek. Ale některé křivky by potřebovaly řídící polygon větší. Proto jak lze vidět na obrázku 20, nejlepší případ nalezené křivky nepokrývá skutečnou křivku, ač druhý a třetí řídící bod byl zvolen na pokraji obrázku. V tomto případě by bylo potřeba řídící polygon rozšířit za hranice obrázku.

Výpočet inlierů

Výpočet inlierů se děje úplně stejně jako u elipsy. Prohledáváme tedy celý obrázek a hledáme v něm nejbližší bod kubické Bézierovy křivky k bodu hrany, který přepočítáváme, zda jej zařadit mezi inliery.

a) **b)**

Obrázek 20: a) Původní obrázek, b) Obrázek s detekovanou kubickou křivkou.

7 Implementace

V této kapitole jsou popsány implementační prvky programu. Takže se zde čtenář může dočíst bližší informace ohledně použitých knihoven a frameworků. Také je zde podkapitola, která obsahuje stručný popis hlavních funkcí, které se musí zavolat pro výpočet křivek.

7.1 Popis použitých knihoven a frameworků

V této krátké kapitole byly použity informace za zdrojů [10], [11] a [12].

AForge.NET Framework

Při vynucení jak metody RANSAC, tak i Houghovy transformace se na obrázek aplikuje funkce *CannyEdgeDetector* z rodiny knihoven *AForge.NET Framework*, která nejprve obrázek převede na odstíny šedi podle rovnice 12 a poté na něj aplikuje Cannyho hranový detektor dle parametrů zvolených uživatelem.

AForge.NET Framework je systém knihoven pro pomoc při vytváření programů v oblasti počítačového vidění a umělé inteligence.

Tento rámec knihoven se dělí na:

a) **AForge.Imaging – knihovna pro běžné zpracování obrázku (filtry, …)**
b) AForge.Vision – knihovna pro počítačové vidění
b) AForge.Neuro – knihovna pro neuronové sítě
c) AForge.Genetic – knihovna pro evoluční algoritmy
d) AForge.MachineLearning – knihovna pro učení počítače
e) AForge.Robotics – knihovna obsahující podporu pro práci s roboty
f) AForge.Video – knihovna pro práci s videem

Cannyho hranový detektor je obsažen v rámci balíčku *AForge.Imaging*, konkrétně tedy v *AForge.Imaging.Filters*. Vytvoříme si tedy novou proměnou s datovým typem *IFilter,* který bude obsahovat objekt *CannyEdgeDetector* a ten následně použijeme na vstupní obrázek.

Emgu CV

Jedná se o obal na *Open CV*, který umožňuje multiplatformnost a mnohojazyčnost.

Z tohoto balíku knihoven jsou využity 2 funkce: funkce pro výpočet elipsy pomocí metody nejmenších čtverců z několika (v aplikaci z pěti) bodů a funkce pro vykreslení elipsy (jen pokud elipsy již nemáme uloženy – to se elipsy vykreslují naimplementovanou funkcí).

Microsoft .NET Framework

Celá aplikace využívá tohoto frameworku. Jedná se o platformu od Microsoftu pro tvorbu a běh aplikací pod OS Windows, která není jazykově omezena.

7.2 Popis hlavních funkcí pro detekci

Houghova transformace

Ve třídě `H_form`, která celá obsahuje výpočet Houghovy transformace, se nachází funkce

```
public Bitmap fHough(Bitmap obrazek, Nastaveni n, ref string v),
```

která se volá pro výpočet detekce touto metodou. Vstupem této hlavní funkce je bitmapa, ve které se mají křivky vyhledat, dále struktura s nastavením vyhledávání a jako poslední je odkazem předaný řetězec znaků, do kterého bude vypisován textový výstup aplikace. Návratový objekt z této funkce je bitmapa, tedy obrázek, ve kterém jsou již detekovány hrany dle zvoleného nastavení.

Funkce nejprve převede obrázek pomocí Cannyho hranového detektoru, poté zaalokuje potřebnou velikost polí, kterým následně vypočte hodnotu (naplní akumulátory) a poté tyto akumulátory filtruje. Jako poslední krok je vykreslení nalezených křivek do připravené bitmapy.

RANSAC

Třída `R_form`, věnující se metodě RANSAC obsahuje funkci:

```
public Bitmap fRANSAC(Bitmap obrazek, Nastaveni nastaveni, ref string
vypis).
```

Tato funkce se volá pro detekci křivek v obraze metodou RANSAC a její vstupní parametry jsou stejně jako u předchozí funkce bitmapa obsahující vstupní obrázek, struktura s nastavením hledání a řetězec znaků do kterého se budou vypisovat textové výstupy. Výstup této funkce je bitmapa obsahující obrázek s detekovanými křivkami.

Funkce postupuje takto: převede obrázek pomocí Cannyho hranového detektoru a body, které tento detektor označí jako hranové, vloží do seznamu bodů. Poté provede celé vyhledání pro přímky. Vybírá tedy ze seznamu bodů 2 náhodné body – vytvoří z nich přímku, vypočte počet inlierů a případně ji vykreslí a toto opakuje tolikrát, kolik je iterací. Analogicky postupuje i pro další křivky – kružnici, elipsy a kubické Bézierovy křivky.

8 Srovnání a vyhodnocení metod

Tato kapitola se bude věnovat, jak již název napovídá, výstupem aplikace. Bude tedy srovnávat výstupy jednotlivých metod a to podle několika kritérií, které jsou děleny do podkapitol.

8.1 Vyhledání

U vyhledání křivek je toto kritérium největší – zda algoritmus spolehlivě detekuje daný typ křivek. Pokud toto kritérium nesplňuje, tak daná metoda není příliš spolehlivá.

Houghova transformace detekuje křivky vždy, jelikož prochází veškeré možnosti. Dalo by se říci, že po této stránce je HT ideální, ale není tomu tak. Houghova transformace implicitně nepracuje s tolerancí při vyhledání. A při vyhledání a počítání parametrů dané křivky dochází k zaokrouhlení na celá čísla tak, aby se mohla jejich hodnota v poli inkrementovat. Takže dochází k takovému efektu, že určitá křivka je v poli díky zaokrouhlení na několika polích (určitý parametr je jen o jedna posunut) a tím pádem nemusí ani jedna buňka splňovat hodnotu potřebného prahu nebo tento práh splňuje hned několik křivek (ač jejím původem je jen jedna). Pokud dojde ke druhému případu, tak nám to příliš nevadí, jelikož paměťový prostor nejprve podrobíme vyhledání lokálních maxim a tím nám zůstane jedno řešení. Pokud však dojde k prvnímu případu, je situace o dost horší. K vyhledání nedošlo, ač je křivka přítomna. Proto se pro Houghovu transformaci musí nastavovat menší práh. Druhá varianta, která není implementována, ale je možná je taková, že počítáme s určitou tolerancí – jak je tomu například u metody RANSAC. Tolerance by se do Houghovy transformace zapracovala tak, že při prohledávání paměťového prostoru bychom se nedívali na jednotlivé buňky polí, ale i na jeho okolí. Toto okolí by bylo tak velké, jak velká by byla nastavena tolerance. A pokud by součet hodnot těchto paměťových buněk přesáhl hodnotu nastaveného prahu, křivku bychom vykreslili. Tato funkce by však zdržela už tak pomalý výpočet, který se pomocí Houghovy transformace děje. A není nezbytná pro správný chod této metody.

Metoda RANSAC je založena na matematickém předpokladu, že nemusíme procházet veškeré kombinace, které se nám nabízejí, abychom zjistili nejlepší řešení, ale stačí nám provést jen tolik výpočtů, které si podle pravděpodobnosti můžeme vypočítat. V podstatě to znamená, že tato metoda postupuje metodou náhodných pokusů. Dle pravděpodobnosti bychom měli vždy objevit všechny křivky. To však není v praxi úplně pravda. Můžeme vybírat stále velmi podobné vzorky a k požadovanému konci by to nakonec nevedlo. Záleží tedy poměrně hodně i na generátoru náhodných čísel. Pokud tedy uvažujeme, že máme dobrý generátor náhodných čísel a tím tedy, že je velká pravděpodobnost, že najdeme veškeré vhodné řešení, tak se nám naskýtá další problém. Druhý problém u metody RANSAC je takový, že i když vybereme všechny vzorky z dané křivky, tak nakonec nemusíme získat její opravdové parametry. Tato situace nastává tehdy, když jsou vzorky příliš blízkou sebe a jelikož má každý pixel určenou pozici v celých číslech, může dojít k odlišnému výpočtu od skutečných hodnot. Takže pokud již uvažujeme, že vybere vždy alespoň jednou všechny vzorky z hledané křivky, tak to neznamená, že dostaneme její skutečné parametry a tím tedy tuto křivku nenajdeme. Jestliže však vybereme vzorky dobré, výpočet parametrů je taktéž správný, tak můžeme nastavit práh na poměrně vysokou hodnotu a křivka stále bude vykreslena. To se stane, pokud budeme mít toleranci nastavenou alespoň na hodnotu 1. Pokud bychom toleranci nechtěli, nemusí dojít k překročení prahu kvůli rozdílu zaokrouhlení hodnot vzdáleností a zaokrouhlení při vykreslení hledané křivky.

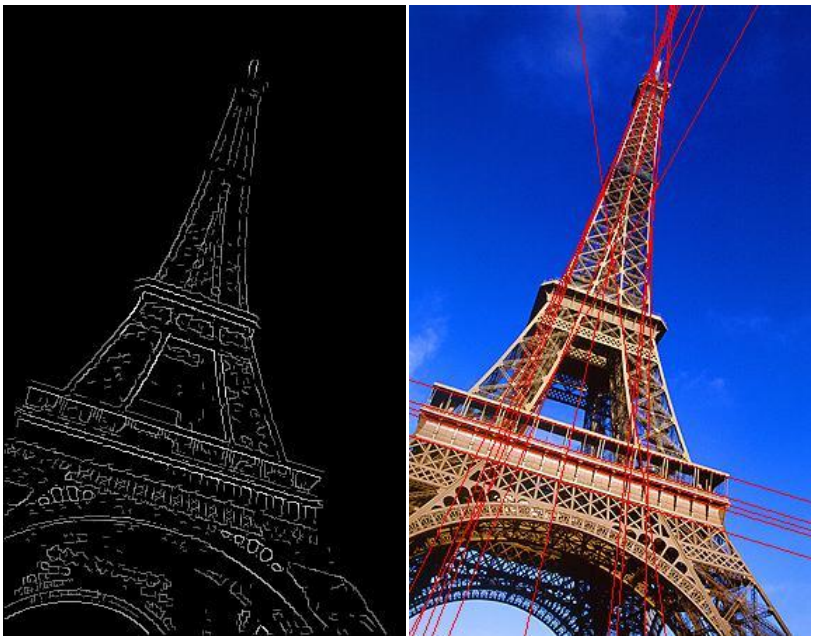
Houghova transformace nám tedy nemusí vrátit veškeré křivky, které se v obraze nalézají, a potřebuje mít práh nastaven poměrně nízko. Zatímco metoda RANSAC práh může mít nastaven poměrně vysoko a existuje malá pravděpodobnost, že hledanou křivku vůbec nenalezne. Toto lze dobře vidět na obrázcích 21 a 22.

Nejprve na obrázku 21a) lze vidět výstup Cannyho hranového detektoru (tedy výstup metody RANSAC) a na odpovídajícím obrázku jen s písmenným označením b) je zobrazen výstup metody RANSAC. Práh pro zobrazení přímek byl nastaven na 120 pixelů, kde velikost obrázku byla 287 x 450 pixelů. Bylo nalezeno několik hlavních přímek. Výstup je odpovídající.

a) **b)**

Obrázek 21: a) Cannyho výstup, b) Výstup metody RANSAC (přímky).

Na obrázku 22c) čtenář může nyní vidět výstup pomocí Houghovy transformace se stejným nastavením, které bylo provedeno u metody RANSAC. Nalezeny byly pouze 3 přímky. Jsou to sice ty nejviditelnější, ale jak vidíme RANSAC byl úspěšnější – díky zabudované toleranci. Proto jsem změnil parametry pro zpracování Cannyho hranovým detektorem. Snížením horního prahu se docílí většího počtu výstupních bodů, které jsou přiřazeny k hraně. Takto je bodů pro zpracování více než dvojnásobně. I přesto jsem snížil i práh pro vykreslení přímek na 109 – tedy zhruba o 10%. Výstup takového nastavení je na obrázku 22b). Tento obrázek je potom trochu podobný výstupu RANSACu. Ale jelikož už bylo oproti RANSACu více bodů pro zpracování, tak jsou detekovány i další přímky. To neznamená, že RANSAC některé nenašel – nemohl je najít, protože v jeho vstupu nebyly.



a) **b)**

c)

Obrázek 22: a) Výstup Cannyho hr. Detektoru, b) Výstup Houghovy transformace (přímky) pro obrázek 22a), c) Výstup Houghovy transformace (přímky) pro 21.a).

8.2 Falešná detekce

To co bylo pro metodu RANSAC výhodou v předchozí podkapitole 8.1 Vyhledání, tak v této podkapitole je to pravý opak. Jedná se o toleranci.

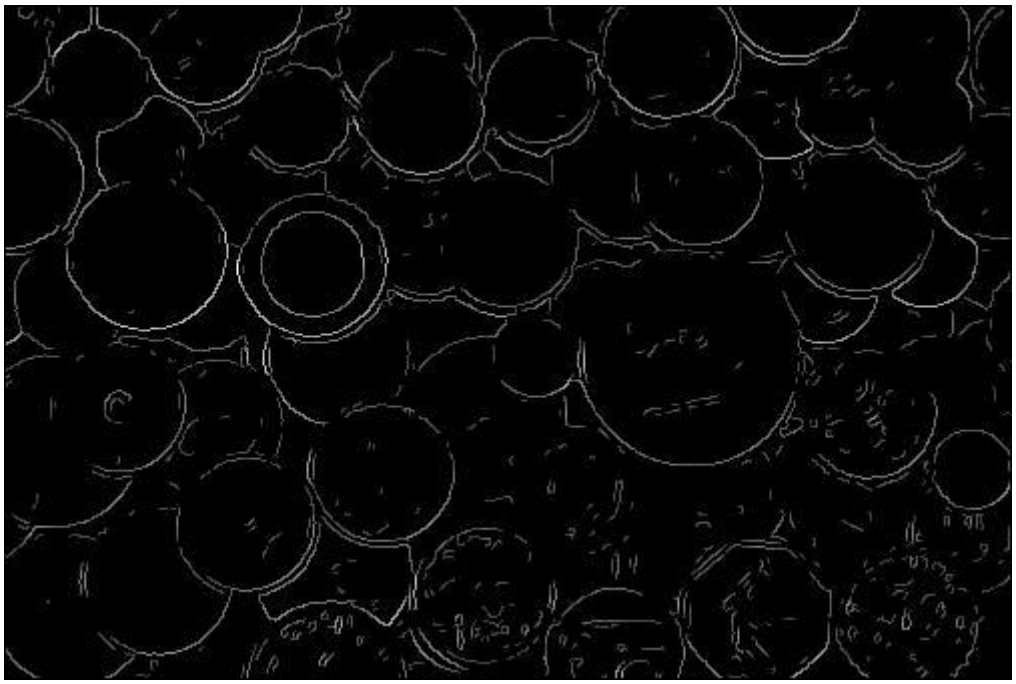
Houghova transformace totiž projde veškeré možnosti a maximálně se nám díky zaokrouhlení náležících bodů stane to, že křivka není vyhledána. Ale nemůžeme ke křivce přiřadit cizí bod, který k ní nepatří. Takže falešná detekce je z tohoto hlediska vyloučena. Může se však stát, že nevhodnou (malou) velikostí prahu pro vykreslení křivky je vykresleno příliš mnoho křivek, protože je splněn

jejich práh. To se však nedá považovat za mylnou detekci. V takovém případě by měly všechny metody stejné výsledky.

RANSAC už ve falešné detekci tak dobré výsledky nemá. Díky tolerance mohou být totiž do k inlierům vypočtené křivky započteny i body, které ke křivce nepatří. To se samozřejmě stává hlavně tehdy, pokud obrázek obsahuje velké množství bodů. Pokud bychom například nechali nastavení Houghovy transformace popsané v předchozí kapitole jejíž výstupem je obrázek 22b), tak u RANSACu by výstup vypadal zcela jinak. Pro toleranci 1, která je pro RANSAC nejvhodnější by byl obrázek celý (až na jeden pixel) zabarven barvou pro vykreslení přímek, kterých je nalezeno téměř 30000. Což je ohromný rozdíl oproti tomu, kdyby tolerance byla nastavena na 0. To potom algoritmus nenašel ani jednu přímku. RANSAC tedy potřebuje pro něj vhodný vstup. Jinak může detekovat spoustu křivek, které v obrázku nejsou.

Pro lepší pochopení a vizuální představu je na následujícím obrázku – tedy na obrázku 23 další příklad. Na obrázku 23a) je výstup z Cannyho hranového detektoru. Na následující části, tedy na 23b) je zobrazen výstup Houghovy transformace. Můžeme vidět, že jeho výstup je poměrně kvalitní. Zatímco metoda RANSAC již tak dobrý výstup při stejných parametrech nemá – obrázek 23c). Najde sice mnoho kružnic, které jsou dobře, ale mimo ně najde mnohem více kružnic, které splňují požadavky jen kvůli své toleranci, která byla nastavena pouze na hodnotu 1. Aby k falešné detekci u této metody nedocházelo, musí být nastaveny správné parametry pro Cannyho hranový detektor a tolerance by měla být nastavena na hodnotu 1. Pokud se tolerance nastaví větší, zvětšuje se tím i pravděpodobnost právě falešné detekce. Pokud se nastaví tolerance na nulu, k detekci křivek dojde, ale nebude jich vyhledáno mnoho.

Srovnání pomocí parametru falešné detekce tedy pro Houghovu transformaci vychází lépe, jelikož u ní k falešné detekci v podstatě nedochází.



a)

b)

c)

Obrázek 23: a) Výstup Cannyho hranového detektoru – vstup pro metody vyhledání, b) Výstup Houghovy transformace pro 23a), c) Výstup metody RANSAC pro 23a).

8.3 Přesnost vyhledání

Přesností vyhledání se myslí to, že pokud algoritmus najde určitou křivku, tak jak skutečně přiléhá ke křivce na obrazu, pokud nedošlo k mylné detekci.

V této, stejně jako v předchozí, podkapitole vítězí opět Houghova transformace. U této metody může dojít k nepřesnosti pouze tehdy, pokud několik sousedních buněk naplněného akumulátoru

obsahuje stejnou hodnotu a my za výsledek nezvolíme středovou buňku. To někdy může být obtížné, kvůli vícedimenzionálnímu uskupení těchto buněk. Jinak je přesnost určena celými čísly. Takže k malé odchylce dochází, ale ta není příliš markantní.

U metody RANSAC přesnost už je poněkud horší, i když také není markantní a je přijatelná. Nepřesnost je způsobena dvěma efekty:

1) Všechny body, které vybereme, nenáleží jedné křivce, ale přesto je křivka uznána za platnou, protože splňuje limit prahu. Tento efekt je ale popsán ve falešné detekci. Ta je ovšem i součástí přesnosti a proto je zde o ní malá zmínka.
2) Body jsou vybrány z jedné křivky. Jelikož se však vybírá minimální počet bodů, které jsou potřeba pro zjištění parametrů hledaného typu křivky, tak kvůli aproximaci může dojít ke zjištění jiných parametrů než jsou skutečné. Tyto parametry mohou být pouze o trochu jiné a tak křivka může splňovat kritéria pro její uznání. Tento efekt se dá odstranit např. tím, že budeme vybírat více bodů než je minimum pro danou křivku. Poté by výpočet byl poměrně přesný, ale toto řešení má jednu nevýhodu - značně by se zvýšil počet potřebných iterací. A odchylka není příliš velká – zanedbatelná.

8.4 Paměťová náročnost

V této části srovnání je vítěz jasný. Houghova transformace je extrémně náročná na paměť, zatímco metoda RANSAC nevyžaduje v podstatě žádné místo.

Paměťová náročnost u Houghova transformace roste exponenciálně s počtem parametrů, kterými musíme popsat hledanou křivku. Nejlépe závislost množství parametrů na potřebném místě půjde vidět graficky resp. v tabulce. Jelikož znázornění hodnot v grafu by nebylo téměř vůbec čitelné a to i s logaritmickou osou, protože se hodnoty velmi rychle zvětšují.

Tabulka tedy obsahuje závislost mezi počtem parametrů, jejich rozdíl maximální a minimální hledaných hodnot, k potřebné paměti. Pro křivku popsanou 2 parametry (např. přímka) je potřebná paměť únosná i pro větší obrázek. Pro křivku popsanou třemi parametry (např. kružnice) už to tak jednoznačně neplatí. Pokud parametry mohou nabývat tisíci hodnot, tak bychom potřebovali téměř 2GB paměti. Což možné je, ale rozhodně ne pro každý počítač. A s každým dalším přidaným parametrem se potřeba místa stále zvětšuje. Pro zajímavost je uvedeno i potřebné místo kdybychom měli křivku popsanou deseti parametry (to může být například konkrétní kubická Bézierova křivka, která může být jakkoliv pootočena a jakkoliv zvětšena). Pokud by parametry mohly nabývat tisíci hodnot (do značné části to závisí na velikosti obrázku), tak bychom potřebovali 1 654 361,23YB, což je 1000^{10}*2B = 2 000 000 000 000 000 000 000 000 000 000B.

Počet parametrů	Rozdíl mezi min. a max. hodnotou parametru	Potřebná paměť	
2	100	20000	B
2	500	488,28	kB
2	1000	1953,13	kB
3	100	1953,13	kB
3	500	238,42	MB
3	1000	1907,35	MB
4	100	190,73	MB
4	500	116,42	GB
4	1000	1862,65	GB
5	100	18,63	GB
5	500	56,84	TB
5	1000	1818,99	TB
10	100	173,47	EB
10	500	1615,59	YB
10	1000	1654361,23	YB

Tabulka 2: Výpočet potřebného místa pro HT.

Samozřejmě můžeme algoritmus modifikovat například tak, jak je to popsáno v této práci, ale to by potom velmi ovlivnilo časovou náročnost. Takto vypočtené hodnoty tedy platí pro ideu Houghovy transformace.

RANSAC je na rozdíl od HT paměťově velmi nenáročný. V podstatě nepotřebuje žádné dodatečné místo, ale aby mohl pracovat rychleji, tak si může uložit např. do pole body, které Cannyho hranový detektor označil za součást hran. Takže celková paměťová náročnost by tedy závisela na množství bodů. Maximální hodnota by však mohla být šířka obrázku x výška obrázku / 2 (hrany mohou pokrývat maximálně polovinu obrázku) * 2 * 32bitů (pozice bodu – x, y a každá je uložena na integeru). Což nám při obrázku 1000x1000 dává jen pár MB.

Myslím si, že slovní shrnutí paměťové náročnosti je dostačující a nejvýmluvnější a proto v této podkapitole nebudu uvádět žádné obrázky, které by čtenářovi ani neposkytly více informací než výše zmíněné.

8.5 Časová náročnost

Časovou náročností se samozřejmě myslí to, jakou dobu potřebný algoritmus potřebuje k jeho celkovému průchodu.

V kritériu časové náročnosti již není jednoznačně vítěz či poražený, ač jsou misky vah nakloněny na jednu stranu. Velmi záleží, jaké křivky hledáme, resp. kolika parametry je daná křivka popsána. V podstatě platí, čím méně parametrů, tím je rychlejší Houghova transformace. To ovšem platí jen do tří parametrů. Pokud je křivka popsána třemi parametry, tak jsou výsledky většinou srovnatelné, pokud jsou alespoň trošičku omezeny velikosti křivky. Pokud ne, RANSAC je o mnoho rychlejší. Takže ty křivky, které jsou popsány více než třemi parametry, je vhodnější, alespoň z časového hlediska, vyhledávat pomocí metody RANSAC.

V následující tabulce (tabulka 3) jsou uvedeny naměřené časy, které byly naměřeny na určitých obrázcích při uvedeném počtu bodů vrácených pomocí Cannyho hranového detektoru. Tyto časy jsou jen orientační a mohou se měnit v závislosti na rozmanitosti, rozlišení obrázku a výkonu procesoru.

Tyto testy byly provedeny na AMD Athlon™ 64 X2 Dual, 2,01GHz. Časy také nejsou zcela stejné, pokud algoritmus pustíme se stejnými hodnotami – vyskytují se mírné odchylky. Časy v jednotlivých řádcích jsou uvedeny pro zhruba stejný počet nalezených křivek, aby nedošlo k ovlivnění měření testu kvůli ukládání do struktur a vykreslování křivek. Tento přibližný počet je v tabulce uveden také.

Dva parametry - přímky

Část tabulky, která se vztahuje k dvěma parametrům, je viditelně větší než ostatní části. Je to způsobeno tím, že další části mají zabudováno adaptivní určení iterací, zatímco pro přímky statický výpočet. Zde je výpočet přijatelný a adaptivní určení by mohlo výsledky jen zhoršit. Jelikož zde nemáme adaptivní určení iterací, můžeme měnit práh a můžeme očekávat výstup o určitém počtu křivek. To v druhém případě nemůžeme, protože pokud změníme práh, může to vézt i k snížení i ke zvýšení iterací, což znamená změna času a nemožnost odhadnout počet výstupních křivek. Takže příklady rozdílnosti nalezení křivek jsou uvedeny jen u přímek – tedy u křivek popsaných dvěma parametry. Test byl prováděn na obrázku 24a), který má rozlišení 287x450 a je poměrně vhodný pro hledání přímek.

Jak lze vidět Houghova transformace je pro přímky velmi rychlá. Až když výstupních bodů z Cannyho hranového detektoru bylo 30000, tak HT potřebovalo na zpracování o něco více než 1s. A počet nalezených přímek ovlivnil čas jen málo a jedná se o zdržení při vykreslení, jelikož zpracování je prováděno pořád stejně.

U RANSACu však můžeme pozorovat větší časové změny. Pokud je bodů pro zpracování málo, rychlost je dobrá. Pokud je však bodů 10000, tak už výpočet trvá pár desítek sekund – záleží na počtu výstupních přímek. Tedy kolik jich chceme najít. Výsledek RANSACu v tomto případě už není tak kvalitní jako u HT, jelikož často nalézá přímky s velmi podobnými parametry.

Tři parametry - kružnice

Měření času u kružnic je rozděleno do několika částí. Zaprvé podle složitosti, resp. uspořádání obrázku a zadruhé podle toho jak byly omezeny parametry (velikost) křivky. To má totiž zásadní vliv na časovou náročnost hlavně u Houghovy transformace. U RANSACu se čas také změní, ale tato změna může být různá – může se jednat o zmenšení i o zvětšení časové potřeby pro výsledek. Především však záleží na správném nastavení tolerance a prahu. Pro metodu RANSAC tedy není v tabulce rozdělen výsledek pro omezené a neomezené parametry.

Test na obrázku 24b), který obsahuje spoustu kružnic podobné velikosti, jež jsou po celém obrázku, dopadl pro RANSAC následovně: čas se markantně zvedal s počtem bodů pro zpracování. Pokud je bodů do několika tisíc, test trval několik sekund. Poté se čas velmi strmě zvedal až nakonec, když byl Cannyho hranový detektor nastaven tak, aby našel veškeré hrany (50000 bodů na obrázku o velikosti 503x337pixelů – takže 30% pixelů bylo vyhodnoceno jako hrana), čas strmě klesl a výsledek byl neuspokojivý. Je to však pochopitelné. RANSAC při každé iteraci, ať vybral jakékoliv body, přiřadil mnoho bodů jako inliery – podmínka splněna – adaptivní počítání iterací proto mnoho iterací nepřidávalo. Teoreticky k tomu nebyl důvod. Proto pokud je obrázek již přesycen body, RANSAC selhává a nenalézá uspokojivé řešení. Na výsledek alespoň nemusíme čekat dlouho.

Pokud poloměr kružnice byl znám, což zhruba můžeme odhadnout, tak velmi dobrých výsledků dosahovala Houghova transformace. Tato metoda byla ještě rychlejší než RANSAC a nalézání kružnic bylo lepší. Když však poloměr zadán nebyl, čas se drasticky změnil. V podstatě se čas pro Houghovu transformaci zvýšil 9x. V tomto případě velmi záleží na velikosti obrázku, jelikož maximální poloměr byl nastaven na velikost úhlopříčky. Pokud by byl tedy obraz větší, doba by byla také větší. Na druhou stranu, pokud jsme provedli tento dlouhý výpočet, máme již veškeré kružnice uloženy v paměti a můžeme jen měnit prahy.

Další test byl proveden opět pro kružnice. Obrázek byl však zcela jiný (24c)) – obsahuje několik kružnic, které jsou rozmístěny na dvou místech v obrázku, který má rozlišení 779x471pixelů. Parametry byly poměrně omezeny, i když jejich rozpětí bylo větší než v předchozím případě (zpomalení pro HT). Také velikost obrázku je větší.

Z toho, že je obrázek větší, plyne pro HT jediné – další zpomalení. Protože se pokouší do každého bodu vložit střed kružnice. Houghova transformace byla oproti předchozímu obrázku pomalejší zhruba 4 až 7 násobně. Záleželo to na počtu zpracovávaných bodů. Čím více jich bylo, tím více byla pomalejší. Když parametry omezeny nebyly vůbec, tak se výpočet zpomalil zhruba trojnásobně. V testu na předchozím obrázku zpomalení bylo markantnější, protože diference minimálního a maximálního poloměru byla menší než v tomto případě. Takže pro omezené parametry byl výpočet rychlejší. U tohoto obrázku, jelikož je větší, jsme nemohli zvolit úplné rozpětí poloměru od téměř minimálního po maximální. Na to už by nestačila paměť. Proto poloměr byl nastaven na maximální možnou míru.

Metoda RANSAC dopadla však jinak. Časy měl většinou rychlejší než u předchozího obrázku (a výstupy také byly lepší). Dokonce i pro mnoho bodů, kde v předchozím testu RANSAC selhal, se tady osvědčil. Body jsou totiž z velké části přítomny na kolech – jen ve dvou oblastech. Je tedy velká pravděpodobnost, že vybere všechny 3 body z jednoho kola a vytvoří tak určitou kružnici, která sice nemusí kopírovat ani jednu kružnici na kole, ale velmi se jí blíží. Opět tedy adaptivní počítání iterací moc nezvyšuje tento počet a výpočet byl hotov za několik málo sekund. Ale záleží také na dobrém nastavení tolerance a prahu. Pokud je zvolíme nevhodně – třeba malý práh bez tolerance – výpočet může trvat třeba několik desítek minut.

U všech předchozích testů platí jedno, pokud známe přibližné parametry hledaných křivek a tyto parametry mají poměrně malé rozpětí (desítky), je vhodnější použít Houghovu transformaci. Ta má ještě jednu výhodu – pro přímky i pro kružnice si celý akumulátor nechává v paměti, dokud se nezmění nějaký parametr, který by hodnoty v akumulátoru mohl ovlivnit. Takže po načtení můžeme změnit práh, aniž by došlo k novému přepočtu. To u RANSACu není možné. Z časového hlediska je to výborné, protože můžeme po jednom načtení jen upravovat práh a hledat jeho ideální hodnotu.

Čtyři parametry – elipsy v normální poloze

Poslední test na společném obrázku byl proveden s elipsami, které mají osy rovnoběžné s osami obrázku. Tento test byl proveden na obrázku 24d). A pro Houghovu transformaci se zde mění jedna podstatná věc – akumulátor již není uložen a nemůžeme jen změnit práh pro vykreslení bez přepočtu.

Pokud parametry zůstanou omezeny na poměrně malé rozmezí, tak je Houghova transformace opět rychlejší než metoda RANSAC. Výpočet i pro poměrně malé obrázky však trvá i několik minut. Většinou bohužel neznáme velikost hledaných křivek anebo je jejich rozptyl větší než jen několik jednotek. V tom případě je lepší použít metodu RANSAC. Pokud je rozptyl parametrů velký – Houghova transformace je nepoužitelná. Tedy použít jí můžeme, výsledky obdržíme solidní, ale z časového se jedná o beznaděj. U nejhoršího případu, kdy jsme parametry neznali a bodů pro zpracování bylo 10 000, vyhledávání pro Houghovou transformaci trvalo více než 27hodin a řešení ještě nebylo nalezeno.

Zhodnocení časové náročnosti jednotlivých metod může být tedy následovné: Pokud je počet parametrů roven dvěma, tak je výhodnější použít Houghovu transformaci. Jestli jsou hledané parametry tři nebo čtyři, tak bychom měli jejich velikost omezit pokud možno nejvíce. Jestli je rozpětí minimální a maximální velikosti jednotlivých parametrů stále vysoké, tak je vhodnější použít metodu RANSAC. V opačném případě doporučuji použít opět Houghovu transformaci. Pokud by bylo hledaných parametrů více než 4, Houghova transformace už by vhodná nebyla – pokud bychom

nehledali konkrétní křivku a ne jen její obecný vzor. V tom případě je vhodnější metoda RANSAC. Ta nejlépe pracuje, pokud v obrázku není příliš mnoho nalezených hran. Proto je lepší zvolit parametry Cannyho hranového detektoru vhodně. Výpočet trvá několik desítek minut a to v případě, že je bodů pro zpracování poměrně málo. Pokud je jich více, tak výpočet trvá neuvěřitelných několik hodin a to pro obrázek veliký jen 253x210 pixelů. Navíc již elipsy nemůžeme kvůli paměťové náročnosti nechávat v paměti, takže při změně prahu se výpočet musí provádět znovu.

počet parametrů	počet bodů	počet nalezených křivek	obrázek	HT		RANSAC
				omezené parametry	neomezené parametry	
2	500	20	24a)	0s		0s
2	500	500	24a)	0s		0s
2	500	1500	24a)	0s		0s
2	3000	30	24a)	0s		0s
2	3000	500	24a)	0s		2,24s
2	3000	1500	24a)	0s		3,43s
2	10000	500	24a)	0s		12,53s
2	10000	3000	24a)	0s		19,34s
2	10000	16000	24a)	0s		35,43s
2	30000	500	24a)	1,23s		2min28s
2	30000	5000	24a)	1,37s		2min49s
2	30000	15000	24a)	1,43s		3min35s
3	1000	1	24b)	5,26s	44,31s	3,7s
3	3000	10	24b)	11,37s	1min 40s	12,20s
3	13000	37	24b)	46,65s	6min 22s	2min 47s
3	50000	5	24b)	2min 28s	22min 41s	11,64s
3	1000	12	24c)	23,76s	1min 20s	2,59s
3	5000	10	24c)	1min 44s	4min 44s	17,2s
3	10000	5	24c)	3min 15s	8min 59s	1min 47s
3	43000	3	24c)	14min 47s	35min 38s	7,56s
4	1000	4	24d)	45,21s	46min 37s	1min
4	3000	8	24d)	5min 52s	2hod 29min	12min 38s
4	10000	13	24d)	7min 29s	>27hod30min	26min 47s

Tabulka 3: Tabulka naměřených časů.

a)

b)

c)

d)

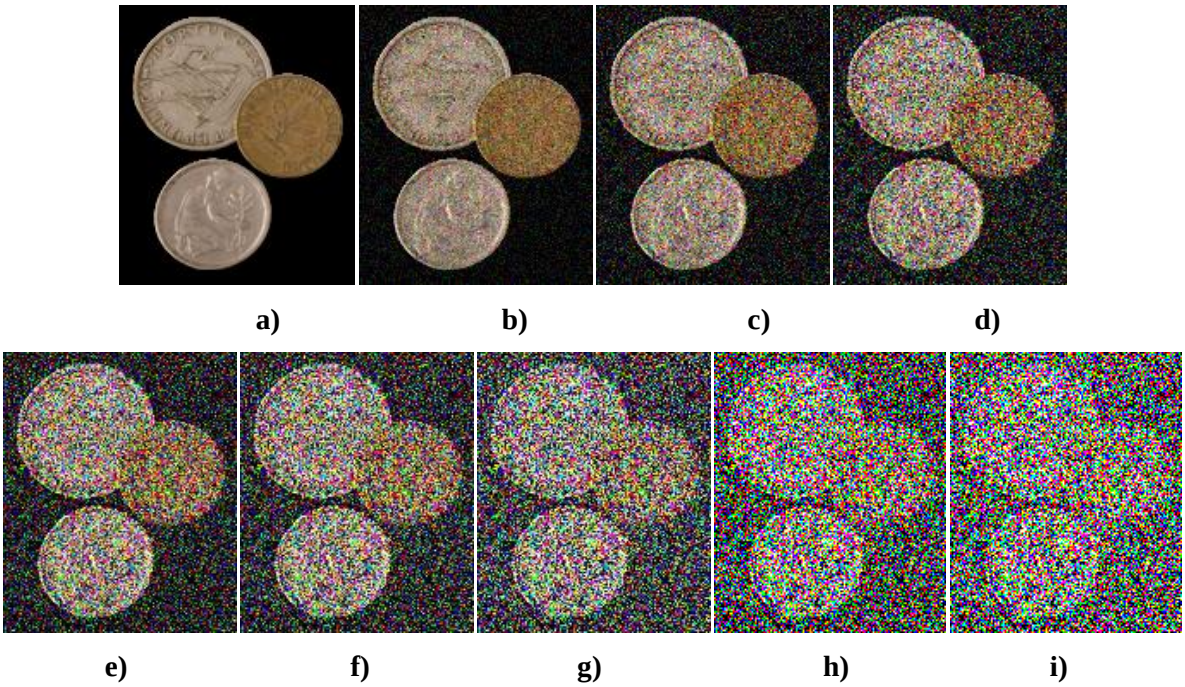
Obrázek 24: Obrázky, které byly zdrojem pro měření času zaznamenané v tabulce 3. a) pro přímky (287x450), b) pro kružnice (503x337), c) pro kružnice (779x471), d) pro elipsy (253x210).

8.6 Odolnost vůči šumu

Další kritérium pro srovnání dvou metod je schopnost algoritmu odolat zašuměnému obrázku. A to ve dvou věcech:

1) Vyhledá všechny křivky, které se v obraze nachází.
2) Nevyhledá křivky, které v obraze nejsou.

Pro test byl zvolen obrázek (obrázek 25a)), který obsahuje 3 mince – kružnice, jež jsou přes sebe překryty. Žlutá mince ani nemá příliš velký kontrast oproti pozadí a tak při vyšším hodnotě horního prahu Cannyho hranového detektoru hrana mince není ani detekována. Do originálního obrázku byl uměle zanesen barevný Gaussův šum. A tím vznikla celá testovací sada. Pokud se čtenář diví, že obrázek 25i) obsahuje 150% šumu a stále je jemně viditelný originální obrázek, tak je to způsobeno tím, že některý bod, který je již šumový, byl překryt jiným šumovým bodem. Takže i přes procento 150 je v obrázku několik originálních pixelů.



Obrázek 25: Obrázky, na kterých byl proveden test na odolnost proti šumu a) originál bez šumu, b) s 10% šumu, c) s 20% šumu, d) s 30% šumu, e) s 40% šumu, f) s 50% šumu, g) se 75%šumu, h) se 100% šumu, i) se 150% šumu.

Jak lze vidět v následující tabulce 4, metody jsou proti šumu poměrně odolné. Veškeré testy, které jsou uvedeny v tabulce pro metodu RANSAC měly nastavenou hodnotu tolerance na 0 a velikost možného poloměru od 25 do 45 pixelů, přičemž kružnice mají poloměry 31, 34 a 40 pixelů. Obě metody měly bezchybné detekce do obrázku 25g), tedy obrázek, kterému bylo přidáno 75% šumu. U tohoto obrázku bychom se již museli rozhodnout, zda detekovat pouze 2 kružnice bez jiných – falešných, nebo jestli detekovat všechny 3 kružnice za cenu několika kružnic, které jsou vytvořeny pouze šumem. Tato situace se opakuje i u dalšího obrázku, který obsahuje 100% šumu. U posledního testovacího obrázku, na kterém již původní kružnice nejdou téměř vidět, byla úspěšnější Houghova transformace. Ta detekovala 2 kružnice správně a 2 špatně. Třetí kružnici detekovat ani nemohla, jelikož horní práh Cannyho hranového detektoru už musel být nastaven na vyšší hodnotu, aby v obrázku nebyly zobrazeny hrany pouze ze šumu a touto hranicí se „objetovala“ i třetí kružnice (obrázek, který metody měly jako vstup, je zobrazen jako obrázek 26). Metoda RANSAC vyhledala pouze jednu kružnici správně a k ní ještě několik špatně.

Obě metody tedy byly na test proti šumu úspěšné. Na posledním obrázku byla úspěšnější Houghova transformace, ale dalo by se říci, že obě metody jsou zhruba stejně odolné vůči šumu a tato odolnost je poměrně vysoká.

Šum [%]	Houghova transformace			RANSAC		
	Horní práh Cannyho hr. detektoru	Práh kružnic	Nalezených kružnic (dobře/špatně)	Horní práh Cannyho hr. detektoru	Práh kružnic	Nalezených kružnic (dobře/špatně)
0	13	53	3/0	100	70	3/0
10	100	51	3/0	100	66	3/0
20	100	43	3/0	100	65	3/0
30	90	40	3/0	90	59	3/0
40	82	41	3/0	90	59	3/0
50	83	48	3/0	90	59	3/0
75	83	52	2/0	100	60	2/0
100	80	51	3/2	140	56	2/0
150	150	27	2/2	150	50	1/4

Tabulka 4: Test odolnosti proti šumu.

Obrázek 26: Výstup Cannyho hranového detektoru pro obrázek 25i) při horním prahu 150.

8.7 Objekty vyhledání – typy křivek

Houghovou transformací můžeme vyhledávat veškeré křivky a objekty jejichž analytický popis známe. Můžeme tedy vyhledat veškeré geometrické tělesa. Pomocí Houghovy transformace můžeme také vyhledávat i křivky, které se nedají analyticky popsat, ale známe jejich přesný tvar. Tyto křivky nebo objekty potom můžeme vyhledávat, i když jsou pootočeny nebo jakkoliv zvětšeny. Obojí znamená o jeden parametr pro vyhledání více.

U metody RANSAC je situace podobná. Dokáže také vyhledávat analyticky popsané křivky. Proto tyto metody jsou také schopny nalezené křivky parametrizovat. Ale tím jeho záběr vyhledávání končí. Nedokáže vyhledat křivky či objekty, které víme jak vypadají, ale matematicky popsat nejdou. Ač se častěji používá ve vyhledání ve 3D prostoru (např. rovin) než HT, tak HT je toho schopna taky.

Ač je RANSAC robustní metodou, tak v této kategorii vítězí Houghova transformace díky větší možnosti vyhledání objektů a křivek.

8.8 Srovnání metod

Tato podkapitola v podstatě jenom shrnuje kapitoly předchozí. Pojednává o rozdílech metod v různých kritériích.

Obě metody jsou založeny na vyhledání křivek, které lze analyticky popsat a tyto křivky poté parametrizovat. Houghova transformace má potom ještě tu výhodu oproti metodě RANSAC, že dokáže vyhledat i křivku, která parametrizovat nelze, ale známe její přesný tvar.

Ve vyhledání křivek je na tom o něco málo lépe také Houghova transformace. Může se stát, že body křivky při počítání se díky potřebnému zaokrouhlení při přepočtu rozprostřou po paměťovém prostoru a křivka tak práh nesplní. Proto se u Houghovy transformace musí nastavovat práh nižší. RANSAC má však problémů s vyhledáním více. Díky tomu, že obraz neprochází systematicky a náhodně vybírá vzorky, nemusí zjistit všechna vyhovující řešení. A i když vybere všechny vzorky ze správné křivky, nemusí vypočítat správné parametry křivky, takže křivka opět nemusí být vyhledána. RANSAC má však jednu výhodu a to toleranci. Ta by neměla být nastavena na velkou hodnotu, ale díky ní dochází k lepšímu vyhledání skutečných křivek.

Další kritérium, podle kterého se metody v této kapitole srovnávají, je falešná detekce. Tedy zda metoda prohlásí, že našla vyhovující křivku, která by takto uznána být neměla. Houghova transformace tento problém opět nemá. Zatímco metoda RANSAC ano, díky své toleranci. Pokud nastavíme vyšší toleranci, máme vysokou pravděpodobnost, že nám nalezne potřebnou křivku (ač ne tak přesně, ale o tom za chvíli), ale nalezne i spoustu falešných křivek, protože díky toleranci budou splňovat potřebný práh.

Přesnost je další vlastnost detektoru křivek. Houghova transformace zjišťuje veškeré možné kombinace křivek v obraze a poté provede vyhledání lokálních maxim. Díky tomuto je metoda přesná. Dochází pouze k malým odchylkám a to tak, že metoda musí pracovat jen s celými čísly. Takže nemůžeme nastavit např. střed kružnice na hodnotu [15,3; 32,8] nebo úhel přímky na 36,4‘. V tomto případě bychom museli nastavit střed na [15; 33] a úhel přímky na 36‘. U RANSACu je situace jiná. Může pracovat i s desetinnými čísly, takže pokud vypočte správné parametry, tak je výstup přesnější než u Houghovy transformace. Bohužel nemůžeme zaručit, že metoda vypočte správné parametry, jelikož počítá s celočíselnými hodnotami pozic pixelů. Tato pozice však byla zaokrouhlena opět z desetinných míst na celočíselnou hodnotu, a tím se dostává u metody RANSAC odchylka do skutečné křivky. Pokud ještě k tomu zvolíme větší hodnotu tolerance, nemůžeme čekat přesný výsledek.

Paměťová náročnost. To je velmi důležité srovnání. Zatím ve všem byla metoda RANSAC maximálně srovnatelná, ale většinou horší, než Houghova transformace. V této části srovnání je vše zcela naopak. Zatímco RANSAC nemá žádné nároky, jen pro lepší a rychlejší práci je vhodné mít uloženy body hrany, tak Houghova transformace má paměťové nároky veliké. Pokud má každý parametr stejnou diferenci maxima a minima, tak je náročnost exponenciální, kde základ je tento rozdíl a exponent je počet parametrů.

U časové náročnosti jednoznačný výsledek není. Záleží na struktuře obrázku, na rozsahu zvolených parametrů, na typu hledaných křivek. Platí však, že pokud rozdíl minima maxima parametrů není příliš velký, tak je lepší zvolit Houghovou transformaci. Pro hledání přímek a kružnic paměť u Houghovy transformace nemažeme, takže při změně prahu ihned vykreslíme potřebné křivky.

Srovnání pomocí odolnosti vůči šumu je nerozhodné vůči oběma metodám. Obě jsou velmi odolné a dosahují dobrých výsledků i při velkém zašumění.

Celkově by se tedy dalo říci, že z výsledků srovnání vyšla lépe Houghova transformace. V podstatě je to pravda, ale má obrovskou spotřebu paměti a pokud vyhledává křivky, u nichž vůbec neznáme rozsah parametrů a bodů pro zpracování je velké množství, tak je ohromně pomalá. Takže záleží na několika parametrech jaká metoda je pro daný obrázek a pro dané křivky vhodná.

9 Závěr

Tato práce se zabývala detekcí křivek v obraze, konkrétněji tedy srovnání metod s parametrickým výstupem – Houghova transformace a metoda RANSAC. Toto srovnání je podrobněji popsáno v kapitole 8 *Srovnání a vyhodnocení metod*. Zjednodušeně by se dalo říci, že Houghova transformace dosahuje lepších výsledků ve většině kritérií. Ale metoda RANSAC není o mnoho horší. Naopak velký a poměrně klíčový rozdíl je v paměťové náročnosti metod, kde Houghova transformace má ohromné nároky, které se exponenciálně zvyšují s každým dalším parametrem navíc. A často by nám bez upravení algoritmu paměť nestačila. Metoda RANSAC paměťové nároky nemá v podstatě žádné – jen si uloží pozici hranových bodů vrácených Cannyho hranovým detektorem. Poměrně velkých rozdílů nabývá i časová náročnost pro zpracování obrázku. Zde však nemůžeme prohlásit ani jednu metodu za lepší – rychlejší. Záleží na několika věcech – struktura obrázku, počet parametrů, které určují hledanou křivku, znalost a rozdíl velikosti maximálních a minimálních hodnot jednotlivých parametrů a počet zpracovávaných bodů. Pokud zvolíme práh či parametry nevhodně, můžeme na výsledek, který nakonec bude neuspokojivý, čekat poměrně dlouho. Ale při správném nastavení je výstup většinou vyhovující. Vyhodnocení ale často trvá i několik minut a to proto, že se vychází z podstaty daných algoritmů a jsou implementovány jen mírné optimalizace. Existují další optimalizace, které však už často nemají za důsledek uspokojivých výstupů.

Algoritmus Houghovy transformace pro hledání elipsy musel být upraven. Tato změna však nesměla změnit základní ideu Houghovy transformace, aby porovnání metod nebylo ovlivněno. Proto myšlenka zjišťování všech možných kombinací parametrů byla zachována. U metody RANSAC došlo také k malé úpravě a to takové, že si na začátku nevypočteme počet potřebných iterací, ale tyto iterace adaptivně zjišťujeme po každém průchodu náhodného výběru bodů.

Další vývoj na aplikaci je tedy možný. Mohly by se zavést optimalizace pro dané metody, které by ovšem často znamenaly omezení výstupu. Popřípadě by se mohly implementovat další, jiné křivky, které implementovány nejsou. Houghova transformace však příliš velký výběr rozšíření křivek bez změnění základní myšlenky nemá. Proto by takto rozšířený program již nemohl být použit pro srovnání těchto metod, protože by se již jednalo o metody hybridní.

Dalším projektem by tedy mohlo být tyto metody skloubit a upravit tak, aby z každé metody byly vybrány ty nejlepší vlastnosti a vznikla tak úplně nová – hybridní metoda.

Literatura

[1] **Španěl, Michal a Beran, Vítězslav.** *Obrazové segmentační techniky.* Brno, Česká republika, Aktualizováno 2006-19-01 [cit. 2008-16-12]. Dostupné na URL: <http://www.fit.vutbr.cz/~spanel/segmentace/.cs.iso-8859-2>.

[2] **Hořčička, Jiří.** *Počítačové zpracování digitálních obrázků – Houghova transformace*. K7 vědecko populární časopis Fakulty mechatroniky TU v Liberci [online]. 2005, roč. 2, č. 4 [cit. 2008-12-16], s. 13-18. Dostupné na URL: <http://k7.tul.cz/download/k7_05_4.pdf>.

[3] **Navrátil, Jan.** *Transformace a RANSAC.* Brno, Česká republika, Aktualizováno 2008-25-11 [cit. 2008-16-12]. Dostupné na URL: <https://www.fit.vutbr.cz/study/courses/POV/private/lectures/pov_09_transformace_a_RANSAC.pdf>.

[4] **Kybic, Jan.** *Active Contours — Snakes* [online]. Aktualizováno 2007-20-11 [cit. 2008-16-12]. Dostupné na URL: <http://cmp.felk.cvut.cz/cmp/courses/ZS1/slidy/snakes.pdf>.

[5] **Kalová.** *Detekce a parametrizace geometrických tvarů* [online]. [cit. 2008-16-12]. Dostupné na URL: <*http://www.uamt.feec.vutbr.cz/vision/TEACHING/MPOV/07%20-%20Detekce%20a%20parametrizace%20geometrickych%20tvaru.pdf*>.

[6] Hough transform. *Wikipedia* [online]. Aktualizováno 2008-16-12 [cit. 2008-16-12]. Dostupné na URL: <http://en.wikipedia.org/wiki/HoughTransform>.

[7] RANSAC. *Wikipedia* [online]. Aktualizováno 2008-11-12 [cit. 2008-16-12]. Dostupné na URL: <http://en.wikipedia.org/wiki/RANSAC>.

[8] Elipsa. *Wikipedia* [online]. Aktualizováno 2009-07-05 [cit. 2009-10-05]. Dostupné na URL: <http://cs.wikipedia.org/wiki/Elipsa>.

[9] Bézierova křivka. *Wikipedia* [online]. Aktualizováno 2009-13-02 [cit. 2009-10-05]. Dostupné na URL: <http://cs.wikipedia.org/wiki/Bézierova_křivka>.

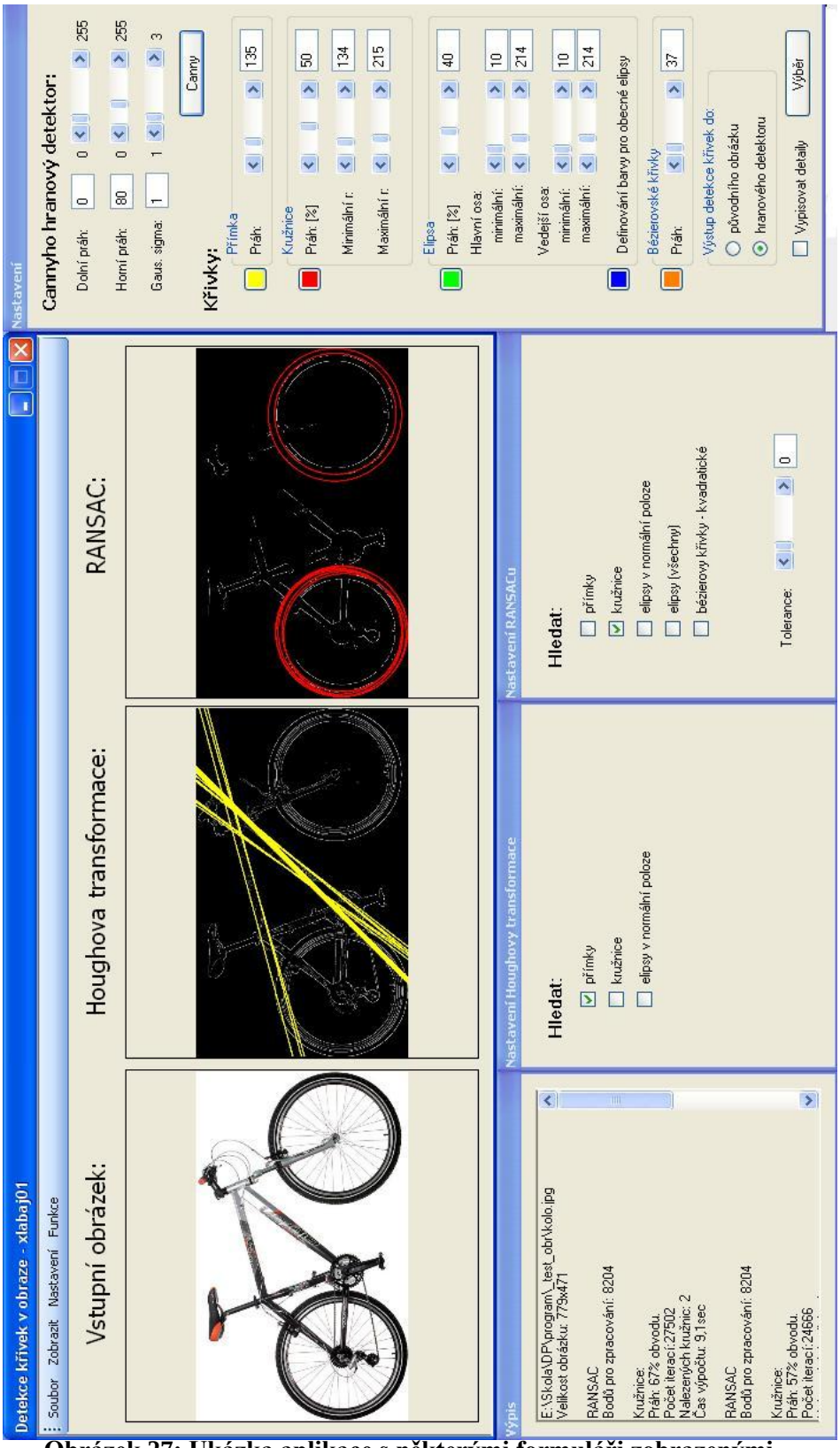
[10] **Kirillov Andrew.** AForge.NET Framework, *Google Code* [online]. [cit. 2009-10-05]. Dostupné na URL: <http://code.google.com/p/aforge/>.

[11] Main Page – Emgu CV, *Emgu CV* [online]. Aktualizováno 2009-29-04 [cit. 2009-10-05]. Dostupné na URL: <http://www.emgu.com/wiki/index.php/Main_Page>.

[12] .NET. *Wikipedia* [online]. Aktualizováno 2009-21-03 [cit. 2009-17-05]. Dostupné na URL: <http://cs.wikipedia.org/wiki/.NET>.

Seznam příloh

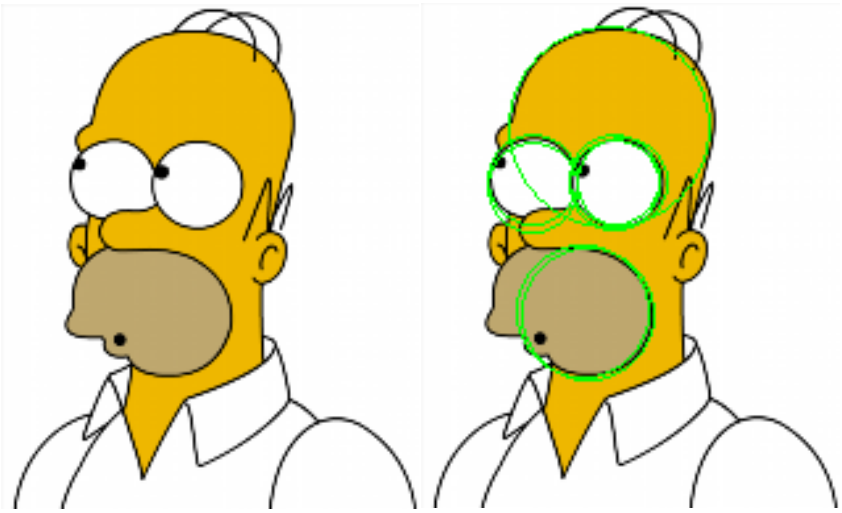
Příloha 1



Obrázek 27: Ukázka aplikace s některými formuláři zobrazenými.

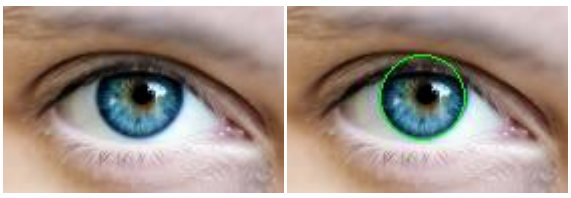
Příloha 2

Obrázek 28: Fotbalové hřiště – elipsy a přímky.



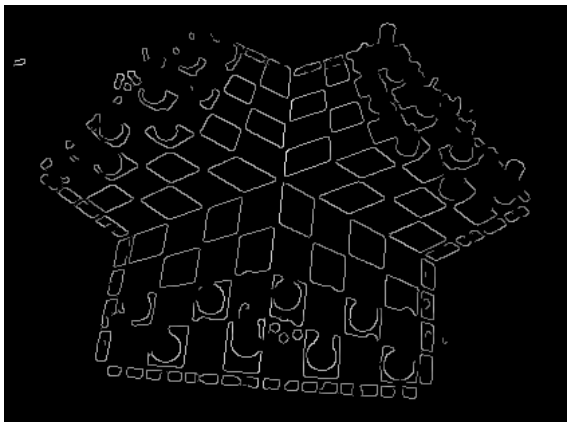
Obrázek 29: Homer Simpson – kružnice.

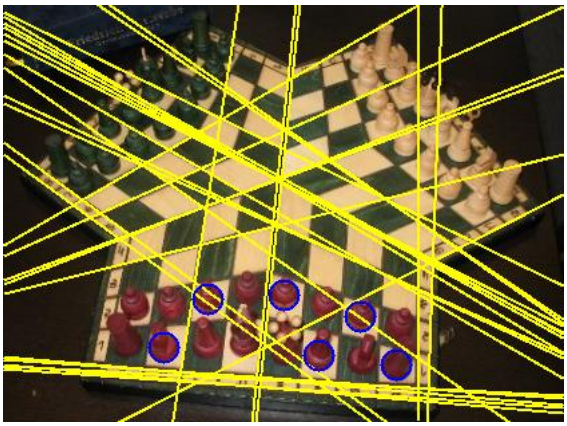
Obrázek 30: Kruhy v obilí – kružnice.



Obrázek 31: Detekce duhovky – kružnice.

Obrázek 32: Kruh z kol – elipsy.





Obrázek 33: Šachovnice pro 3 – přímky, kružnice.

Obrázek 34: Mince zvlášť – kružnice.

Obrázek 35: Mince na hromadě – kružnice.