

Laboration 5

Stereoproblemet

Laboration i Bilder och Grafik

Gunnar Farnebäck

februari 2001
reviderad december 2002

1 Syfte

Laborationens syfte är att i matlab implementera och testa en enkel men komplett disparitetsestimeringsalgoritm, vilken är beskriven i kursmaterialet om stereoproblemet, och därigenom få fram djupestimater från ett stereopar. Testningen görs på det stereopar som används som exempel i kursmaterialet.

2 Stereopar

Stereoparet laddas in i matlab genom att göra `load treeimages`. Detta kommer att placera vänstra bilden i matrisen `il` och högra bilden i matrisen `ir`.

3 Uppgifter

1. Implementera polynomexpansion i en enstaka punkt i enlighet med stycke 4 i kursmaterialet.
2. Testa för en valfri punkt i en av bilderna att din egen implementering ger samma resultat som funktionen `polyexp` (beskriven nedan).
3. Implementera disparitetsestimeringsalgoritmen som är beskriven i stycke 6 i kursmaterialet.
4. Testa algoritmen på stereoparet. Undersök vilken effekt det har att variera parametrarna N_1 , σ_1 , N_2 och σ_2 . Försök att få resultat som är jämförbara med (eller bättre än) dem i kursmaterialet.

4 Särskilda matlabfunktioner

För den här laborationen finns tre specialskrivna matlabfunktioner tillgängliga:

polyexp Snabb implementation av polynomexpansion i varje punkt i bilden. Gör `help polyexp` för att ta reda på hur den används. Observera att koordinatsystemet är riktat så att x_1 är nedåt och x_2 åt höger.

show_image Praktiskt gränssnitt mot funktionen `image` för att visa en matris som en bild. `show_image(im)` automatskalär värdena i matrisen `im`. `show_image(im, [a b])` trunkerar värden utanför intervallet $[a, b]$ och skalar bilden till det givna intervallet.

disparity_image Specialfunktion för att visa färgkodade disparitetsvärden överlagrat på en gråbild. Anropet är `disparity_image(im, disparity, [a b])` där `im` är gråbilden, `disparity` innehåller disparitetsvärdena och $[a, b]$ är det disparitetsintervall som ska färgkodas.

5 Matlabtips

- Gör `help` och eventuellt också `type` på alla kommandon som du undrar hur de fungerar eller hur de är implementerade.
- Figurer är alltid värdefulla för att inspektera resultat och mellanresultat. Förutom de specialfunktioner som nämndes ovan kan `plot`, `mesh` och `surf` vara användbara.
- Lösning av ekvationssystemet $\mathbf{Ax} = \mathbf{b}$ utförs i matlab med fördel av `A\b`.
- Enklaste sättet att räta ut en matris `A` till en vektor är konstruktionen `A(:)`. Omvända operationen kräver kommandot `reshape`. En vektor kan placeras i diagonalen av en matris (som i övrigt är noll) med hjälp av `diag`. Kommandot `size` talar om hur stor en matris/vektor/array är.
- För att lagra till exempel en 2×2 -matris i varje punkt i en bild är det lämpligt att använda 4-dimensionella arrayer av storlek $M \times N \times 2 \times 2$, där $M \times N$ är bildstorleken. Bland annat ger kommandot `polyexp` sina resultat på den här formen. En liten fälla är att `A(m,n,:,:)` visserligen extraherar 2×2 -matrisen som hör till punkten (m, n) , men resultatet blir inte en matris utan en array av storlek $1 \times 1 \times 2 \times 2$. `squeeze(A(m,n,:,:))` eliminerar de överflödiga dimensionerna och ger den önskade matrisen.
- `meshgrid` är användbart för att skapa koordinatmatriser. Speciellt är

```
[x,y] = meshgrid(-a:a, -a:a);  
g = exp(-(x.^2 + y.^2)/(2*sigma^2));
```

 ett enkelt sätt att skapa en 2-dimensionell Gaussfunktion.
- Faltning av 2-dimensionella signaler görs med funktionen `conv2`. I bildsammanhang vill man i stort sett alltid använda den på formen `conv2(im, kernel, 'same')` för att få ett resultat av samma storlek som bilden `im`.

6 Prestandatips

- Underlätta för matlabs minneshantering genom att preallokera stora matriser som fylls i efter hand. Till exempel är hastighetsskillnaden mellan

```
clear a;
for i = 1:100000
    a(i) = i^2;
end
och
a = zeros(1,100000);
for i = 1:100000
    a(i) = i^2;
end
större än en faktor 100.
```

- Loopar i matlab är långsamma. Exemplet ovan kan skrivas om som

```
a = (1:100000).^2
och går då ytterligare en faktor 100 snabbare. Det är inte alla loopar som
kan skrivas om på liknande sätt, men det är möjligt oftare än man tror. I
koden nedan är A en  $M \times N \times 2 \times 2$ -array, b en  $M \times N \times 2$ -array och vi
vill för varje punkt beräkna produkten av en  $2 \times 2$ -matris och en vektor
av längd 2. Med en dubbelloop låter detta sig göras enkelt
c = zeros(M,N,2);
for m = 1:M
    for n = 1:N
        AA = squeeze(A(m,n,:,:));
        bb = squeeze(b(m,n,:));
        c(m,n,:) = AA * bb;
    end
end
```

men tyvärr ganska långsamt. Genom att uttrycka multiplikationen explicit i de ingående elementen så kan vi snabba upp operationen genom att göra beräkningen i alla punkter parallellt, till priset av något mindre lättgenomskådad kod.

```
c(:,:,1) = A(:,:,1,1) .* b(:,:,1) + A(:,:,1,2) .* b(:,:,2);
c(:,:,2) = A(:,:,2,1) .* b(:,:,1) + A(:,:,2,2) .* b(:,:,2);
```

I laborationen är det aktuellt att lösa ekvationssystem snarare än att multiplicera, men principen är densamma. Det är lämpligt att i förväg räkna ut den explicita lösningen till

$$\begin{pmatrix} a & c \\ c & b \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} d \\ e \end{pmatrix}.$$

- 2-dimensionell faltning kan snabbas upp väsentligt om faltningskärnan är kartesiskt separabel. Närmare bestämt gäller det att om $\mathbf{h} = \mathbf{h}_1 \mathbf{h}_2^T$, där \mathbf{h} är en kvadratisk matris och \mathbf{h}_1 och \mathbf{h}_2 är kolumnvektorer av samma längd så kan

```
conv2(im, h, 'same')
```

bytas ut mot

```
conv2(conv2(im, h1, 'same'), h2, 'same')
```

för att snabba upp koden. Det är bra att komma ihåg att Gausskärnan är kartesiskt separabel.