Vaja 4: Značilne točke, Ujemanje, Homografije

Laboratorijske Vaje

3. december 2011

Povzetek

V nalogi se bomo posvetili problemu avtomatskega iskanja korespondenc med slikami. Korespondence v slikah se v praksi uporabljajo za samodejno grajenje panoram z večjega števila slik, iskanje deformabilnih objektov v slikah ali pri poravnavi več zajetih posnetkov iste scene v različnih časovnih obdobjih. Vse te metode temeljijo na iskanju značilnih točk v sliki. Te značilne točke so oznake tistih struktur v sliki, za katere je velika verjetnost, da jih bomo detektirali, če tudi slikamo sceno pod malce dugačnimi pogoji ali kotom opazovanja. V prvih dveh nalogah bomo zato implementirali dve metodi iskanja značilnih točk, ki sta osnova za vse naprednejše metode. Nato se bomo v tretji in četrti nalogi posvetili zapisu značilnih točk in iskanju ujemanj med značilnimi točkami v različnih slikah. V zadnji nalogi bomo uporabili naše detektorje/opisnike za izgradnjo preprostega postopka za avtomatsko poravnavo slik.

1 Naloga: Hessov detektor značilnih točk

Demonstracijsko kodo za to vajo pišite v skripto '`vaja4_naloga1.m`'. V tej vaji bomo implementirali Hessov detektor značilnih točk [1](str. 44), ki ste ga obravnavali na predavanjih. Detektor temelji na matriki drugih odvodov $\mathbf{H}(x, y)$ (imenovani Hessova matrika) pri koordinatah slike (x, y):

$$\mathbf{H}(x,y) = \begin{bmatrix} I_{xx}(x,y;\sigma) & I_{xy}(x,y;\sigma) \\ I_{xy}(x,y;\sigma) & I_{yy}(x,y;\sigma) \end{bmatrix}, \tag{1}$$

kjer smo s σ posebej izpostavili, da so drugi odvodi izračunani na *glajenih slikah*. Hessov detektor izbere (x, y) lokacijo v sliki za značilno točko, če je determinanta Hessove matrike večja od predpisane pragovne vrednosti t. Če želimo odziv Hessovega detektorja narediti neodvisnega od skale (od velikosti Gaussovega filtra s katerim filtriramo), moramo vključiti še normalizacijski faktor σ^4 in dobimo pravilo:

$$det(\mathbf{H}(x,y)) = \sigma^4(I_{xx}(x,y;\sigma)I_{yy}(x,y;\sigma) - I_{xy}(x,y;\sigma)^2) > t. \tag{2}$$

(a) Implementirajte funkcijo `hessov_detektor.m`, ki izračuna Hessovo determinanto po formuli (2) za vsak slikovni element vhodne slike. Splača se vam razmisliti, kako implementirati zadevo kot vektorsko operaciijo, namesto uporabe `for...end` zank. Funkcijo zaženite na `data/graf0.png` (ne pozabite je pretvoriti v sivinsko sliko), in rezultat vizualizirajte.

```
function I_hess = hessov_detektor( I, sigma )
 ...
```

(b) Razširite zgornjo funkcijo `hessov_detektor` tako, da izvede še dušenje ne-lokalnih maximumov (angl., non-maximum suppression) na sliki determinant Hessove matrike in vrne koordinate točk, pri katerih je determinanta večja od pragovne vrednosti `thresh`. V `px` in `py` vrne `x` in `y` koordinate značilnih točk.

```
function [px, py] = hessov_detektor( I, sigma, thresh )
 ...
```

(c) Uporabite spodnjo funkcijo `drawpoints` za izris detektiranih točk preko originalne slike. Naložite si sliko `data/graf.png` (ali manjšo sliko `data/graf0.png`), zaženite vašo kodo in izrišite sliko hessove determinante ter vhodno sliko z detektiranimi značilnimi točkami (uporabite `subplot()`). Za začetek nastavite vrednosti parametrov na `thresh=100` in $\sigma = 1$, lahko pa jih tudi spreminjate, da dobite občutek za parametere. Na kakšnih strukturah v sliki se pojavljajo značilne točke?

```
function h=drawpoints(img, px, py, color)
 imagesc(img); colormap gray; hold on ;
 for i=1:length(px)
    plot( px(i),py(i),strcat(color,'+') );
 end
 hold off ;
```

2 Naloga: Harrisov detektor značilnih točk

To vajo pišite v skripto '`vaja4_naloga2.m`'. Harrisov detektor značilnih točk temelji na tako imenovani avtokorelacijski matriki $\mathbf{C}$, ki meri koliko je okolica neke točke podobna sama sebi za majhne premike. Na predavanjih ste slišali, da Harrisov detektor izbere lokacijo (x, y) za značilno točko, če sta obe lastni vrednosti avtokorelacijske matrike, ki jo izračunamo v okolici (x, y), dovolj veliki. V grobem to pomeni, da se v okolici (x, y) nahajata dve dobro definirani približno pravokotni strukturi – torej oglišče. Avtokorelacijsko matriko lahko izračunamo preko prvih parcialnih odvodov slike, ki jih naknadno zgladimo z Gaussovim filtrom v okolici točke (x, y):

$$\mathbf{C}(x, y; \sigma, \tilde{\sigma}) = \sigma^2 \begin{bmatrix} G(x, y; \tilde{\sigma}) * I_x^2(x, y; \sigma) & G(x, y; \tilde{\sigma}) * I_x I_y(x, y; \sigma) \\ G(x, y; \tilde{\sigma}) * I_x I_y(x, y; \sigma) & G(x, y; \tilde{\sigma}) * I_y^2(x, y; \sigma) \end{bmatrix}, \quad (3)$$

kjer je $*$ operator konvolucije. Računanje lastnih vrednosti λ_1 in λ_2 matrike $\mathbf{C}(x, y; \sigma, \tilde{\sigma})$ je potratna operacija, zato raje uporabimo naslednje relacije[1]

$$\det(\mathbf{C}) = \lambda_1 \lambda_2 \quad (4)$$
$$\mathrm{trace}(\mathbf{C}) = \lambda_1 + \lambda_2 \quad (5)$$

in izračunamo razmerje $r = \lambda_1/\lambda_2$. Če upoštevamo

$$\frac{\mathrm{trace}^2(\mathbf{C})}{\det \mathbf{C}} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2\lambda_2} = \frac{(r+1)^2}{r}, \quad (6)$$

lahko izrazimo pogoj za prisotnost značilne točke na lokaciji (x, y) takole:

$$\det(\mathbf{C}) - \alpha \mathrm{trace}^2 \mathbf{C} > t. \quad (7)$$

V praksi za vrednosti parametrov uporabimo $\tilde{\sigma} = 1.6\sigma$, $\alpha = 0.06$. Kot v prejšnji nalogi razmislite, kako implementirati enačbo (7) brez eksplicitnega vmesnega izračuna avtokorelacijske matrike $\mathbf{C}$ in brez uporabe odvečnih `for ... end` zank.

(d) Implementirajte funkcijo `harrisov_detektor.m`, ki izračuna vrednost (7) za vse slikovne elemente, na rezultatu izvrši non-maximum suppression in vrne koordinate vseh točk (`[px, py]`), ki so nad pragovno vrednostjo `thresh`.

[1] Zaradi preglednosti bomo v nadaljevanju v oznaki za avtokorelacijsko matriko $\mathbf{C}(x, y; \sigma, \tilde{\sigma})$ spustili indekse $(x, y; \sigma, \tilde{\sigma})$ in namesto tega pisali kar $\mathbf{C}$.

```
function [px, py] = harrisov_detektor( I, sigma, thresh )
 ...
```

(e) Naložite sliko `data/graf.png` (ali manjšo `data/graf0.png`) in izračunajte Harrisove značilne točke. Rezultat primerjajte s Hessovimi značilnimi točkami. Eksperimentirajte z različnimi vrednostmi parametrov. Ali se pojavljajo točke obeh detektorjev na istih strukturah v sliki?

3 Naloga: Opisniki značilnih regij

Z značilnimi točkami iščemo *podobne strukture* v različnih slikah. To lahko uporabljamo npr. za poravnavanje panoramskih slik, detekcijo v naprej naučenih objektov v slikah, itd. Zato pa potrebujemo vizualne opisnike regij, ki obdajajo značilne točke. V tem delu naloge bomo implementirali nekaj preprostih opisnikov regij, ki smo jih obravnavali v sklopu 'Vaje 2'. To vajo pišite v skripto '`vaja4_naloga3.m`'.

(a) Kot primer imate spodaj funkcijo `descriptors_rg`, ki za vhod vzame vhodno sliko in vektor lokacij točk, izračuna pa (r, g) histogram v regiji velikosti $m \times m$ v okolici vsake točke (s pomočjo funkcije `histrg`). Histograme vrne kot vrstice v matriki `D`.

```
function D = descriptors_rg(img, px, py, m, bins)
  rad   = round((m-1)/2);
  [h w c] = size(img);
  D = zeros(length(px),bins^2);
  for i=1:length(px)
    minx = max([px(i)-rad,1]);
    maxx = min([px(i)+rad,w]);
    miny = max([py(i)-rad,1]);
    maxy = min([py(i)+rad,h]);

    imgWin = img(miny:maxy, minx:maxx, :);
    hist   = histrg(imgWin,bins);
    D(i,:) = hist';
  end;
end
```

Napišite podobno funkcijo za izračun deskriptorjev `descriptors_maglap.m`, ki izračuna *mag*, *lap* histogram (histogram magnitude odvodov in Laplaca Gaussa) okoli vsake značilne točke. Za izračun mag/lap histogramov uporabite priloženo funkcijo `h = histmaglap(img,sigma,bins)` (Za to funkcijo boste rabili nekatere metode (npr., `gradmag()`, `laplace()`), ki ste jih implementirali v Vaji 3. Tiste metode si skopirajte v delovno mapo Vaje 4 in popravite kodo za izračun histograma, če je to potrebno!).

(b) Napišite funkcijo, ki vzame kot vhod dva nabora deskriptorjev D_1 in D_2 istega tipa (npr., rg histogrami ali mag/lap histogrami) in poišče za vsak deskriptor v D_1 najbolj podoben deskriptor v D_2 z uporabo Hellingerjeve razdalje:

```
function [Idx, Dist] = findnn_hellinger( D1, D2 )
 ...
```

V zgornji funkciji naj bo `Idx` vektor indeksov ujemanj, `Dist` pa razdalje/podobnosti ujemanj. Na primer, če imate `Idx(5)=7`, pomeni, da je petemu elementu v D_1 najbolj podoben sedmi element v D_2, razdalja med njima pa je `Dist(5)`.

Če želite, lahko implementirajte bolj splošno funkcijo, ki ji daste za parameter ime mere za primerjavo histogramov – te mere smo obravnavali v sklopu Vaje 2: (i) χ^2,

(ii) razdaljo L_2 in (iii) presek.[2]
in za vsako poiščite njej najbolj podobno med vsemi.

4 Naloga: Iskanje ujemanj

Sedaj imamo vse potrebne komponente, da implementiramo preprosto aplikacijo za iskanje ujemanj lokalnih regij. V mapi `data/graf5/` se nahaja referenčna testna slika in njene rotacije. Za izbrano rotirani sliki bomo poskusili poiskati korespondenčne točke v njeni referenčni (nerotirani) sliki.

(a) Najprej bomo preizkusili barvni opisnik regij. Napišite skripto `vaja4_naloga4a.m`, ki Naloži sliki `data/graff5/img1_0.ppm` ter `data/graff5/img2_0.ppm` in izvrši naslednje korake:

i. Izračuna Harrisov detektor značilnih točk na obeh slikah.

ii. Izračuna *mag/lap* histograme za vse detektirane točke.

iii. Poišče najboljša ujemanja med histogrami v levi in desni sliki s Hellingerjevo razdaljo, ki ste jih implementirali zgoraj (lahko tudi eksperimentirate z razdaljami). Najprej izračunajte za histograme v levi sliki najbolj podobne v desni, nato pa v desni poiščite najbolj podobne v levi sliki. Izberite samo tista ujemanja, za katera je to ujemanje "simetrično". Pravimo, da je ujemanje simetrično, če točka P_{1i} v levi sliki "izbere" za najbolj podobno točko P_{2j} v desni sliki in hkrati točka P_{2j} v desni sliki izbere za najbolj podobno točko P_{1i} v levi sliki. Tako dobimo podmnožico parov točk, kjer vsaki točki v levi sliki ustreza samo ena točka v desni in obratno.

iv. Uporabite spodnjo funkcijo `displaymatches` za prikaz `N=inf` simetričnih ujemanj. Kaj opazite pri izrisanih korespondencah? Za začetne vrednosti parametrov uporabite: velikost lokalnih regij za deskriptor `m=41` pikslov, število celic v histogramu `bins=16`.

```
function displaymatches(I1, px1, py1, I2, px2, py2, N)
% I1, I2 ... slika 1 in slika 2
% px1,py1 ... x in y koordinate koresp. točk v I1
% px2,py2 ... x in y koordinate koresp. točk v I2
% N ... število točk za prikaz
%       (negativen N pomeni naključna permutacija "-N" elementov)
 if N < 0
    N = min([-N, length(px1)]) ;
    rand_idx = randperm(length(px1)) ;
    Idx = rand_idx(1:N) ;
 else
    N = min([N, length(px1)]) ;
    Idx = 1 : N ;
 end
 px1 = px1(Idx) ; py1 = py1(Idx) ; px2 = px2(Idx) ; py2 = py2(Idx) ;
 [h1, w1, cn1] = size(I1) ; [h2, w2, cn2] = size(I2) ;
 h = max([h1, h2]) ; w = w1 + w2 + 1 ; cn = max([cn1,cn2]) ;
 I = zeros(h,w,cn) ;
 I(1:h1,1:w1,1:cn1) = I1 ;
 I(1:h2,w1+1:w1+w2,1:cn2) = I2 ;
```

[2]Za testiranje in razhroščevanje lahko uporabite kar sliko `data/graf1.png`. Detektirajte vse točke na sliki `graf1.png` in njene deskriptorje spravite v D_1, ter za razhroščevanje kar iste deskriptorje spravite tudi v D_2. Tako boste videli, če vam vaša funkcija vrača smiselne rezultate v `Idx` in `Dist`.

```
clf ; imagesc(uint8(I)) ; hold on ;
axis equal ; axis tight ; colormap gray ;
plot([w1+1,w1+1],[0,h], 'r', 'LineWidth', 4) ;
px2 = px2 + w1+1 ; plot([px1', px2'], [py1', py2'], 'oy') ;
for i = 1 : length(px1)
  plot([px1(i),px2(i)], [py1(i),py2(i)], 'g-') ;
end
```

(b) Napišite funkcijo `M=najdi_vsa_ujemanja(Slika1,Slika2)`, kateri daste za vhod dve sliki, ta pa vrne v matriki M vse ujemajoče pare (Slike1 v Slika2) in za vsak par tudi podobnost histogramov. Matrika M naj vsebuje te stolpce: prva dva stoplca x, y koordinate značilnih točk v prvi sliki, druga dva stolpca x, y koordinate korespondenčnih značilnih točk v drugi sliki, zadnji stolpec razdalja/podobnost med pari korespondenčnih točk. Namesto Harrisovega detektorja naj funkcija uporablja Hessov detektor. Ponovite eksperiment s prejšnje točke, izrišite vsa simetrična ujemanja, rezultat pa napišite v skripto '`vaja4_naloga4b.m`'.

5 Naloga: Ocenjevanje Homografije

To vajo pišite v skripto '`vaja4_naloga5.m`'. Ker naše testne slike s prejšnje naloge vsebujejo samo planarne (ravninske) scene, lahko poskusimo oceniti homografijo $\mathbf{H}$, ki preslika eno sliko v drugo. V nadaljevanju bomo na kratko ponovili metodo za iskanje takih preslikav, več detajlov pa najdete v literaturi za vaš predmet [1]. V literaturi se metoda imenuje direktna linearna transformacija (DLT).

Za referenčno točko $\mathbf{x}_r$ v prvi sliki in njen ekvivalent $\mathbf{x}_t$ v transformirani sliki, lahko napišemo transformacijo kot:

$$\begin{aligned} \mathbf{H}\mathbf{x}_r &= \mathbf{x}_t \\ \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} &= \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} \quad ; \quad \text{pri} \quad \frac{1}{z'_t} \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix}, \end{aligned} \tag{8}$$

kjer smo točki $\mathbf{x}_r$ in $\mathbf{x}_t$ zapisali v *homogenih koordinatah*. Na hitro rečeno, homogene koordinate za 3D točko dobimo tako, da ji četrto koordinato postavimo na 1. Z enačb (8) dobimo sistem linearninh enačb z osmimi neznankami $h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}$:

$$\frac{h_{11}x_r + h_{12}y_r + h_{13}}{h_{31}x_r + h_{32}y_r + 1} = x_t \tag{9}$$

$$\frac{h_{21}x_r + h_{22}y_r + h_{23}}{h_{31}x_r + h_{32}y_r + 1} = y_t, \tag{10}$$

kar lahko spremenimo v

$$h_{11}x_r + h_{12}y_r + h_{13} - x_t h_{31}x_r - x_t h_{32}y_r - x_t = 0 \tag{11}$$

$$h_{21}x_r + h_{22}y_r + h_{23} - y_t h_{31}x_r - y_t h_{32}y_r - y_t = 0. \tag{12}$$

Če želimo oceniti 8 neznanih parametrov homografije, potrebujemo vsaj 4 pare ujemajočih točk. Natančnost ocenjene homografije pa lahko povečamo, če uporabimo večje število ujemajočih točk $(\mathbf{x}_1, \mathbf{x}'_1), \ldots, (\mathbf{x}_n, \mathbf{x}'_n)$. S tem dobimo *predeterminiran sistem* enačb:

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{13}$$

$$\begin{bmatrix} x_{r1} & y_{r1} & 1 & 0 & 0 & 0 & -x_{t1}x_{r1} & -x_{t1}y_{r1} & -x_{t1} \\ 0 & 0 & 0 & x_{r1} & y_{r1} & 1 & -y_{t1}x_{r1} & -y_{t1}y_{r1} & -y_{t1} \\ x_{r2} & y_{r2} & 1 & 0 & 0 & 0 & -x_{t2}x_{r2} & -x_{t2}y_{r2} & -x_{t2} \\ 0 & 0 & 0 & x_{r2} & y_{r2} & 1 & -y_{t2}x_{r2} & -y_{t2}y_{r2} & -y_{t2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{rn} & y_{rn} & 1 & 0 & 0 & 0 & -x_{tn}x_{rn} & -x_{tn}y_{rn} & -x_{tn} \\ 0 & 0 & 0 & x_{rn} & y_{rn} & 1 & -y_{tn}x_{rn} & -y_{tn}y_{rn} & -y_{tn} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}, \quad (14)$$

ki ga lahko rešimo kot minimizacijo srednje kvadratične napake. Če je $\mathbf{A}$ kvadratna matrika, potem dobimo eksaktno rešitev. V primeru prederminiranega sistema (npr., $n > 4$), pa matrika $\mathbf{A}$ ni diagonalna. Ta problem klasično [1] rešujemo preko psevdo-inverza $\mathbf{A}^T\mathbf{A}$, ki je kvadraten in ga lahko razcepimo na lastne vektorje in lastne vrednosti. Rešitev sistema je enotski lastni vektor od $\mathbf{A}^T\mathbf{A}$, ki ustreza najmanjši lastni vrednosti (v Matlab/Octave ukaz `eig`). Enako rešitev lahko učinkoviteje dobimo preko SVD (dekompozicije po singularnih vrednostih) kot enotski lastni vektor, ki pripada najmanjši lastni vrednosti A (v Octave ukaz `svd`).

$$\mathbf{A} \overset{svd}{=} \mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{U} \begin{bmatrix} \lambda_{11} & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_{99} \end{bmatrix} \begin{bmatrix} v_{11} & \dots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T \quad (15)$$

Elemente homografije $\mathbf{h}$ tako dobimo z zadnjega stolpca $\mathbf{V}$, in ker zahtevamo $\mathbf{h}_{33} = 1$, normaliziramo z vrednostjo v_{99}:

$$\mathbf{h} = \frac{[v_{19}, \dots, v_{99}]^T}{v_{99}}. \quad (16)$$

(a) Napišite funkcijo `oceni_homografijo`, ki aproksimira homografijo med dvemi slikami s pomočjo nabora ujemajočih se točk v levi in desni sliki:

   (a) Zgradite matriko $\mathbf{A}$ po enačbi (13).
   (b) Izvedite dekompozicijo matrike $\mathbf{A}$ s SVD: `[U,S,V]=svd(A)`.
   (c) Izračunajte $\mathbf{h}$ z enačbo (16).
   (d) Preuredite elemente $\mathbf{h}$ v 3×3 matriko $\mathbf{H}$ z uporabo funkcije `reshape`.

   Implementirajte novo funkcijo `oceni_homografijo`:

```
function H = oceni_homografijo(px1, py1, px2, py2)
  ...
```

(b) Naložite si dve sliki New Yorka posnetega z zraka, `data/NewYork/im1_small.pgm` in `data/NewYork/im5_small.pgm`, in štiri ročno označene koresponenčne točke `data/newyork_1_to_5_points_small.txt`[3] za ti dve sliki. Uporabite funkcijo `displaymatches(I1,px1,py1,I2,px2,py2,[])` in prikažite ujemajoče se pare točk v slikah. S temi pari ocenite homografijo H s slike `im1` v `im2` in uporabite `showTransformedImage(in1,H)` za transformacijo in prikaz slike `im1` preko homografije. Vizualizirajte si rezultat. Poskusite še s slikami: `data/img/img0.pgm`, `data/img1.pgm` in točkami `data/img_0_to_1_points.txt`

(c) Ko vam ocenjevanje homografije deluje pravilno, ponovite postopek v zgornji nalogi, vendar tokrat uporabite vašo funkcijo `M=najdi_vsa_ujemanja(,)`, da določite ujemanja avtomatsko. Prvih 10 najboljših ujemanj uporabite za izračun homografije in si vizualizirajte rezultat s transformacijo prvotne slike, prav tako pa si izrišite

[3] V datoteki sta prvi dve vrstici (x,y) koordinate točk v prvi sliki, drugi dve vrstici pa korespondenčne koordinate v drugi sliki

tudi ujemanja. Kako dobro ste uspeli oceniti homografijo? Ali se ocena izboljša, če uporabite več/manj detektiranih parov točk? Kaj se zgodi, če se med ujemanji pojavijo nepravilna ujemanja? Nastavite parametre tako (izbira detektorja, opisnika, parametri detektorja in opisnika, število uporabljenih točk, itd.), da vam metoda deluje najboljše – od kvalitete delovanja bo odvisna ocena te naloge.

Literatura

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