Vaja 3: Robovi in Houghova transformacija

Laboratorijske Vaje

3. november 2011

V vaši delovni mapi kreirajte mapo '`vaja3/`', ter si s spletne strani predmeta vanjo razpakirajte datoteko '`vaja3.zip`'. Rešitve nalog boste pisali v Matlab/Octave skripte v mapi '`vaja3/`' in jih zagovarjali ob zagovoru nalog. Pri nekaterih nalogah so vprašanja, ki zahtevajo razmislek. Odgovore na ta vprašanja si zabeležite v pisni obliki in jih prinesite na zagovor. Na zagovoru morate znati teorijo, ki jo tukaj uporabljate, kakor tudi poznati delovanje funkcij, ki so že napisane za in so priložene v tekstu. Za dodatno razlago teorije, ki jo potrebujete v teh nalogah, glejte predavanja prof. Leonardisa, predvsem pa znanstveno literaturo s to temo [2, 5]. Verzijo knjige od Forsyth&Ponce lahko najdete tudi na spletu [3] – vsebina te vaje se nanaša na teorijo v poglavju 8 in 15.

1 Naloga: Določanje odvodov slike

V naslednjih treh nalogah se bomo posvetili problemu določanja robov v sliki. Robove iščemo tako, da analiziramo lokalne spremembe sivinskih nivojev v sliki. Matematično to pomeni, da *računamo odvode slike*. Slabost neposrednega odvajanja slike v neki točki je v tem, da zaradi prisotnosti šuma lokalne spremembe niso jasne in so tudi ocene odvodov šumne. V praksi zato sliko najprej zgladimo z majhnim filtrom, $I_b(x, y) = G(x, y) * I(x, y)$ in šele nato izračunamo odvod.

Navadno za filtriranje uporabimo Gaussov filter. Ker bomo v nadaljevanju uporabljali parcialne odvode, se naprej posvetimo dekompoziciji parcialnega odvoda Gaussovega jedra. Spomnimo se z Vaje 2, da lahko 2D Gaussovo jedro zapišemo kot produkt dveh 1D jeder:

$$G(x, y) = g(x)g(y), \tag{1}$$

filtriranje slike $I(x, y)$pa kot

$$I_b(x, y) = g(x) * g(y) * I(x, y). \tag{2}$$

Upoštevajoč naslednjo lastnost konvolucije $\frac{d}{dx}(g * f) = (\frac{d}{dx}g) * f$, lahko parcialni odvod *glajene slike* po x zapišemo kot

$$I_x(x, y) = \frac{\delta}{\delta x}[g(x) * g(y) * I(x, y)] = \frac{d}{dx}g(x) * [g(y) * I(x, y)]. \tag{3}$$

To pomeni, da vhodno sliko najprej filtriramo z Gaussovim jedrom po y osi in *rezultat filtriramo* z odvodom Gaussovega jedra po x osi. Podobno lahko izpeljemo drugi parcialni odvod po x, vendar si moramo zapomniti, da vedno pred odvajanjem sliko filtriramo. Drugi odvod po x je tako definiran kot parcialni odvod že odvajane slike:

$$I_{xx}(x, y) = \frac{\delta}{\delta x}[g(x) * g(y) * I_x(x, y)] = \frac{d}{dx}g(x) * [g(y) * I_x(x, y)]. \tag{4}$$

(a) Sledite zgornjim izpeljavam in izpeljite enačbe za izračun prvega in drugega odvoda po osi y, $I_y(x,y)$, $I_{yy}(x,y)$, ter mešanega odvoda $I_{xy}(x,y)$.

(b) Implementirajte funkcijo za izračun odvoda 1D Gaussovega jedra. Enačba odvoda Gaussovega jedra se glasi (za vajo lahko preverite analitično):

$$\frac{d}{dx}g(x) = \frac{d}{dx}\frac{1}{\sqrt{2\pi}\sigma}\exp(-\frac{x^2}{2\sigma^2}) \quad (5)$$

$$= -\frac{1}{\sqrt{2\pi}\sigma^3}x\exp(-\frac{x^2}{2\sigma^2}). \quad (6)$$

Jedro implementirajte v funkciji `gaussdx(x,sigma)`. Kot smo normalizirali diskretizirano Gaussovo jedro, je tudi priporočljivo, da normaliziramo odvod jedra. Vendar, ker je odvod Gaussovega jedra liha funkcija, ga normaliziramo tako, da je vsota absolutnih vrednosti elementov vsake od polovic enaka 1. Torej moramo jedro deliti z $0.5\sum abs(g_x(x))$.

```
function gd = gaussdx(x,sigma)
 ... % ne pozabite jedro na koncu normalizirati
```

(c) Lastnosti filtra lahko analiziramo preko tako imenovanega impulznega odziva filtra $f(x,y)$, ki je definiran kot konvolucija Dirakove $\delta(x,y)$ delte z jedrom $f(x,y)$: $f(x,y) * \delta(x,y)$. Zato najprej naredite sliko, ki ima vse vrednosti nič, razen centralnega elementa:

```
imgImp = zeros(25,25) ; imgImp(13,13) = 255 ;
```

Sedaj generirajte naslednji 1D jedri G in D:

```
sigma = 6.0;
x = [round(-3.0*sigma):round(3.0*sigma)];
G = gauss(x,sigma);
D = gaussdx(x,sigma);
```

Kaj se zgodi, če aplicirate naslednje operacije na sliko `imgImp` (ali je zaporedje ukazov pomembno?):

(a) Najprej konvolucija z G in potem konvolucija z G'.

(b) Najprej konvolucija z G in potem konvolucija z D'.

(c) Najprej konvolucija D in potem konvolucija z G'.

(d) Najprej konvolucija G' in potem konvolucija z D.

(e) Najprej konvolucija D' in potem konvolucija z G.

Izrišite si slike impulznih odzivov in uporabite '`imagesc`' za boljši prikaz. Rešitev pišite v skripto `vaja3_naloga1c.m`.

(d) Implementirajte funkcijo, ki uporablja vaši funkciji `gauss` in `gaussdx`, in izračuna parcialni odvod slike po x in parcialni odvod po y. Funkcijo preizkusite na sliki `lenaGray_mini.png` in izrišite rezultat odvajanja. Rešitev pišite v skripto `vaja3_naloga1d.m`.

```
function [Ix,Iy]=gaussderiv(img,sigma)
 ...
```

(e) Podobno kot zgoraj implementirajte funkcijo `gaussderiv2`, ki vrne parcialne odvode drugega reda. Funkcijo preizkusite na sliki `lenaGray_mini.png` in izrišite rezultate odvajanja. Rešitev pišite v skripto `vaja3_naloga1e.m`.

```
function [Ixx,Iyy,Ixy] = gaussderiv2(img,sigma)
 ...
```

(f) Implementirajte funkcijo `gradmag`, ki za vhod vzame sivinsko sliko `img`, vrne pa matriko magnitud odvodov `Imag` in matriko kotov odvodov `Idir` vhodne slike. Magnitude izračunate po formuli
$m(x,y) = \sqrt{(I_x(x,y)^2 + I_y(x,y)^2)}$, kote pa po formuli $\phi(x,y) = \arctan(I_y(x,y)/I_x(x,y))$. Namig: (i) uporabite matrično obliko za hitrejši izračun, (ii) v izračunu kotov se lahko ognete problemom deljenja z nič, če uporabite funkcijo `atan2(,)`. Funkcijo preizkusite na sliki `lenaGray_mini.png` in izrišite rezultate. Rešitev pišite v skripto `vaja3_naloga1f.m`.

```
function [Imag, Idir] = gradmag(img,sigma)
 ...
```

(g) Implementirajte funkcijo `laplace`, ki vrne Laplaca od Gaussa za vsak slikovni element. Laplace je definiran kot $L = \sigma^2(I_{xx}(x,y)+I_{yy}(x,y))$. Funkcijo preizkusite na sliki `lenaGray_mini.png` in izrišite rezultat. Rešitev pišite v skripto `vaja3_naloga1g.m`.

```
function L = laplace(img,sigma)
 ...
```

2 Naloga: Detekcija roba v sliki

(a) Na predavanjih ste se spoznali s Cannyjevim filtrom, ki je eden on najbolj razširjenih detektorjev robov v slikah. V nadaljevanju bomo implementirali preprostejši detektor robov, ki pa bo sicer demonstriral elemente Cannyjevega filtra. Implementirajte funkcijo `najdirobove.m`, ki vzame za parameter vhodno sliko, izračuna sliko magnitud gradientov `Imag` in vrne binarno sliko robov `Ie`, ki označuje magnitude večje od predpisane pragovne vrednosti `theta`:

$$Ie(x,y) = \begin{cases} 1 & ;\mathrm{Imag(x,y)} \geq \mathrm{theta} \\ 0 & ;sicer \end{cases} \quad (7)$$

Implementirajte naslednjo funkcijo, jo zaženite na sliki `lenaGray_mini.png` in si izrišite rezultat za nekaj vrednosti parametera `theta`. Ali lahko nastavite parameter `theta` tako, da so dobro vidni prav vsi robovi v sliki? To nalogo pišite v skripto '`vaja3_naloga2_a.m`'.

```
function Ie = najdirobove(I,sigma, theta)
% 1. izračunaj magnitude gradientov
% 2. upraguj magnitude ;
```

(b) Zgornja koda nam vrne sliko v kateri so robovi pogosto debeline nekaj slikovnih elementov. V praksi pa si pogosto želimo, da bi bil odziv roba velikosti zgolj enega slikovnega elementa. Uporabite v `najdirobove` spodnjo funkcijo in z njo filtrirajte `Imag` pred upragovanjem. Spodnja koda demonstrira preprosto implementacijo dušenja lokalnih nemaksimumov (angl. nonmaxima suppression). Za vsak slikovni element pogleda ali se pravokotno na lokalni rob v magnitud odvodov `Imag` nahaja element z večjo magnitudo. Če se ne nahaja, potem sprejme trenutno lokacijo za točko roba, sicer ne. Razložite si kaj počne posamezna vrstica v spodnji kodi. Preizkusite dopolnjeno funkcijo `najdirobove` na sliki `lenaGray_mini.png`. To nalogo pišite v skripto '`vaja3_naloga2_b.m`'.

```
function imgMax=nonmaxsup(imgMag,imgDir)
 [h , w] = size(imgMag);
 imgMax = zeros(h,w);
 offx = [-1 -1 0 1 1 1 0 -1 -1];
 offy = [ 0 -1 -1 -1 0 1 1 1 0];
 for y = 1 : h
  for x = 1 : w
    dir = imgDir(y,x); % poglej orientacijo slik. elementa
    idx = round(((dir+pi)/pi)*4) + 1; % pretvori orientacijo za vpogledno tabelo
    y1 = y + offy(idx) ; x1 = x + offx(idx) ;
    y2 = y - offy(idx) ; x2 = x - offx(idx) ;
    % omejitve koordinat
    x1 = max([1,x1]) ; x1 = min([w,x1]) ;
    y1 = max([1,y1]) ; y1 = min([h,y1]) ;
    x2 = max([1,x2]) ; x2 = min([w,x2]) ;
    y2 = max([1,y2]) ; y2 = min([h,y2]) ;
    % preveri ali je lokalni max
    if( (imgMag(y,x) >= imgMag(y1,x1))&&(imgMag(y,x) >= imgMag(y2,x2)) )
            imgMax(y,x) = imgMag(y,x) ;
     end
    end
 end
```

3 Naloga 3: Houghova transformacija

V tej nalogi boste implementirali Houghov transform, ki ste ga na predavanjih obravnavali kot metodo za iskanje parametriziranih oblik (npr., ravne črte in krogi) v slikah. V nadaljevanju bomo na kratko obnovili bistvo tega pristopa za iskanje premic v sliki, za več detajlov pa poglejte zapiske s predavanj in literaturo [**?**], kakor tudi spletne applete, ki demonstrirajo delovanje Houghovega transforma, npr., [1, 4].

Zamislimo si neko točko $p_0 = (x_0, y_0)$ na sliki. Če vemo, da je enačba premice $y = mx + c$, katere vse premice potekajo skozi točko p_0? Odgovor je preprost: vse premice, katerih parametra m in c ustrezata enačbi $y_0 = mx_0 + c$. Če si fiksiramo vrednosti (x_0, y_0), potem zadnja enačba opisuje zopet premico, vendar tokrat v prostoru (m, c). Če si zdaj zamislimo novo točko $p_1 = (x_1, y_1)$, njej prav tako ustreza premica v prostoru (m, c), in ta premica se seka s prejšnjo v neki točki (m', n'). Točka (m', c') pa ravno ustreza parametrom premice v (x, y) prostoru povezuje točki p_0 in p_1.

Torej, če želimo poiskati vse premice v sliki, moramo slediti sledečem pristopu. Parametrični prostor (m, c) najprej kvantiziramo v matriko *akumulatorjev*. Za vsak slikovni element, ki je kandidat za rob v vhodni sliki, "narišemo" pripadajočo premico v prostoru (m, c) in inkrementiramo akumulatorje preko katerih ta premica poteka. Vsi slikovni elementi, ki ležijo na isti premici v vhodni sliki bodo generirali premice v prostoru (m, c), ki se bodo sekale v isti točki in tako poudarile vrednost pripadajočega akumulatorja. To pomeni, da lokalni maximumi v (m, c) prostoru določajo premice, na katerih leži veliko slikovnih elementov v (x, y) prostoru.

V praksi je zapis premice v odvisnosti od m in n neučinkovit, še posebej, ko gre za navpične črte, saj takrat postane m neskončen. Temu problemu se preprosto ognemo tako, da premico parametriziramo s polarnimi koordinatami

$$x\cos(\theta) + y\sin(\theta) = \rho. \tag{8}$$

Postopek iskanja parametrov z akumulatorji je nespremenjen, razlika je samo v tem, da točka v (x, y) prostoru generira namesto premice sinusoido v prostoru (θ, ρ).

V naslednjih podnalogah bomo postopoma implementirali Houghovo transformacijo za iskanje premic v sliki. Implementirajte funkcijo `moj_houghTransform.m`, ki bo vaša glavna funkcija:

```
function [out_ro, out_theta]=moj_houghTransform(Ie,nBinsRo,nBinsTheta,thresh)
 ...
```

Vmesne segmente vaše kode lahko testirate na dveh zelo preprostih slikah `enapermica.png` in `pravokotnik.png`. Sledite korakom:

(a) Zgradite matriko akumulatorjev `A` za parametrični prostor (ρ, θ). θ naj teče od $-\pi/2$ do $\pi/2$, ρ pa naj teče od $-D$ do D, kjer je D dolžina diagonale slike. Parametra `nBinsRo` in `nBinsTheta` naj določata število celic vzdolž parametrov ρ in θ. Na začetku inicializirajte matriko akumulatorjev in vrednosti postavite na nič.

(b) Za vsak slikovni element v sliki, ki je kandidat za rob, generirajte krivuljo v prostoru (ρ, θ) preko enačbe (8) za vse možne vrednosti θ in inkrementirajte pripadajoče akumulatorje. Prikažite si 2D matriko akumulatorjev Houghovega prostora kot 2D sliko z `imagesc`.

(c) Opazite lahko, da se zaradi šuma in kvantizacije krivulje v Houghovem prostoru ne sekajo nujno v enem samem akumulatorju. Zato implementirajte funkcijo `nonmaxsuppression2D.m`, ki bo postavila na nič vse akumulatorje, ki niso lokalni maksimumi. To dosežete tako, da za vsako celico preverite ali je njena vsebina manjša od njenih 8 neposrednih sosedov. Če je večja ali enaka, jo sprejmete kot lokalni maksimum. Tako filtrirano matriko akumulatorjev si zopet izrišite v kot 2D sliko.

```
function imgResult=nonmaxsuppression2D(imgHough)
 ...
```

(d) Preiščite fitriran Houghov prostor in si zapomnite vse vrednosti (ρ, θ) katerih pripadajoči akumulatorji so večji od vrednosti `thresh`. Dobljene premice si lahko izrišete s funkcijo `narisi_premice(ro, theta, w, h)`, ki je priložena nalogi. Ker je parameter `thresh` odvisen od tega koliko slikovnih elementov pade v celico, to pa je odvisno od velikosti slike in kvantizacije, uredite premice po velikosti vrednosti pripadajočih akumulatorjev. Izrišite si tri najmočnejše premice.

(e) Naložite sliko `examGrade.jpg`. *Sliko spremenite v sivinsko*, in na njej detekritajte robove s funkcijo `najdirobove()`, ki ste jo implementirali v Nalogi 2b. Na tej sliki robov izvršite vašo Houghovo transformacijo in si izrišite tri premice, ki ustrezajo prvim trem lokalnim maksimumom v Houghovem prostoru. To vajo pišite v skripto '`vaja3_naloga3e.m`'. Poskusite spreminjati število celic v akumulatorju. Kako to vpliva na premice, ki jih najdete?

(f) Problem Houghovega transforma je v tem, da za vsak parameter našega modela potrebujemo novo dimenzijo v matriki akumulatorjev, kar pri kompleksnejših parametričnih modelih močno upočasni iskanje. Takim problemom se lahko ognemo, če nam uspe zmanjšati parametrični prostor. Spomnimo se, da smo se v Nalogi 2 poleg magnitude gradienta naučili izračunavati tudi smer (θ) gradienta (ki je pravokoten na rob). To smo zapisali v matriki `Idir`. Če poznamo θ nam v Houghovem prostoru ni treba tabelirati preko vseh možnih vrednosti *theta* ampak izračunamo pripadajoči ρ samo za enega. Razširite vašo kodo tako, da bo vrednost kota na nekem slikovnem elementu prebrala direktno z `Idir`. Upoštevajte tudi, da ste `Idir` izračunali z uporabo `atan2(dy,dx)`, ki daje vrednosti med

$-\pi$ in π. Zato morate najprej primerno zasukati kote določenih elementov v `Idir`, da bodo ležali na intervalu $[-\pi/2, \pi/2]$. Rezultat preizkusite na sliki '`examGrade.jpg`'. To vajo pišite v skripto '`vaja3_naloga3f.m`'. *V razmislek: Včasih se pojavijo premice, ki ne ležijo na ravnih linijah v sliki. Zakaj je temu tako? Kakšno postprocesiranje bi uporabili, da bi se znebili teh odvečnih premic?*

(g) Samo v razmislek, brez implementacije: kako bi spremenili vašo kodo za Houghov transform, da bi lahko iskali krožnice s poljubnim radijem v sliki?

Literatura

[1] C. Brechbühler. Interactive hough transform. `http://users.cs.cf.ac.uk/Paul.Rosin/CM0311/dual2/hough.htm`.

[2] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2002.

[3] D. A. Forsyth and J. Ponce. Computer vision: A modern approach (online version). `http://www.cs.washington.edu/education/courses/cse455/02wi/readings/book-7-revised-a-indx.pdf`, 2003.

[4] M. A. Schulze. Circular hough transform. `http://www.markschulze.net/java/hough/`, 2003.

[5] R. E. Woods, R. C. Gonzalez, and P. A. Wintz. *Digital Image Processing, 3 ed.* Pearson Education, 2010.